Copyright

by

Anh Luong

2014

The Report Committee for Anh Luong
Certifies that this is the approved version of the following report:

ClosetStylist
From closet to style control – an Android app can rule them all

APPROVED BY
SUPERVISING COMMITTEE:

Supervisor: ________________________________
Adnan Aziz

____
Christine Julien

ClosetStylist
From closet to style control – an Android app can rule them all

by

Anh Luong, B.S.E.E.

Report

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

December 2014

Dedication

To my parents and my wife.

Acknowledgements

I would like to thank my supervisor, Professor Adnan Aziz, who has provided guidance on the ClosetStylist project, and to my reader Professor Christine Julien, who has helped review this paper. I would also like to thank Ile Jugovski and Truong Nguyen for their support to create a beautiful and user friendly UI for ClosetStylist. Most importantly, I want to thank my parents and my wife for all the hard work and sacrifices they have always made to support me throughout my work and studies.

ClosetStylist

From closet to style control – an Android app can rule them all

Anh Luong, M.S.E.

The University of Texas at Austin, 2013

Supervisor: Adnan Aziz

ClosetStylist is an Android app that helps users digitize their clothing inventory for better usage, manage worn history, laundry bags, and last but not least, suggest what to wear based on occasion and weather. The app utilizes a variety of free-of-charge services such as location and weather services combining with a proprietary clothes matching algorithm to recommend the most suitable outfit to users. In addition to the main features, ClosetStylist presents a friendly user interface that is easy to navigate control flow to keep users excited when they use the app.

Chapter 1 Introduction

1.1 Motivation

Wardrobe stylists are often hired by celebrities, models, public figures, and wealthy individuals to select their clothing for public appearances, or by professionals in entertainment industry for special events. Their services usually deem too expensive for majority of people whose budget is tight. Therefore, while most people love fashion and desire to look fashionable, not many people can afford these expensive types of services. The goal of the app to is help people with limited time and style to favorably present themselves and efficiently organize their closet.

We spend thousands of dollars every year on new clothes, but most of the time they end up getting lost in our closet after a couple times of usage. There are quite a few issues that this app is aspired to address. How should we organize all the items in our closets? How do we mix and match them wisely to utilize all items in our wardrobe without breaking our bank account for expensive consultation from costly stylists? Another often asked question is how do we know when our laundry bag is full to avoid running out of clothes? For those people without a washer and dryer at home, doing laundry could cost several hours waiting in the Laundromat (if we do not want to pay some extra bucks to have it done and folded, but we do not know how careful and gentle the assistant would be to our clothes).

Another problem that I usually face myself as a male individual without a delicate appetite for fashion is that I would go to the first item that I come across after rushing out of bed in the morning. As I open to door and get ready to leave for work, suddenly I realize the outfit I just choose several minutes ago is either too warm or too cold because I forget to check the weather when picking out the outfit. The temperature is not likely to be the same inside and outside the house. At that time, I wish there is an app that could tell me what I should wear based on the weather at my location. For those whose morning is a rush to prepare for a busy working day, an app like this could be of great help.

1.2 VISION

ClosetStylist is an Android app developed to address problems above. Its core functionalities includes assist women and men to pick the right outfit from their clothing inventory, organize their closets digitally, manage their laundry bags, and keep track of worn history. The ultimate goal is to help clients to get the most fashion value for their dollar by helping them to manage their closets wisely.

1.3 REPORT ORGANIZATION

This report is organized as followings: chapter 1 provides an introduction, chapter 2 discusses the user interface including mockups and workflow of the app, chapter 3 reviews the technology stack used in the app, chapter 4 describes the results and pain points, chapter 5 ends the report with summary, related and future work.

Chapter 2 User Interface Design

2.1 OVERVIEW

In this section, we explain the user interface design by first providing some typical user stories to guide how users can use this app. We then describe in detail some use cases to clarify the UI/UX flow for the main features of the app. We end this section with some mockups to show the look and feel of the app.

2.2 USER STORIES

The below stories highlight some features that ClosetStylist has achieved:

- Users can easily flip through every item in their own closet, which is a digital storage of pictures of their clothes taken by built-in camera phone or imported from gallery.
- Users can find out how many dirty items in their laundry bag and schedule for laundry.
- Users can go through their worn history and look for what garments they have worn on any particular date in the past.
- Users can choose any outfits that the app has programmatically picked from their closets based on occasion and the weather at the current location, and they can mix match with other garments if they do not like the app's initial suggestion.

2.3 USE CASES

[Amb] has illustrated an effective methodology to model and document the structures and behavior of software projects. The use cases presented in this section followed this Agile modeling approach to depict the interaction between user and the ClosetStylist app. Each use case consists of a UML activity diagram between two actors -

the user and the ClosetStylist Android app, a precondition that must be satisfied before starting this case, a purpose (or result) of this use case which describes the achievement after following the procedure, and the steps to achieve this result.

2.3.1 Register

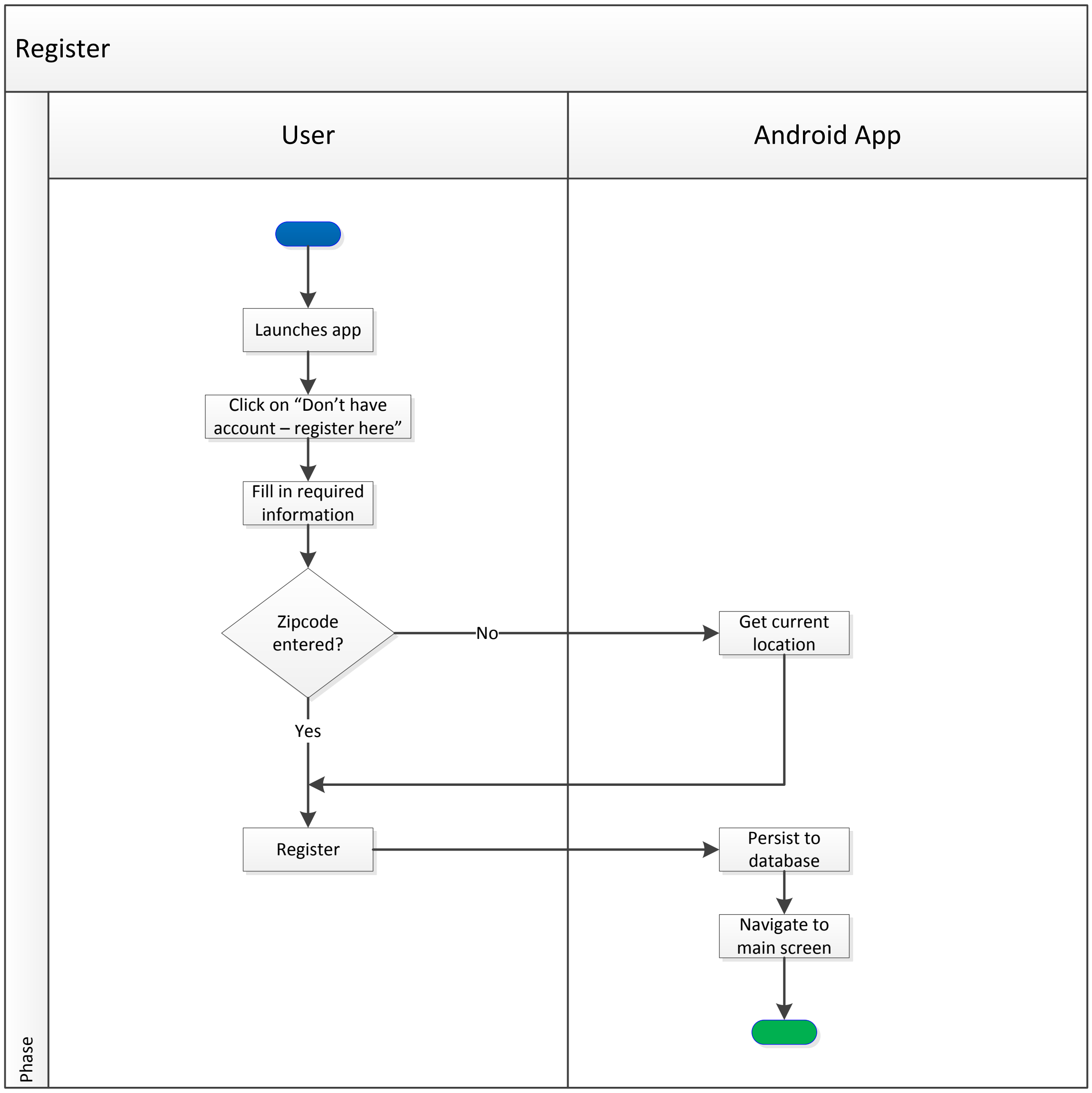


Figure 2.1: Register diagram

The precondition: ClosetStylist has already been installed on the device under test.

The purpose: show how the user can register an account to use the app for the very first time and how to populate all the required fields.

The steps: the app has its own simple authenticating method to validate, independent of any social networks, so that the users can still use the app if they choose to not enable any social network feature. When this is the very first time the app is launched, users have to click on the “Don’t have account – register here” and fill in the required fields, one of which is the postal code. This field is mentioned here because it is treated as the default location that the app will use if for any reason it fails to obtain the current location. If users do not know their current zipcode, they can click on “Get Location” and the app will find the zip code of the current location. Once the users have filled in all the required fields, they can click on the “Register” button to log in to the app. The users’ information is also persisted to the database so that users can login the next time without repeating the registration step.

2.3.2 Login

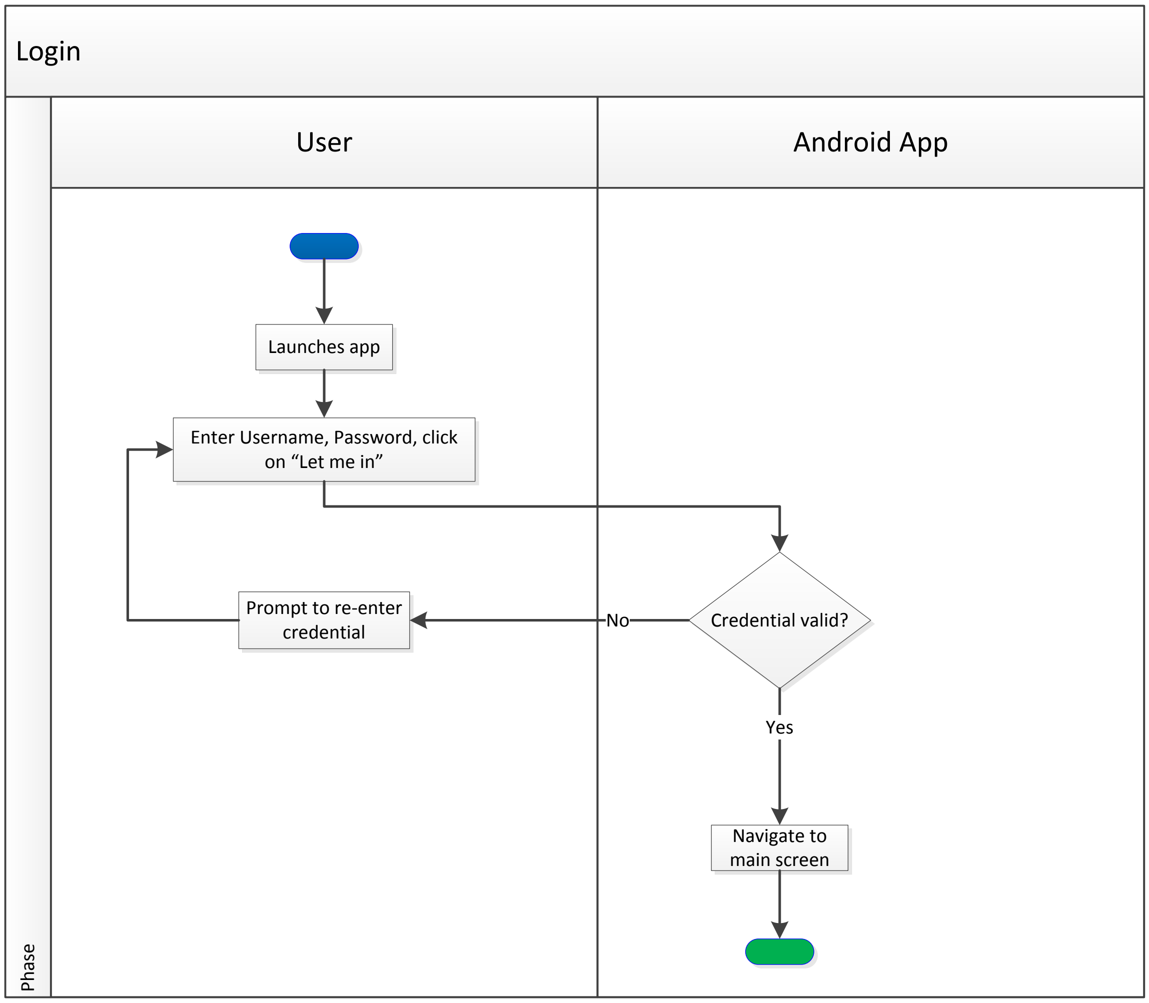


Figure 2.2: Login diagram

The precondition: users have already registered.

The purpose: users have to enter their credentials to login after registration step.

The steps: users launch the app and enter their username and password. The app will navigate to the main screen where users find helpful information such as the current location, date, weather, and they can proceed to any of the four main pages: Outfit of the day, My Closet, My Laundry Bag, My Outfit History.

2.3.3 Add new item

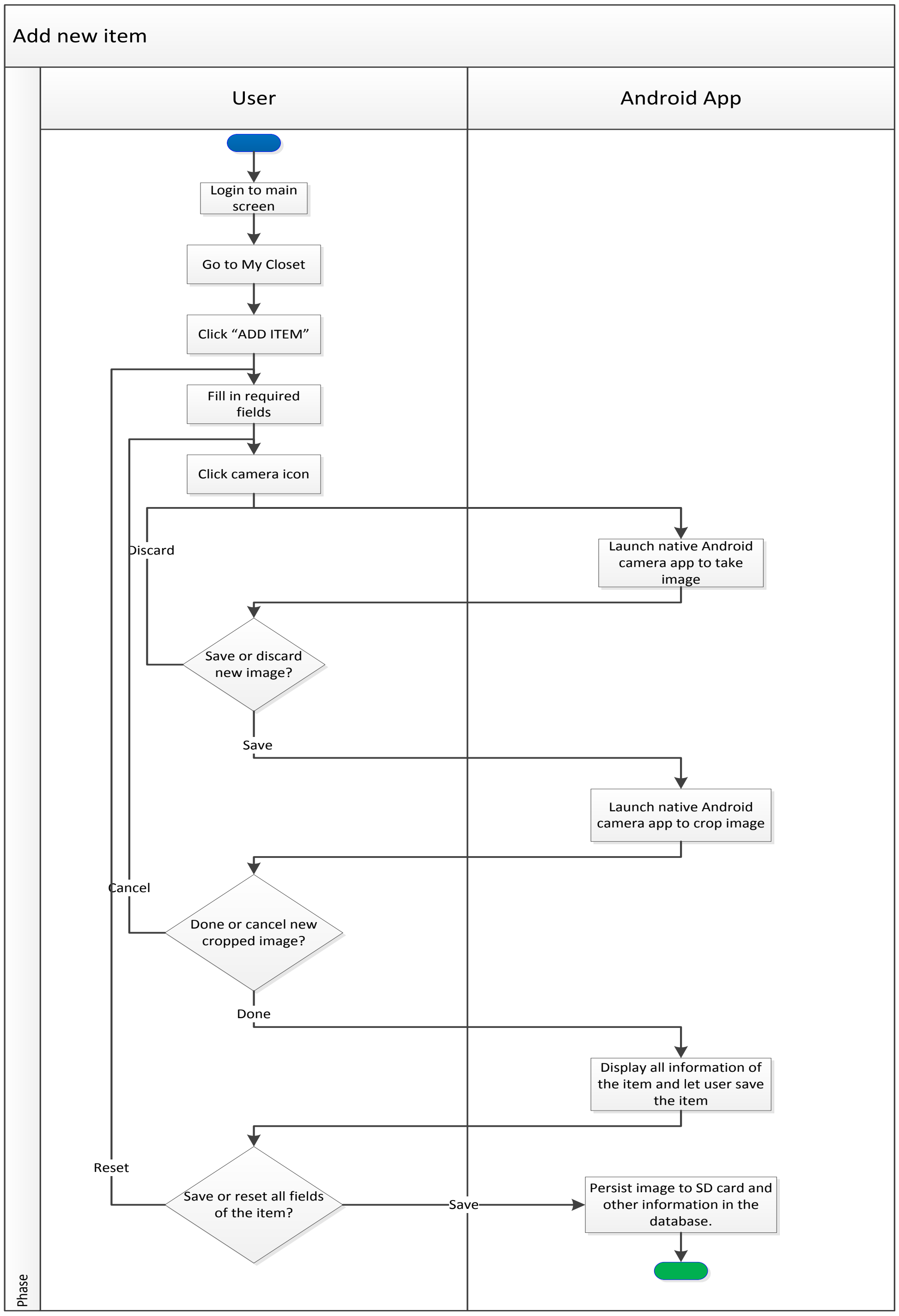

Add new item
User
Android App
Phase
Login to main screen
Go to My Closet
Click "ADD ITEM"
Fill in required fields
Click camera icon
Launch native Android camera app to take image
Discard
Save or discard new image?
Save
Launch native Android camera app to crop image
Cancel
Done or cancel new cropped image?
Done
Display all information of the item and let user save the item
Reset
Save or reset all fields of the item?
Save
Persist image to SD card and other information in the database.

Figure 2.3: Add new item diagram

The precondition: users are logged in.

The purpose: users have to populate their closets with the pictures of their clothes taken by built-in camera.

The steps: From the main screen, users click on “My Closet”. In the bottom of the My Closet page, there is “ADD ITEM” button. After filling in the required fields, users can take picture of the clothes by clicking on the camera icon. Once saved, users can crop the newly taken picture to get rid of the unnecessary parts. Users can choose either “Reset” all of the fields to their default values or “Save” the detail of this item by clicking on the corresponding button.

2.3.4 View or edit an item

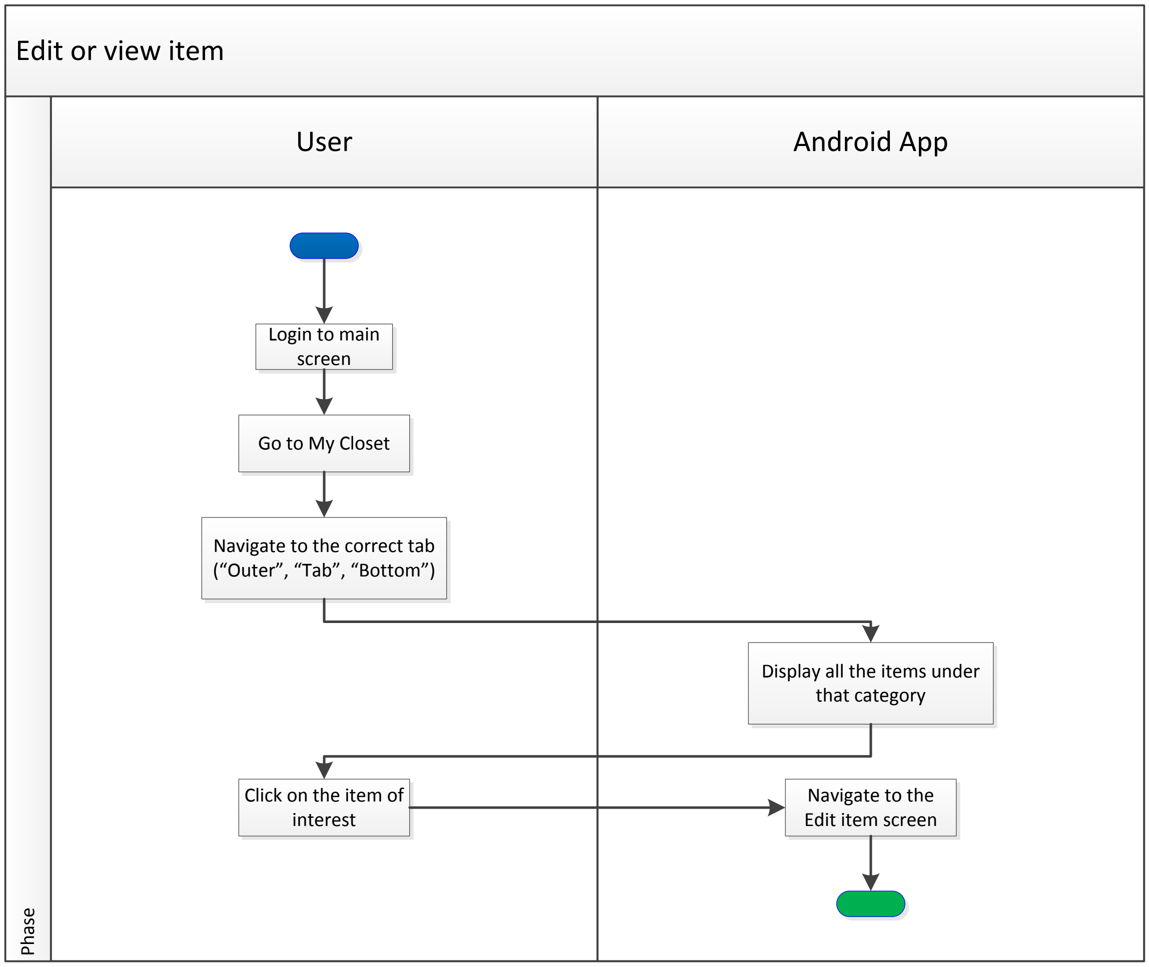

Edit or view item
User
Android App
Login to main screen
Go to My Closet
Navigate to the correct tab (“Outer”, “Tab”, “Bottom”)
Display all the items under that category
Click on the item of interest
Navigate to the Edit item screen
Phase

Figure 2.4: View or edit item diagram

The precondition: the item has been added to the closet.

The purpose: users can view the detail of an item and update the information if needed.

The steps: From the main screen, users click on “My Closet”. The wardrobe is categorized as “Outer”, “Top”, and “Bottom”. Users choose the tab that the wardrobe belongs to and click on the items they want to see. They can change any of the fields, or can even mark an item is dirty to be sent to laundry bag.

2.3.5 Pick an outfit

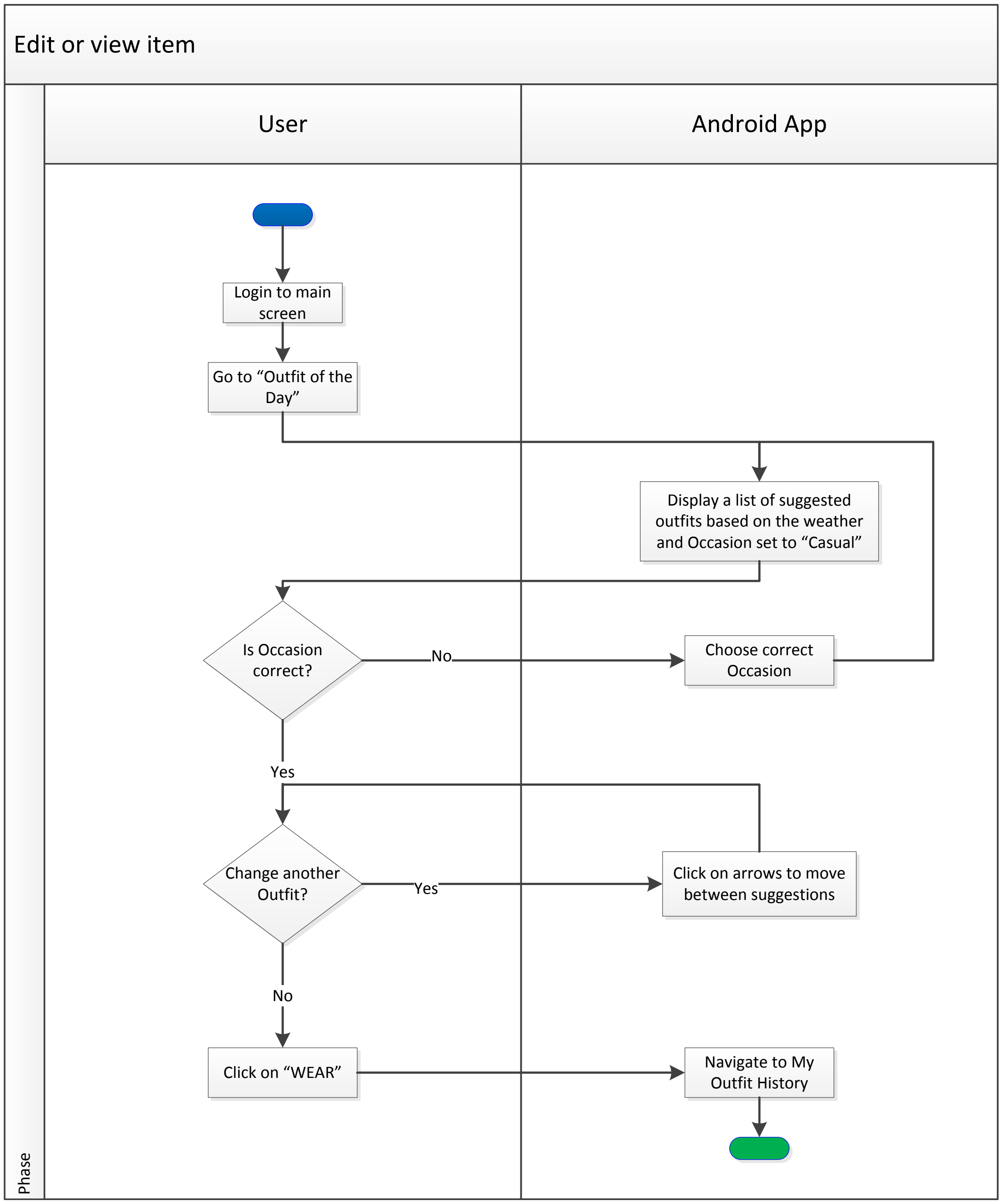

Edit or view item
User
Android App
Phase
Login to main screen
Go to “Outfit of the Day”
Display a list of suggested outfits based on the weather and Occasion set to “Casual”
Is Occasion correct?
No
Choose correct Occasion
Yes
Change another Outfit?
Yes
Click on arrows to move between suggestions
No
Click on “WEAR”
Navigate to My Outfit History

Figure 2.5: Pick an outfit diagram

The precondition: “My Closet” has been populated with some items in both Top and Bottom.

The purpose: the app suggests a list of outfits that best fit the user based on the weather and the chosen occasion.

The steps: From the main screen, users click on “Outfit of the Day”. The app will display a list of suggested outfits based on the current weather and the occasion set to “Casual”. There are five options for Occasion – “Formal”, “Semi_Formal”, “Casual”, “Day_Out”, “Night_Out” and users can choose the Occasion best fit their situation. There are arrows to switch to another Top or Bottom. There are double-arrows to let the user traverse through the list of suggested outfits. Once the users decide to choose a particular outfit, they can click on the “WEAR” button at the bottom and they will be navigated to the “My Outfit History” page. There is also a rank to inform the users how far they are from the first suggestion.

2.3.6 View outfit history

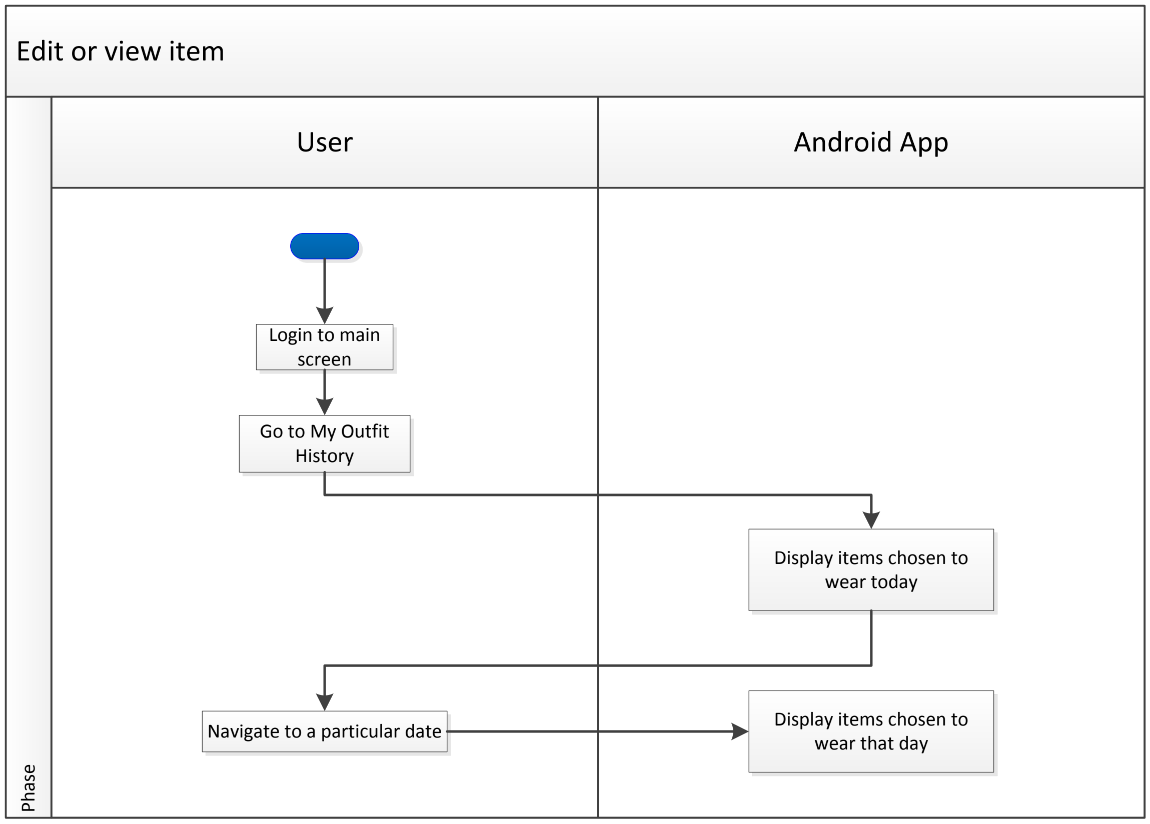

Edit or view item
User
Android App
Phase
Login to main screen
Go to My Outfit History
Display items chosen to wear today
Navigate to a particular date
Display items chosen to wear that day

Figure 2.6: View outfit history diagram

The precondition: users have already chosen to wear some outfits.

The purpose: the app displays the outfits that users have already worn on any particular day.

The steps: from the main screen, users click on “My Outfit History”. The page displays the outfits that users have worn on a particular day, starting from today. If users have worn several outfits on the same day, all of them will be listed in chronological order, starting with the one worn earliest on that day. User can click on any of them and they will be navigated to the “Outfit Preview” to see a more detail picture of the outfit.

2.3.7 Laundry bag

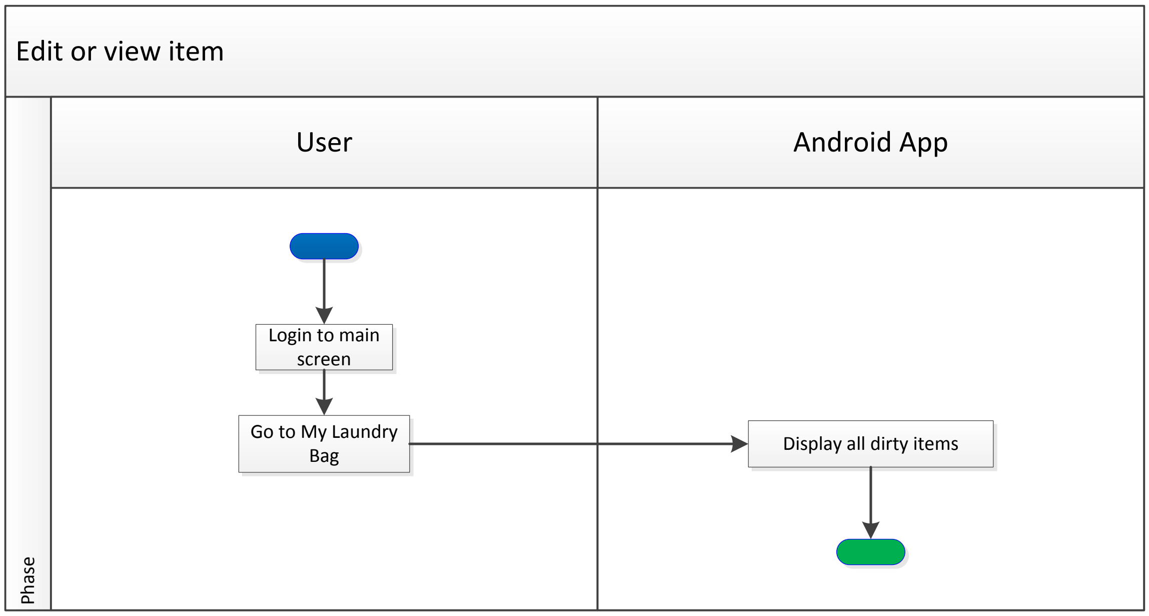


Figure 2.7: Laundry bag diagram

The precondition: users have already chosen to wear some outfits.

The purpose: the app displays the dirty items so that the users can schedule to wash them.

The steps: from the main screen, users click on “My Laundry Bag”. The page will display all the dirty items as a list. Users can click on any of them to view more detail.

2.4 MOCKUPS

Balsamiq [Bal] was initially used to create mockups as it was user friendly and its online version was free for students. Figure 2.8 is an example of original mockups:

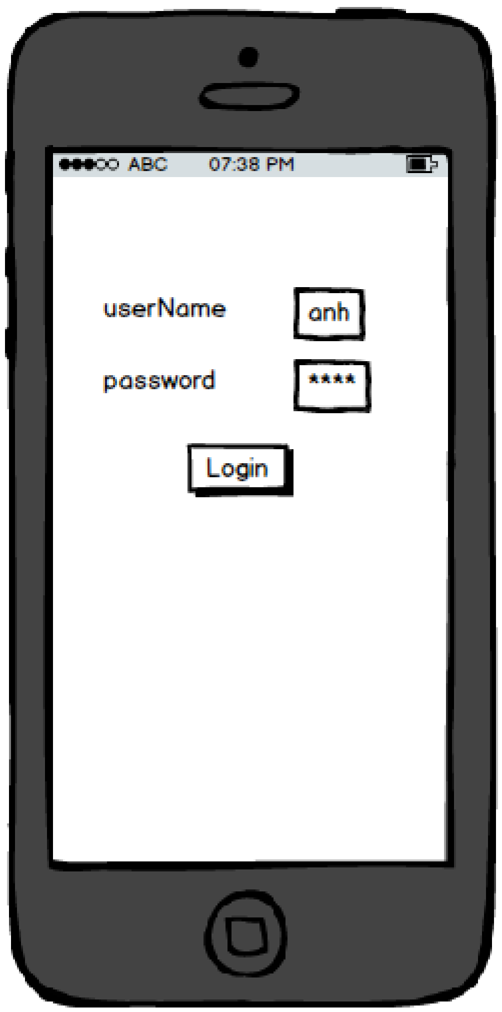
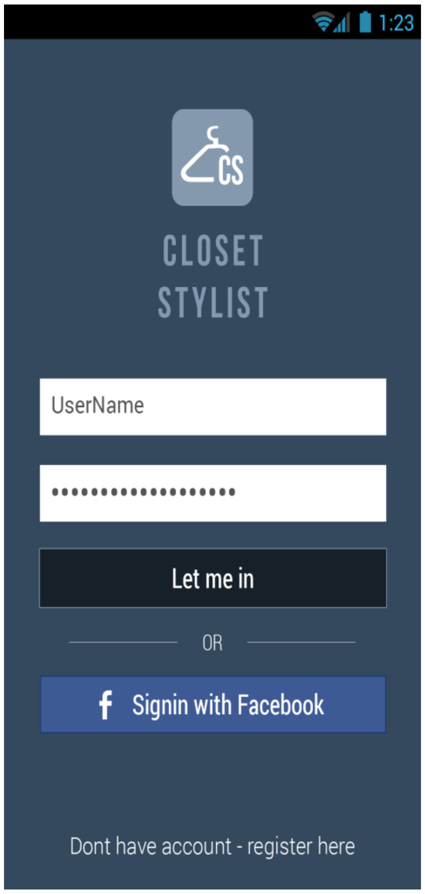


Figure 2.8: Balsamiq user login mockups.

As the development continued, some limitations of Balsamiq such as the limitations of Android UI elements and difficulty in sharing feedbacks showed up. We switched to InVision App [Inv], an extremely powerful design tool to create a fully interactive prototypes and wireframes, as well as collaborate to share vision and gain feedbacks from all stakeholders. All of the mockups from InVision App will be illustrated in the following section together with a storyboard created to help readers to easily visualize the workflow of the app.

2.4.1 Login and Registration



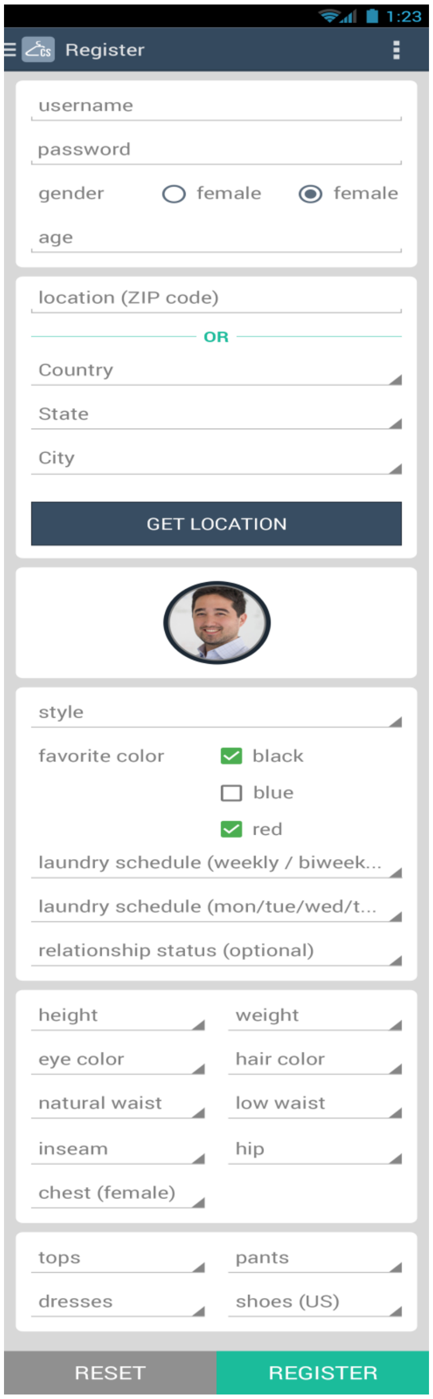


Figure 2.9: User login and registration mockups.

To begin with, the new user will register with the app their username, password, default location. After registration is done, the user can login and logout of the app.

2.4.2 Main Screen and Side Menu

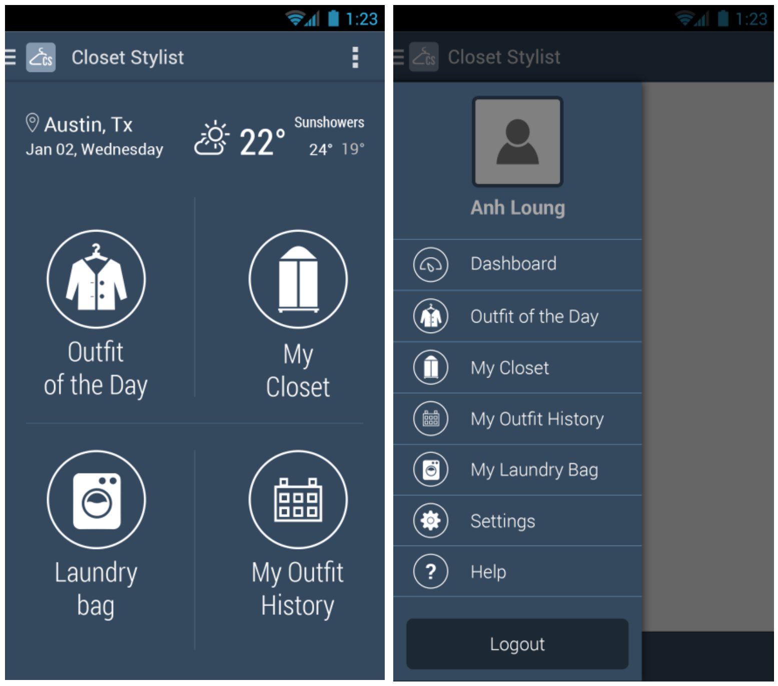
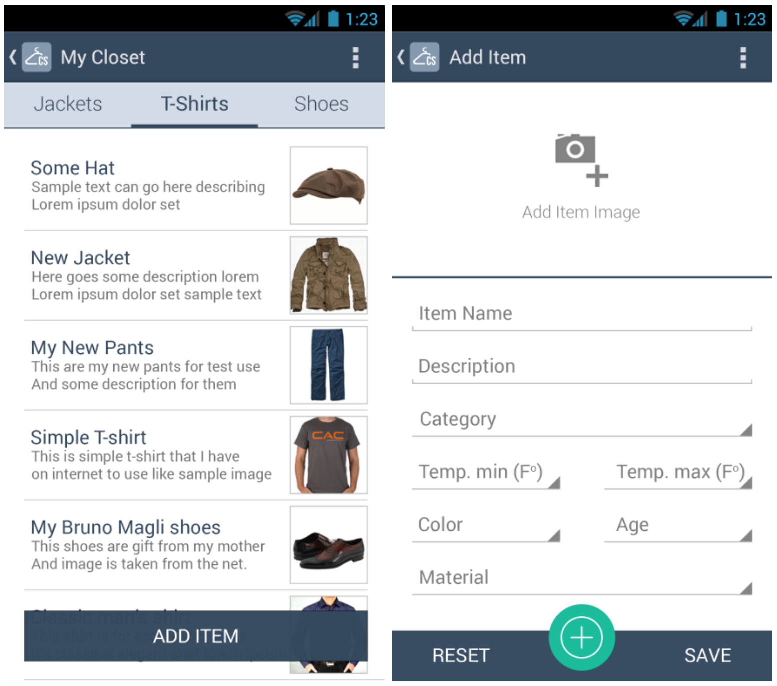


Figure 2.10: Main Screen and Side Menu mockups.

After registration for the first time or after login, users will be navigated to the main screen in Figure 2.10 which displays useful information about the current location, date, temperature. It also provides options to navigate to the main features of the app such as suggesting picking Outfit of the Day, organizing My Closet, managing Laundry bag, or viewing My Outfit History.

To make navigation between screens in the app easier, user can take advantage of the provided a side menu (or drawer as Android term).

2.4.3 My Closet and Add Item



Some Hat
Sample text can go here describing
Lorem ipsum dolor set

New Jacket
Here goes some description lorem
Lorem ipsum dolor set sample text

My New Pants
This are my new pants for test use
And some description for them

Simple T-shirt
This is simple t-shirt that I have
on internet to use like sample image

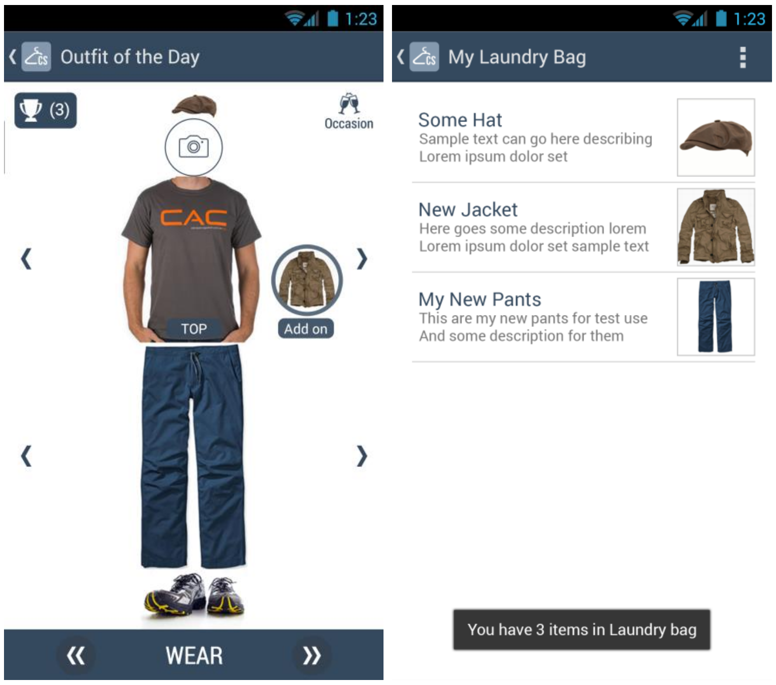
My Bruno Magli shoes
This shoes are gift from my mother
And image is taken from the net.

Figure 2.11: My Closet and Add Item mockups.

In order to use the app, user needs to import photos of their clothes from his/her phone's built-in camera and enters additional information about the items. Each item will be categorized as either top or bottom, together with its corresponding styles, materials, color.

After the item is saved in the closet, user can also edit or delete the item from the closet. Once the closet is fully populated with all the items, my closet should look like the mockup in Figure 2.11.

2.4.4 Outfit of the Day and Laundry bag



Some Hat
Sample text can go here describing
Lorem ipsum dolor set

New Jacket
Here goes some description lorem
Lorem ipsum dolor set sample text

My New Pants
This are my new pants for test use
And some description for them

You have 3 items in Laundry bag

Figure 2.12: Outfit of the Day and Laundry bag mockups.

When the user is ready to pick the outfit, he/she can navigate to the Outfit of the Day shown in Figure 2.12 and find the list of suggested outfits based on today's weather and the occasion. The selection algorithm ranks different outfits (combination of top, bottom, and may be outer if the weather is cold). User can choose among different occasions including formal, semi-formal, casual, day-out, night-out and different outfits will be recommended. A mix and match option is also provided through the arrows next to top and bottom if user wants a different piece in the recommended outfit. User can move back and forth between suggested outfits by pressing double arrows at the bottom of the screen.

The user can decide to wear the outfit by clicking on the "Wear" button, and items will be processed to determine if it is dirty or not and placed in laundry bag as shown in Figure 2.12. The reason for processing an item instead of moving it to the laundry bag right away because there are certain pieces of clothes that we can wear more than once such as jackets or jeans.

2.4.5 Outfit History and Outfit Preview

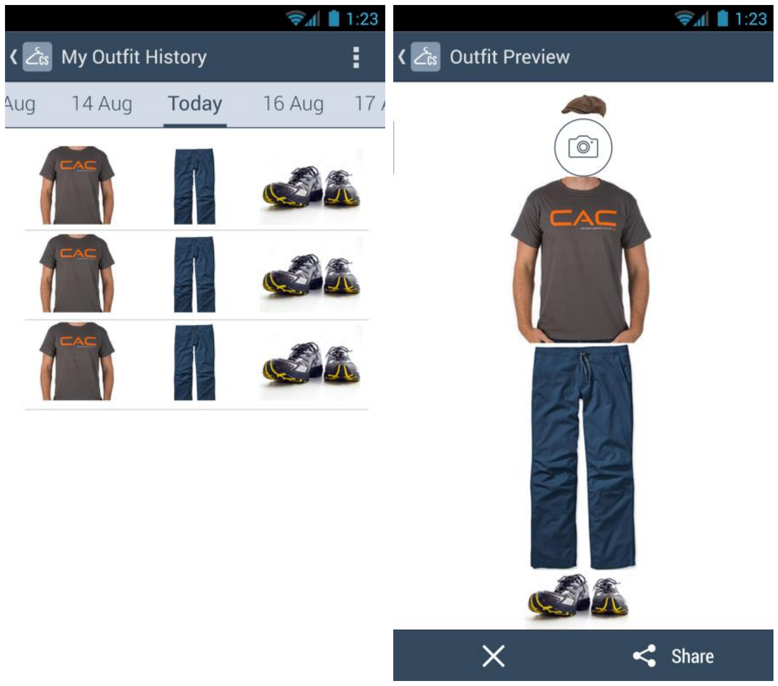


Figure 2.13: Outfit History and Outfit Preview mockups.

When the user clicks on the "Wear" button as described in the previous section, the app will take the user to the Outfit History screen which lists all the outfits have been chosen today and in the past as shown in Figure 2.13. This screen can also be accessed from the main screen or the side menu.

Once the user is in the Outfit History screen, outfit worn today is shown first and if there is more than one outfit, they will be displayed in chronicle order. User can find what he/she has worn on any particular day in the past by moving to the tab for that day, clicking on an outfit entry in the list, and the outfit will be displayed in the Outfit Preview screen as in Figure 2.13.

Chapter 3 Implementation

3.1 TECHNOLOGY STACK

ClosetStylist is an Android app and hence Java is used as the main programming language. Table 3.1 below shows a summary of my development environment.

Language	Java
IDE	Eclipse Juno 4.2, Kepler 4.3
Additional code editor	GVIM
SDKs	JDK 1.6, Android 2.3.3, Facebook 3.0
Test equipment	Samsung S3
System	Windows 7 64-bit, Ubuntu 12.04 32-bit, Ubuntu 12.10 64-bit.

Table 3.1: Development environment.

The app was initially developed on a powerful laptop with Intel i7-3720 2.6GHz, 16GB RAM, Windows 64-bit to run on the Android simulator. When I needed to run the app on a Samsung S3 device, the development was moved to a laptop with Intel Core 2 duo, 4GB RAM, Ubuntu 12.10 32-bit, and a desktop with AMD Quad-Core, 16GB RAM, Ubuntu 12.04 64-bit.

In addition to the development environment, several technologies were applied in this app; some were open-source while some were proprietary. We are going to describe each of the main technologies next and the reason why we chose them over the others and how did we deploy them.

3.1.1 Location Service

In this app, GeoNames database was used as a service to convert geographic coordinates (the longitude and latitude) of a location to postal code. Geonames was chosen over Yahoo service (Yahoo BOSS PlaceFinder) because it was free, and Yahoo service used proprietary WOEID (Where On Earth Identification number). Besides, the later was subject to change by Yahoo, and would cause problem if we decide to switch to another service.

Geonames provided a lot of services in many formats such as XML and JSON. In this app, we employed the service that converted geographic coordinates to postal code (and then city and country) and vice versa by sending a HTTP request with the following URLs:
http://api.geonames.org/postalCodeSearchJSON?postalcode=78758&maxRows=10&username=demo and http://api.geonames.org/findNearbyPostalCodes?lat=30.4883997&lng=-97.7175117&username=demo. The response was in JSON format and our tasks were to collect and parse the response, then display it in the main screen of the app.

3.1.2 Weather Service

We looked at several weather services including Yahoo Weather, World Weather Online, and Open Weather Map. [SWA] provided the sample code to retrieve weather information from Open Weather Map and made it an ideal choice for our weather service. The weather response consisted of a lot of information including but not limiting to geographic coordinates, temperatures, humidity, pressure, wind, rain. As of this writing, only a subset of the code was used including geographic coordinates, temperatures, and rain.

3.1.3 Clothes Matching Service

This smart service is our proprietary algorithm to provide suggestions on which wardrobe users should put on based on the available items in their closets, the current weather information, and the occasion of the event, and gender.

3.1.3.1 High-level design

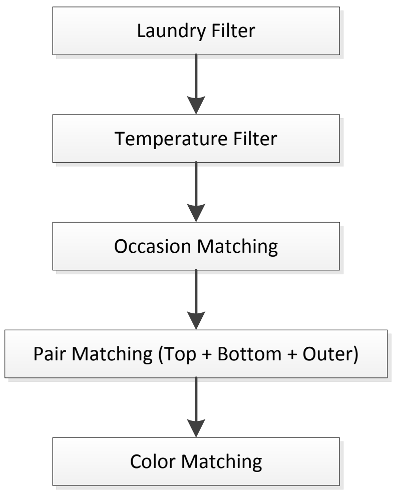


Figure 3.1: 5-step clothes matching algorithm.

There are five steps to create the list of suggested outfits. Each step is an essential part of the algorithm and must be executed in the same order described in Figure 3.1. The inputs to the algorithm are all of the factors mentioned above and the output is a list of outfit in descending order of score. Each outfit consists of a top, bottom, and an optional outer if the temperature is in a certain range. While the first two steps are used to obtain a valid set of items to select from, the last 3 steps are used to score points for each outfit based on several factors. An overview of each step is given below.

Laundry Filter: eliminate dirty items out of the process.

Temperature Filter: eliminate items that do not cover the range of today's maximum temperature and minimum temperature of the current location.

Occasion Matching: each item is given a score for the chosen occasion. For example, a short is graded low for the "Formal" occasion but it gets high score in the "Day_Out" occasion.

Pair Matching: each combination of a top item with a bottom item (and optionally an outer) will be scored based on its category. This point is added together with the point of each item in the previous step to the outfit.

Color Matching: the color combination of the top, the bottom, and the outer of the outfit will be given a score. This is added to the points from the previous steps to produce the final score. The result is a list of outfits in the order of descending points.

3.1.3.2 Low-level design:

A deeper discussion on the low-level design and implementation is provided for each of the five steps.

3.1.3.2.1 Step 1 - Laundry Filter

This step is pretty simple, given that each item has a "dirty" attribute to specify an item is clean or dirty. The implementation is simply a query the database of clothes in the closet to obtain a list of clean items.

3.1.3.2.2 Step 2 - Temperature Filter

Each item is assigned a range of temperature in which it can be worn. Among different attributes of an item, material and style are the two that we think the temperature could be based on. After some careful consideration, style was chosen over material because it was more relevant to temperature. The rationale of this choice is that if an item of a specific material is not warm enough, then another item of the same or different material can still be worn outside. With regards to style, once a particular style is chosen, it is more difficult to match with another style to keep warm.

The look-up Table 3.2 below is used to define the range of each style per gender.

Style - Men	Temp Min	Temp Max
Pants	-999	999
Jeans	-999	999
Shorts	-999	999
Dress_Shirt	-999	999
Casual_Button_Down_Shirt	-999	999
Polo	-999	999
T_Shirt_Short_Sleeve	-999	999
T_Shirt_Long_Sleeve	-999	999
Sweater_And_Sweatshirt	-999	999
Coat_And_Jacket_Light	-999	75
Coat_And_Jacket_Heavy	-999	40
Style - Women	Temp Min	Temp Max
Pants	-999	999
Jeans	-999	999
Legging_Skinny	-999	999
Shorts	-999	999
Skirts	-999	999
Collared_And_Button_Down	-999	999
Blouse_Short_Sleeve	-999	999
Blouse_Long_Sleeve	-999	999
Blouse_Sleeveless	-999	999
T_Shirt_Long_Sleeve	-999	999
T_Shirt_Short_Sleeve	-999	999
Tank_Camisoles	-999	999
Party_Top	-999	999
Tunic	-999	999
Pull_Over	-999	999
Cardigan	-999	999
Sweater_And_Sweatshirt	-999	999
Vest	-999	999

Coat_And_Jacket_Light	-999	75
Coat_And_Jacket_Heavy	-999	40

Table 3.2: Temperature range per Style.

Summary of step 1 and 2: based on the current maximum and minimum temperature, we made two queries the database of wardrobes to obtain the two separate lists of clean items in our closet, one for top and one for bottom.

3.1.3.2.3 Step 3 - Occasion Matching

Each item is given different point based on gender, category, style, and occasion. This step has higher weight than pairing and color matching steps because occasion matching is more important to the final outfit in our opinions.

In this step, the list of items outputted from the above steps is processed together with the selected occasion by the methods of the OccasionMatching object, and the result is two lists of items with score, one for top and one for bottom.

Table 3.2 and Table 3.3 show the score tables of Occasion Matching of male and female, respectively.

		Point				
Category	**Style**	**Formal**	**Semi_Formal**	**Casual**	**Day_Out**	**Night_Out**
Bottom	Pants	40	40	5	0	20
Bottom	Jeans	5	20	40	20	40
Bottom	Shorts	0	0	20	40	0
Top	Dress_Shirt	40	40	5	0	5
Top	Casual_Button_Down_Shirt	5	20	20	20	40
Top	Polo	5	20	40	5	20
Top	T_Shirt_Short_Sleeve	0	0	20	40	5
Top	T_Shirt_Long_Sleeve	0	5	20	5	20
Top	Sweater_And_Sweatshirt	0	5	20	20	5
Top	Coat_And_Jacket_Light	40	40	5	0	20
Top	Coat_And_Jacket_Heavy	5	5	5	5	20

Table 3.2: Occasion Matching score table for male.

		Point				
Category	**Style**	**Formal**	**Semi_Formal**	**Casual**	**Day_Out**	**Night_Out**
Bottom	Pants	40	40	5	0	20
Bottom	Jeans	5	20	40	20	20
Bottom	Legging_Skinny	0	5	20	20	40
Bottom	Shorts	0	0	5	40	5
Bottom	Skirts	5	20	20	20	20
Top	Collared_And_Button_Down	40	20	0	0	5
Top	Blouse_Short_Sleeve	20	40	20	20	20
Top	Blouse_Long_Sleeve	40	40	5	5	20
Top	Blouse_Sleeveless	5	20	20	40	20
Top	T_Shirt_Long_Sleeve	5	20	40	5	5
Top	T_Shirt_Short_Sleeve	0	5	40	40	5
Top	Tank_Camisoles	0	0	5	40	20
Top	Party_Top	0	0	5	20	40
Top	Tunic	0	5	20	20	20
Top	Pull_Over	0	0	20	20	0
Top	Sweater_And_Sweatshirt	20	20	5	5	20
Top	Coat_And_Jacket_Light	40	20	0	0	20
Top	Cardigan	40	40	20	5	20
Top	Vest	40	20	5	0	20
Top	Coat_And_Jacket_Heavy	5	5	5	5	20

Table 3.3: Occasion Matching score table for female.

3.1.3.2.4 Step 4 - Pair matching

Similar to previous steps, we create a look-up table (Table 3.4 and Table 3.5) for each combination of a top item and a bottom item, and an optional outer piece based on the weather and gender. One thing to notice here is that the grading scale is lower than that of the occasion matching step because this step is not as important and as a result should have less weight on the final score.

In this step, each of the items in the top list is paired with an item in the bottom list outputted from the previous step, and an optional outer item picked from the top list if the temperature is within a certain range by running methods of the PairMatching object. At the end of this step, we obtain a single list, in which each entry contains a combination of a top item, a bottom item, optionally an outer, and the total score of this combination.

Bottom	Top	Point	Outer
Pants	Dress_Shirt	20	No
Pants	Casual_Button_Down_Shirt	5	No
Pants	Polo	20	No
Jeans	Casual_Button_Down_Shirt	5	No
Jeans	Polo	16	No
Jeans	T_Shirt_Short_Sleeve	20	No
Jeans	T_Shirt_Long_Sleeve	20	No
Shorts	Casual_Button_Down_Shirt	6	No
Shorts	Polo	6	No
Shorts	T_Shirt_Short_Sleeve	20	No
Shorts	T_Shirt_Long_Sleeve	5	No

Table 3.4: Pair Matching score table for male.

Bottom	Top	Point	Outer
Pants	Collared_And_Button_Down	20	No
Pants	Blouse_Short_Sleeve	16	No
Pants	Blouse_Long_Sleeve	16	No
Pants	Blouse_Sleeveless	16	Yes
Pants	Tank_Camisoles	10	Yes
Pants	Party_Top	10	No
Jeans	Blouse_Short_Sleeve	10	Yes
Jeans	Blouse_Long_Sleeve	10	No

Jeans	Blouse_Sleeveless	10	Yes
Jeans	T_Shirt_Long_Sleeve	20	No
Jeans	T_Shirt_Short_Sleeve	20	No
Jeans	Tank_Camisoles	16	Yes
Jeans	Party_Top	16	No
Jeans	Pull_Over	16	No
Legging_Skinny	Blouse_Short_Sleeve	10	No
Legging_Skinny	Blouse_Long_Sleeve	10	No
Legging_Skinny	Blouse_Sleeveless	10	Yes
Legging_Skinny	T_Shirt_Long_Sleeve	6	No
Legging_Skinny	T_Shirt_Short_Sleeve	6	Yes
Legging_Skinny	Tank_Camisoles	6	Yes
Legging_Skinny	Party_Top	20	No
Legging_Skinny	Tunic	20	No
Legging_Skinny	Pull_Over	6	No
Legging_Skinny	Collared_And_Button_Down	6	No
Shorts	Blouse_Short_Sleeve	10	No
Shorts	Blouse_Long_Sleeve	10	No
Shorts	Blouse_Sleeveless	10	Yes
Shorts	T_Shirt_Long_Sleeve	16	No
Shorts	T_Shirt_Short_Sleeve	20	Yes
Shorts	Tank_Camisoles	20	Yes
Shorts	Party_Top	6	No
Shorts	Pull_Over	10	No
Skirts	Collared_And_Button_Down	10	No
Skirts	Blouse_Short_Sleeve	20	No
Skirts	Blouse_Long_Sleeve	16	No
Skirts	Blouse_Sleeveless	20	Yes
Skirts	T_Shirt_Long_Sleeve	16	No
Skirts	T_Shirt_Short_Sleeve	16	Yes
Skirts	Tank_Camisoles	10	Yes
Skirts	Party_Top	20	No
Skirts	Pull_Over	6	No

Table 3.5: Pair Matching score table for female.

3.1.3.2.5 Step 5 - Color matching

It is impossible to list every existing shade of color because there are just too many. We decided on twelve basic colors: beige, black, blue, brown, gray, green, orange, pink, red, violet, white, yellow, and an additional option of 'multicolor_pattern' to accommodate items with more than one color. Therefore, we have a total of 13 colors to work with. These 13 colors can further be divided into two groups – "Color" (blue, green, violet, red, yellow, orange, pink, multicolor_pattern) and "Neutral" (gray, white, black, brown, beige). [CWL] has listed which colors are complimentary to one another. Besides, "Neutral" colors can be easily matched with other while the "Color" colors are more restricted. These relations are expressed through the points given to each combination of these colors in the color score Table 3.6.

In this step, the resulted list from the above step will be run with the ColorMatching object to create a final list of the same object as in step 4, with the score updated to include the color factor. It should be emphasized that there is only one table for both genders as we did not see any extra benefit to separate color scoring scheme based on gender. Nevertheless, our design can be easily expanded and modified to include different tables for male and female if there is a need to down the road.

Bottom	**Top**	**Point**
Beige	Black	2
Beige	Blue	4
Beige	Brown	22
Beige	Gray	24
Beige	Green	20
Beige	Multicolor_Pattern	6
Beige	Orange	10
Beige	Pink	8
Beige	Red	16
Beige	Violet	18

Beige	White	12
Beige	Yellow	14
Black	Beige	4
Black	Blue	6
Black	Brown	24
Black	Gray	20
Black	Green	18
Black	Multicolor_Pattern	8
Black	Orange	10
Black	Pink	12
Black	Red	22
Black	Violet	14
Black	White	2
Black	Yellow	16
Blue	Beige	6
Blue	Black	8
Blue	Brown	10
Blue	Gray	12
Blue	Orange	2
Blue	White	4
Brown	Beige	2
Brown	Black	16
Brown	Blue	10
Brown	Gray	12
Brown	Green	8
Brown	Multicolor_Pattern	8
Brown	Orange	10
Brown	Pink	12
Brown	Red	18
Brown	Violet	14
Brown	White	4
Brown	Yellow	6
Gray	Beige	24
Gray	Black	20
Gray	Blue	10
Gray	Brown	18
Gray	Green	12
Gray	Multicolor_Pattern	8

Gray	Orange	2
Gray	Pink	6
Gray	Red	4
Gray	Violet	14
Gray	White	22
Gray	Yellow	16
Green	Beige	14
Green	Black	12
Green	Brown	10
Green	Gray	8
Green	White	6
Green	Yellow	4
Green	Blue	2
Multicolor_Pattern	Beige	2
Multicolor_Pattern	Black	4
Multicolor_Pattern	Brown	10
Multicolor_Pattern	Gray	8
Multicolor_Pattern	White	6
Orange	Beige	10
Orange	Black	4
Orange	Blue	2
Orange	Brown	6
Orange	Gray	12
Orange	White	8
Pink	Beige	6
Pink	Black	2
Pink	Brown	4
Pink	Gray	10
Pink	White	8
Red	Beige	6
Red	Black	10
Red	Brown	8
Red	Gray	4
Red	White	2
Violet	Beige	8
Violet	Black	10
Violet	Brown	12
Violet	Gray	14

Violet	White	4
Violet	Yellow	2
Violet	Pink	6
White	Beige	10
White	Black	2
White	Blue	12
White	Brown	8
White	Gray	24
White	Green	20
White	Multicolor_Pattern	4
White	Orange	18
White	Pink	16
White	Red	6
White	Violet	14
White	Yellow	22
Yellow	Beige	8
Yellow	Black	4
Yellow	Brown	6
Yellow	Gray	10
Yellow	Green	12
Yellow	Violet	2
Yellow	White	14

Table 3.6: Color Matching score table for male and female.

3.1.3.2.6 Pre-load tables in the app

All of the above score look-up tables were created in Excel. To transform these tables into datasets in the app database, we first exported the Excel tables in CSV format and saved them in res/raw folder of the Android project. Several methods were implemented to read these tables and create SQLite tables to be preloaded with the app at built time.

The goal of doing this is future change to the score tables for certain combinations will be integrated into our code without any code modifications. The steps include exporting CSV files, updating the files in the res/raw folder, and rebuilding the project.

3.1.4 Robotium

[Rob] Robotium is a powerful Android test automation tool for both emulator and real devices. It was applied to run several ClosetStylist's unit test cases that do not span over two applications due to limitation of Robotium. The test cases include registering, logging in, verifying that the main screen displays correct information, checking "My Closet", picking "Outfit of the Day", managing "Laundry Bag", and traversing "Outfit History".

This tool has been very helpful to catch bugs and unexpected behavior every time our code was modified, or new features were added.

3.2 ARCHITECTURE

ClosetStylist design is composed of three main layers: presentation layer, application layer, and data layer. The layer design is mainly for code reusability and portability. In addition, multiple design patterns have been applied to provide flexibility to switch between different services.

Figure 3.2 below shows the top-level architectural design of ClosetStylist.

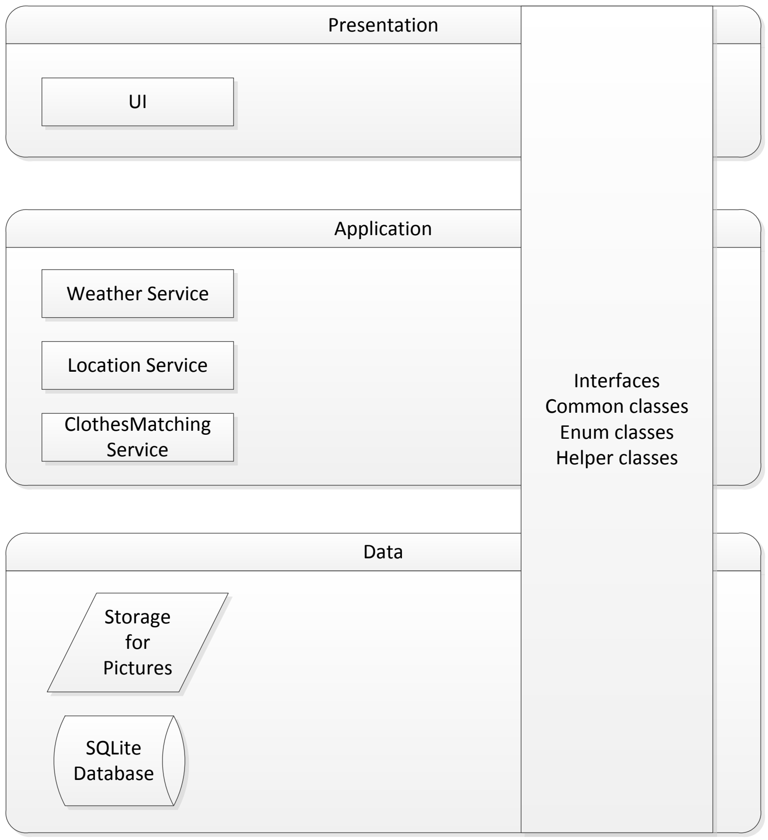


Figure 3.2: ClosetStylist top-level architecture.

3.2.1 Core

The core code includes common classes, common interfaces, helper classes, enum classes that are used to glue different layers together. The purpose of this layer is to connect different layers through common interfaces and features so that upper layers can work seamlessly with the information of the lower layers.

3.2.2 Presentation layer

This layer contains UI and UX modules of the app. The purpose of this layer is to implement the behavioral logic and to provide good user experience with the app's flow as well as look and feel of the app. The design of this layer strictly follows that of

Android design. We first drafted our own design, and then we consulted with professionals UI/UX designer and UI developer to implement the prototype.

3.2.3 Application layer

The application layer consists of all the services employed in this app, including weather service, location service, and clothes matching service. The purpose of this layer is to implement application logic and provide all the features of the app including organizing user's closet, programmatically suggest outfits, keep track of outfit history, and managing laundry bag.

3.2.4 Data layer

This layer includes two main components: the storage to store big-size pictures and the database to store smaller information about the user's profile, clothing items, outfit history, and look-up tables for matching service. The purpose of this layer is to provide data management for the app. SD card was chosen over Cloud storage as picture storage because we wanted to keep the picture-retrieving latency low which would help the responsiveness of the app. For small information, we used Android built-in database SQLite to manage. It should be noted that for each item, only the path to the SD card is saved in the SQLite of the item table and this is how we can link the information in the SQLite database and the pictures in SD card.

An overview of the data tables created for the app is shown in Figure 3.3.

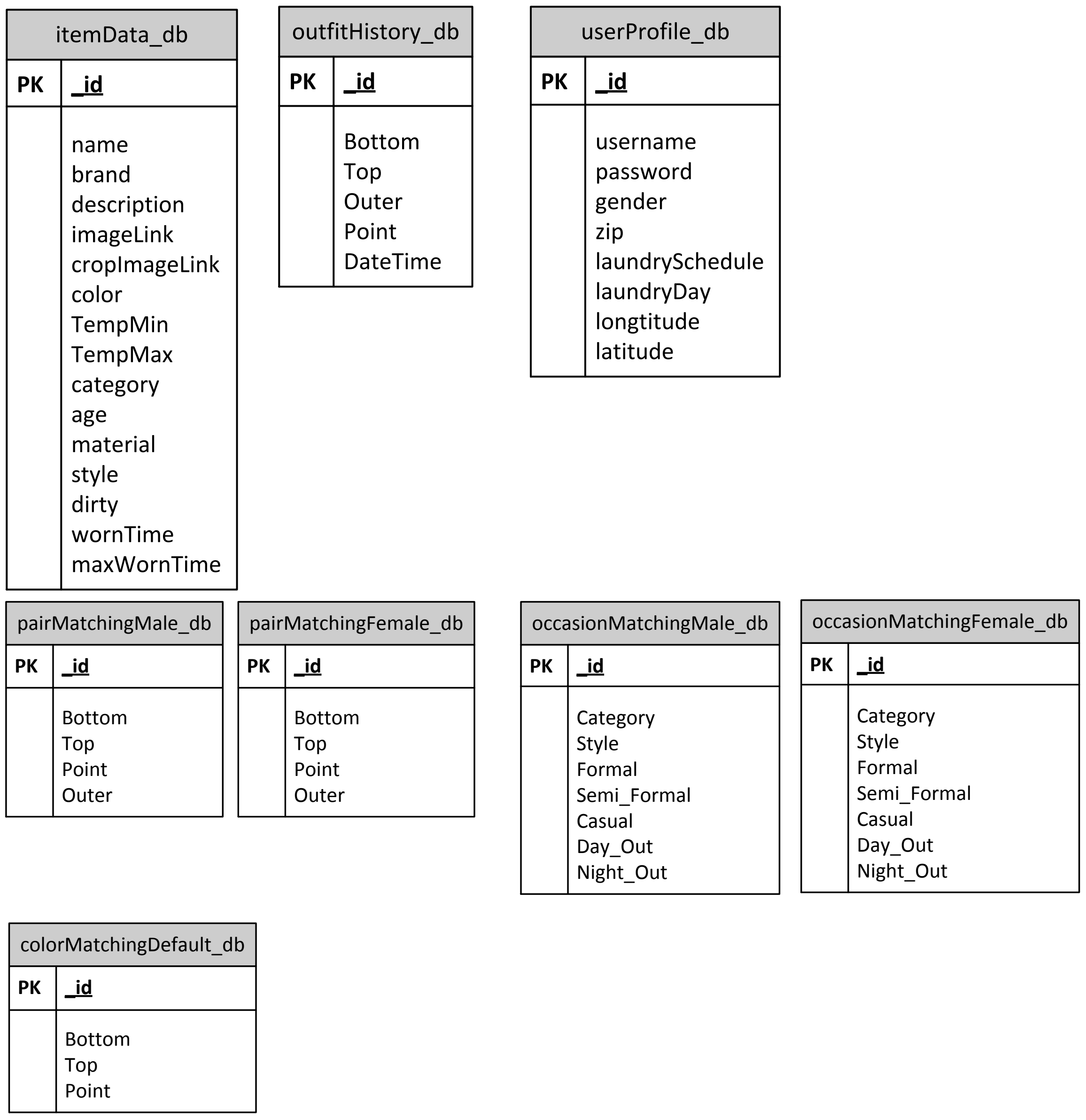


Figure 3.3: Database tables.

3.2.5 Design patterns

[HFDP] shows many design patterns to provide flexibility for future expansion while keeping closed for code modification. In this section, we describe how some of these are employed in our design for this app.

3.2.5.1 Factory Method Pattern

This pattern is applied to create different concrete storage types. SD card is chosen at the moment is the main storage for pictures, but the design is opened to replace SD card with another type of storage such as Google App Engine with much of the code intact because the creator class is written without knowledge of the actual products that will be created. In other words, the implementation of the product is decoupled with its use. In addition, new storage it will not affect Creator class. Figure 3.9 follows the Factory Method Pattern defined in [HFDP] to illustrate the deployment in our app.

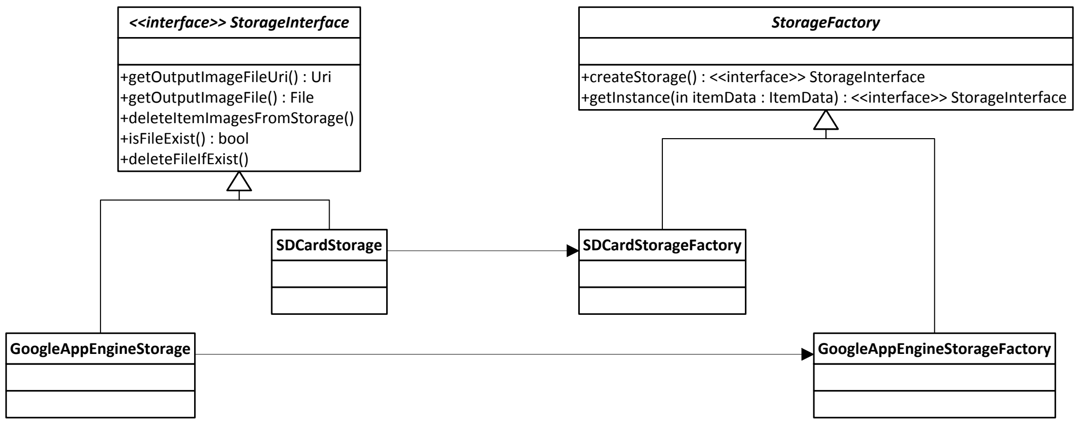

<<interface>> StorageInterface
+getOutputImageFileUri() : Uri
+getOutputImageFile() : File
+deleteItemImagesFromStorage()
+isFileExist() : bool
+deleteFileIfExist()
StorageFactory
+createStorage() : <<interface>> StorageInterface
+getInstance(in itemData : ItemData) : <<interface>> StorageInterface
SDCardStorage
SDCardStorageFactory
GoogleAppEngineStorage
GoogleAppEngineStorageFactory

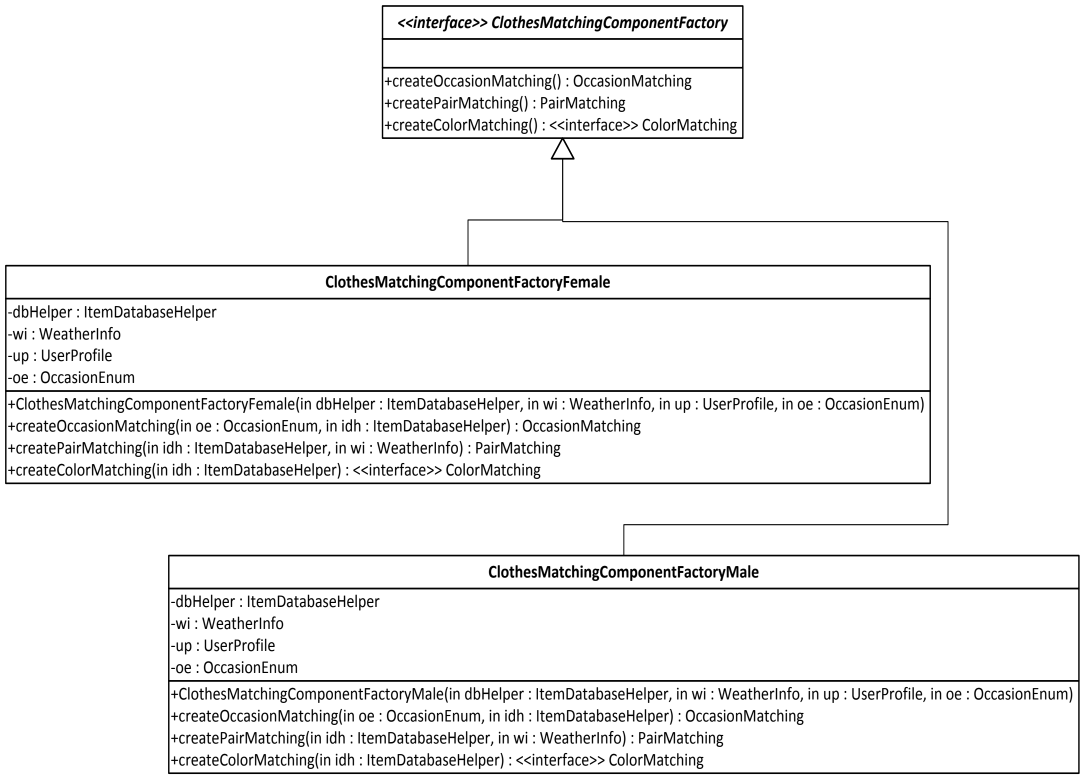
Figure 3.4: Factory Method Pattern for storage.

In Figure 3.4, the abstract Creator class is StorageFactory and the concrete Creator classes are SDCardStorageFactory and GoogleAppEngineStorageFactory. The

Product is StorageInterface; the concrete Product classes implementing this interface are SDCardStorage and GoogleAppEngineStorage; the Factory Method is createStorage.

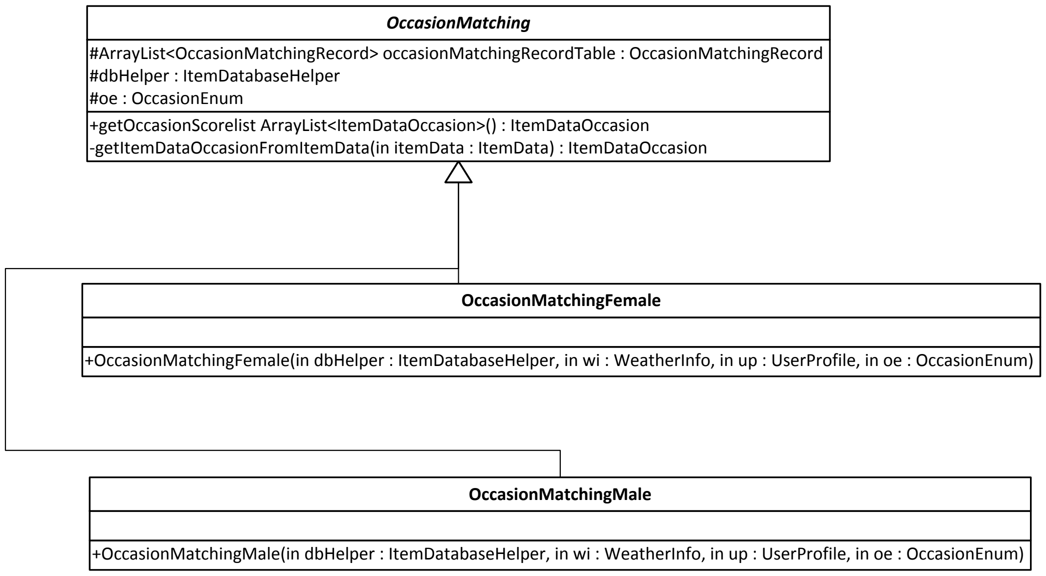
3.2.5.2 Abstract Factory Pattern

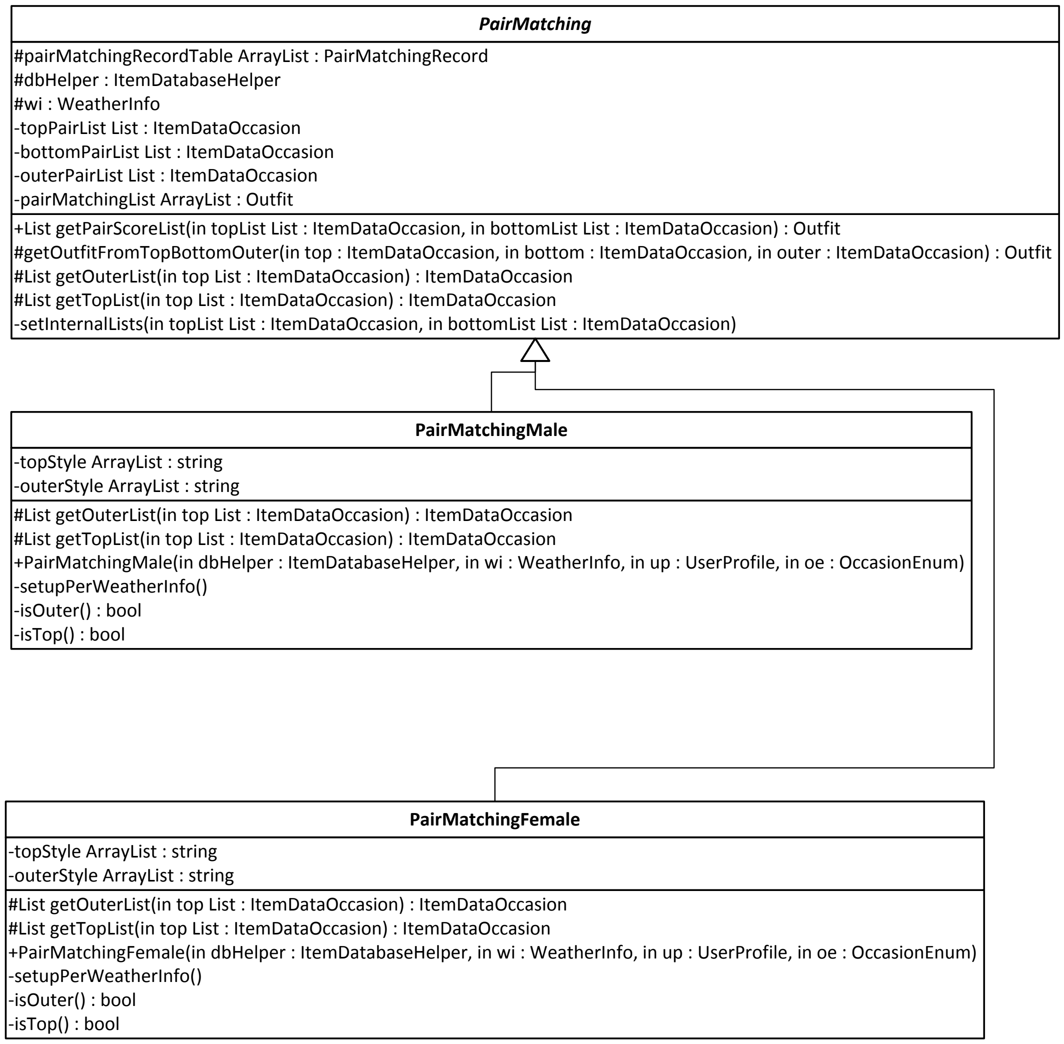
Clothes Matching service is consisted of five steps. While step one and two can be filtered by querying the database, steps three to five require more complicated implementation based on gender (limited to male and female for now). Abstract Factory pattern was applied to provide an interface to create a family of matching steps: occasion matching, pair matching, color matching. Writing code using this interface helped us decouple our code from actual factory that created these concrete matching steps (i.e., object classes). This also led us expand to a variety of genders if we need to in the future. Once the users register their genders, we can subsequently assign the correct matching steps. Figure 3.5 and Figure 3.6 follows the Abstract Factory Pattern defined in [HFDP] to illustrate the deployment in our app.



<<interface>> ClothesMatchingComponentFactory
+createOccasionMatching() : OccasionMatching
+createPairMatching() : PairMatching
+createColorMatching() : <<interface>> ColorMatching
ClothesMatchingComponentFactoryFemale
-dbHelper : ItemDatabaseHelper
-wi : WeatherInfo
-up : UserProfile
-oe : OccasionEnum
+ClothesMatchingComponentFactoryFemale(in dbHelper : ItemDatabaseHelper, in wi : WeatherInfo, in up : UserProfile, in oe : OccasionEnum)
+createOccasionMatching(in oe : OccasionEnum, in idh : ItemDatabaseHelper) : OccasionMatching
+createPairMatching(in idh : ItemDatabaseHelper, in wi : WeatherInfo) : PairMatching
+createColorMatching(in idh : ItemDatabaseHelper) : <<interface>> ColorMatching
ClothesMatchingComponentFactoryMale
-dbHelper : ItemDatabaseHelper
-wi : WeatherInfo
-up : UserProfile
-oe : OccasionEnum
+ClothesMatchingComponentFactoryMale(in dbHelper : ItemDatabaseHelper, in wi : WeatherInfo, in up : UserProfile, in oe : OccasionEnum)
+createOccasionMatching(in oe : OccasionEnum, in idh : ItemDatabaseHelper) : OccasionMatching
+createPairMatching(in idh : ItemDatabaseHelper, in wi : WeatherInfo) : PairMatching
+createColorMatching(in idh : ItemDatabaseHelper) : <<interface>> ColorMatching

Figure 3.5: AbstractFactory classes and ConcreteFactory classes of Abstract Factory Pattern applied in Clothes Matching service.


OccasionMatching
#ArrayList<OccasionMatchingRecord> occasionMatchingRecordTable : OccasionMatchingRecord
#dbHelper : ItemDatabaseHelper
#oe : OccasionEnum
+getOccasionScorelist ArrayList<ItemDataOccasion>() : ItemDataOccasion
-getItemDataOccasionFromItemData(in itemData : ItemData) : ItemDataOccasion
OccasionMatchingFemale
+OccasionMatchingFemale(in dbHelper : ItemDatabaseHelper, in wi : WeatherInfo, in up : UserProfile, in oe : OccasionEnum)
OccasionMatchingMale
+OccasionMatchingMale(in dbHelper : ItemDatabaseHelper, in wi : WeatherInfo, in up : UserProfile, in oe : OccasionEnum)


PairMatching
#pairMatchingRecordTable ArrayList : PairMatchingRecord
#dbHelper : ItemDatabaseHelper
#wi : WeatherInfo
-topPairList List : ItemDataOccasion
-bottomPairList List : ItemDataOccasion
-outerPairList List : ItemDataOccasion
-pairMatchingList ArrayList : Outfit
+List getPairScoreList(in topList List : ItemDataOccasion, in bottomList List : ItemDataOccasion) : Outfit
#getOutfitFromTopBottomOuter(in top : ItemDataOccasion, in bottom : ItemDataOccasion, in outer : ItemDataOccasion) : Outfit
#List getOuterList(in top List : ItemDataOccasion) : ItemDataOccasion
#List getTopList(in top List : ItemDataOccasion) : ItemDataOccasion
-setInternalLists(in topList List : ItemDataOccasion, in bottomList List : ItemDataOccasion)
PairMatchingMale
-topStyle ArrayList : string
-outerStyle ArrayList : string
#List getOuterList(in top List : ItemDataOccasion) : ItemDataOccasion
#List getTopList(in top List : ItemDataOccasion) : ItemDataOccasion
+PairMatchingMale(in dbHelper : ItemDatabaseHelper, in wi : WeatherInfo, in up : UserProfile, in oe : OccasionEnum)
-setupPerWeatherInfo()
-isOuter() : bool
-isTop() : bool
PairMatchingFemale
-topStyle ArrayList : string
-outerStyle ArrayList : string
#List getOuterList(in top List : ItemDataOccasion) : ItemDataOccasion
#List getTopList(in top List : ItemDataOccasion) : ItemDataOccasion
+PairMatchingFemale(in dbHelper : ItemDatabaseHelper, in wi : WeatherInfo, in up : UserProfile, in oe : OccasionEnum)
-setupPerWeatherInfo()
-isOuter() : bool
-isTop() : bool

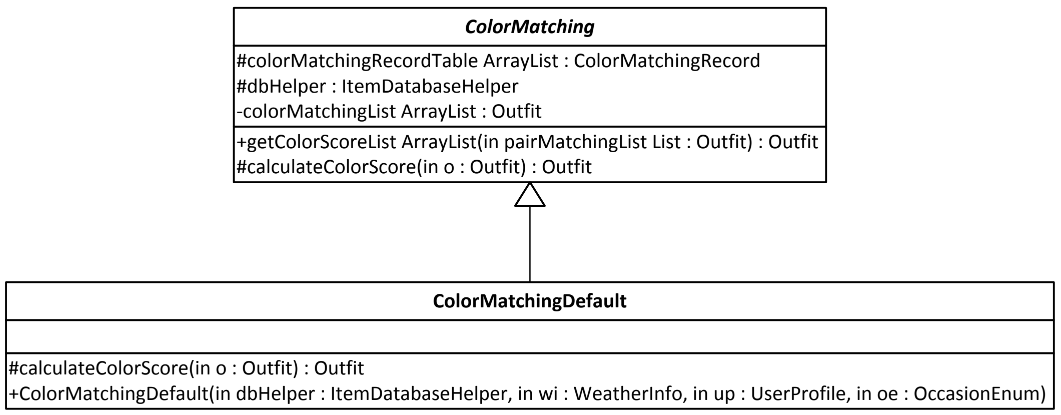

ColorMatching
#colorMatchingRecordTable ArrayList : ColorMatchingRecord
#dbHelper : ItemDatabaseHelper
-colorMatchingList ArrayList : Outfit
+getColorScoreList ArrayList(in pairMatchingList List : Outfit) : Outfit
#calculateColorScore(in o : Outfit) : Outfit
ColorMatchingDefault
#calculateColorScore(in o : Outfit) : Outfit
+ColorMatchingDefault(in dbHelper : ItemDatabaseHelper, in wi : WeatherInfo, in up : UserProfile, in oe : OccasionEnum)

Figure 3.6: AbstractProduct classes and ConcreteProduct classes of the Abstract Factory Pattern applied in Clothes Matching service (OccasionMatching, PairMatching, and ColorMatching classes).

In our design, the AbstractFactory interface is the ClothesMatchingComponentFactory, and the ConcreteFactory classes are ClothesMatchingComponentFactoryMale, and ClothesMatchingComponentFactoryFemale. There are several AbstractProduct classes OccasionMatching, PairMatching, ColorMatching and the corresponding concrete Product classes are OccasionMatchingMale, OccasionMatchingFemale, PairMatchingMale, PairMatchingFemale, ColorMatchingDefault.

3.2.5.2 Template Pattern

This pattern is applied to the ClothesMatching class to encapsulate the five-step algorithm described above. ClothesMatchingMale and ClothesMatchingFemale are the two subclasses of ClothesMatching and we can modify the implementation steps if we need to tailor our need for each gender. This provides a framework to plug in new gender in the future. Besides, the algorithm lives in one place (ClothesMatching class), so it is easy for code change later on. ClothesMatching focuses on the algorithm and let subclasses such as ClothesMatchingMale and ClothesMatchingFemale redefine certain steps of that algorithm without changing the algorithm's five-step structure.

3.2.5.2 Other Design Patterns

Other design patterns applied in this app include strategy pattern, command pattern, etc. Some of these are inherited from Android architecture (e.g., AsyncTask for Command Pattern), while some are used to implement features in this app.

3.3 CLASS DIAGRAMS

Figure 3.6 displays a simplified class diagram which consists of main classes such as item information, user profile, place record (or location), weather information, 4 main UI fragments, and how they are linked together. It is easy to recognize most of the UI fragments classes were designed to decouple as much as possible from the core classes for expansibility and flexibility. Throughout this project, this goal is always treated as the highest priority in our design.

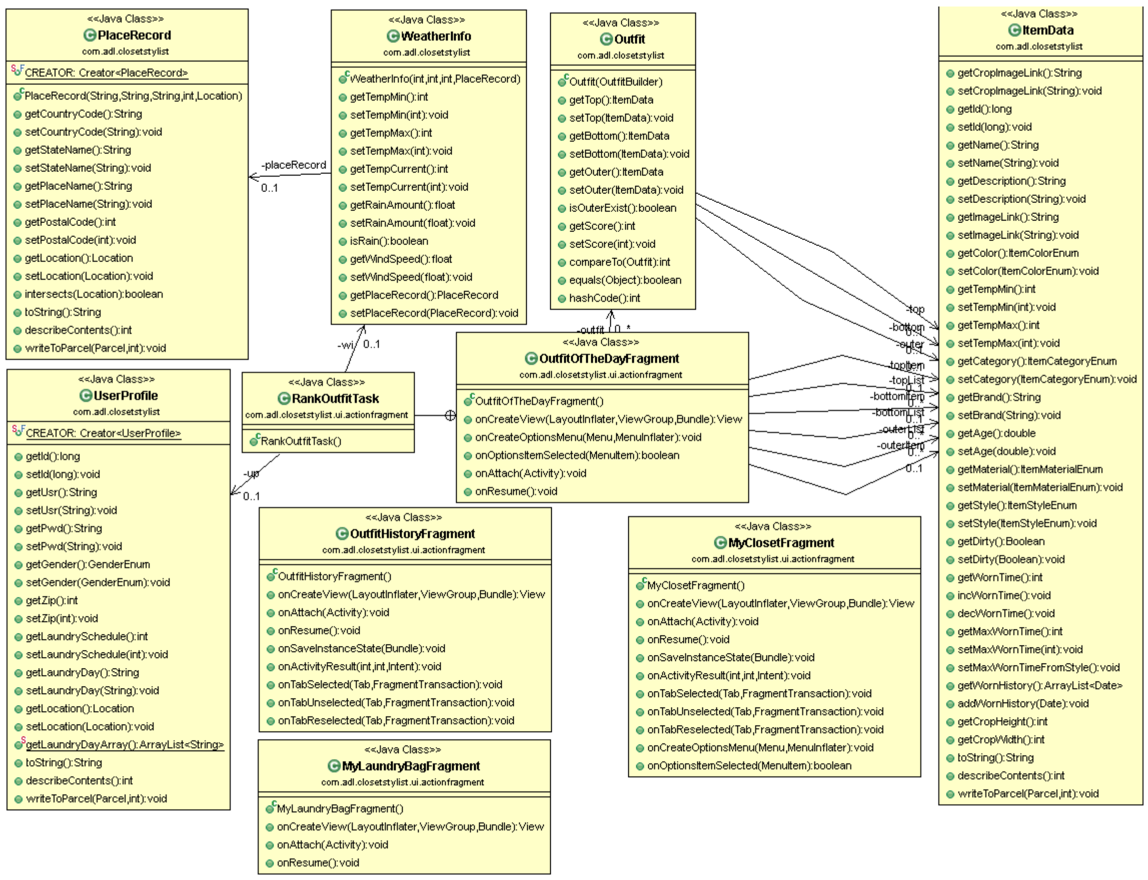


Figure 3.6: ClosetStylist class diagram.

Chapter 4 Results

Closetstylist utilizes both off-the-shell technologies and our own proprietary service; therefore, it is critical that each one fulfills its part and works together smoothly to provide satisfying user experience. Besides, it is also important for the app to deliver accurate weather information, to give reasonable outfit suggestions, and to be responsive to users.

4.1 OUTFIT OF THE DAY RESULT

The five-step algorithm is considered the most important feature of the app. Therefore, many different trials were executed to alter the knobs of the five-step clothes matching service. We tried making the temperature filter dependent on both style and material, but the material turned out to be not an obvious indicator (as explained above, because we could layer up the clothing pieces). Upon realizing that the list of suggestions did not change much among Occasion options, we tried increasing the scale of Occasion matching. This change proved to be helpful as it made the algorithm produce the results as expected.

Although the app works fine in many scenarios, we are aware of certain limitations to the algorithm:

- If the closet does not have many items in various styles, the service will recommend very similar outfits.
- If there are many items in the same style, the algorithm tends to provide the same suggestion lists for different occasions.
- At the moment, the suggestion list is not much difference between these 2 pairs of occasions: Formal and Semi_Formal, Day_Out and Night_Out.

- When the weather is cold, the outer item is not changed drastically when traversing through the suggested outfit list.

4.2 DISPLAY PICTURE

Loading images of garments is critical for most of the features of this app. To display the pictures taken by the cell phones bears a lot of unanticipated problems. Android devices have a limitation of 16MB memory for an application due to the constrained system resource of handheld devices. Rich images taken by cell phones usually have the size in Mega Bytes and can easily exhaust per-app limit on some devices. When the bitmap object is loaded, it consumes the entire available memory budget, the app usually crashes with the following message "java.lang.OutofMemoryError: bitmap size exceeds VM budget." To avoid these types of exceptions, images must be processed before loaded in the app.

[DAN] provides guidelines and sample code to process images off the UI thread, and then load them efficiently to the app. We applied the guidelines to sample images using AsyncTask on a different thread from the UI thread. Once the images were resized, we displayed the newly processed images on the screen. This allowed us to display list of images in My Closet screen or multiple images in Outfit of the Day screen smoothly as the user scrolled up and down.

4.3 WEATHER SERVICE AND LOCATION SERVICE

The app uses the location service from geonames [Geo] to find the city and country based on current location, and then obtains the weather information from Open Weather Map service.

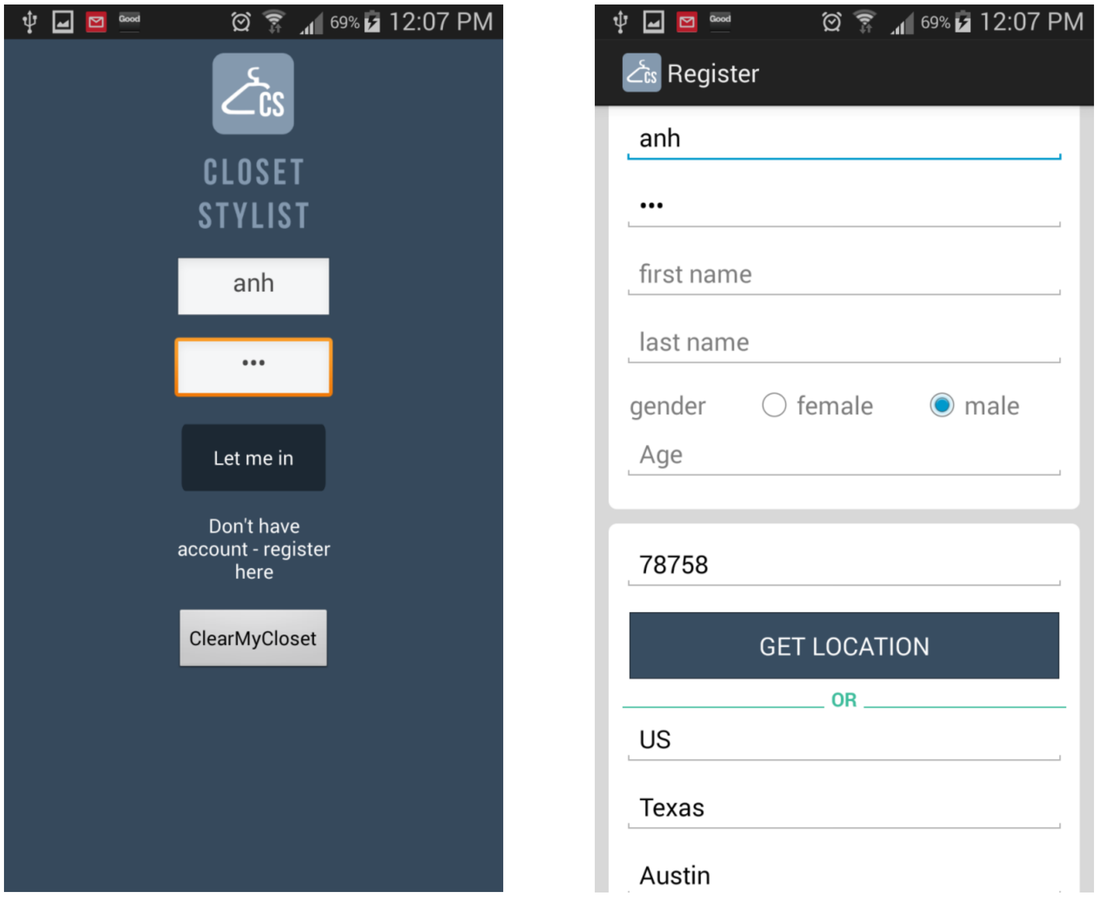
The result for location service provides the correct city and country, but the zip code is not quite exact. However, this result is acceptable for our app because we do not need exact location as other tracking app with the assumption that the weather within a city does not change significantly.

Regarding the weather, we compare with the weather.com information and it is within the -5 to +5 Fahrenheit range. This is acceptable because the granularity in our algorithm is bigger than this.

4.4 SCREENSHOTS

In this section, some screenshots whose mockups were presented earlier are shown to compare between the original design and the result. There are many factors led to modifications from the originals, for example, change in design, imperfect pictures of clothes, etc. All the change will be explained in the followings:

4.4.1 Login and Registration



anh
•••

anh
•••
first name
last name
gender ○ female [✓ male]
Age
78758
US
Texas
Austin

Figure 4.1: User login and registration screenshots.

Figure 4.1 shows the screenshots of mockups in Figure 2.9. In the user login screenshot, the Facebook login was removed because it was decided to implement our own login and Facebook login is treated as an option to enable certain social features. The other method we could have chosen was to utilize Facebook login to authorize people using our app. The decision was made to give users a freedom to opt out social features if they want to and also to reduce some features in this first ClosetStylist prototype. In the register mockups, there is a user's profile picture, which was intended to be used in the Outfit of the Day screen to give user a hearty feeling when they try different outfits. However, that screen already looked a little busy with too many items, and hence the profile picture was removed in the registration step.

4.4.2 Main Screen and Side Menu

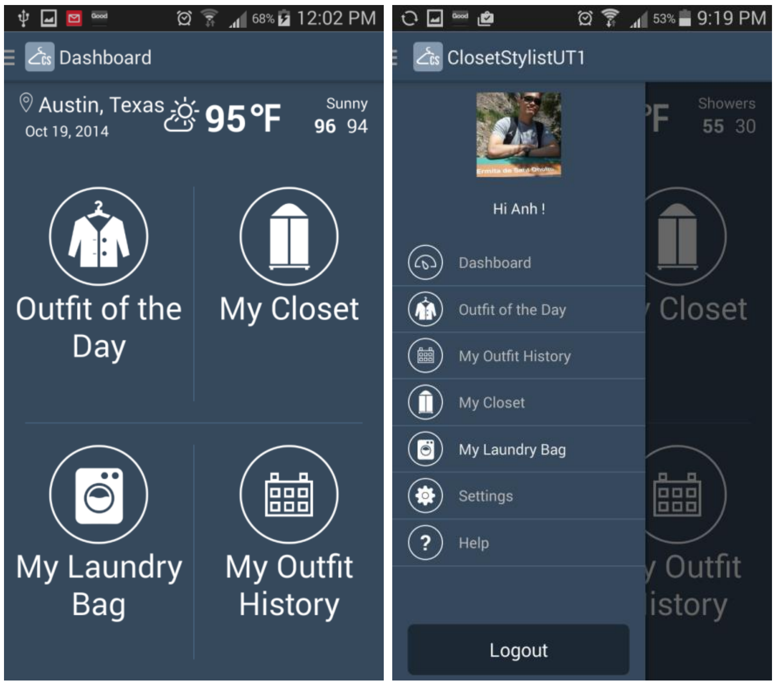


Figure 4.2: Main Screen and Side Menu screenshots.

The screenshots in Figure 4.2 look alike their mockups in Figure 2.10 and there was no change from the original design.

4.4.3 My Closet and Add Item

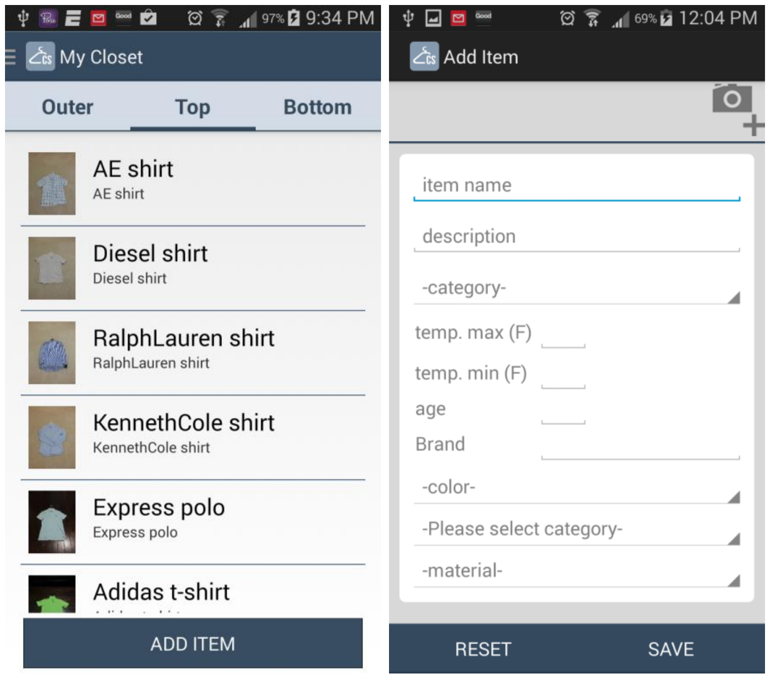


Figure 4.3: My Closet and Add Item screenshots.

Compared to their mockups in Figure 2.11, there was some cosmetic change to the screenshots in Figure 4.3. In My Closet screen, the tabs' titles were changed to more generic terms such as "Jacket" to "Outer", "T-Shirt" to "Top", while "Shoes" was omitted as it would be too complex for the first prototype. The fields in Add Item were re-arranged to fit longer category, style, and material fields.

4.4.4 Outfit of the Day and Laundry bag

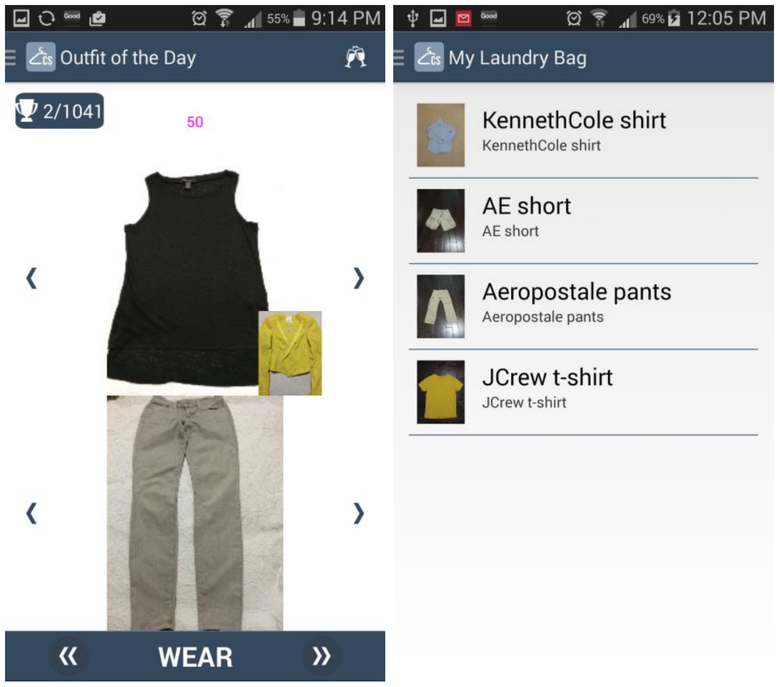


Figure 4.4: Outfit of the Day and Laundry bag screenshots.

There were some changes from the screenshots in Figure 4.4 compared to their counterpart in Figure 2.12. It can be easily noticed that hat, shoes, and user’s profile picture was omitted from the original design due to space limited on the screen. It was very complex to scale all items as in the original design because each image could be taken at different angles and different zoom levels. To simplify our app, some items were omitted. There was not much change in My Laundry bag.

4.4.5 Outfit History and Outfit Preview

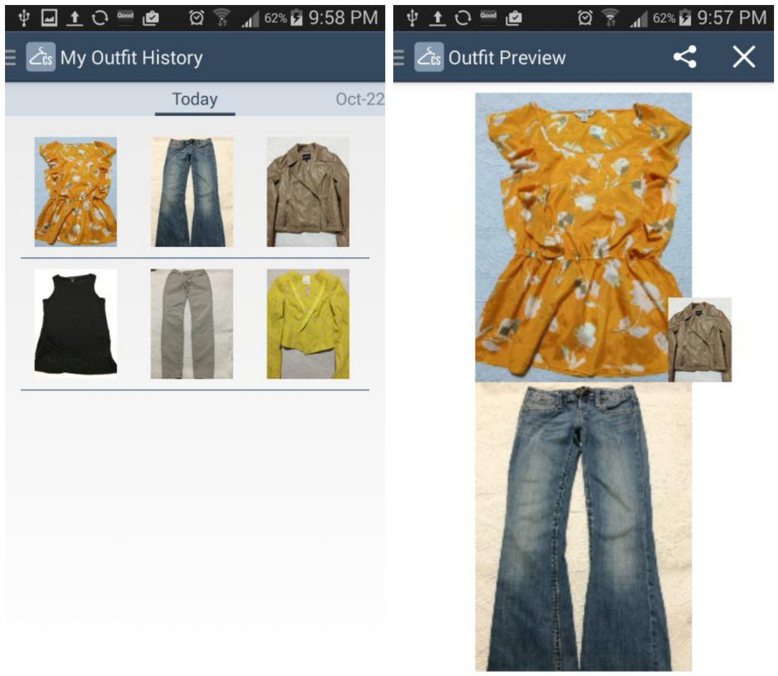


Figure 4.5: Outfit History and Outfit Preview screenshots.

Similar to the change in Figure 4.4, screenshots in Figures 4.5 had the hat, shoes, and user's profile picture deleted.

4.4 COSTS AND LEVEL OF EFFORT

During development, GitHub was used as the source control tool, and the project was left as public to be used for free. Table 4.1 shows the cost of equipment and services spent on the ClosetStylist app.

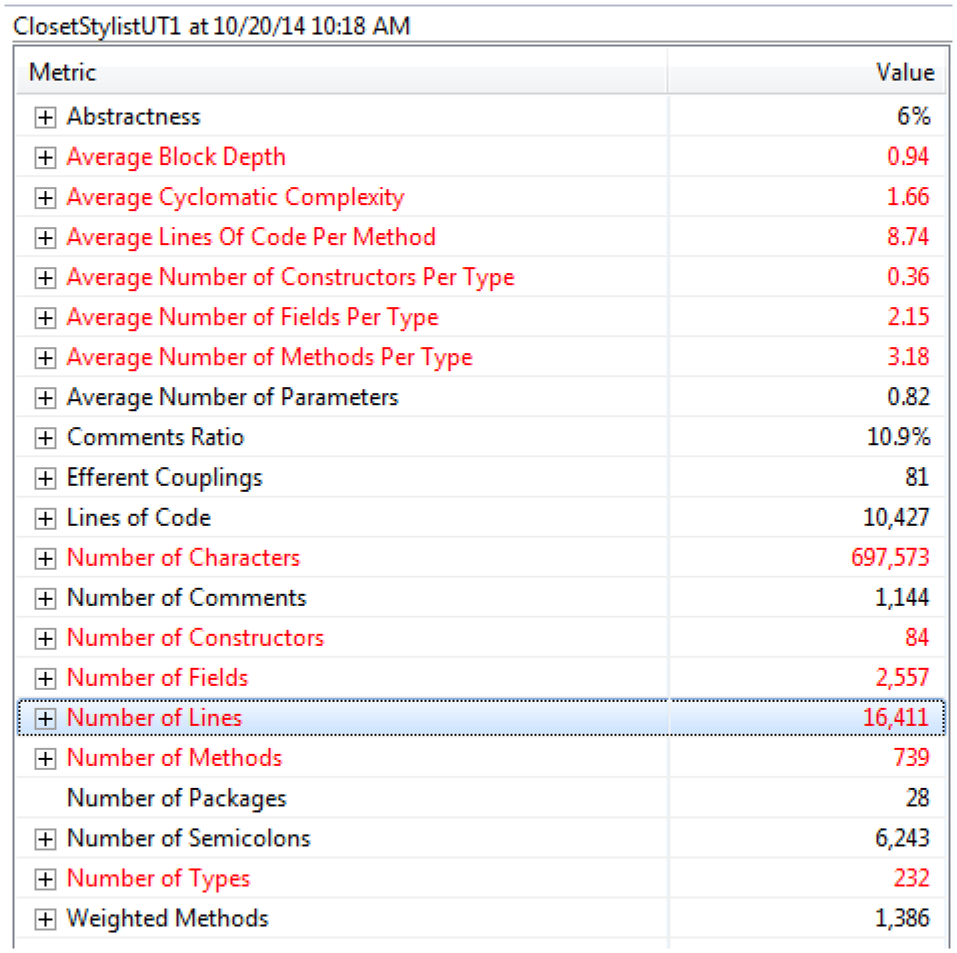
Item	Costs
Samsung S3	Free
UI/UX Design	$150
UI Development	$500

Table 4.1: ClosetStylist development costs.

The app was developed from March to August of 2014. The following section shows how many hours were spent on different aspects of the projects:

- 40 hours for architecture design.
- 30 hours for original UI design.
- 10 hours for final UI design.
- 20 for UI development collaboration.
- 260 hours for studying Android, researching, coding, and testing.
- 30 hours for writing the report.

Figure 4.5 shows the result after running CodePro AnalytiX tool [Cpro] against the multiple metrics. An entry in red means there is some code violation in some predefined criteria of metric, and there is room for improvement. For example, the cyclomatic complexity violation simply means the average number of branched keywords per method such as “if”, “while”, “for”, etc. is above a predefined threshold in the metric. There is some other basic information about code such as 10427 lines of code were written in 92 Java files and 39 XML files, the average number of lines per method were 8.74, etc.

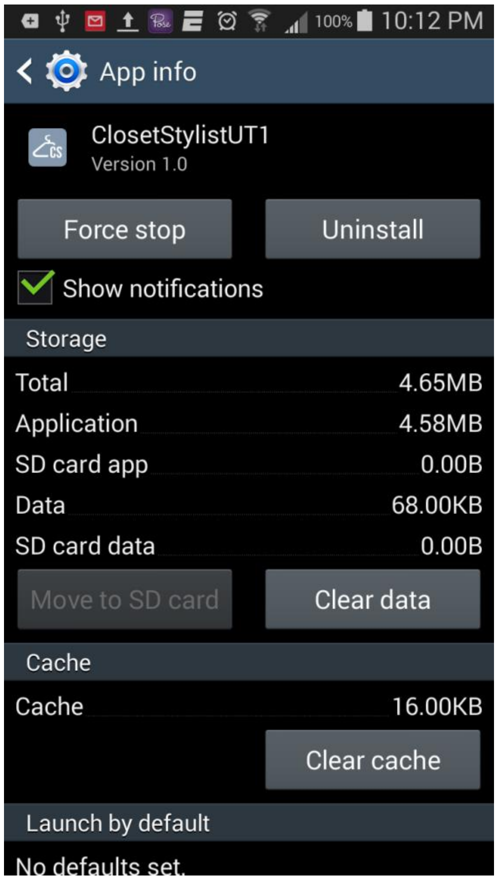


ClosetStylistUT1 at 10/20/14 10:18 AM

Metric	Value
Abstractness	6%
Average Block Depth	0.94
Average Cyclomatic Complexity	1.66
Average Lines Of Code Per Method	8.74
Average Number of Constructors Per Type	0.36
Average Number of Fields Per Type	2.15
Average Number of Methods Per Type	3.18
Average Number of Parameters	0.82
Comments Ratio	10.9%
Efferent Couplings	81
Lines of Code	10,427
Number of Characters	697,573
Number of Comments	1,144
Number of Constructors	84
Number of Fields	2,557
Number of Lines	16,411
Number of Methods	739
Number of Packages	28
Number of Semicolons	6,243
Number of Types	232
Weighted Methods	1,386

Figure 4.6: Metrics with CodePro AnalytiX.

Figure 4.7 shows the foot print of ClosetStylist from the Android Application Manager, it occupies the total of 4.65 MB memory, of which 4.58 MB is for application code and 68 KB is for data.


100% 10:12 PM

App info

ClosetStylistUT1
Version 1.0

✓ Show notifications

Storage

Total	4.65MB
Application	4.58MB
SD card app	0.00B
Data	68.00KB
SD card data	0.00B

Cache

Cache	16.00KB

Launch by default

No defaults set.

Figure 4.7: Foot print of Closet Stylist.

4.4 LESSONS LEARNED

Android is a very powerful framework and it takes a lot of time and effort to master and use it efficiently. Starting to learn Android from the beginning of 2014, I had encountered quite a few challenges while implementing the app. Nevertheless, as Android is being used in hundreds of millions of devices, the eco system is huge and it was often easy to find the solution for problems I was facing as someone else had dealt with similar problems. Stackoverflow and Android developer websites are my companions throughout the project.

Git is another new thing I learned in this project. Although I was familiar with SVN, another source control tool, I decided to learn Git and used it as the source control for this project. It did take me a lot of time to get used to running Git from command line because I often worked with TortoiseSVN – a UI tool on Windows system.

Below are the most highlighted things to do and not to do that I collected after finishing the first prototype of ClosetStylist:

Do:

- Design UI/UX carefully to avoid missing any features, especially missing features near the release. Hiring a professional designer to assist you is a great idea because there are many subtle front-end elements that back-end developers may consider trivial but could turn out to be quite significant to users.
- Use Robotium to leverage test efforts. Although this tool has certain limitations, it is still a very powerful tool that can save you a lot of time and effort.
- Put more effort into processing images taken from built-in camera or imported from a gallery. Taking pictures of clothes is not as easy as it may seem and it takes a lot of time even with a handy camera phone. A big challenge was to take appealing pictures of your clothes as displayed on branded clothing websites. Even taking brand new items with tags at home did not help as there was no good place to pose them, for example, hanging the clothes will make the sleeve look saggy because gravity pulls them down. We ended laying them on the floor and the couch to take picture, but then we need to be careful not to include the shadow in the picture. If we want to release this app, we must figure out a better way to tackle this issue.

Don't:

- Couple UI with the core. To some degree, Android architecture has provided a tight couple between UI and core. Attention needs to be paid to avoid this coupling as it will be catastrophic if any change in UI (which happens quite often) requires a change in backend code or vice versa.
- Wait until the last minute to integrate social media, especially Facebook. For a simple post, it is straightforward with the provided sample code. For customized post including pictures, mastermind the sample code and the APIs is essential. Another problem is their APIs change more often than their guidance and many samples are obsolete due to deprecated APIs. Also, beware that their APIs may not be compatible with the latest development Android version.

Chapter 5 Conclusion

5.1 SUMMARY

After six months of development, the first prototype of ClosetStylist app was completed and met all of the original goals. The architecture was designed with certain flexibility and extensibility for future work. Some off-the-shell technologies such as location service and weather service were integrated to work coherently with proprietary clothes matching service to deliver the following key features of the app: picking outfit programmatically based on weather and occasion options, viewing outfit history, organizing closet, and managing laundry.

With all that said, the app still leaves a lot to be desired. The image processing method needs to be enhanced to provide user with more edit capability than just cropping. Social media needs to be integrated more aggressively than just simply a login to Facebook such as letting friends vote on the outfits and sharing them. Items such as hat, bags, and shoes are part of a complete outfit and should be part of the recommendation.

In conclusion, the prototype of ClosetStylitst app has integrated well with multiple technologies to implement features that can help people get the most out of what they already had in their closets. It has laid out a good foundation for future work towards releasing an official Android app.

5.2 RELATED WORK

There are many fashion apps available but not many of them offer all of the features of ClosetStylist. [5FADYC] lists five apps that can help to digitize our closets, and while all of them support iOS, only two of them support Android. The following section will review the most interesting ones of them and compare with supported features in ClosetStylist.

5.2.1 Stylebook and Stylebook Men

This is an iOS app and it has has the most similar functionalities with our ClosetStylist app. Their goal is to curate customers wardrobe's and choose new pieces that fit into their current closet.

The app provides many neat functionalities: match other pieces in the closet with a specific top, make quick outfit collage, search for the right items to buy by using shopping features, mix match different pieces in the outfit editor, suggest which items in closet should be replaced.

Although our ClosetStylist app offers some similar features, we differentiate from this app by programmatically suggesting outfits based on the current weather.

5.2.2 Pose

Pose a tool to keep track of daily outfits and it is available in both iOS and Android. It is deeply integrated into social networks and resembles Instagram in many ways, for example, it let users share and discover inspiring looks from other users as well as your own garments' pictures.

Although this offers great experience in social network, it does not provide some fundamental features of ClosetStylist such as managing laundry bag or suggesting outfit to wear.

5.2.3 Netrobe

This app is only available in iOS. It offers quite appealing features includes managing clothes, and mixmatching outfits from the garments populated. The ClosetStylist features lacked in this app are laundry bag maintenance and outfit suggestion.

5.3 FUTURE WORK

This ClosetStylist prototype serves multiple purposes: prove a concept, learn how to program Android, learn how to manage a smart phone app development with a professional UI/UX designer and a UI developer. Although we were successfully developed a prototype, the features offered at the moment is still a very small subset of the full feature set that could attract high adoption. Some of them are discussed in the following sections.

5.3.1 Integration with social networks such as Facebook

We were able to login to Facebook, did not have enough bandwidth to implement sharing the outfit on Facebook because sharing images of the outfit is a complicated process. Given that this app requires displaying pictures, Twitter may not be a good social network to share. Instagram may be a better choice.

Without social media, it is very difficult to promote the app, and that is why this is the highest priority in our to-do list.

5.3.2 Detect the item’s color automatically

Manually entering color is not too much work, but it would be nice if we can detect the color of the item and fulfill it automatically. The challenge is with multi-color items. Another obstacle is how to distinguish between the item and the background. Although this is a nice feature to have, the effort would be massive unless we can find a library or tools out there that already support this.

5.3.3 Support more items

Currently, the app can handle regular “Tops” items, such as blouse, shirt, t-shirt, etc. but not dress. Other things users would like to put together when going out including hats, shoes, bags, belts are not supported. These bears a lot of work because not only displaying them will make the phone screen too crowded but also the algorithm to choose pick an outfit will be much more complex. Nevertheless, these are essential to make a fully functional app.

5.3.4 Add support for travelling

Adding support for travelers to pick the items for their trip is another functionality that we would like to add in the future. User will enter their destination or a list of destinations together with the begin and end date, the app will programmatically suggest the outfits they should pack to be most efficient for their trip based on the weather forecast at the destinations.

5.3.5 Create app for iOS

Although Android powers 70 percent of the mobile device, iOS is still a very big player in this area, especially in terms of money. Implement an iOS version is a required step to make this app popular and bring in revenue.

References

[Amb] "UML 2 Activity Diagramming Guidelines." UML 2 Activity Diagramming Guidelines. Ed. Scott Ambler. Ambysoft Inc. Web. 23 Oct. 2014. <http://www.agilemodeling.com/style/activityDiagram.htm>.

[Bal] "Balsamiq." . Rapid, Effective and Fun Wireframing Software. Balsamiq. Web. 23 Oct. 2014. <http://www.balsamiq.com>.

[Ecl] "Eclipse Downloads." Eclipse RSS. Eclipse Foundation, 1 Jan. 2004. Web. 23 Oct. 2014. <http://www.eclipse.org/downloads/>.

[SWA] Azzola, Francesco. "Surviving W/ Android." Android Weather App: JSON, HTTP and Openweathermap. 21 May 2013. Web. 23 Oct. 2014. <http://www.survivingwithandroid.com/2013/05/build-weather-app-json-http-android.html>.

[Inv] "Free Web & Mobile (iOS, Android) Prototyping and UI Mockup Tool | InVision." InVision. InVision. Web. 23 Oct. 2014. <http://www.invisionapp.com/>.

[Geo] "GeoNames." GeoNames. Web. 23 Oct. 2014. <http://www.geonames.org/>.

[Rob] Reda, Renas. "User Scenario Testing for Android." Robotium - The World's Leading Android™ Test Automation Framework. Google. Web. 23 Oct. 2014. <https://code.google.com/p/robotium>.

[TCW] Centeno, Antonio. "The Color Wheel – Color Coordination for Men." Real Men Real Style. RMRS. Web. 23 Oct. 2014. <http://www.realmenrealstyle.com/color-wheel-color-coordination-men/>.

[HFDP] Elisabeth Freeman , Eric Freeman , Bert Bates , Kathy Sierra, Head First Design Patterns, O' Reilly & Associates, Inc., 2004

[DAN] "Displaying Bitmaps Efficiently." Android Developers. Google. Web. 23 Oct. 2014. <http://developer.android.com/training/displaying-bitmaps/index.html>.

[Cpro] "Java Developer Tools." CodePro Analytix User Guide. Google. Web. 23 Oct. 2014. <https://developers.google.com/java-dev-tools/codepro/doc/>.

[5FADYC] "5 Fashion Apps to Digitize Your Closet." Mashable. Mashable, 1 Jan. 2013. Web. 23 Oct. 2014. <http://mashable.com/2012/07/13/closet-management-apps/>.