* **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.

1. 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) | 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 stardardization 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 |

1. 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.

* 1. Unit Test 1 – Empty Expense Fields (**Figure 1)**

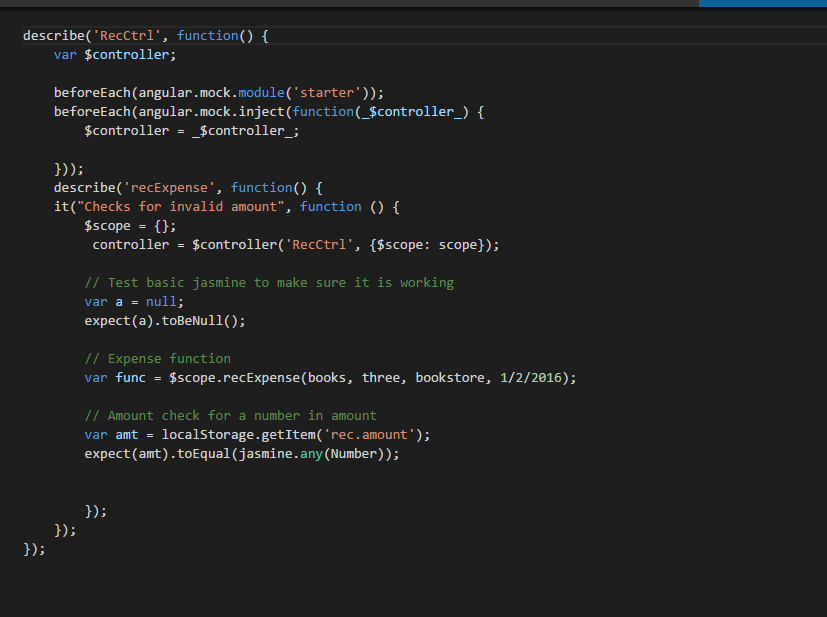
**Figure 1**. Null Submission Unit Test



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.

* 1. Unit Test 2 – Valid Amount (**Figure 2)**

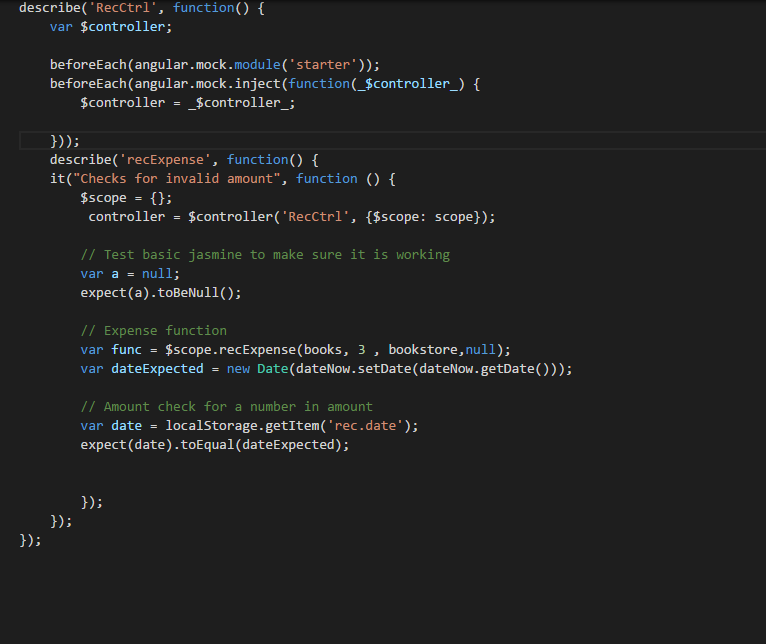
**Figure 2**. Valid Amount Unit Test

* 1. 

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.

* 1. Unit Test 3 – Default Date (**Figure 3)**

**Figure 3**. Valid Date Unit Test



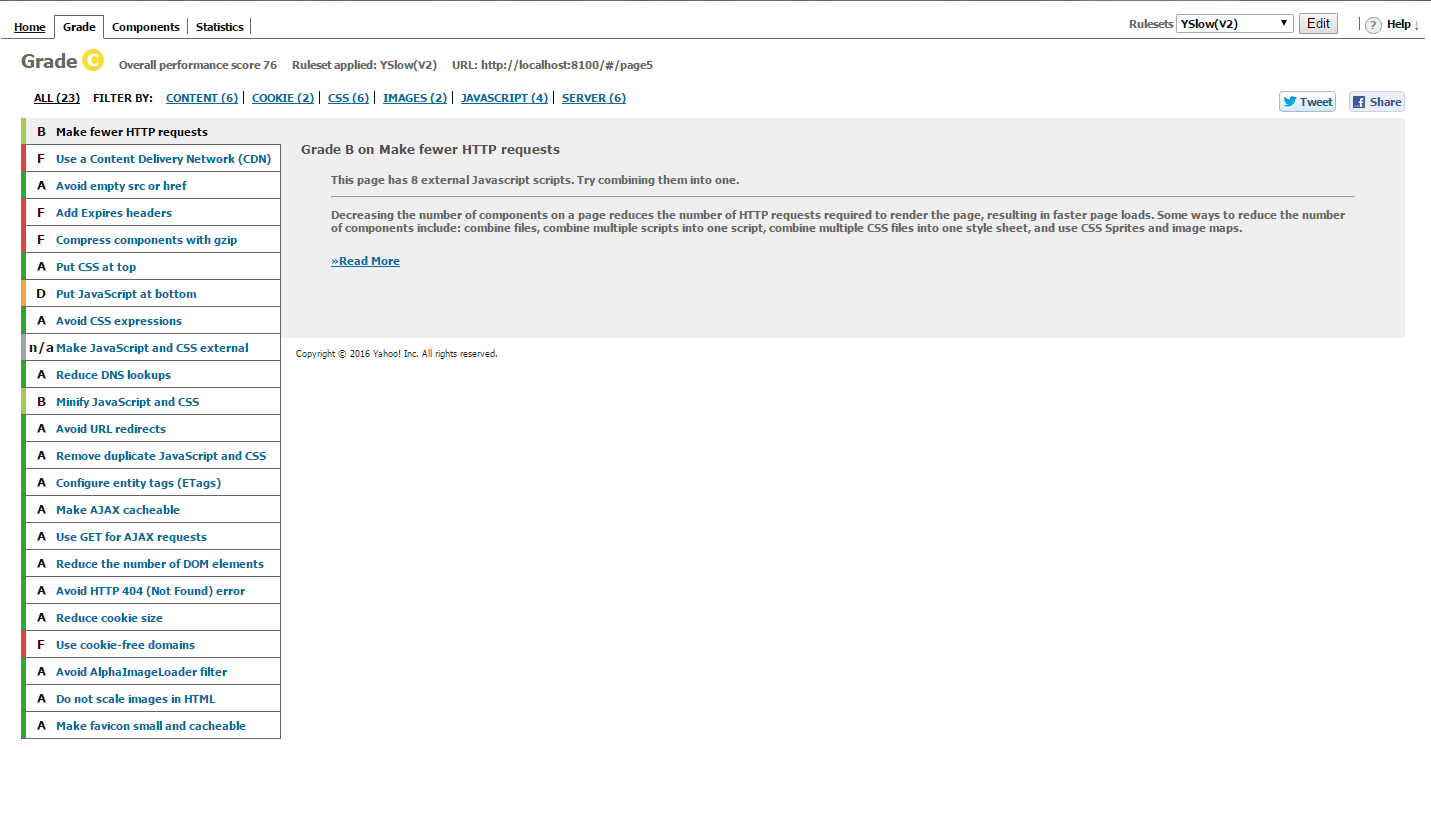
The first test focused on if a user could leave the date field blank and still get a valid date. (which would save the user from having to use the picker for an expense recorded on the same day) The test failed and users would have to use the picker to get a date. The issue was resolved by automatically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

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

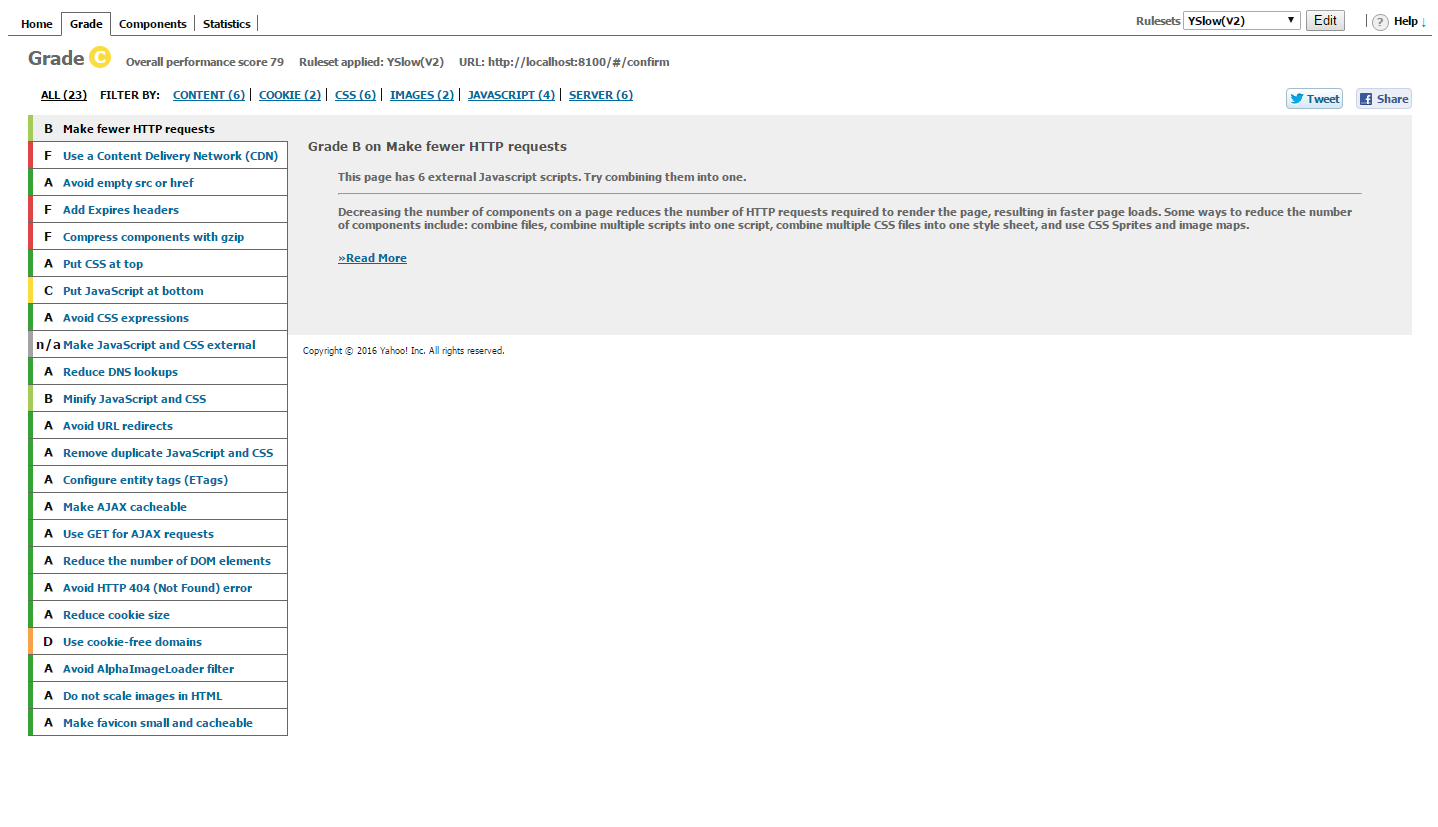
The performance of a loading page can have a major effect on site traffic and usage. Poor performance can cause users to avoid a site or application. In addition, loading issues can increase support requests and result in poor app reviews. Since site and app visits can result in add revenue and perform user statistics which can be sold to various companies, page 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 4**. Home Page Speed Test



**Figure 5**. 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 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.

1. Virtual Device Test

At least two different virtual environments were used in testing

1. Android Studio Virtual Device
2. 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, 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 leeks) | Yes |

1. 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 |
| c. | Any suggested changes? | Add a nicer background, easier navigation |

1. 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.

1. 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.

1. 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

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 save 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.

1. 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.

1. 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

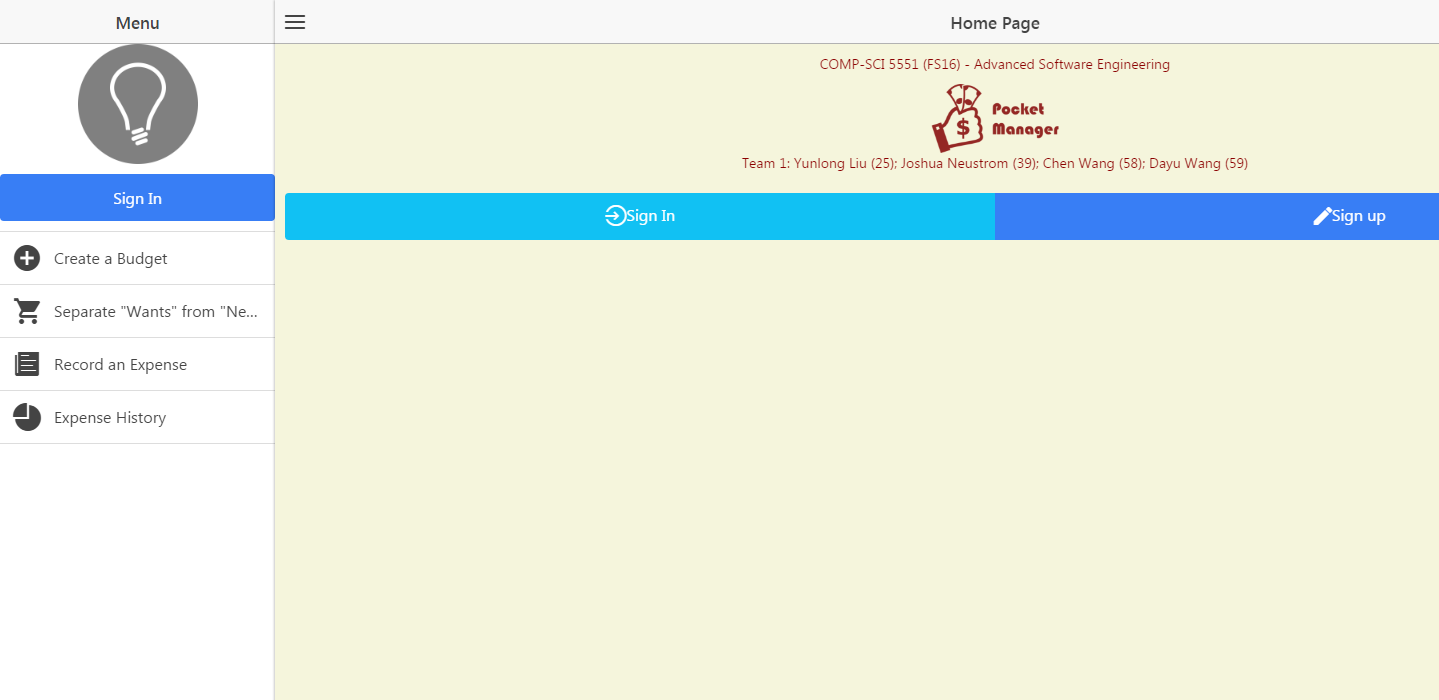
Deployment includes leading the app onto devices, taking screenshots with detailed descriptions, and creating a wiki in GitHub. The overall purpose of the section of the process is to deliver the completed application to market (or in the case of the second increment, the teachers). With the fast pace 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.

1. Screenshots

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

1. Login Screen (Figure 20)



**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.

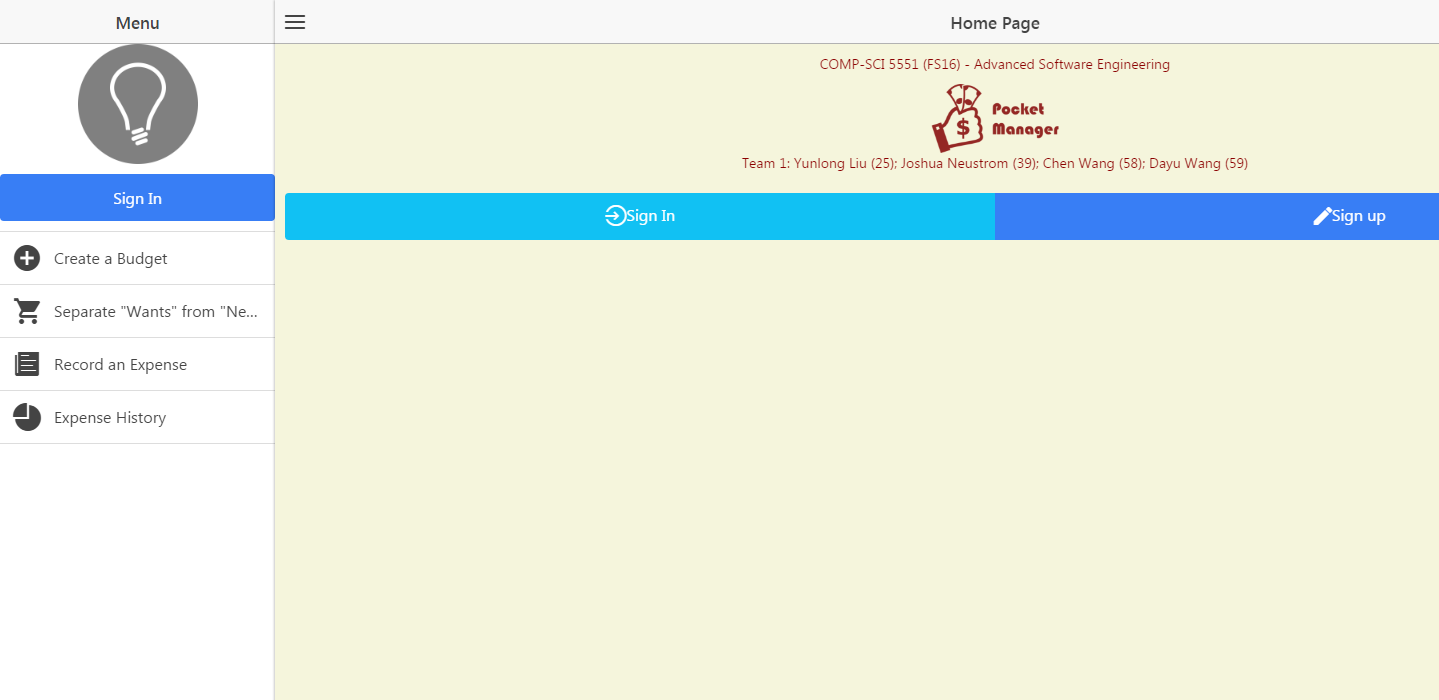
For branding purposes, the Pocket Manager logo is central to the page creating a visual association in the users mind. The logo also allows for a user to know if they are using the real Pocket Manager Application.

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.

1. Main Menu (Figure 21)



**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.

1. Create a Budget (Figure 22)

**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 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.

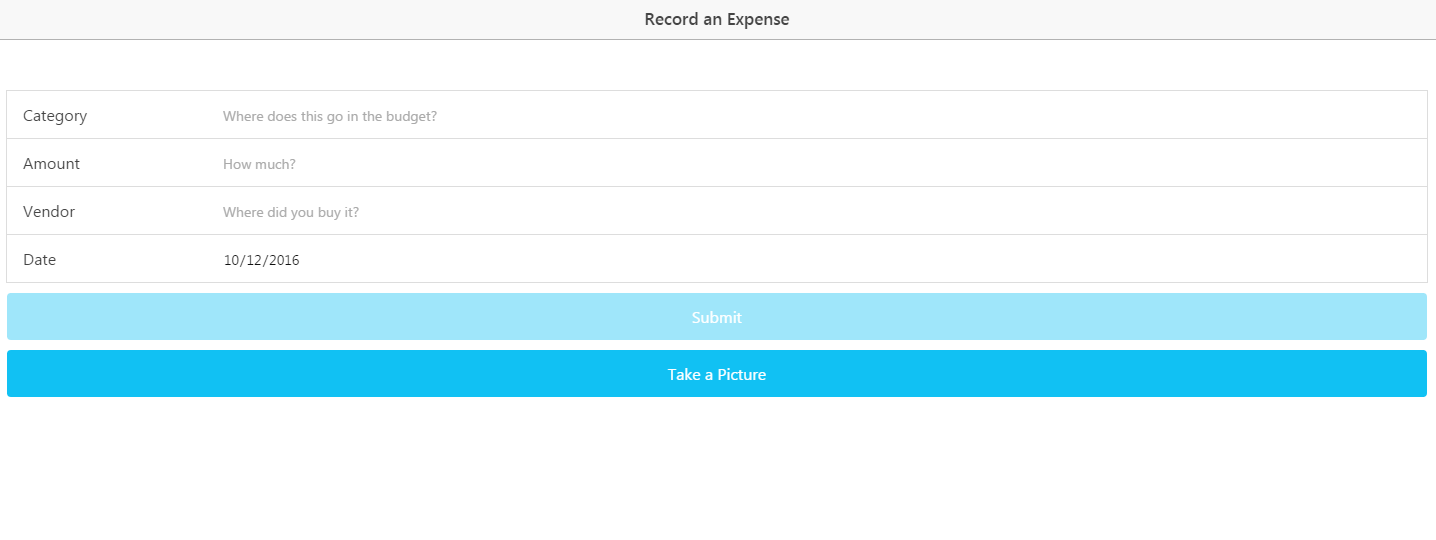
1. Separate Wants from Needs (Figure 23)

**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 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.

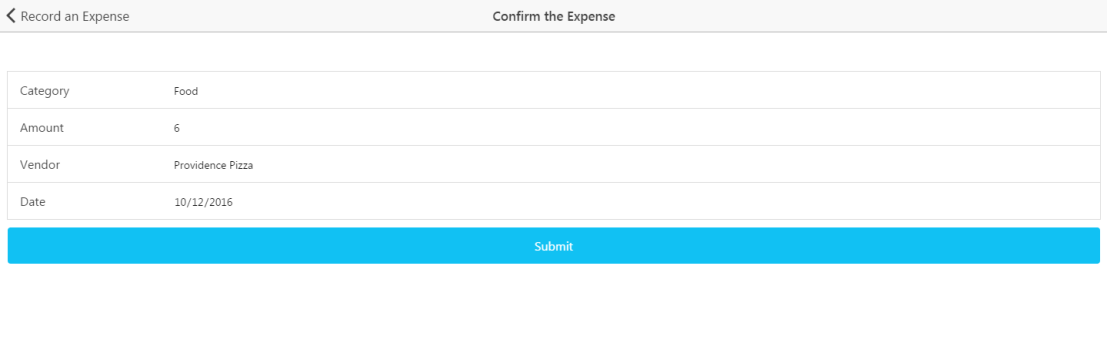
1. Record An Expense (Figure 24 - 26)



**Figure 24**. Record an Expense.

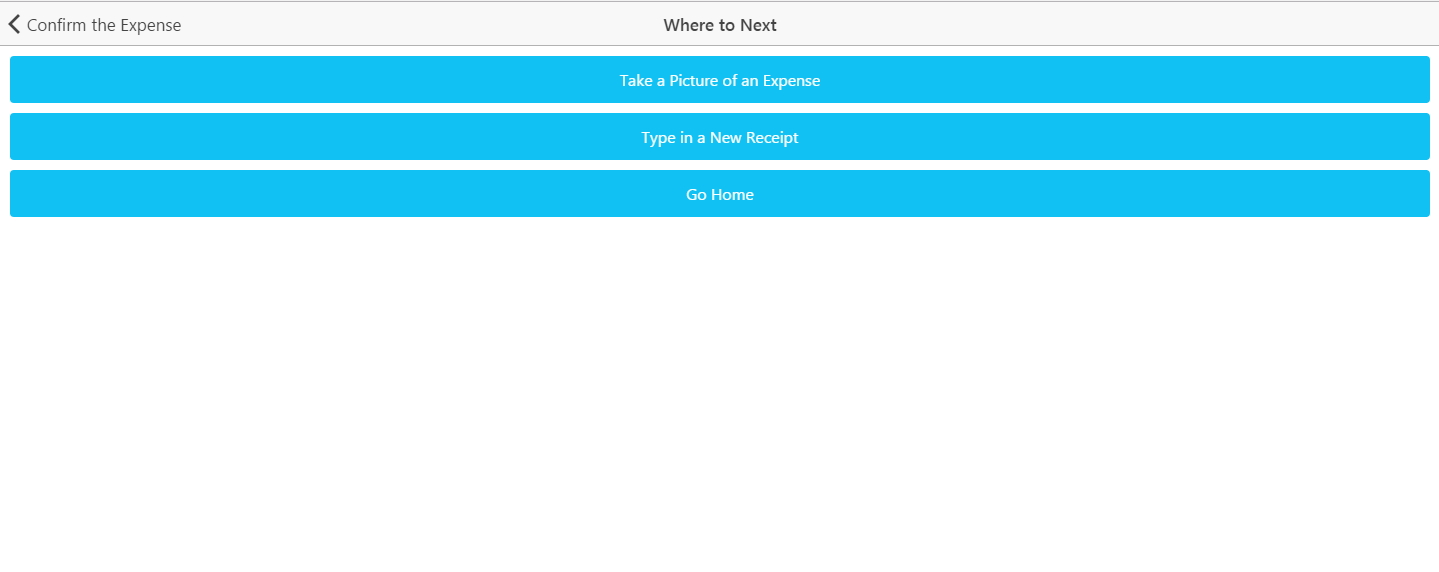
The initial expense recording page provides the user an opportunity to populate key information on money they have spent including the budget category, amount, vendor, and date. All fields are required and today’s date is populated into the form by default. If any field is touched and left blank, the 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.



**Figure 25**. 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.



**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.

1. Expense History (Figure 27)

**Figure 27**. Expense History.

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

1. 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 istself 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>

1. 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.

* **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>