COMP-SCI 5551 (FS16) - Advanced Software Engineering
Project Team 1 - The Brokers

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

Project Increment 1 - Report

(Due Sep 23rd, 2016)

- **Introduction**

The fall semester is well underway. Millions of college freshmen headed off to school last month. Managing money as a college student is critical. It's likely a priority for every student, unless you recently won the lottery. As we can remember, the first thing before we come to the college is shopping for the daily necessities, food, gathering textbooks, and packing up entire room. To most of these freshmen, it's the first time live to independently in their lives, and paying more attention to their personal finance. The average college student graduates with $24,000 in debt, according to the Project on Student Debt[1]. Most of the students will find on-campus or off-campus job to keep their financial independence, while some students still relying on their family. Most of the college students may roll their eyes at making a budget, they know the significances of managing money to college experience, but they don't know how to start and cultivating these good habits, they are paying their own way; receiving parents' help or get budget problems. Even there are several tips for these freshmen, create a budget, start from the bank statement; spend on the right things, take the time to impress the importance of using debt wisely; Look for student discounts, college students should become masters at exploring the ways how their educational status can see the money; avoid full-price textbook, as we know college textbook is very expensive, so it's a good idea to borrow from library or buy a used one; Set financial limits, it sounds like a most simple way to avoid over budget, but it's really hard to figure out how to keep the balance within certain limit.

- **Project Goal and Objectives** *(revised from the project proposal)*

1. Overall Goal

Base on the situation above, it's urgent to find a solution. And when we investigate and research all kinds of budget managing application, we have not found any application that targeted to college students, and that's the reason we have this idea, make an application for the college student, and help this group of people who need this most. Our application is called *Pocket Manager*, take care of the expense habit of college students, and help them to have the habit of recording every single expense, and review that, then have a clear spending plan in their mind. And at the very beginning, our application will provide these solutions for the users, we will have these system features: login system, budget creation, record a payment, optical input, ask for an item, get statistical result, and graphical representation.

Unlike other similar software which just help user understand the budget and check the bill seems mediocre.*Pocket Manger* try to help users to have a good expense habit, and make this process smooth which let user accept this easier.

2. Specific Objectives

Based on our summarization and research, there are four most common problems that may trouble college students (see Figure 1).

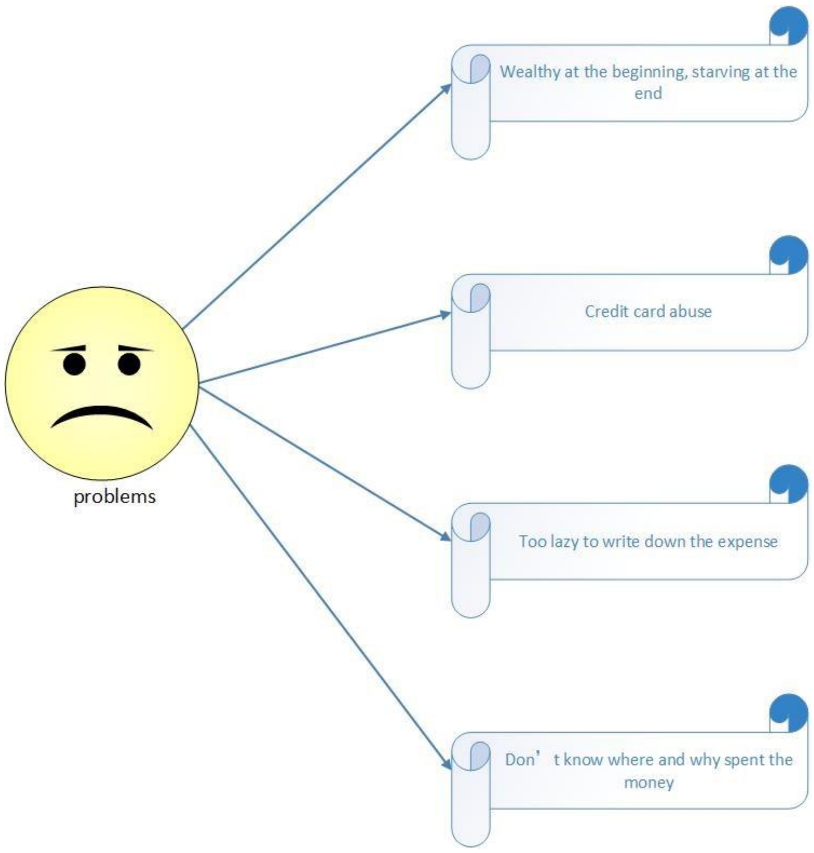

problems
Wealthy at the beginning, starving at the end
Credit card abuse
Too lazy to write down the expense
Don't know where and why spent the money

Figure 1. Expense problems that frequently occur amongst college students.

Problem 1 - So wealthy at the beginning of a month and so starving at the end of the month.

Many students may find that, every beginning of the month, when they get the salary from their on-campus or off-campus jobs, they are kind of wealthy, and spend half of the money in one week by eating in the restaurants every meal, shopping mindlessly, etc. and then find they become very pool, and live off balance. And starving at the end of the month, unfortunately, at the beginning of next month, repeat this situation, they may realize this, but don't know how to stop this circulation.

Problem 2 - Credit card abuse always occurs.

It's not really about the Credit Card fraud, it's more likely a kind of over budget, serious overdraft, three-quarters of college students have at least one credit card and nearly half have four or more, according to a research by Take Charge America Institute[2]. It's really important for students to get their debts under control before they graduate. This situation not only occurs among college freshmen, based on this research, senior student have more risky business. It's really important to think carefully every time before you slide the credit card (see Figure 2).

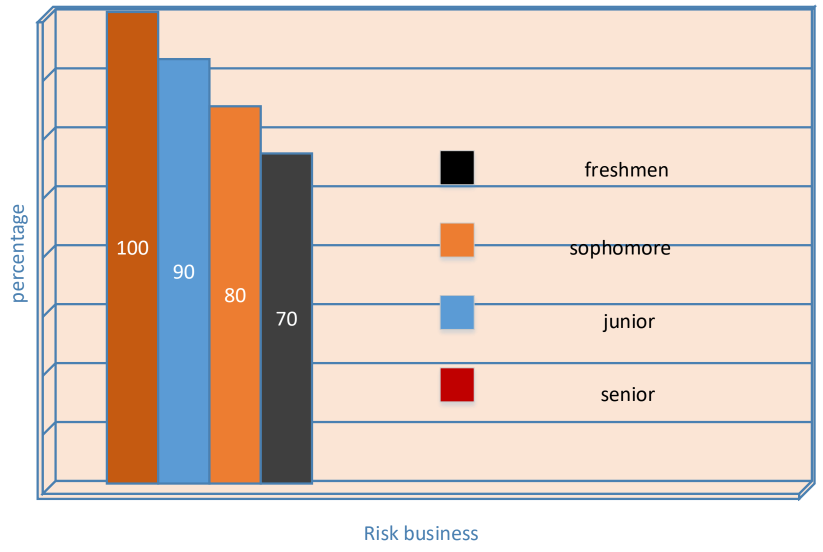


Figure 2. Risk business percentage relates to college students.

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

Every time you use credit card, the bank will have a record, at the end of the month, the bank will send you a bank statement, every single expense will be on that statement, but please remember, not every expense was done with credit card, what about the payment with cash? You may won't remember how and when you spent that amount money, and another thing is, the bank bill will be sent to you at the end of the month, you will know nothing about this before that, so write down the detail of expense every will remind you what you have done this day, you will get a plan in your mind, how much money i have now, how much money I've spent. And realize that, what kind of things I shouldn't consider buying.

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

This problem seems very common, we make payment so many times a month, it's hard to have this habit, write down everything you've bought, write down every detail of the expense, so, that may cause us forget what we've bought, the students may don't care about certain single payment, but within a month, too many that kind of expense may be a large amount of money, but they may even don't realize, where, when or why they spent that number of money, that all because they don't have a review.

Since we have summarize these serious problems, let us talk about how solve these case, here are some solutions we can provide in our application.

Solution 1 - Create a budget.

Based on the monthly income of a user (college student), our system helps him/her to create a reasonable monthly budget. For example, the system tells the student how much you can spend on food, entertainment, exercises, and so on.

Solution 2 - Separate want from needs.

Several rules are generated based on normally how people understand what is a college student's need and what is a want. The rule differs from every major field for a student.

Solution 3 - Character/number recognition API applied.

Character/number recognition API will be applied in our system, which definitely facilitates the input of their expense by just taking a picture of their payment receipt.

Solution 4 - "Grading" of your last month performance in money management.

Statistical data will send to the user monthly to analysis his/her performance in expenses. Also, when accumulated expense in a certain aspect is approaching the budget, warning messages will be also delivered to the user.

3. Specific Features (see Figure 3)

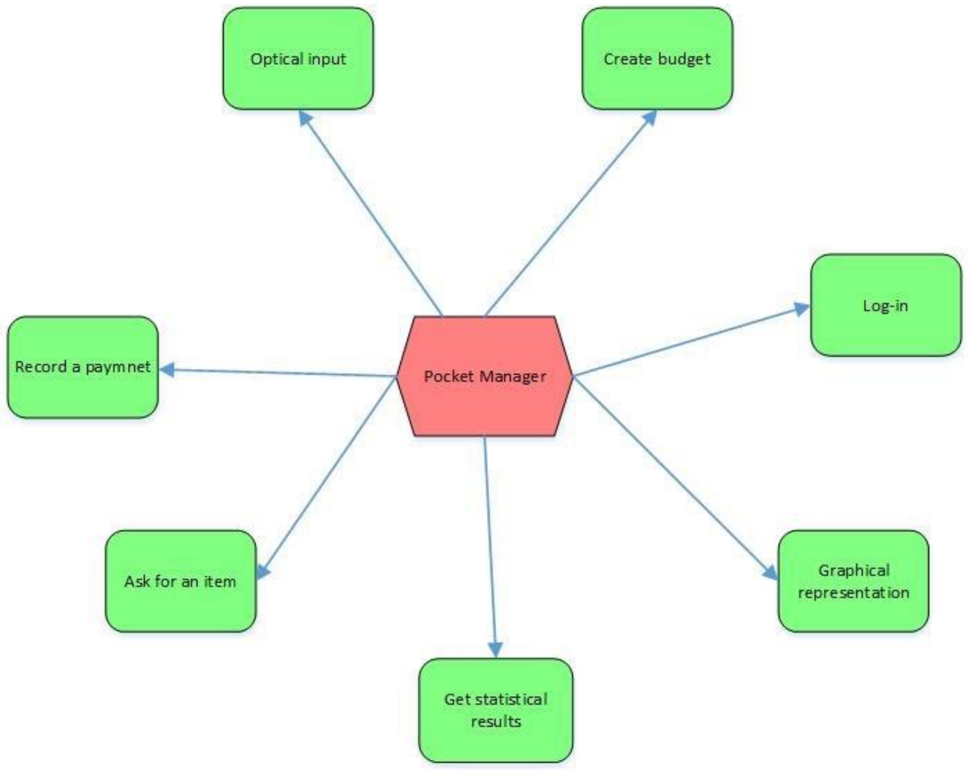


Figure 3. Features involved in the *Pocket Manager* system.

Feature 1 - Log-in system

This application will provide different functions and solutions for the users in different situations, so the log-in system is necessary, user may login with unique user's name or email address.

Feature 2 - Budget creation

This features existing for collecting all the user's income information, since our application focus on students, so we will probably have three suggestive monthly budgets which are depend on the income levels, because we want to design personal budget plan for every user, and base on our users are supposed to be college students. Three different budgets are created and may be the feasible levels of the budgets. This is called Budget Creation. But this part of income should be manageable income which is excluding rent, loan, etc. Because usually rent and loan are unchangeable or too stable to be managed.

Feature 3 - Optical input

This is a very characteristic feature; we will achieve the function with a few relative API, such as character or number recognition API. Users can simply scan or take picture of the bar codes or receipt number, it's very convenient for user record the payment, as we all know, and it's too hard to remember the prices of everything we've bought. And it's better than type the expense details manually.

Feature 4 - Record a payment

Though we will have the optional input feature, but it is still necessary to record some specific payment manually, as we know, sometimes we can't have the picture of the receipts, or even we don't have the receipts. And in this part we will classify the expenses into several categories, such as food, traffic, book, entertainment, etc. Users want to know more than a bank statement, they need to know how and why the spent the money.

Feature 5 - Ask for an item

This feature is more like a wish list, as the users may have items that they want to buy but may cause over budget. Based on the prices of these items and the monthly income, this feature will decide whether the item is necessary or not, this is called separate needs from wants, what is more, it will generate a time to suggest when the user can consider buying it.

Feature 6 - Get statistical results

This feature allows users check the statistical expense result of this monthly or previous month, absolutely, based on several categories, users always want more than a bank statement, and it is better to show the expenses in a form or separated pages than just a single list.

Feature 7 - Graphical Representation

This feature applies the graphical API, such as high charts, it will show the statistical result graphically, sector diagrams is a good choice to show the expense of every aspect, and also the percentage.

- **Project Plan**

The plan of developing the system of *Pocket Manager* requires us to estimate the difficulty of the implementation of the features mentioned in the project proposal prior to creating a realistic plan of development. Also, since we are just located at the very first stage of the system development at this moment, we further classified each feature as *compulsory* and *optional*, where compulsory feature means that the feature is fundamental to our entire system (e.g. the communication with Mongo DB in that a user's expense history is always securely stored and accessed) or will mark our system distinguishable from other similar systems (e.g. the *lazy recording* feature that allows the user to input an expense of him/her by just taking a picture of the payment receipt), and the optional feature means that the feature includes time-consuming or challenging techniques for us to accomplish under the several-week intense time pressure as a class project. For example, the *expense predicting* feature, which requires a complicated *machine learning* model in our system designation, is categorized as an optional feature to us, though it is a very good idea to remind the user of any bad expense habit detected by the machine learning component before the user actually suffers from such ill habit. Definitely, a world-class nice Android application requires a huge team of even 100 people to work together for from several months to even several years. For a small group like us, we would like our developed system to be rather a "completed" system, than a "cutting-edge" system but lack of integration. Therefore, if time allows, we would be happy to develop those optional features to make our final project more beautiful; but if we are falling behind with the original schedule, then we have to focus ourselves on the compulsory features to accomplish the project on time. The classification of our planned features in our final system is categorized in Table 4.

Table 4. Classification of each planned features to be developed in the *Pocket Manager* system based on the estimated difficulty of each feature. The estimated difficulty values vary from 1 (easy) to 5 (very hard) and were generated by our group discussion and/or experienced class teaching assistant.

System Feature	Estimated Difficulty	Compulsory/Optional	Note
Login System	2	Compulsory	Regular
Social Login System	3	Optional	Google, Facebook, etc.
Expense History	2	Compulsory	Communicate with Mongo DB
Expense Recording (by text)	1	Compulsory	User manually enters the amount.
Expense Recording (by receipt photo)	3	Compulsory	User takes a picture of the receipt.
Algorithm of Creating Budgets	3	Compulsory	Create 3 different budgets to choose.
Algorithm of Separating Wants from Needs	4	Compulsory	Avoid *credit card abuse*.
Statistics in previous expense behavior	1	Compulsory	Mathematical models
Expense Supervising Monitor	3	Compulsory	Give warnings when the user is approaching a certain budget.
Graphical Demonstration	3	Compulsory	Using Highcharts
Expense Prediction	5	Optional	Machine Learning Model

Another factor considered in generating our project plan is the *dependency* amongst all the desired features to be developed. Obviously, some features need to be implemented after some other features completely implemented. Therefore, when we are coming up to a project plan, a consideration of the sequence of features to be developed matters a lot in the working efficiency. This dependency is different from the streaming architecture or the workflow diagram of the system. Workflow is a kind of streaming diagram, but for implementation and unit testing, several components can be implemented and unit-tested in parallel. Also, when a feature is half accomplished implementation, another feature may be already good to start coding. Therefore, figuring out a "fine" dependency will help us increase our working efficiency and save us a lot of time (especially "waiting others" time).

Therefore, we would like out project plan to be finely structured before we put a realistic and efficient plan in a ZenHub. Figure 5 is a graphical representation of how the entire project can be divided into smaller pieces. In Figure 5, each horizontal level represents tasks that can be worked on in parallel. When the regular login system has been built up, then the next level can be started working on without the accomplishment of the social login system. Similarly, receipt scanning input can be somehow "independent" since the downstream components can be implemented when the text receipt input has been implemented. The wants separation feature is isolated from the rest part of the system, since it is targeted to a rectification of bad money habit amongst college students, and the budget creation feature embraces the related algorithm design. The advantage of this diagram is that if our plan changes in future, a new feature is added or a current feature is removed, then by appropriately placing the new feature into the diagram, we can visually and easily understand how our project plan can be changed.

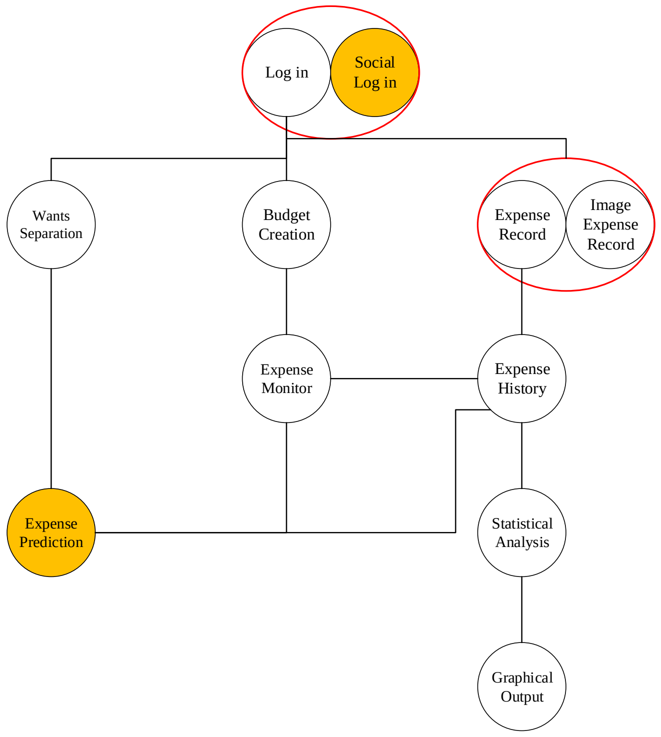
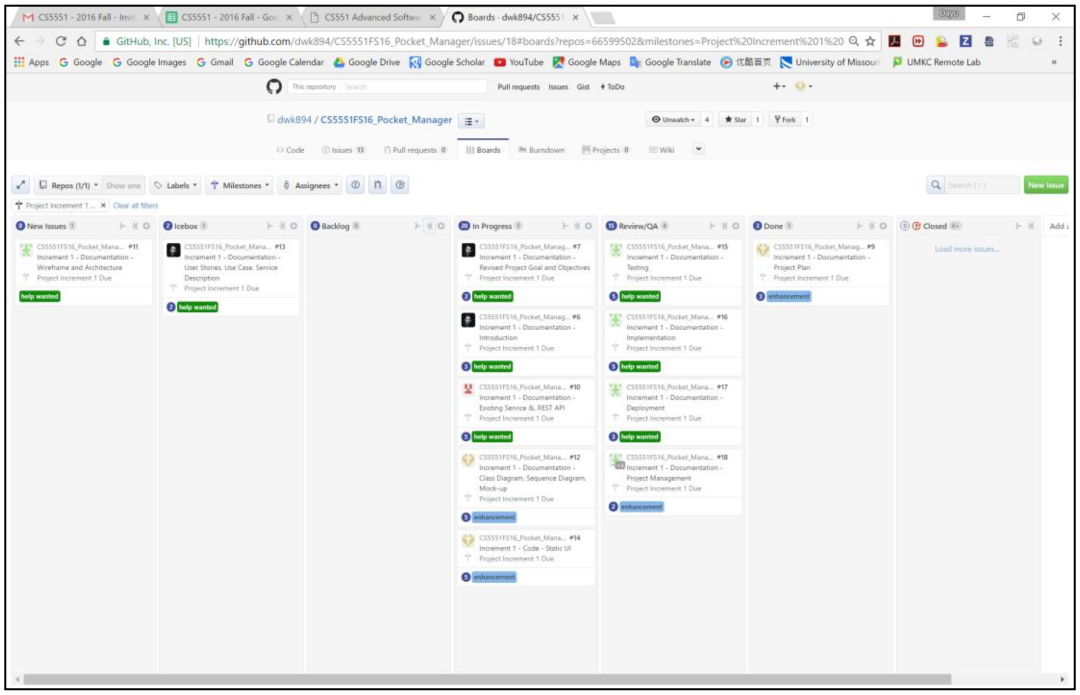


Figure 5. Analyzed dependencies amongst different features to be developed in the *Pocket Manager* system. The orange circles represent optional features and the others are compulsory features. The oval elements means that there are "sub-features" in this big feature and the sub-features can be accomplished by different people in different time.

1. Schedule for the Four Different Increments

Based on the factors mentioned above, we finally created a project plan. Using the ZenHub tool, iterations of our project plan to develop the entire system (also based on the due day of increment reports) is shown in Figure 6, which is a series of screenshots. For more detailed information, you can look at our actual GitHub repository at https://github.com/dwk894/CS5551FS16_Pocket_Manager. Note that since the report guideline is only available for the first increment, our plan for the increments 2-4 is mainly based on the programming issues. Also, our plan is just good for now and may change or tremendously change in future.

(a)



0 New Issues 1
CS5551FS16_Pocket_Mana... #11
Increment 1 - Documentation - Wireframe and Architecture
Project Increment 1 Due
help wanted

2 Icebox 1
CS5551FS16_Pocket_Mana... #13
Increment 1 - Documentation - User Stories. Use Case. Service Description
Project Increment 1 Due
2 help wanted

0 Backlog 0

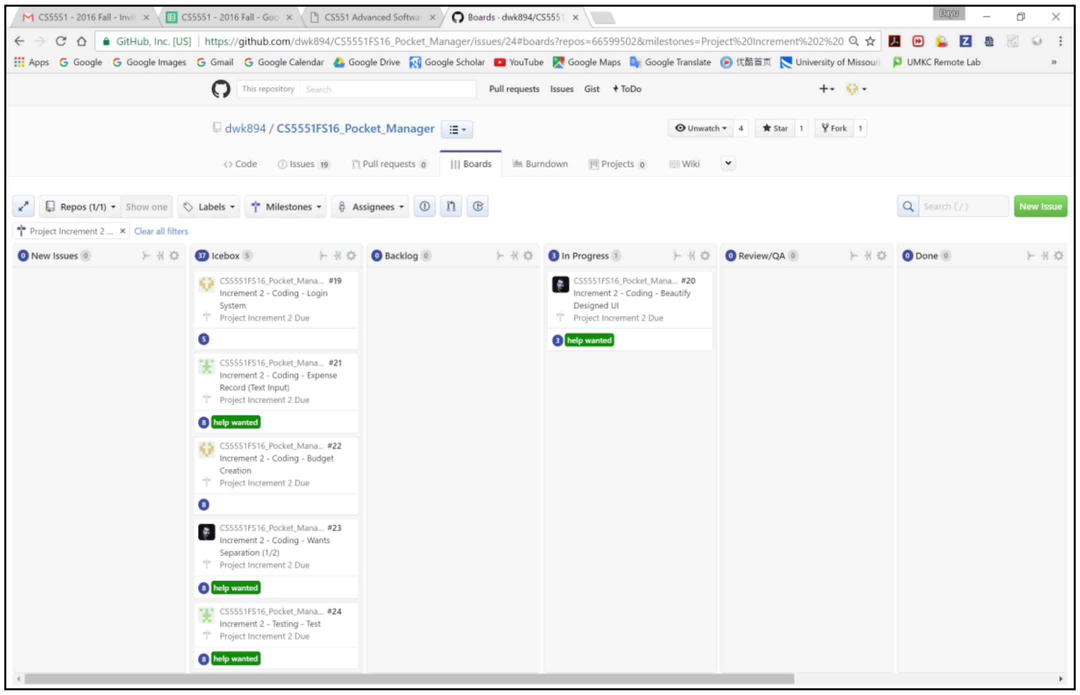
20 In Progress 5
CS5551FS16_Pocket_Manag... #7
Increment 1 - Documentation - Revised Project Goal and Objectives
Project Increment 1 Due
2 help wanted
CS5551FS16_Pocket_Manag... #6
Increment 1 - Documentation - Introduction
Project Increment 1 Due
3 help wanted
CS5551FS16_Pocket_Mana... #10
Increment 1 - Documentation - Existing Service & REST API
Project Increment 1 Due
5 help wanted
CS5551FS16_Pocket_Mana... #12
Increment 1 - Documentation - Class Diagram, Sequence Diagram, Mock-up
Project Increment 1 Due
5 enhancement
CS5551FS16_Pocket_Mana... #14
Increment 1 - Code - Static UI
Project Increment 1 Due
5 enhancement

15 Review/QA 4
CS5551FS16_Pocket_Mana... #15
Increment 1 - Documentation - Testing
Project Increment 1 Due
5 help wanted
CS5551FS16_Pocket_Mana... #16
Increment 1 - Documentation - Implementation
Project Increment 1 Due
5 help wanted
CS5551FS16_Pocket_Mana... #17
Increment 1 - Documentation - Deployment
Project Increment 1 Due
3 help wanted
CS5551FS16_Pocket_Mana... #18
Increment 1 - Documentation - Project Management
Project Increment 1 Due
2 enhancement

3 Done 1
CS5551FS16_Pocket_Manag... #9
Increment 1 - Documentation - Project Plan
Project Increment 1 Due
3 enhancement

0 Closed 0+

(b)



0 New Issues 0

37 Icebox 5
CS5551FS16_Pocket_Mana... #19
Increment 2 - Coding - Login System
Project Increment 2 Due
5
CS5551FS16_Pocket_Mana... #21
Increment 2 - Coding - Expense Record (Text Input)
Project Increment 2 Due
8 help wanted
CS5551FS16_Pocket_Mana... #22
Increment 2 - Coding - Budget Creation
Project Increment 2 Due
8
CS5551FS16_Pocket_Mana... #23
Increment 2 - Coding - Wants Separation (1/2)
Project Increment 2 Due
8 help wanted
CS5551FS16_Pocket_Mana... #24
Increment 2 - Testing - Test
Project Increment 2 Due
8 help wanted

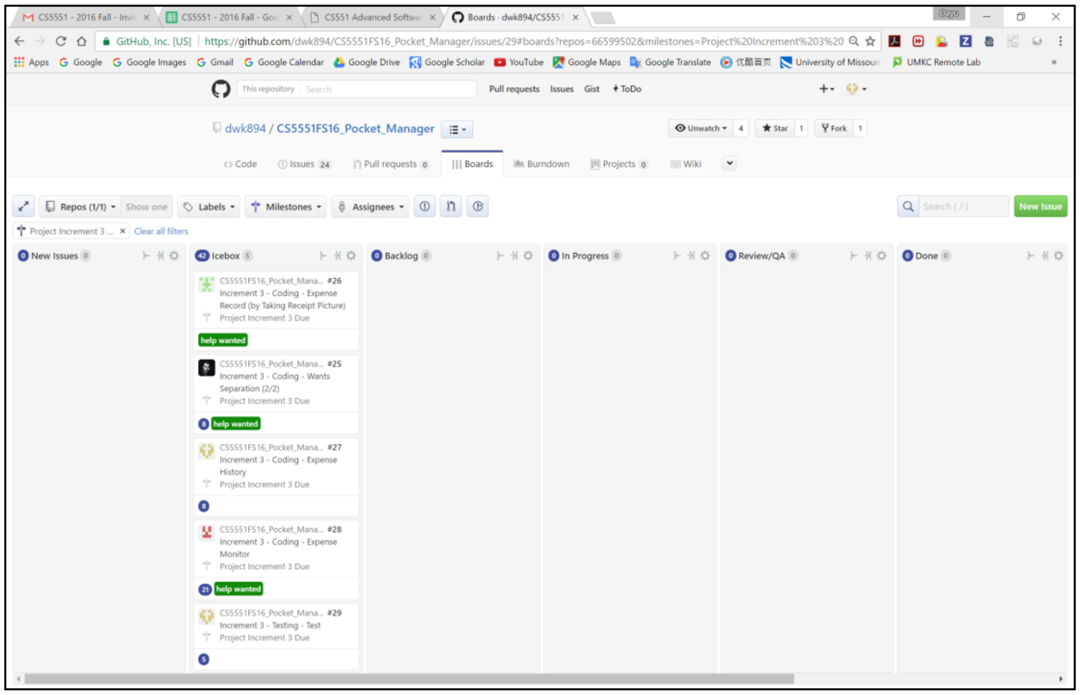
0 Backlog 0

3 In Progress 1
CS5551FS16_Pocket_Mana... #20
Increment 2 - Coding - Beautify Designed UI
Project Increment 2 Due
3 help wanted

0 Review/QA 0

0 Done 0

(c)



CS5551FS16_Pocket_Mana... #26 Increment 3 - Coding - Expense Record (by Taking Receipt Picture) Project Increment 3 Due help wanted

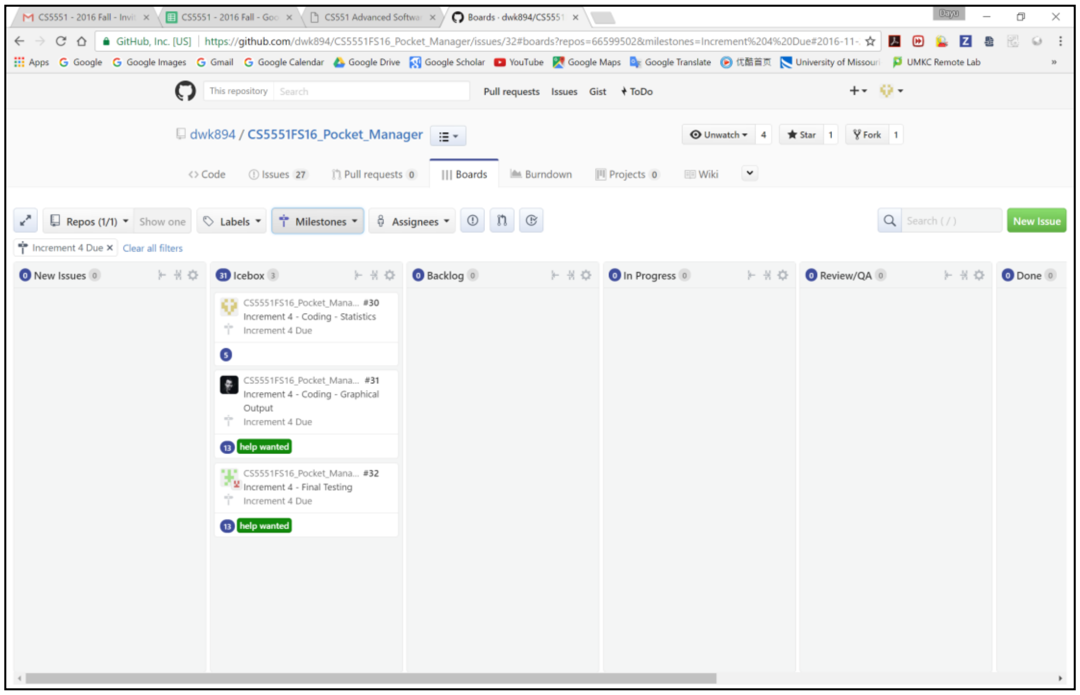
CS5551FS16_Pocket_Mana... #25 Increment 3 - Coding - Wants Separation (2/2) Project Increment 3 Due help wanted

CS5551FS16_Pocket_Mana... #27 Increment 3 - Coding - Expense History Project Increment 3 Due

CS5551FS16_Pocket_Mana... #28 Increment 3 - Coding - Expense Monitor Project Increment 3 Due help wanted

CS5551FS16_Pocket_Mana... #29 Increment 3 - Testing - Test Project Increment 3 Due

(d)



CS5551FS16_Pocket_Mana... #30 Increment 4 - Coding - Statistics Increment 4 Due

CS5551FS16_Pocket_Mana... #31 Increment 4 - Coding - Graphical Output Increment 4 Due help wanted

CS5551FS16_Pocket_Mana... #32 Increment 4 - Final Testing Increment 4 Due help wanted

Figure 6. Screenshots from project ZenHub for the four different iterations (*a*: increment 1; *b*: increment 2; *c*: increment 3; *d*: increment 4) of the development of the *Pocket Manager* system. At this moment, most of the future plans are located in the icebox since they could not be started this early. For more detailed information, please look at the project GitHub repository at https://github.com/dwk894/CS5551FS16_Pocket_Manager.

2. Project Timelines, Members, Task Responsibility

Since we tried our best to divide the entire implementation work equally into four independent series and assign one category to each member in the group, it is not considered possible at this moment to build up a realistic timeline for the project development because each member's personal time schedule varies greatly for us. As a team leader (*Dayu Wang*), I respect every gentleman in our team to control his time schedule by himself, with only the due days and tasks were assigned in advance. Instead, we are using the four increments of the project as the time spot to draw something that is similar to a timeline, which is a high-level overview. Nevertheless, the amount of coding work for each body can be summarized based on the estimates of coding workload for each assignment appeared in the ZenHub board (Figure 6). The real timeline for our project is demonstrated in Figure 7.

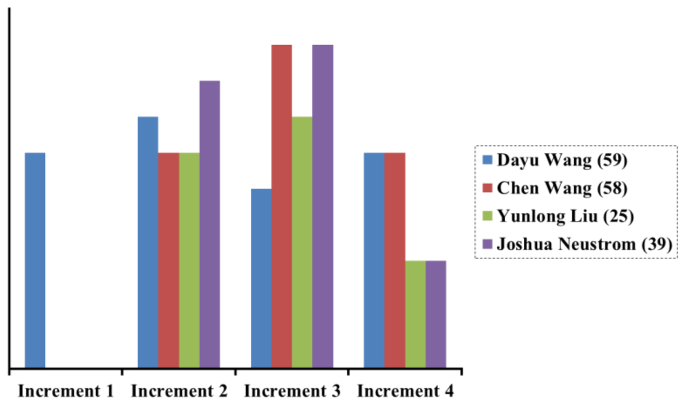


Figure 7. Coding workload for each member of our group amongst the four different project increment iterations. Note that a lower workload of coding does **not** necessarily indicate the lower overall contribution to the entire project, since the project is not just only coding. Also, this is the planned workload for each person and is subjected to change in future.

3. Burndown Chart

Figure 8 is the burndown chart obtained from the screenshot of our GitHub repository.

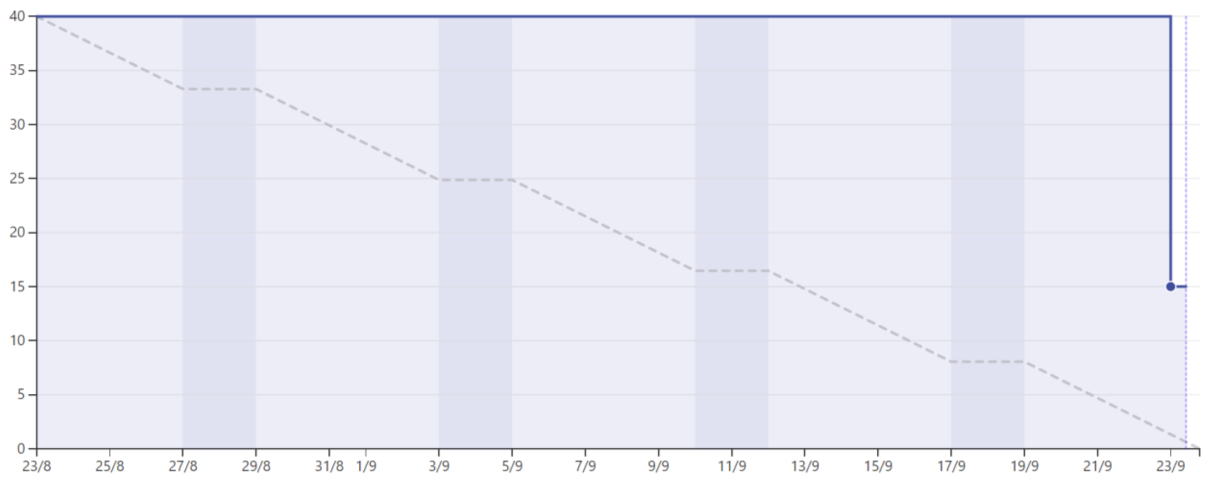
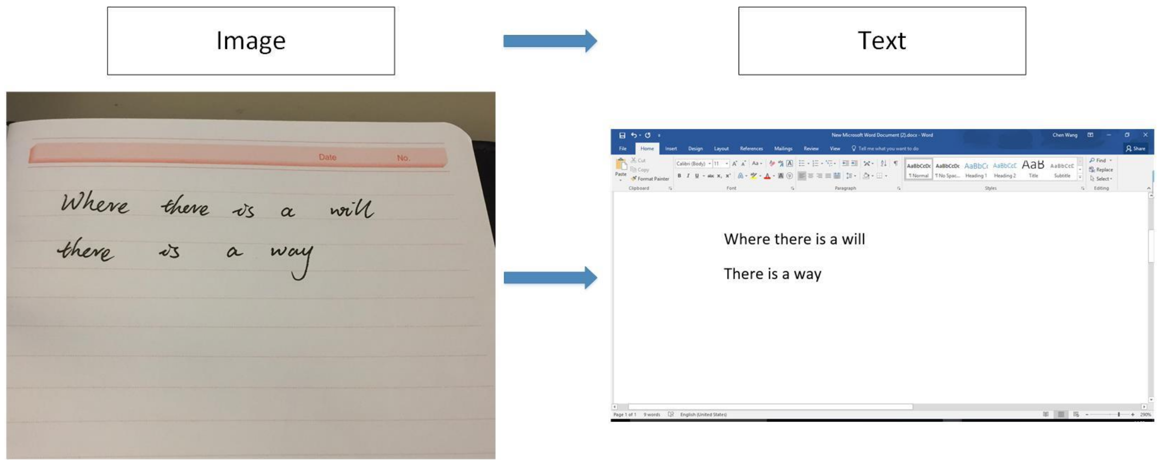


Figure 8. Burndown chart of the project of the *Pocket Manager* system.

- **Existing Services/REST API**

1. Google Cloud Vision - Google Cloud Vision is an API which used the optical character recognition (OCR) technology[3].

   The OCR is the mechanical or electronic conversion of images of typed, handwritten or printed text into machine-encoded text, whether from a scanned document, a photo of a document, a scene-photo on an image. For example, it can transform a photo which with a novel in it to a text with a simple novel. OCR technology always is used for the automated recognition of documents, credit cards, recognizing and translating signs on billboards—all of this could save time for collecting and processing data. Figure 9 has showed the OCR technology visually. In general, it needs some step to implement it: first, preprocessing the images. It needs to find the object we need in the picture, such as a novel in the picture, the book on the desk should be select at first. Second is to find the text. Before the transformation, finding the characters is required. Third is the recognition. Recognition and transform the characters in images to the text are the last but most important step.



Where there is a will
there is a way

Where there is a will
There is a way

Figure 9. Visual demonstration of an OCR service. OCR is the technology that extracts words from the images and transforms those into editable text.

   Google cloud vision is a very good API for the OCR technology, it can detect broad sets of objects in images, which means the cars, trees, sun, birds and so on in images can be detected and return the user as text information. It can also detect inappropriate contents, powered by Google Safe Search, easily moderate content from the crowd sourced images. Vision API enables users to detect different types of inappropriate content from adult to violent content. Google cloud vision even can do analysis the sentiment in the image. Like joy, sad, sorrow and angry can be detected and analysis. It can also detect the label, Logo, landmark, face and image attributes. A matter of a course, the Google cloud can extract the text in images, along with automatic language identification. Figure 10 has showed the functions of the Google cloud.

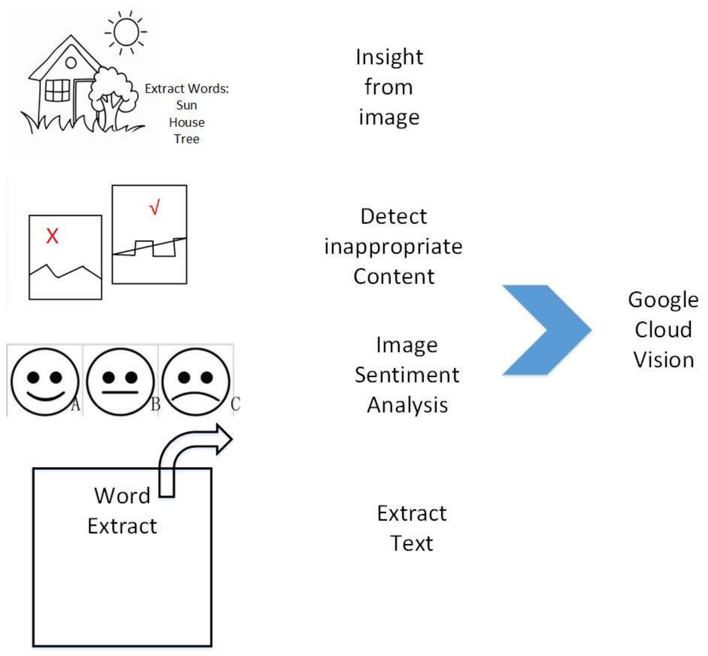


Figure 10. Functions of Google Cloud Vision. It includes insight the objects in images, detect inappropriate content, analysis the sentiment in images and extract text from images.

For our project, receipts can be scan and store automatically. So the camera and the google cloud vision API will be used for take photos for the receipts, extract the details of it and store them to the database.

2. Mongo DB[4]

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.

NoSQL is a database type which includes a wide variety of different database technologies. It is based on scale-out architecture using open source software, commodity servers and cloud computing instead of a whole large server and storage. With it, developer can work with applications that create a lot of volumes of new and rapidly changing datasets. It made the development cyclically then the small team can work iterating quickly and pushing code in very short time. And it can access by many devices and scaled globally on amount of users. The NoSQL database type has a lot of benefits, the most important is that it is more scalable and provides superior performance and their data model addresses issues that the relational model is not designed to address. On the large volumes of rapidly changing dataset, it worked very well. And it has quick schema iteration, frequent code push and easy to programming.

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. Figure 11 showed the details about the data exchange and store of the users.

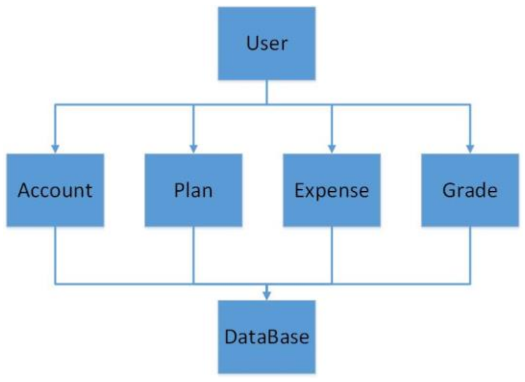


Figure 11. The usage of database. The users' account, plan, expense, and grade have been stored in the database.

3. Highcharts[5]

Highcharts is an application that we can use it to create interactive charts. It can be used for the SaaS projects, web applications, intranets, and websites. It is open source for the users, it based on native browser technologies and using the HTML5, all users can fork them on GitHub and participate in tech discussions. It can support a lot kind of charts for the user such as the basic line charts, area charts, column and bar charts, Pie charts, scatter and bubble charts, combination charts, dynamic charts, 3D charts, gauges charts, heat and tree maps. Figure 12 showed the kinds of charts that Highcharts work.

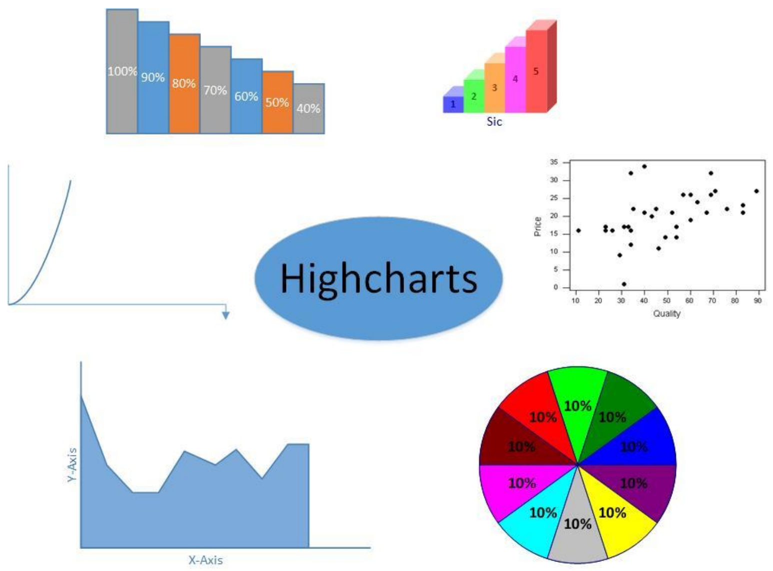


Figure 12. Functions of Highcharts. It supports the line charts, area charts, column and bar charts, pie charts, scatter and bubble charts and so on.

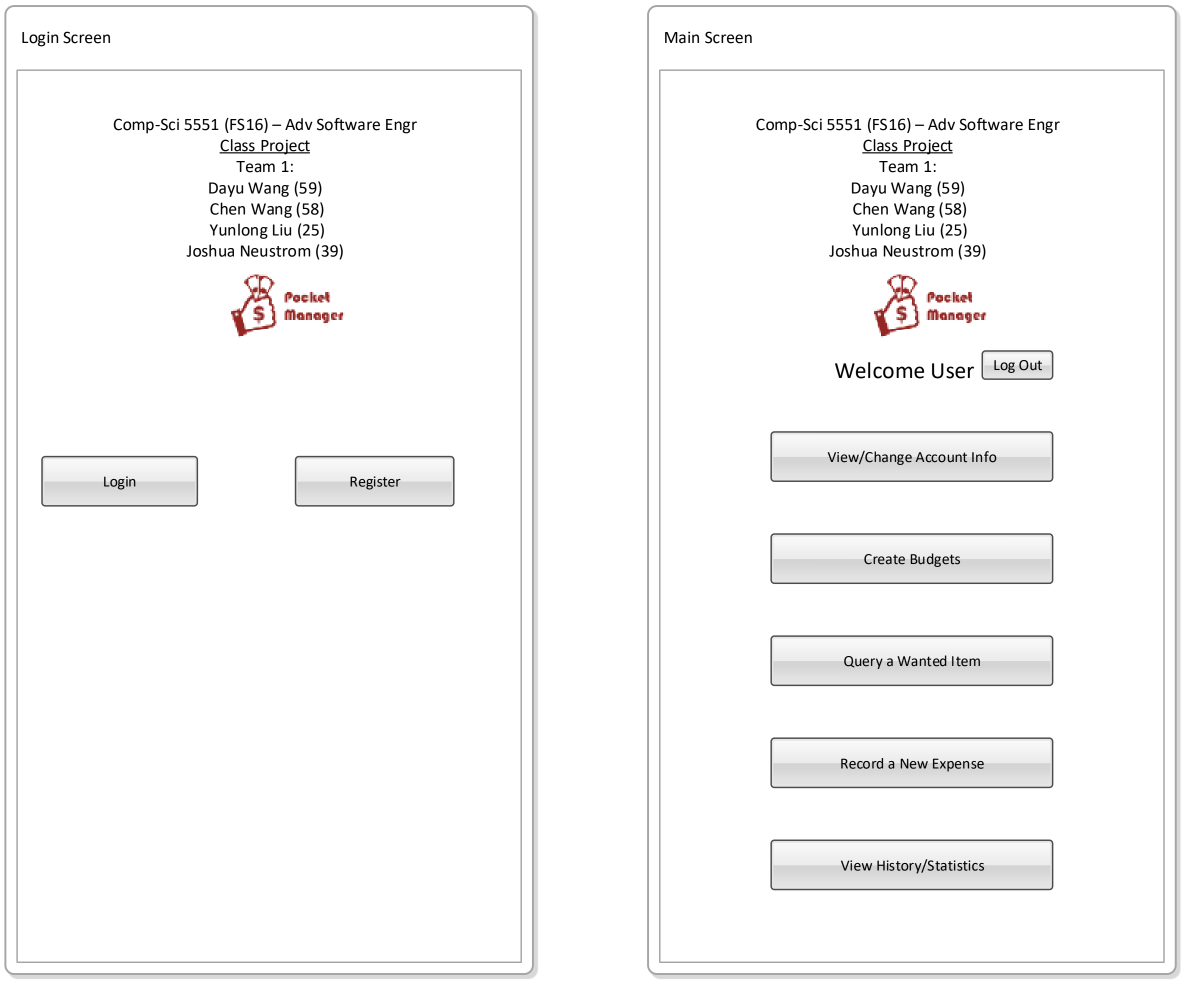
The most popular charts tools are the Highcharts and google charts, on the aspects of usability, setup, maintenance, product directions, google charts is better than Highcharts, but on the parts of meeting requirement, support, ease of business, the Highcharts get the higher score, so more small business and individuals choose the Highcharts for their project. A lot of people think that the Highcharts is more users friendly; it is easy to use and produces great results.

For our project, when the user chooses a plan, and when they review their expense in one mouth, the chart will be posted for the user then they can see their expenses in different parts visual. Then they can pay attention and improve their expense plan.

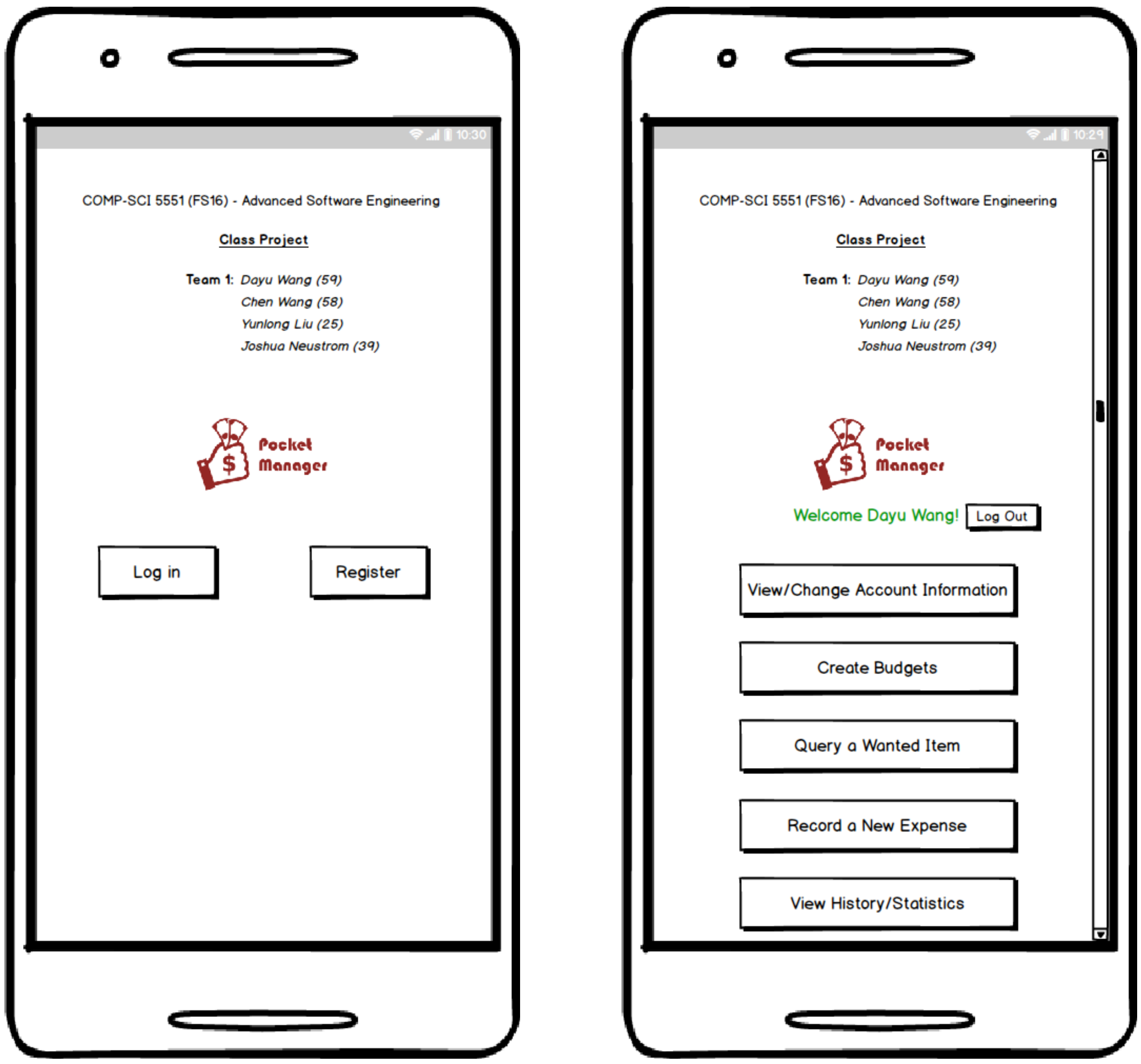
- **Detail Design of Features**

1. Wireframes and Mockups

The high-level wireframes and mockups of the *Pocket Manger* system to be developed are depicted in Figure 13, which act as the shell of our entire system that shows the main components in our system. For each smaller functional portion of the system, in future, whoever is assigned to a feature should design the UI of the feature by himself.



(a)



(b)

Figure 13. Wireframes and Mockups of the designation of user interfaces in the *Pocket Manager* system. These pictures are the shell model for our system. The further details of the UI of each feature will be designed by whoever assigned for that feature during the implementation process.

2. Architecture Diagram/Sequence Diagram/Class Diagram

Software architecture is defined by New York University (NYU) as the structure of structures for a system[6]. 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. Various API provide data to the application including the Google Sign In which provides authentication and user information with only a few lines of JavaScript and HTML. For assistance categorizing transactions several APIs are being considered including Google Shopping. 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 14.

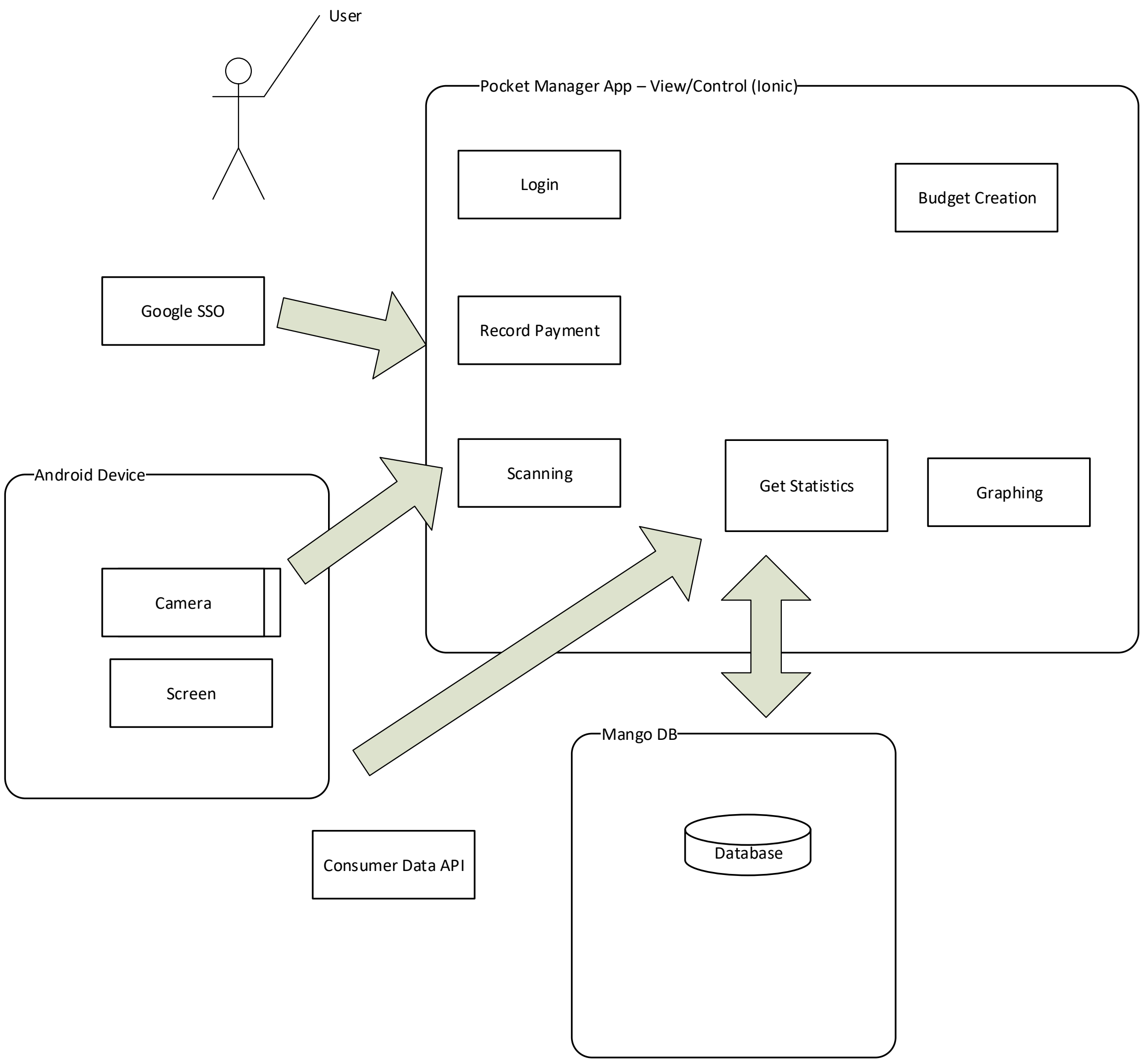


Figure 14. System architectural designation of the *Pocket Manager* to be developed.

Figure 15 and Figure 16 are the high-level UML class diagram and sequence diagram, respectively.

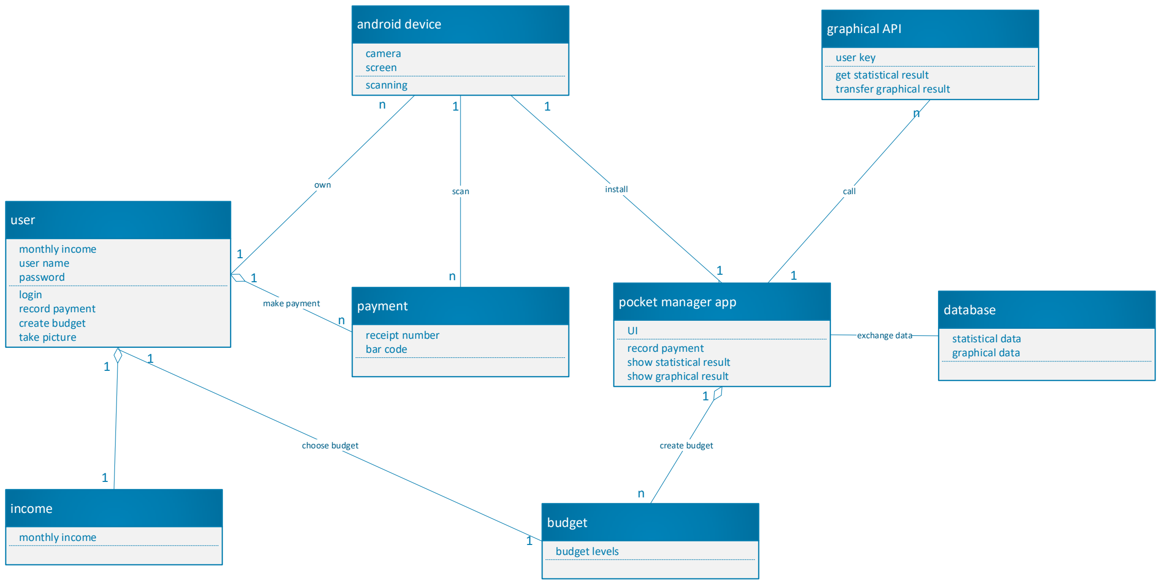


Figure 15. High-level UML class diagram for *Pocket Manager* system designation.

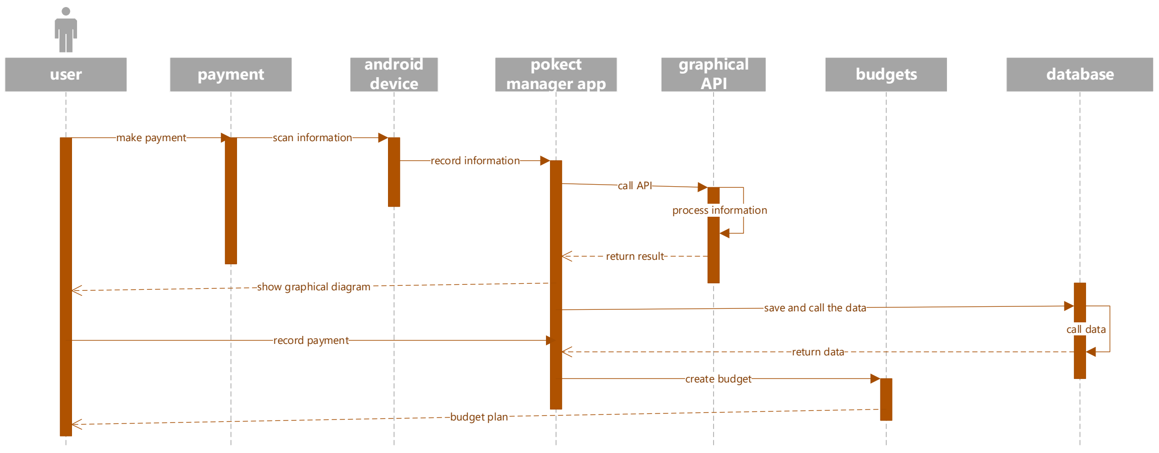


Figure 16. High-level UML sequence diagram for *Pocket Manager* system designation.

- **Testing**

1. Overview

Testing of the application involved a multi-step process including pre-submission audit, virtual device testing, and physical device testing. 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)	HTML could use a few extra comments to help it stay organized in the future
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?	N/A

Item	Feedback	Response
a.	Were major issues submitted to GitHub?	No major issues
b.	Do you have any major concerns about your code?	Not yet

3. Virtual Device Test

At least two versions of devices are used in the virtual testing including the following

a) Phone - Nexus 5x with 5.2in screen and API Level 23
b) Tablet - Nexus 9 with 8.83in screen and API Level 23
c) Wear – Android Wear Square with 1.65in screen and API Level 23

As part of the Virtual Testing the following checklist is used to make sure all of the features are fully tested. The list must be completed by a group member before the coding submission is considered complete.

The checklist 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. Virtual device 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?	N.A

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

4. 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.

Currently, we are searching for physical devices for application install. Some group members use iPhones and others have older versions of Android OS. Several tablets rom Dr. Song's lab may be used, but they may not be a valid test since they are rooted for other experiments. As a result, the following feedback was from virtual devices until we can secure the proper physical devices. With the large variety of devices in the android ecosystem, the long-term goal is to secure multiple types of devices for testing.

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.

Due to the simplicity of the first increment the user feedback was combined on a single form. In future increments, the amount of feedback is expected to expand. As a result, user feedback forms will be individually reported with a brief summary in the increment documentation (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?	Id numbers were confusing, did not look very flashy
c.	Any suggested changes?	Add a nicer background

5. 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, a support request and feedback button is expected to be added to give the user a way to quickly send feedback to a shared email account. Having a high number of outside testers provides a way to both develop marketing material and an existing set of users before the official application release to the market.

- **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 first 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 initial increment focused on the delivery of the skeleton of our app including the user interface and development process. The user interface was developed in ionic with the main objective being to make sure our development tools worked properly to create a hybrid application. In the case of some group members the development environment alone took several days to get up and running. With the successful setup of ionic and the development of the core pages, a solid foundation was built for future increments. Since the look and feel of an application creates a first impression with the user critical to marketing, the deployment of the interface design was intended to get feedback early and thereby improve the interface early in the process. Overall, changing the interface early in the process is preferable in order not to confuse the user by a modifying where items are found in the application. As a result, the initial deployment focused on CSS, HTML and other view components.

Along with the interface, the initial increment's deployment focused on refining our processes for development and release of the application. Along with a basic structure for the app in GitHub and Ionic, the process of submitting, testing, and deploying the software was a key deliverable for the initial increment.

The major concern with the deployment is the limited number of compatible devices available for testing the application. Due to the ever growing variety or Android devices, a large number of devices would be preferred for initial deployment. Since our application is not on the app market yet, the team relies on devices available through people known by members to deploy and test the devices.

2. Screenshots

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

a. Login Screen (Figure 20)

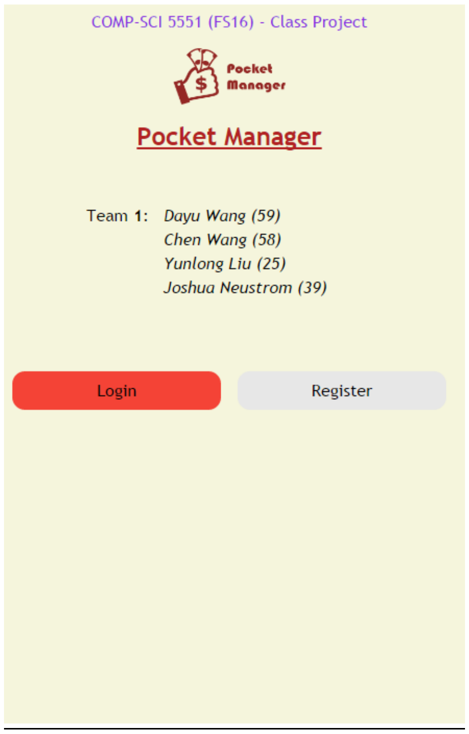


Figure 20. Login screen.

The initial login screen provides the users first interaction with the application. A darker tan colored background was selected to be easier on user eyesight while still allowing for ample contrasts for users with color blindness.

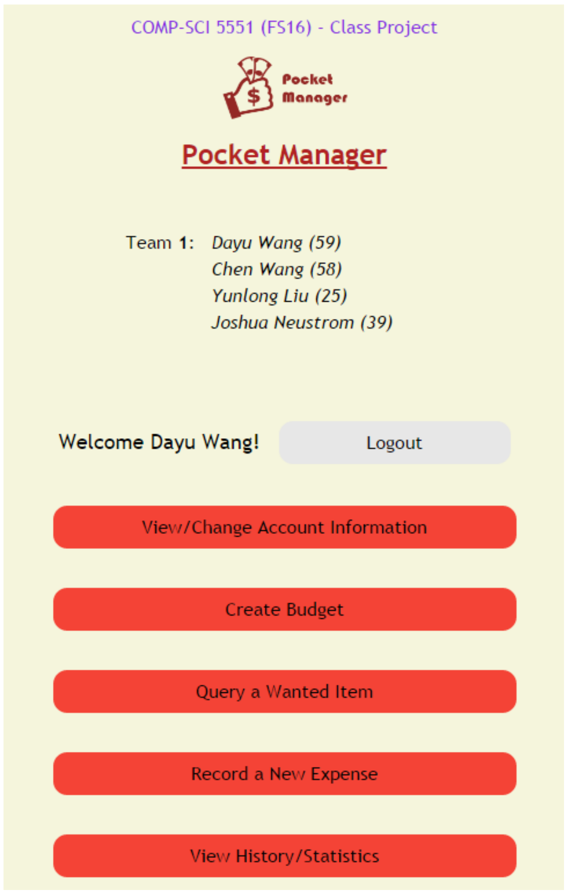
The two primary buttons are for login and registration of the user. The buttons are designed to be responsive to user touch or the hovering of a mouse by having two side arrows appear. The buttons also have unique colors for easier differentiation by users. The login button allows users to access the main page with all of the key functionality and confirms that the user is properly authenticated. The registration button is intended to take users to the registration page to collect key user information including preferred credentials for the application database.

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.

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

b. Main Menu (Figure 21)


COMP-SCI 5551 (FS16) - Class Project

Pocket Manager

Team 1: *Dayu Wang (59)*
Chen Wang (58)
Yunlong Liu (25)
Joshua Neustrom (39)

Welcome Dayu Wang!

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 logout button to take the user back to the login page and revoke their session access to the main menu. Features accessed include the option for changing account information initially registered by the user including the important function of updating the password. The next button is 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.

3. Wiki Page

A Wiki page was created for the increment 1 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-1
- 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 initial 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