



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 2

(Due Oct 14th, 2016)

1. Introduction

A lot of college students are confused on this question: Where is my money? I always control my expense, why I always have no money on the end of the month? As everyone knows, a lot of college and university student are financial independence, or at least they need make a plan for their expense. But now there are a lot of student don't know where their money has been used, they always feel they have a lot of money at the beginning of the month or when the salary just been issued. But at the end of the month, they always have no money and drive a hard bargain for each expense. The most important reason for this situation is that the student doesn't know how to plan and manage the expenses correctly. At the students' views, small amount of expenses is easily to be ignored, they always consider and manage their large amount of expenses such as the rental, traveling and so on. But they don't know most of the money always be wasted on lots of small amount of expenses. The expense for one meal looks very small, but if a student ate all the meal outside, it is also a very big expense. So how to plan and manage the expense is a very important question for the students.

But there is no any application which is analyzing and managing the expense that targeted to college students. For this phenomenon, we develop this expense control and management application, Pocket Manager. This "pocket" at here has two different meanings: It not only means that the application can help student to manage their money in their pocket, but also means that user can easily manage their expense in the phone which is very convenient to put in pocket. So this application can help students manage their "pocket money" whenever and wherever possible.

2. Objectives

2.1. Overall objective

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. Specific Objectives

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.

3. Features (Introduction)

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.

4. Existing Services/API

4.1. 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 a estimated future time to buy it.

Here are some most popular APIs from eBay list below.

4.1.1. Shopping API

This API provides a function for developers to access to the shopping information. There are several calling types which are quite useful in developing relevant application (see [Table 4-1](#)), such as finding products' information and shipping status.

Table 4-1. Basic eBay shopping API calls.

API Calls	Interpretation
FindProducts	It searches for stock product information (stock description and item specifics), such as information about a particular kind of product like iPhone.
GetCategoryInfo	This type of call can make you retrieve high level category information.
GetSingleItem	It gets publicly visible details about specific items.
FindItems	This type of call is the one we used during designing and achieve the Separating Wants from Needs part. We modified the API calling string and let it return JSON files instead of XML files, additional string "best match" also help us sort the result as we want.



4.1.2. Trading API

This kind of call will provide users the trading information, such as message, account information, my eBay selling items information, refund, feedback and so on (see [Table 4-2](#) for details).

Table 4-2. Basic eBay trading API calls.

API Calls	Interpretation
GetAccount	It returns a seller's invoice data for their eBay account, including the account's summary data.
GetAllBidders	This is the base request type for the GetAllBidders call, which is used to retrieve bidders from an active or recently-ended auction listing.
GetCategories	It retrieves the latest eBay category hierarchy for a given eBay site. Information returned for each category includes the category name and the unique ID for the category (unique within the eBay site for which categories are retrieved). A category ID is a required input when you list most items.
GetFeedback	It retrieves one, many, or all Feedback records for a specific eBay user.
GetItem	It retrieves item data such as title, description, price information, seller information, and so on, for the specified ItemID .
GetMyMessages	It retrieves information about the messages sent to a user.
GetOrders	It retrieves the orders for which the authenticated user is a participant, either as the buyer or the seller.
GetUserDisputes	It requests a list of disputes the requester is involved in as buyer or seller. eBay Money Back Guarantee Item Not Received and Significantly Not As Described cases are not returned with this call.



4.1.3. Finding API

It's difficult to tell all the differences among these three type of API, because some of the will provide the same result such as item id, description, pictures, prices. Finding call, also provide some specific string (see details in [Table X](#)).

Table X. Basic eBay finding API calls.

API Calls	Interpretation
findCompletedItems	It returns items whose listings are completed and are no longer available for sale on eBay.
findItemsAdvanced	It finds items by a keyword query and/or category and allows searching within item descriptions.
findItemsByCategory	It returns items in a specific category. Results can be filtered and sorted.
findItemsByKeywords	It returns items based upon a keyword query and returns details for matching items.
findItemsByProduct	It returns items based on ISBN, UPC, EAN, or ePID.
getSearchKeywordsRecommendation	It checks specified keywords and returns correctly spelled keywords for best search results.
GetUserDisputes	It returns items in eBay stores and can search a specific store or can search all stores with a keyword query.

4.2. 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.

The features of MLab MongoDB are listed in **Table X**.

Table X. Basic MLab features.

Features	Interpretation
Cloud	It creates databases and replica set clusters on-demand. Scale up and down with ease. Never think about machines and never install software.
SSDs Performance	It has 100% SSD-backed disks and dedicated MongoDB processes on all of our for-pay plans.
Maximum Uptime	Mongo DB has a team of expert database manage on-call to make sure the database stays up and healthy.
Expert Support	Also the expert team manages and optimizes more than 300,000 databases worldwide.
Total Data Protection	Mongo DB offers flexible, multi-zone replication options as well as the ability to create recurring backup plans (hourly/daily/weekly/monthly) and initiate one-time snapshots to any S3 bucket.
Web-based Database Management	MLab allows users to search, visualize, and modify data with ease as well as help you optimize your queries and indexes.



4.3. Google Charts - a powerful, simple to use, and free tool to customize chart for your application

Google Charts provides a solution to visualize data on your application. From simple line charts to complex hierarchical tree maps, the chart gallery includes a large amount of ready-to-use chart templates.

Users can load the Google Chart libraries, transfer the data to be charted, select options to customize your chart, and finally create a chart object with an id that you choose. The most popular way to use Google Charts is using the simple JavaScript that you defined in your web application. Then, later in the web page, users create `<div>` with that id to display the Google Chart.

Here are some examples of the google chart gallery; it provides some ready-to-use templates (see details in [Table X](#)).

Table X. Template graphs in Google Charts.

Features	Interpretation
Pie Chart	A pie chart that is rendered within the browser using SVG or VML. Displays tooltips when hovering over slices.
Bar Chart	Google bar charts are rendered in the browser using SVG or VML, whichever is appropriate for the user's browser. Like all Google charts, bar charts display tooltips when the user hovers over the data. Column Chart: A column chart is a vertical bar chart rendered in the browser using SVG or VML, whichever is appropriate for the user's browser. Like all Google charts, column charts display tooltips when the user hovers over the data.
Line Chart	A line chart that is rendered within the browser using SVG or VML. Displays tooltips when hovering over points.
Donut Chart	A donut chart is a pie chart with a hole in the center.
Org Chart	Org charts are diagrams of a hierarchy of nodes, commonly used to portray superior/subordinate relationships in an organization. A family tree is a type of org chart.



5. Detailed Design of Features

4.1. Wireframes/Mockups - **Mockup Tool: Balsamiq Mockups Version 3.5.5**

Figure X is the designed mockup of the home page of the *Pocket Manger* system. 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.

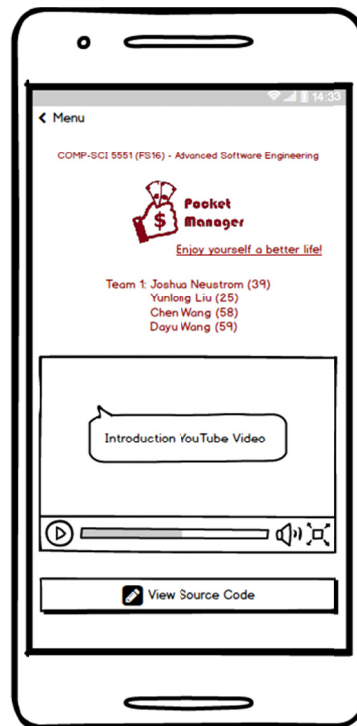


Figure X. Refined mockup of the home page of the *Pocket Manager* system.

Figure X 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.

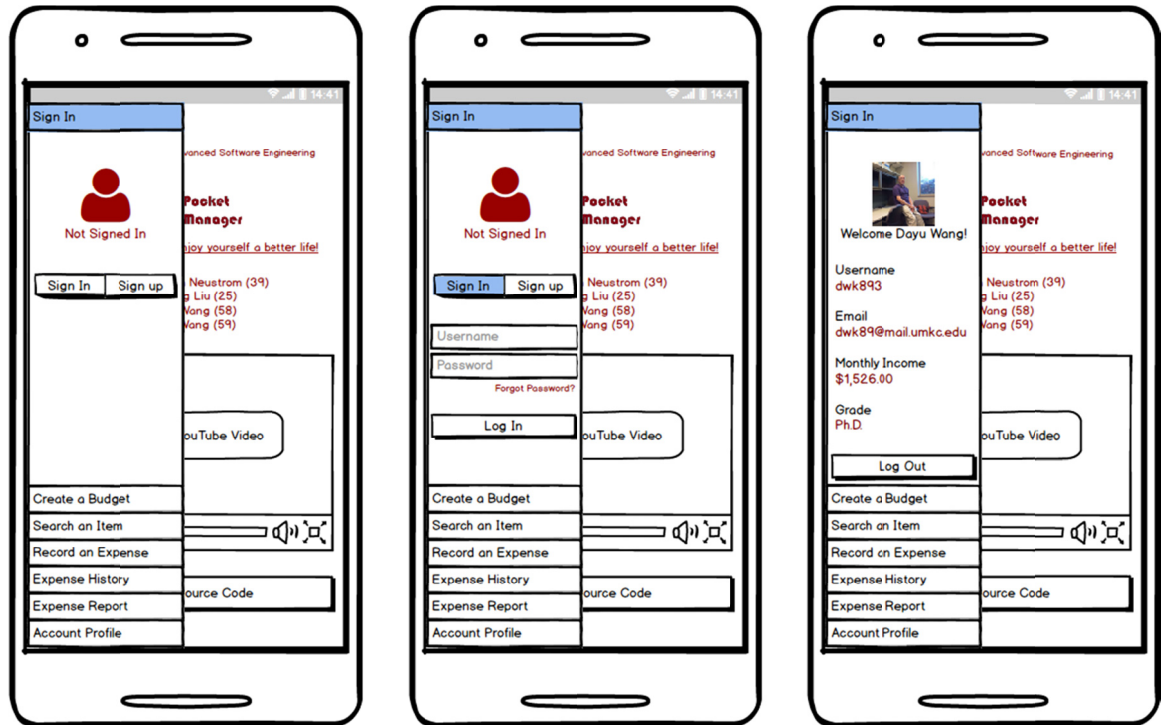


Figure X. Mockup pictures of the login part of the *Pocket Manager* system. The left picture reveals the initial appearance and the state when the user has been logged out. The middle picture is when the user is attempting to login. The right picture demonstrates the view when the user has been successfully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

Figure X 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.

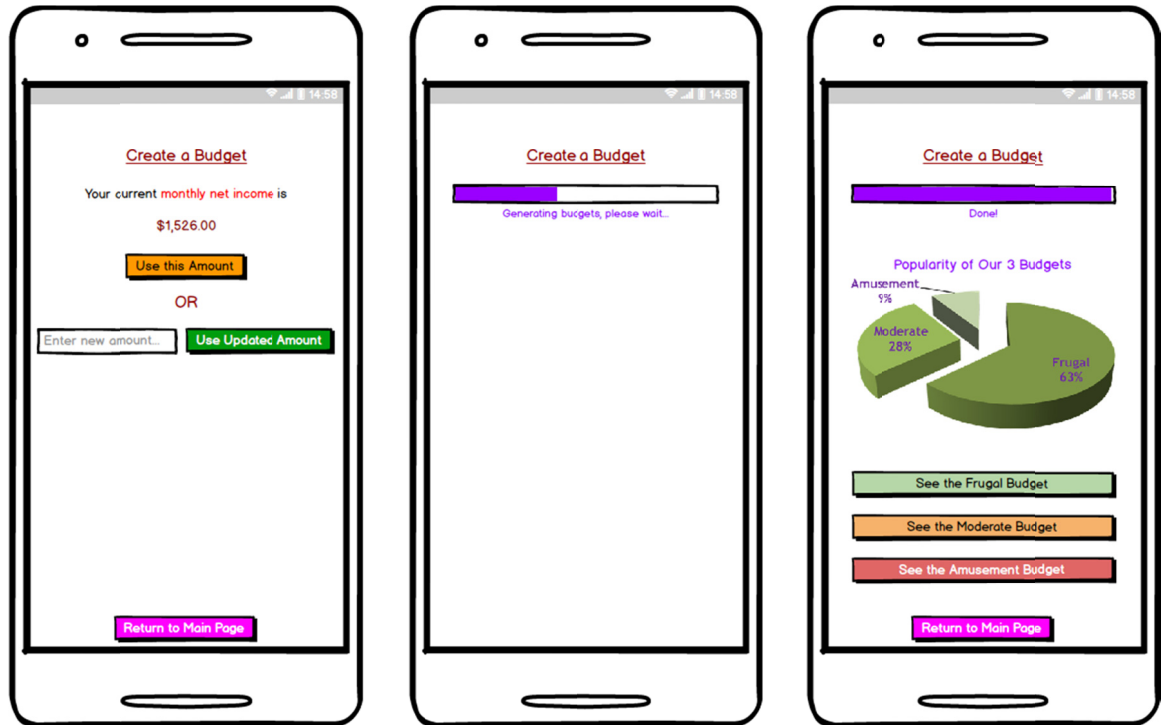


Figure X. Mockup pictures of the feature of *Create a Budget* in the *Pocket Manager* system. The left picture is the initial view when the user chooses the feature. It asks the user to confirm his/her monthly income or update his/her monthly income before the system can calculate budgets for the user.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

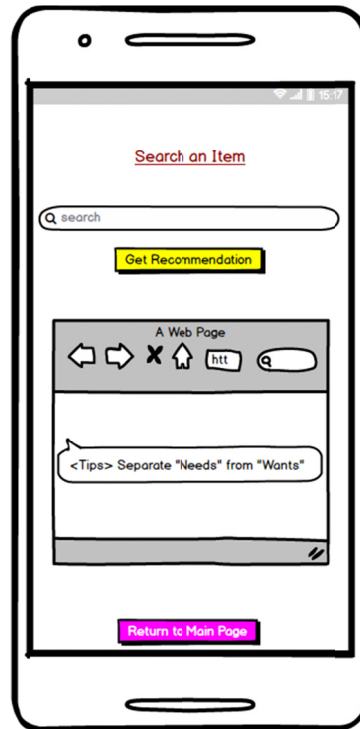


Figure X. Mockup picture of the principal view of the feature of *Search an Item* in the *Pocket Manager* system.

Figure X 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.

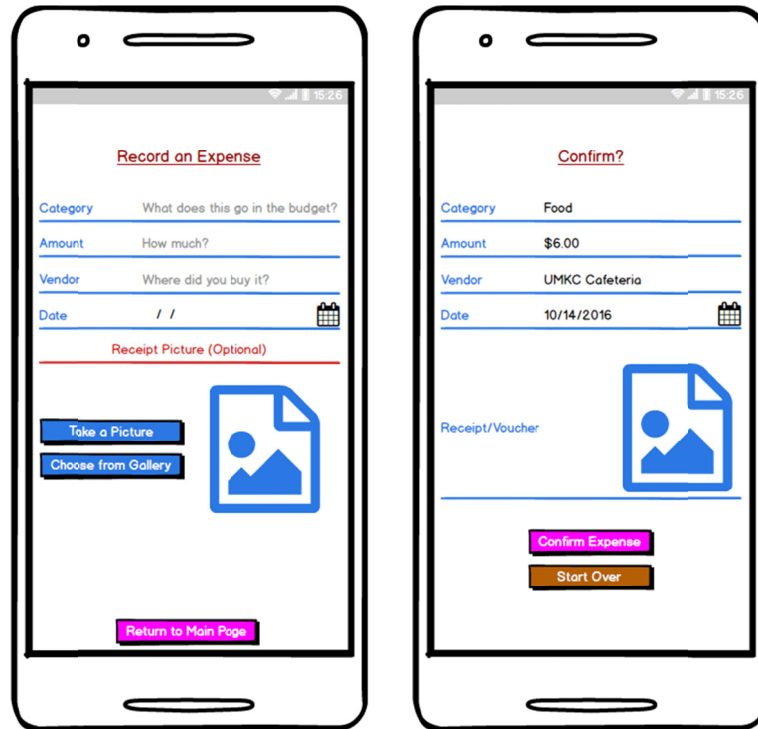


Figure X. Mockup pictures of the *Record an Expense* feature in the *Pocket Manager* system. The left picture is the view when the user is inputting an expense. The right picture is the confirmation page appeared right after the user's input.

5.2. UML Diagram/Sequence Diagram/Class Diagram/Activity Diagram

5.2.1. UML Class Diagram (Figure X)

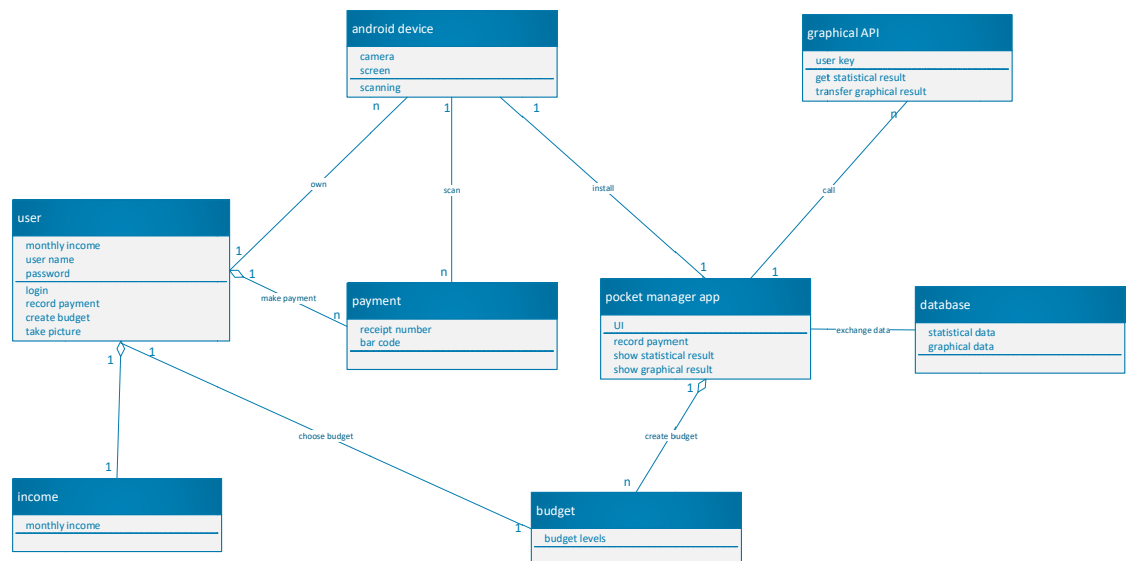


Figure X. High-level UML class diagram of the entire *Pocket Manager* system.

5.2.2. UML Activity Diagram (Figure X)

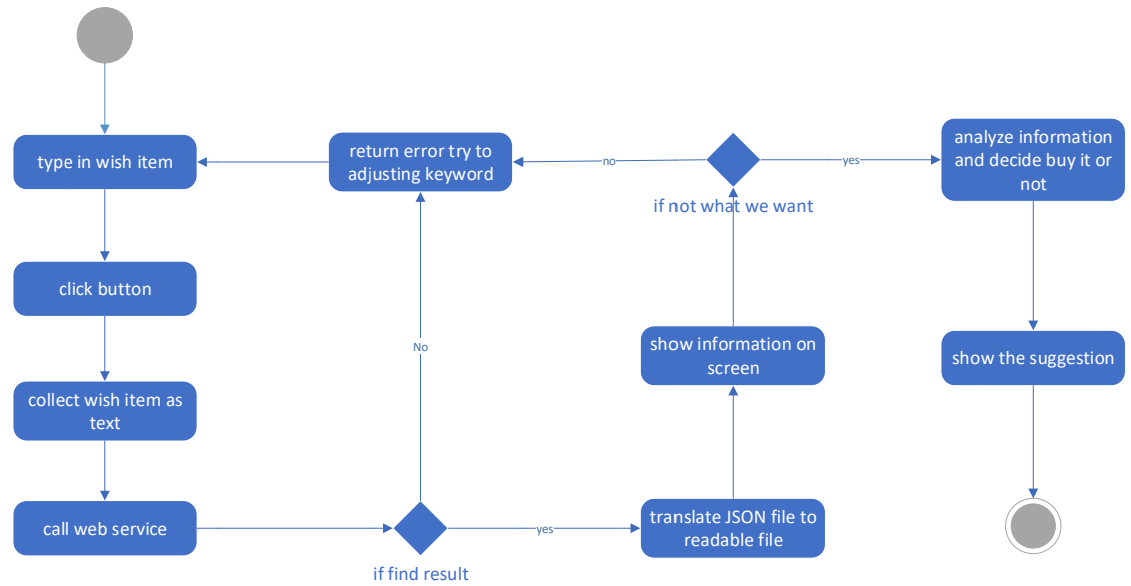


Figure X. High-level UML activity diagram of the *Separate Needs from Wants* part in the *Pocket Manager* system.



- 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)	HTML could use a few extra comments to help it stay organized in the future, code could use more header comments
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 was completed using Karma, Jasmine, and Chrome for the Ionic Framework components. Several areas were tested including the receipt recording section. The tests focused on the expense recording feature which was one of the primary developed functions for this increment. If expenses are not properly recorded, then the key Increment 3 features such as expense monitoring will not have realistic data to display.

a. Unit Test 1 – Empty Expense Fields (Figure 6-2)

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

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

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

  describe('recExpense', 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 func = $scope.recExpense(null, null, null, null);

      // Category
      var cat = localStorage.getItem('rec.category');
      expect(cat).not.BeNull();

      // Amount
      var amt = localStorage.getItem('rec.amount');
      expect(amt).not.BeNull();

      // Vendor
      var ven = localStorage.getItem('rec.vendor');
      expect(ven).not.BeNull();

      // Date
      var ven = localStorage.getItem('rec.date');
      expect(date).not.BeNull();
    });
  });
});
```

Figure 6-2. Null Submission Unit Test.

The first test focused on if a user could submit expense fields as blank in the initial recording of data to local storage. The test failed and users could submit blank data. The form was modified to both require fields to have data in the form and warn the user if the field was touched and left blank.

b. Unit Test 2 – Valid Amount (Figure 6-3)

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

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

  beforeEach(angular.mock.module('starter'));
  beforeEach(angular.mock.inject(function(_$controller_) {
    $controller = _$controller_;
  }));
  describe('recExpense', 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 func = $scope.recExpense(books, three, bookstore, 1/2/2016);

      // Amount check for a number in amount
      var amt = localStorage.getItem('rec.amount');
      expect(amt).toEqual(jasmine.any(Number));
    });
  });
});
```

Figure 6-3. Valid Amount Unit Test.

The first test focused on if a user could submit invalid expense fields in the initial recording of data to local storage. The test failed and users could letters for expense amounts and unreasonable dates. The form was modified to both allow only numbers for the amount that ranged between a certain reasonable positive and negative amount.

c. Unit Test 3 – Default Date (Figure 6-4)



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

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

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

  describe('recExpense', 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 func = $scope.recExpense(books, 3 , bookstore,null);
      var dateExpected = new Date(dateNow.setDate(dateNow.getDate()));

      // Amount check for a number in amount
      var date = localStorage.getItem('rec.date');
      expect(date).toEqual(dateExpected);

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

Figure 6-4. Valid Date Unit Test.

The first test focused on if a user could leave the date field blank and still get a valid date. (which would save the user from having to use the picker for an expense recorded on the same day) The test failed and users would have to use the picker to get a date. The issue was resolved by automatically adding the current date is automatically populated to the form unless the user

specifies otherwise. The local storage also accepts the current date by default. Furthermore, the date was limited to only the years around 2016.

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

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.

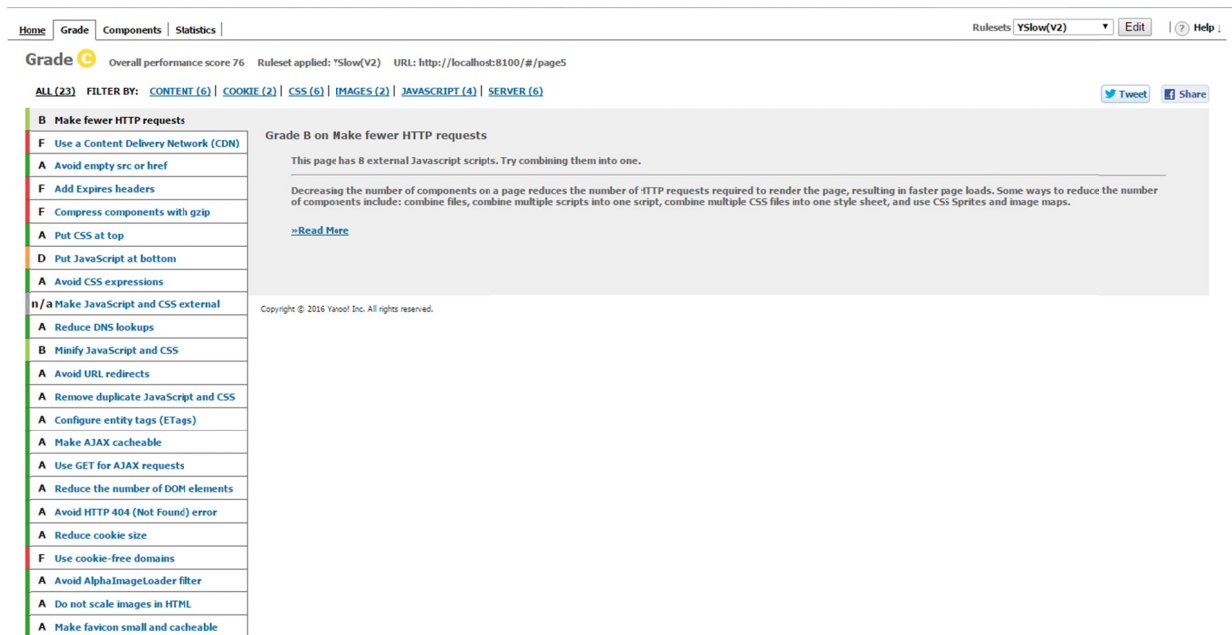


Figure 6.5. Home Page Speed Test.

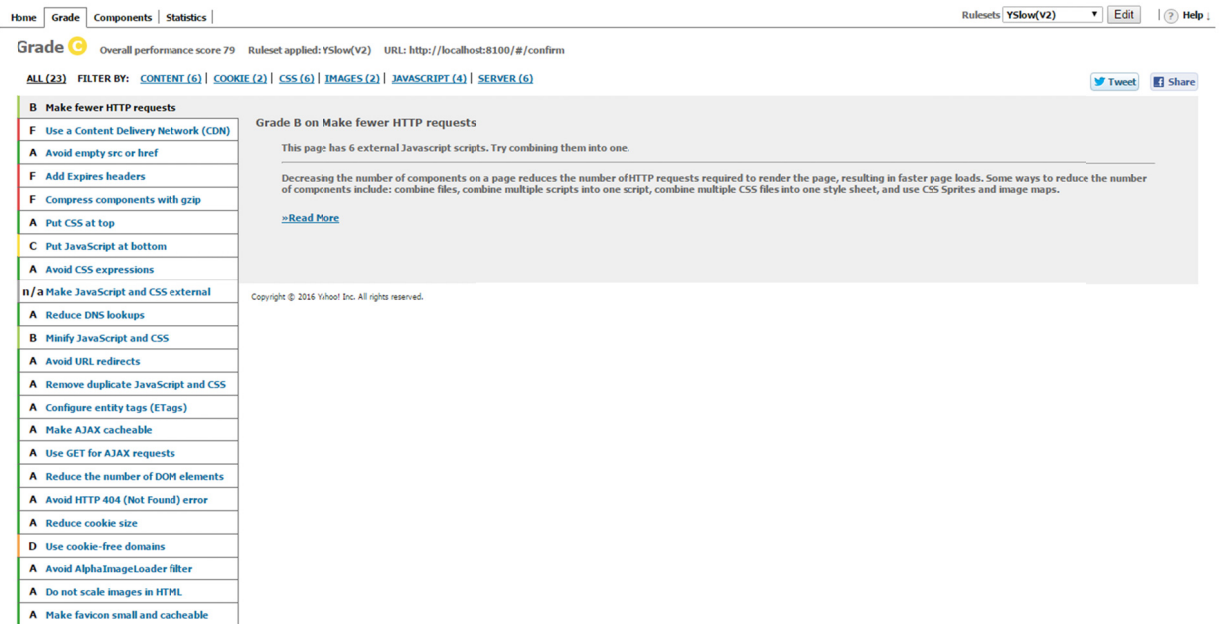


Figure 6.6. Rec Record Speed Test.

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 inherit 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
- Ennymotion 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, some buttons do point to features planned for increment 3
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

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.

User feedback for testing will become a means of marketing and deployment. Future increments are expected to develop tools to monitor behavior by the users testing the application in order to have mathematical data on how the application is actually utilized. In addition, robust unit testing is expected to be used to stress different aspects of the application and what type of data it can handle. Finally, performance testing is expected to become a more critical part of the testing as the application uses additional external services and stores additional data externally.

• **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. Only the camera, clock, and data connection are expected to be utilized on the phone.

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.

API - https://api.mlab.com/api/1/databases/pocket_manager/collections/

Collection - pocket_manager/

ApiKey - Omq-HhXv0WUnDNEVey9TQdBhsEEFDtHo

Table 6-9. Mongo Database Info

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

4. Services

Outside of the database, services were not currently utilized for the application. The second information focused on building the core application for the phone. Future increments are expected to incorporate APIs for login, picture reading, and visualization of data.

5. Going Forward

Long-term, Ionic will continue to be the application framework. Additional Mongo collections and outside services are expected to be added.

• **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 second 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 second increment focused on the delivery of the core mobile application including user interface and core functions for each tab. Building off of the welcome page was 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)

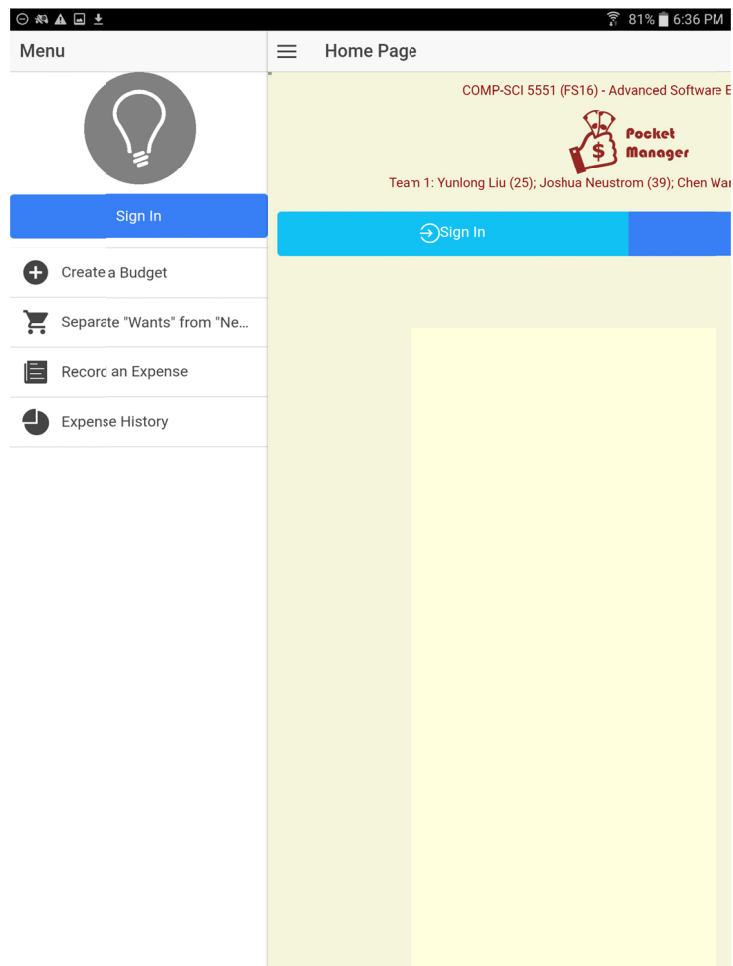
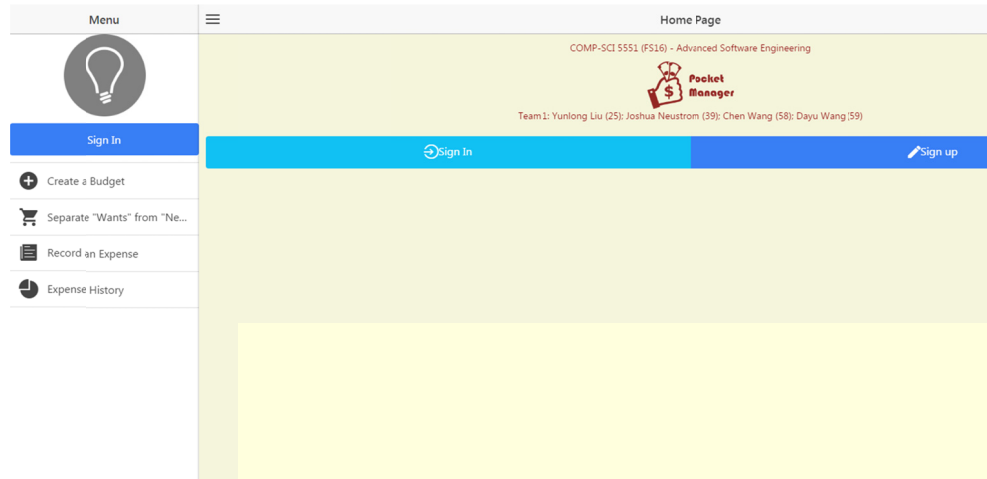


Figure 6-10. Login screen.

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.

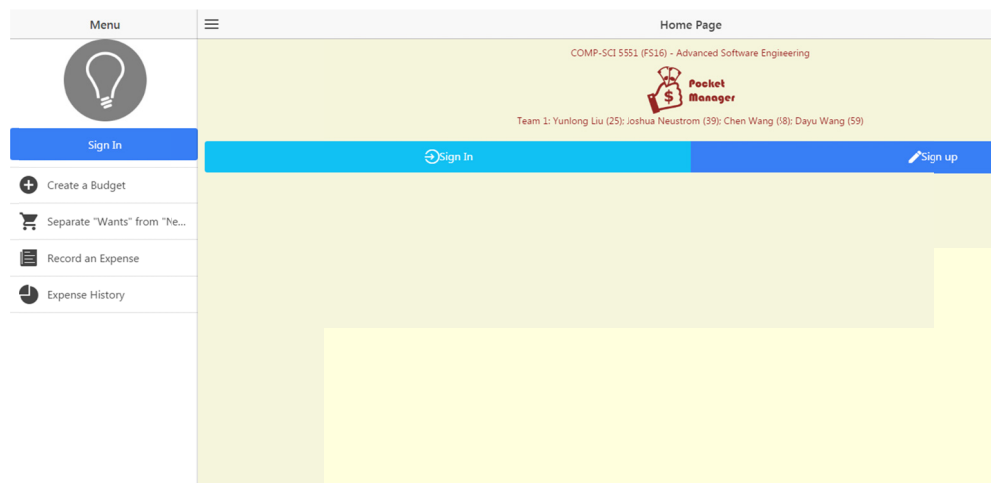
For branding purposes, the Pocket Manager logo is central to the page creating a visual association in the users mind. The logo also allows for a user to know if they are using the real Pocket Manager Application.

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)



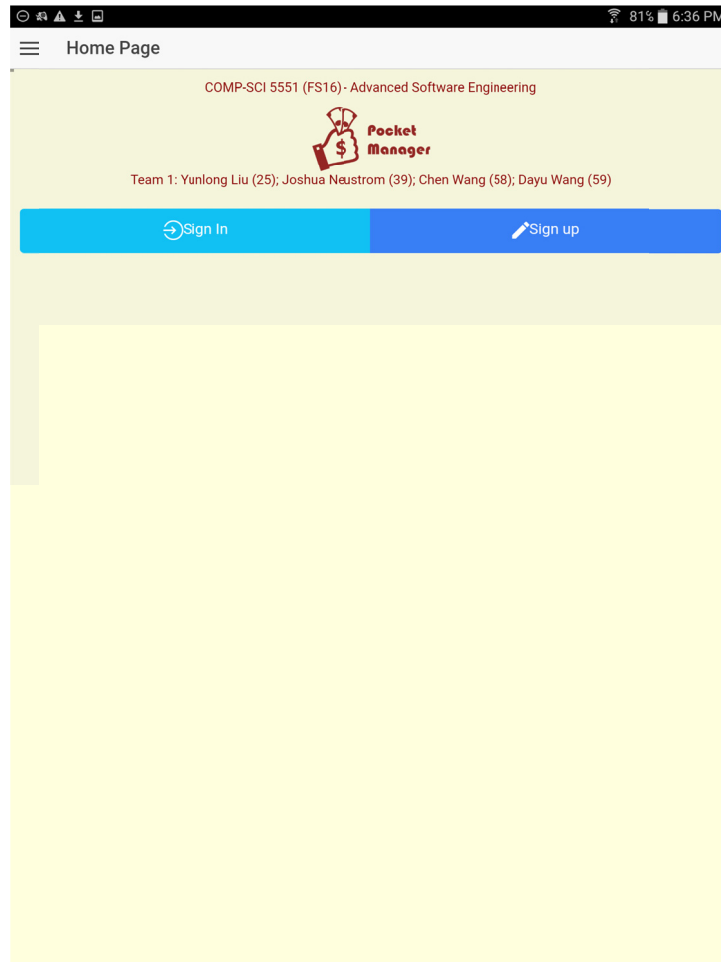


Figure 6-11. Main Menu Screen.

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)

The figure displays two mobile application screens side-by-side. The left screen, titled "Create Budget", shows a status bar at the top with signal strength, time (12:34 PM), and battery (100%). Below the title, it prompts the user to "Please input your income (per month):" and features a text input field containing the word "Income". A large blue button labeled "Confirm" is positioned at the bottom. The right screen, titled "Expense Model Select", also has the same status bar. It prompts the user to "Select your favourite type of expense:" and displays four stacked blue buttons: "Frugal Model", "Moderate Model", "Amusement Model", and "Back".

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 setup is required before recorded expenses can provide meaning full data.

Future increments are expected to enhance the user interface with more data based on our history from other users.

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

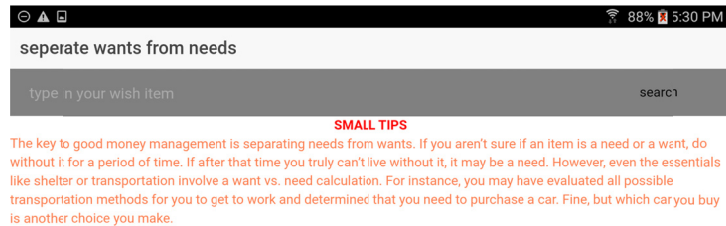
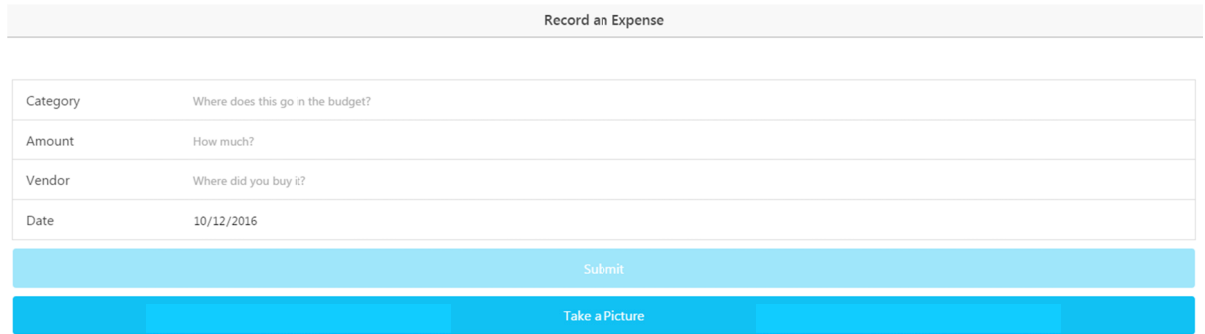


Figure 6-13. Wants vs Needs.

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 second increment focused on the user interface for basic expense submission.

Future increments will add the functionality of returning a smart answer on if the expense is truly a need.

e. Record An Expense (Figure 6-14 to Figure 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

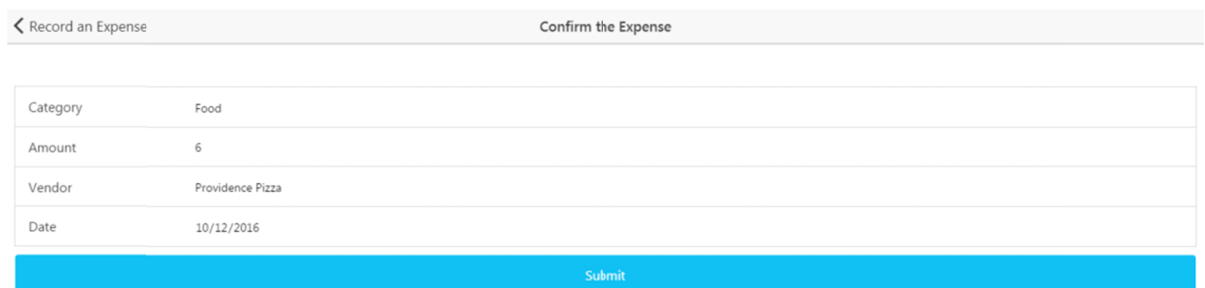
Submit

Take a Picture

Figure 6-14. Record an Expense.

The initial expense recording page provides the user an opportunity to populate key information 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 Take Picture button is present and the functionality is expected to be fully implemented in the next increment.



< Record an Expense		Confirm the Expense
Category	Food	
Amount	6	
Vendor	Providence Pizza	
Date	10/12/2016	

Submit

Figure 6-16. Confirm an expense.

The next screen takes a user to a version of the form populated by their entries from the local 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.

Confirm the Expense	Where to Next
Take a Picture of an Expense	
Type in a New Receipt	
Go Home	

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.

3. Wiki Page

A Wiki page was created for the increment 2 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-2

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

Future increments are expected to build on the structure created in the second increment. 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">• <code><code></code> Separate Wants from Needs• <code><documentation></code> System UML class diagram• <code><documentation></code> Activity diagram for feature <i>Separate Wants from Needs</i>• <code><documentation></code> UML sequence diagram for feature <i>Create a Budget</i>• <code><report></code> Existing services/APIs
Joshua Neustrom (39)	<ul style="list-style-type: none">• <code><code></code> Record Expenses• <code><documentation></code> System Architecture• <code><documentation></code> Manual Expense Recording• <code><documentation></code> Mongo setup, test, and initial data submission• <code><documentation></code> Wiki page creation• <code><report></code> Testing, implementation, and deployment
Chen Wang (58)	<ul style="list-style-type: none">• <code><code></code> Create a Budget• <code><report></code> Introduction• <code><report></code> Objectives• <code><report></code> Features
Dayu Wang (59)	<ul style="list-style-type: none">• <code><code></code> System Shell Model• <code><documentation></code> Wireframe/Mockup Views• <code><report></code> Detail Design of Features• <code><report></code> Final formatting and submission

- **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