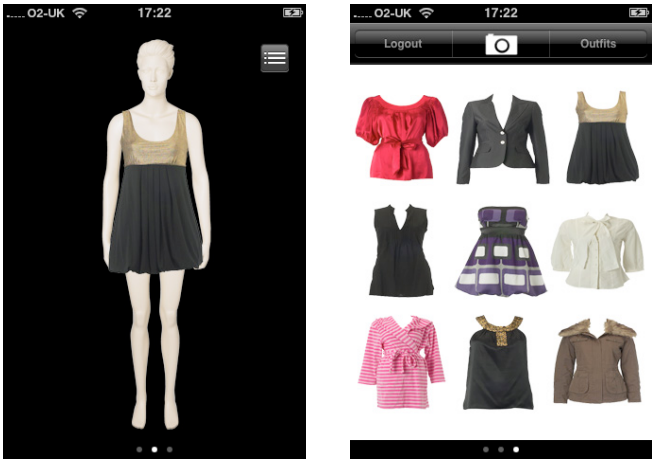
UNIVERSITY COLLEGE LONDON
DEPARTMENT OF COMPUTER SCIENCE

iPhone Application Development for FashionEye.com



Author: Khiet Le

Degree: BSc Computer Science

Supervisor: Prof. Philip Treleaven

Submission Date: 30th April, 2010

Disclaimer

Abstract

This dissertation presents the development of an iPhone application for FashionEye.com (FashionEye) which is an ongoing project lead by Prof. Philip Treleaven at Department of Computer Science at UCL. The project involves collaboration with the website team.

Rationale: Today, social networking websites have been phenomenally successful.
The well-known examples are Facebook and Twitter, both are amongst 15 most visited websites in the world, 2nd and 11th respectively (Alexa Internet, Inc. 2010). However, little has been developed to integrate the features with fashion and none has been a success. FashionEye and is aiming at the niche market and the project, in parallel with the development of the FashionEye website, adds an extra value to the FashionEye paradigm by providing a mobile companion.

Overview: FashionEye is an online social networking featuring fashion. The innovative features of FashionEye include *Mix & Match* of clothes on mannequin, *Closet* allowing users to store, share and upload clothes, and *Recommender System* for suggesting clothes based on users' preferences and tags on clothes. These features are combined with social networking features such as, adding friends, posting comments, and searching and sharing in the social networking space.

The challenges involved in this project is to transform the ideas of FashionEye into a mobile device. At minimum, the design must be customized to suit the space limitation of the device and the code must be optimized to address the resource limitation while offering iPhone-specific features such as, uploading a garment image taken with the built-in camera and zooming a garment image by *multi-touch*.

Therefore, a great deal of careful thoughts are put on usability in the project to ensure user interactions and visuals are presented correctly on a small-screen device.

The design of the application is followed by complex implementations. Since FashionEye is highly interactive in nature e.g. *Mix & Match* of clothes involve many events and state changes, it is a demanding programming task to ensure the interactivity of the application.

Furthermore, developing for the iPhone platform requires programming in Objective-C. The Objective-C is object-oriented extension to C and requires memory management which is notably difficult to constantly get it right (M. Handley 2010) and notoriously the most common cause of bugs (C. Clack 2010).

The integration with the FashionEye website is also an important consideration in the project. The product should be compatible with the existing database. Therefore, intensive collaboration is the nature of the project to ensure the iPhone project works seamlessly with the current project.

Report Structure: The report details the software development cycle undertaken to achieve the goal of providing a companion iPhone application to FashionEye.com. From understanding the problem's purposes and context, designing and implementing highlighting significant choices made, and testing to ensure the build quality lives up to an acceptable standard. The report concludes with the evaluation of the project and any recommendations for carrying it to the future.

Notes to the Reader

As a primer to the dissertation, I would like to draw the reader's attention to a number of points regarding the nature of the project and the amount of work required to deliver a product.

The project is not limited to the iPhone application development.

Throughout the project, a large amount of time was devoted to the development of FashionEye. These include improving the website's UI design, addressing problems and suggesting solutions to those problems, and ideas for incomplete functionalities. An example of tangible contributions is creating informative GIF animations for the front page of the website. The animations are included in the attached CD.

The program contains over 5,000 lines of code with 24 classes written in Objective-C.

The numbers reflect the complexities in resolving many interactions and state changes that are inherent in the application. The entire programing was done from scratch and completed to fully meet all the core functionality requirements except where functionalities depend on the implementation of the FashionEye website.

The emphasis on non-technical details.

Rather than simply meeting the functional requirements. Usability was treated as a critical factor to achieve high end-user satisfaction as described in "*ISO 9241-11: Guidance on Usability*":

"*Usability is the extent to which a product can be used by specified users to achieve specified goals with* ***effectiveness, efficiency and satisfaction*** *in a specified context of use.*".

As such, whenever appropriate efforts were made to take a non-standard way of implementing the iPhone User Interface (UI), which is usually hard to achieve due to less documentation and examples available.

The iPhone application development involves learning a new language Objective-C and iPhone SDK as part of strict terms and conditions imposed by Apple.

The "*iPhone Developer Program License Agreement*" by Apple Inc. (Apple) must be agreed with before using the Apple Software and related services, such as releasing an application on the App Store. The agreement restrict the use of languages to C, C++, Objective-C and and tools are limited to iPhone SDK. This restricts the number of solution domain, therefore demanded creativity in solving certain problems, and required careful attention when using open-source libraries.

The project involved collaboration with other developers and UCL students.

The project involved collaboration with the FashionEye website team and closely with another final year BSc Computer Science student, Ralfas Jegorovas. The meetings were held weekly for the collaboration to be effective. The declaration of collaborative work with Ralfas Jegorovas is included in Appendix A.

Contents

1

Introduction

This chapter presents the motivations for developing a companion iPhone application to FashionEye, summarizing the underlying objectives and scope of this dissertation and concluding with an overview of the report structure.

1.1 Motivation

Today's online services are both diverse and dynamic. Some areas of online services have been established to transform the way we used to do things, such as booking flight tickets. One area that is noticeably successful and increasingly popular is social-networking. Facebook is now the second most visited website in the world after Google (Alexa Internet, Inc. 2010) and had a massive 566% increase in the time spent on its website between Dec. 2007 and Dec. 2008 (Nielsen Online 2009). However, little has been developed to integrate the social networking with fashion and none has been so successful that it is part of what people do online. FashionEye is aiming to establish the niche market where the users can have their collections of fashion items in the virtual world and interact with their collections and, at the same time, express themselves in the social-networking space.

The iPhone companion gives huge advantages to FashionEye. One reason is simply because it is the third widely used smartphone worldwide (BBC 2009) with the fastest 100 percent unit growth between 2009 and 2010 (Apple Inc. 2010), making it one of the preferable platform to reach the wider audience. The second reason comes from the iPhone's capabilities in implementing and extending the FashionEye's paradigm. The iPhone's multi-touch[2] interface is ideal for high-level interactions that is suitable for *Mix & Match*, the built-in camera means a quick and easy way to allow users to take and upload garment photos, and of course the mobility and networking capability are the inherent strengths in the smartphone, which enable users to take their virtual closets anywhere they like and update information with minimum efforts.

---

[2]*Multi-touch is an enhancement to touchscreen technology, which provides the user with the ability to apply multiple finger gestures simultaneously onto the electronic visual display to send complex commands to the device.*

1.2 Initial Brief

The section discusses the initial brief in terms of the project's domain and the project's technical and non-technical objectives.

1.2.1 Domain

The project challenge is to realize the central ideas of FashionEye on a mobile device. The project domain comprises of a number of subject areas that are core to the development of the iPhone application. These include, web-technologies, mobile-technologies and the integration of both.

1.2.2 Initial Concepts

FashionEye's aimed strengths, as compared to other fashion related website are the abilities for users to have their own fashion space in the form of content creation and interaction. The core features include *Mix & Match* of clothes (garments and outfits), *Recommender System* of clothes and *Sharing* of their clothes. Since not all the features of FashionEye.com are ideal for a mobile device to be implemented, the iPhone application is designed and implemented to offer the user as much "look and feel" of FashionEye as possible with iPhone-specific extra features, such as, *Image Uploading*.

1.2.3 Technical Objectives

The technical objectives describe what the core features the iPhone application should deliver functionally, and cover those features in five topics: *Mix & Match*, *Closet*, *Searching*, *Sharing* and *Image Uploading*.

Mix & Match: To allow users to put clothes on and off mannequin from *Closet* and *Searching.*

Closet: To allow users to have a virtual closet where clothes are kept for *Mix & Match* and for merely keeping a collection of clothes.

Searching: To allow users to have access to all the clothes in the entire FashionEye space (database). The functionality serves to search users' own clothes as well as shared outfits by other users.

Sharing: To allow users to share their own outfits in the entire FashionEye space. There are three levels of sharing: Public (shared with all users), Friends (shared with friends) and Private. Only outfits tagged as Public and Friends will be found by *Searching.*

Image Uploading: To allow garment images to be taken by the built-in camera and uploaded.

1.2.4 Non-technical Objectives

The technical objectives describe the application's intangible requirements and cover the three central topics: *Usability*, *Extensibility* and *Quality of Execution*.

Usability: A great deal of attention must be put on usability to *"achieve specified goals with **effectiveness, efficiency and satisfaction** in a specified context of use.*" as described in "*ISO 9241-11: Guidance on Usability"*.

Extensibility: It is highly anticipated that the ongoing FashionEye project, upon which the iPhone project is based, is adding more ideas to the current model. Therefore, it is a natural objective to build an application with extensibility in mind. Also, on successful completion, the application is likely to be handed over to other developers in the future, the documentation and the comments in source code were considered as an important part of ensuring hight-level build quality.

Quality of Execution: The FashionEye is a real-world project which will be commercially deployed in the future. The iPhone application is not an exception and delivering a professional level application is essential. Especially when every iPhone application will go through the Apple's review process for approval to be put on their online App Store. The development strictly follows the Apple's "*iPhone Developer Program License Agreement*" and "*Apple Human Interface Guidelines*" for the application to be commercially deployable in the future.

1.3 Project Roles

The following table describes the project actors and roles:

Name	Role	Description
Prof. Phillip Treleaven	Supervisor and Client	Responsible for overseeing the project as a supervisor as well as engaging in the project as a client.
Ralfas Jegorovas	Collaborator	Final year individual project collaborator, participating in some parts of the project individually.
Boril Bogoev	Collaborators	Developer for the FashionEye website, joined until 2010.
Bogdan Batrinca Ma Ma Thet Mon Aye	Collaborators	Developers for the FashionEye website, joined since 2010.

Table 1.1: Project Roles

1.4 Report Structure

The report consists of eight chapters. Chapter one and two introduce the project's background information and the application's purposes and context. Chapter three describes the process of requirement gathering to develop a formal set of requirements upon which the later software development will be based. Chapter 4 provides the design phase where application's UI and the system schema were formed. Chapter 5 and 6 are implementation and testing that details the programming work undertaken to realize requirements developed. Chapter 7 and 8 provide critical evaluation and conclusion of the overall project.

2

Background

This chapter provides the background information of the project, aiming to make the problem context and the project's purposes clear.

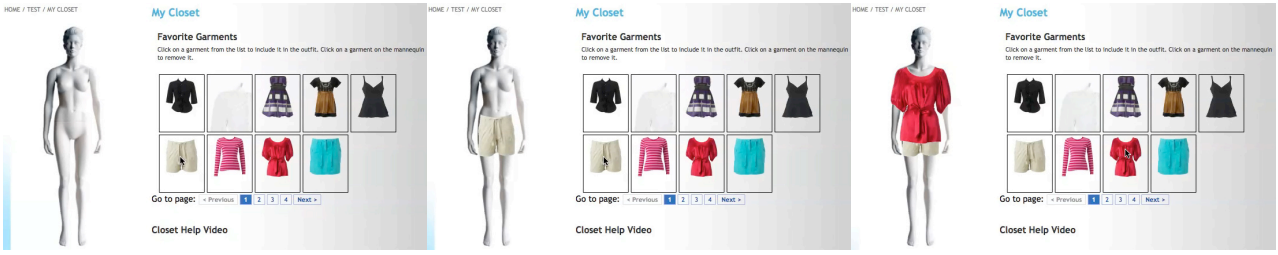
2.1 Introduction

The dissertation involves developing an iPhone application that is based on the FashionEye model. This chapter provides the reader with contextual information required to understand the problem context and appreciate the issues that the application aims to solve.

2.2 Project Context

2.2.1 FashionEye Web

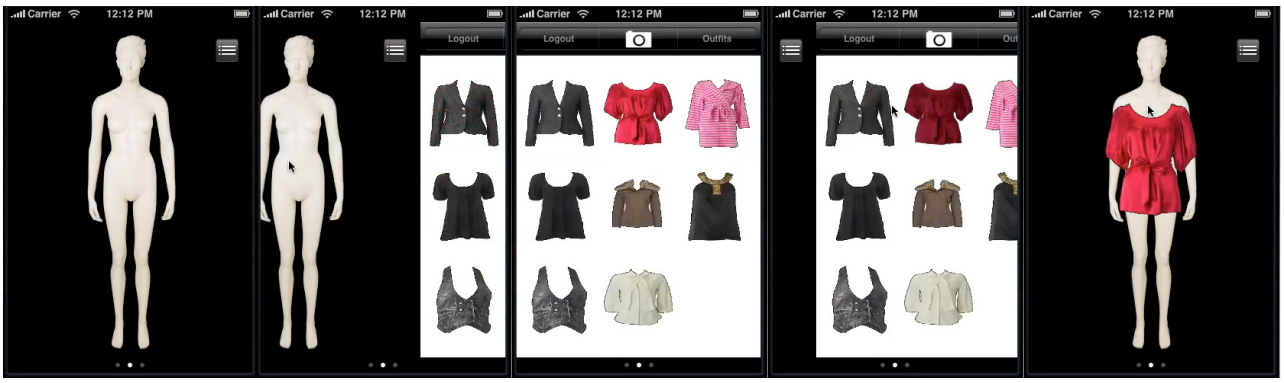
The FashionEye started as a website project and has been in development for five years. Prof. Phillip Treleaven at Dept. of Computer Science at UCL is the project leader, and also both the supervisor and the client for the iPhone project. The effort of FashionEye is to integrate fashion and social networking, one of the most popular features online, and the core features of fashion have been identified to be *Mix & Match* of clothes on mannequin, *Closet* allowing users to store, share and upload clothes, and *Recommender System* for suggesting clothes based on users' preferences and tags on clothes. The development work is in progress to complete the functionalities for those features. On top of those fashion related features, the social networking features, such as, adding friends, posting comments, and searching and sharing in the social networking space are to be added. The following is an example screenshots for *Mix & Match* on the website:



View Transition 2.1: *Mix & Match* of clothes from closet on mannequin on website.

2.2.2 FashionEye iPhone

With the increasing popularity of smartphones (comScore, Inc. 2010), the mobile version of FashionEye was wanted in order to support and extend FashionEye. It is expected to include most of the core features of FashionEye (some features need to wait for the website to complete, such as, *Recommender System*) by transforming the website design into the iPhone design, and as well as taking advantage of what the iPhone's capabilities and providing iPhone-specific features using the iPhone's multi-touch, built-in camera and the inherent mobility. The following is an example screenshots for *Mix & Match* on the iPhone as a comparison to the website's equivalent functionality (*View Transition 2.1*):



View Transition 2.2: *Mix & Match* of clothes from closet on mannequin on iPhone.

3

Requirement Gathering

In order to achieve a high quality of design and implementations, studies were carried out to identify the (prospective) customers' needs and the application's features in demand. This chapter describes the process.

3.1 Introduction

The importance and complexities of the requirements gathering is well described by Frederick P. Brooks, Jr in his paper, "*No Silver Bullet: Essence and Accidents of Software Engineering*":

"No other part of the conceptual work is as difficult as establishing the detailed technical requirements, including all the interfaces to people, to machines, and to tother software systems. No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later." (Brooks 1986)

As the requirements gathering is usually the first part in the life cycle of any software development project and forms a base for the whole development effort, careful studies were conducted to identify the (prospective) customers' needs and the application's features in demand.

3.2 Approach

The following diagram illustrates the activities undertaken during the requirement gathering and discussed in turn:

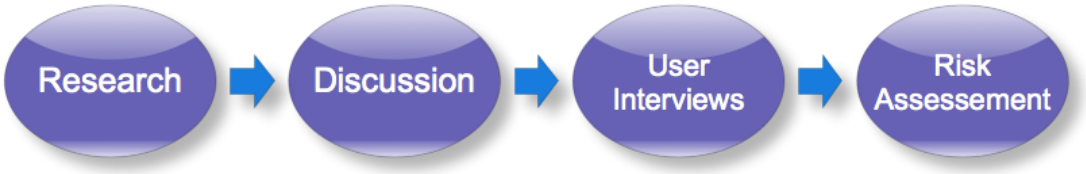


Figure 3.1: Requirement Gathering Activities

3.3 Research

The researches carried out mainly in two areas. The investigation into the FashionEye website and into the competitors' websites and iPhone applications. As the iPhone version is based upon the website version, the first area of research was essential in order to have a clear understanding of

what the current FashionEye model is and what the iPhone version should offer to give extra values to FashionEye. The detailed analysis of the FashionEye website was carried out through out the course and an example of the early stage of it is in Appendix B.

The second area of research is to look into the kinds of features that the competitors and fashion-social related websites and iPhone applications were offering. A number of fashion-social related websites and iPhone applications were studied in terms of their features in 10 categories: Communication, Content, Content Creation, Sales, Mannequin, Cost, Partners, Mobile Support, Game, Favourite and iPhone Specific Features. The result of analysis is provided in Appendix C.

3.4 Discussion

The weekly meetings with Prof. Phillip Treleaven and my collaborator, and later with other students from CS Dept. involved, were used to exchange ideas of FashionEye, both websites and iPhone. The result of researches were discussed and the feedbacks reflected the way choices were made to design the application.

3.5 User Interviews

The user interviews were conducted to collect opinions of the potential users on the kinds of features they would want to expect from a fashion application. The user involvement also helped in differentiating important features and less-important features from the end user's point of view. The result of the study is included in Appendix D.

3.6 Risk Assessment

A set of risks were identified and listed in categories to act as a reminder of predictable risks during the project. The Appendix E details the assessed risks.

3.7 Findings

The research, discussion and the other two methods helped identifying and categorizing the features that are found to be important for the iPhone version of FashionEye and the prospective users. The requirement gathering went into the process of refining the requirements. One to two months were spent in this process to allow careful thoughts about the (prospective) customers' needs and the application's features in demand and feedback from the people involved in the project.

4

DESIGN

This chapter describes the process of forming the design based on the requirements gathering stage.

4.1 Formal Statement of Requirements

The following table shows the requirements of the application in the MoSCoW method (Clegg and Barker 1994), giving a clear picture of an agreed set of functional and non-functional requirements of the application:

Priority	Description
M (**M**ust-have)	MUST have this.
S (**S**hould-have)	SHOULD have this if at all possible.
C (**C**ould-have)	COULD have this if it does not affect anything else.
W (**W**ould-have)	WON'T have this time but WOULD like in the future.

List 4.1 MoSCoW priorities

Priority	Functional Category	Description
M	FE core	View Garments in Closet
M	FE core	View Outfits in Closet
M	FE core	View Outfit details on Mannequin
M	FE core	View Garment details
M	FE core	Edit Outfit on Mannequin: Name, Sharing Status, etc.
M	FE core	Add/Save/Remove Outfit to Closet
M	FE core	Add/Save/Remove Garment to Closet
M	FE core	Mix & Match of garments and outfits from closet
M	FE core	Sharing of outfits
S	FE core	Search Garments/Outfits
S	iPhone-specific	Image Uploading
S	iPhone-specific	Zooming in/out a garment HQ image
W	iPhone-specific	Barcode Scanning
M	administration	Login/Logout

Priority	Functional Category	Description
S	administration	Register with FashionEye
S	general	Password Reminder
W	general	History [Undo actions]
Category	**Non Functional Category**	**Description**
M	FE core	Mix & Match should be smooth
S	iPhone-specific	Minimize networking
M	iPhone-specific	Coded in Objective-C

List 4.2: Requirements in MoSCoW

4.2 VIEWS WITH FEATURES

Since the iPhone's UIs are provided with views, made up of windows, controls and other elements that the user can see and interact with, it is important to design and incorporate the views with the features in mind. In this process, the sets of necessary views were identified for all the features identified in the requirement gathering stage with detailed use cases with the views. The result is provided in Appendix F.

4.3 USER INTERFACE DESIGN

The next phase of the design was to finally form the ideas and concepts developed at earlier stages. The "*iPhone Human Interface Guidelines*" (Apple Inc. 2010) document from Apple describes, in-details, the principles of the iPhone user interface (UI) and both suggests and requires developers to design and implementation the UI in the way Apple agrees with. Following the guideline is not only a good idea but also an essential since the application not conforming to the guideline can be rejected to be released on their online App Store. Therefore, the guideline was used as a reference during the UI design and mockup phase of the project.

The mock-ups were created for each event and each transition between different views. The mock-ups were produced by Ralfas Jegorovas using Adobe Photoshop. The UI designs are included in Appendix G.

For illustrative purposes, the following presents the three views for the Mannequin view, the Outfit view(showing the details of the outfit on mannequin) and the Sharing view (specifying the sharing status of the outfit):

View 4.1: View mock-ups produced by a collaborator Ralfas Jegorovas using Photoshop

4.4 System Schema

The following system schema illustrates the overview of the system interactions including the iPhone and the API (red allow in *Figure 4.1*) to access the FashionEye database:

The API was written by the project collaborator, Ralfas Jegorovas to facilitate communication between the iPhone and the FashionEye server. The reason for the API development was discussed in the next section *4.5 Database Design*.

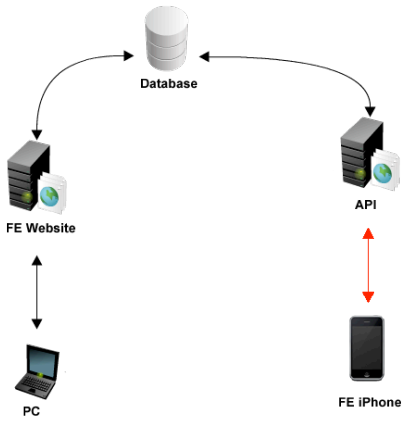


Figure 4.1: System Schema of FashionEye, the red arrow shows interactions between iPhone and API.

4.5 Database Design

It is important to mention that the iPhone version of FashionEye uses the existing FashionEye database in order to enable sharing of data between the website and the iPhone transparently to the user. For this reason, the database development was not part of the project. Rather, how the iPhone application should use the existing database was the problem needed to be addressed. Writing an API to achieve the communication was the answer to the problem.

5

IMPLEMENTATION

This chapter provides the implementation details for all core functionalities together with each views (UI) that support those functionalities.

5.1 Core Technologies

This section briefly explains the core technologies used for the development work:

- iPhone SDK
- Objective-C
- JSON
- PHP
- MySQL

List 5.1: Core technologies for the iPhone application development

5.1.1 iPhone SDK

The iPhone application development must be done by Apple's official iPhone SDK (Standard Development Kit). One of the most important elements included with iPhone SDK is Xcode, Apple's official IDE (Integrated Development Environment). Xcode provides rich sets of tools for writing and debugging code, compiling applications and performance tuning the compiled applications (D. Mark and J. LaMarche 2009). The iPhone SDK 3.2.1 and 3.2.2 through out the project.

5.1.2 Objective-C

Objective-C is the primary development language for the iPhone platform. It is an Object Oriented extension to C language and is a strict superset of C, making Objective-C compiler possible to compile C code (Apple Inc. 2009). The project's version is Objective-C 2.0.

5.1.3 JSON

JSON was selected for a data interchange format between the API and the iPhone application. JSON is an primarily used data interchange format between server and a web application, serving

an alternative to XML (Wikipedia 2010). The JSON's popularity is the much better human readability over other widely-used data interchange format, such as, XML.

5.1.4 PHP & MySQL

PHP and MySQL are used for creating the FashionEye website. Therefore, it was an reasonable choice to use PHP to write an API. The API was written by a collaborator, Ralfas Jegorovas (details of the work are found in his part of the report).

5.2 iPhone Implementations

This section presents details of the most technically challenging part of the project, programming. The application ended with 17 view controllers, out of all 24 classes. As iPhone applications are build using the Model-View-Controller (MVC) paradigm, the view controllers provide a vital link between the view objects and the rest of the application in the MVM model. The following is the list of all the classes and the view controllers are highlighted in blue:

1. User
2. Garment
3. Outfit
4. MannequinViewController
5. GarmentViewController
6. GarmentZoomViewController
7. OutfitViewController
8. NameViewController
9. SharingStatusViewController
10. SearchViewController
11. SearchOutfitViewController
12. SearchStoresViewController
13. ClosetGarmentViewController
14. GarmentUploadViewController
15. ClosetOutfitViewController
16. LoginViewController
17. ForgotPasswordViewController
18. RegisterViewController
19. GenderViewController
20. BirthdayDatePickerViewController
21. FashionEyeAppDelegate
22. MultiTapView
23. DragView.m
24. MD5Extensions

List 5.2: View Controller Classes

The following section provides discussions of the seven core implementations and the references to the appropriate views and classes and view controllers are made in each section:

1. Mix & Match
2. Closet
3. Image Uploading
4. Searching
5. Login, Register and Password Reminder
6. API and Server Communications
7. Other Details

List 5.3: Core implementations

NOTE:
The code presented in the chapter is not necessarily the exact code and details are occationally omitted for better readability and space-efficiency in the document. The exact code can be found in the attached CD.

The view of the view controller is referred to as the view controller's class name without controller e.g. the "garment view" is the view for the "GarmentViewController".

5.2.1 Mix & Match

The mix & match, adding and removing a garment on and from mannequin, is the core interactive feature of FashionEye. The addition and removal of a garment is achieved with the following high-level steps:

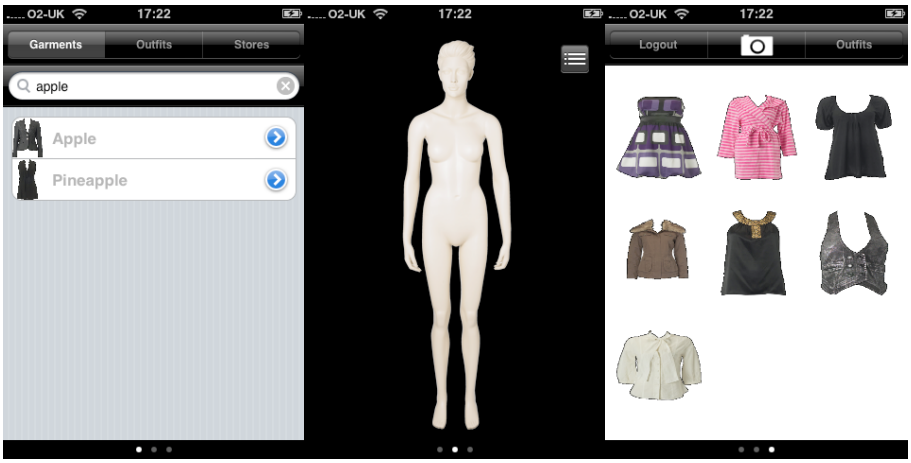
STEP 1. The user selects (touches) a garment from the garment closet.
STEP 2. The selected garment is put on mannequin.

Action 5.1: adding a garment on mannequin.

STEP 1. The user selects the garment on mannequin.
STEP 2. The selected garment is removed from mannequin.

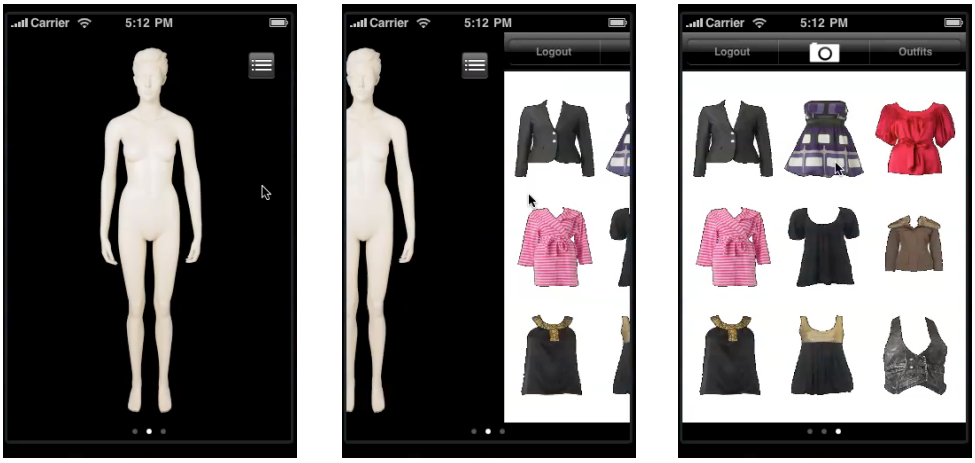
Action 5.2: removing a garment from mannequin.

With the design of our three-view model (View 1.1), the transitions between the garment closet view and the mannequin view is made easy by just flipping left to right and vice versa.

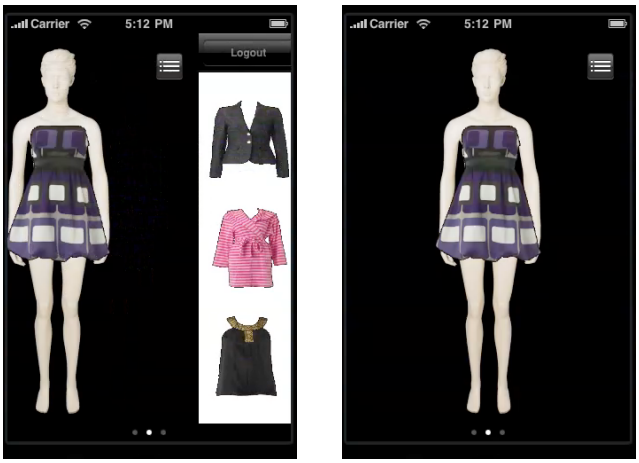


View 5.1: Search (Left), Mannequin (Middle), Closet (Right)

The following screenshots illustrate the transitions between the mannequin view and the garment closet in adding a garment on mannequin:



View Transition 5.1: Slides (left-to-right) from Mannequin to Closet by flipping a finger from right-to-left.



View Transition 5.2: Auto-slides (right-to-left) from Closet back to Mannequin after a garment is selected.

5.2.1.A MannequinViewController

The mannequin view is the implementation of the MannequinViewController class which is responsible for adding and removing garments on and from mannequin from closet. The following is the methods for addition and removal:

```
- (void)dressUpMannequinWith:(Garment *) garment {
    // adds a garment to an array of garments currently on mannequin:
    [((NSMutableArray *) [fe_application getGarmentsFromUserOutfitOnMannequin]) addObject:garment];
    // creates an image view with an image from the selected garment:
    UIImageView *garmentImageView =
        [[[UIImageView alloc] initWithImage:garment.image] autorelease];
    // set the positions of the image view:
    garmentImageView.frame = CGRectMake(g.x, g.y, g.width, g.height);
    // add the garment image view on top of the mannequin image view:
    [self.mannequinImageView addSubview:garmentImageView];
}
```

Code 5.1: Adding a garment in closet on mannequin

When a garment is selected from the garment closet view, the ClosetGarmentViewController will call the `dressUpMannequinWith` method of the MannequinViewController with the selected garment instance held in the garment closet view (discussed in section *5.2 Closet*). The code (*Code 1.1*) with comments should be straight-forward to the reader.

```
- (void)removeGarmentAt:(CGPoint) touchLocation {
    // loop all the garments on mannequin
    for (int i = [self.mannequinImageView.subviews count] - 1; i >= 0; i--) {
        // center of the garment image view
        CGPoint center =
        [((UIImageView *) [self.mannequinImageView.subviews objectAtIndex:i]) center];
        // if touch is 40 pixels around center of the garment image view then remove it and return
        if (center.x > touchLocation.x - 40 && center.x < touchLocation.x + 40 &&
            center.y > touchLocation.y - 40 && center.y < touchLocation.y + 40) {
            // remove the gearment image view from the mannequin view
            [((UIImageView *) [self.mannequinImageView.subviews objectAtIndex:i]) removeFromSuperview];
            // remove from the garment object from mannequin (the outfit object)
            [[fe_application getGarmentsFromUserOutfitOnMannequin] removeObjectAtIndex:i];
            return;
        }
    }
}
```

Code 5.2: Removing a garment off mannequin

The algorithm used for garment removal off mannequin is slightly more complicated. This is due to the work of detecting the touch position on screen for removing an appropriate garment image view. The algorithm loops through all the garment image views (top-to-bottom layers) on the mannequin view and removes the first matching garment image view whose centre is within 40 pixels of the the touch-detected position. In order to return the touch location, MannequinViewController extends MultiTouchView which is a custom class written for returning the location information. The following is the code for the `touchesEnded` method in the MultiTouchView class for the purpose:

```
- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event {
    UITouch *touch = [[event allTouches] anyObject];
    CGPoint location = [touch locationInView:touch.view];
    // NSLog(@"touchesBegan: (%f,%f)", location.x, location.y);
    // call the MannequinViewController's removeGarmentAt with the touched location
    [delegate removeGarmentAt:location];
}
```

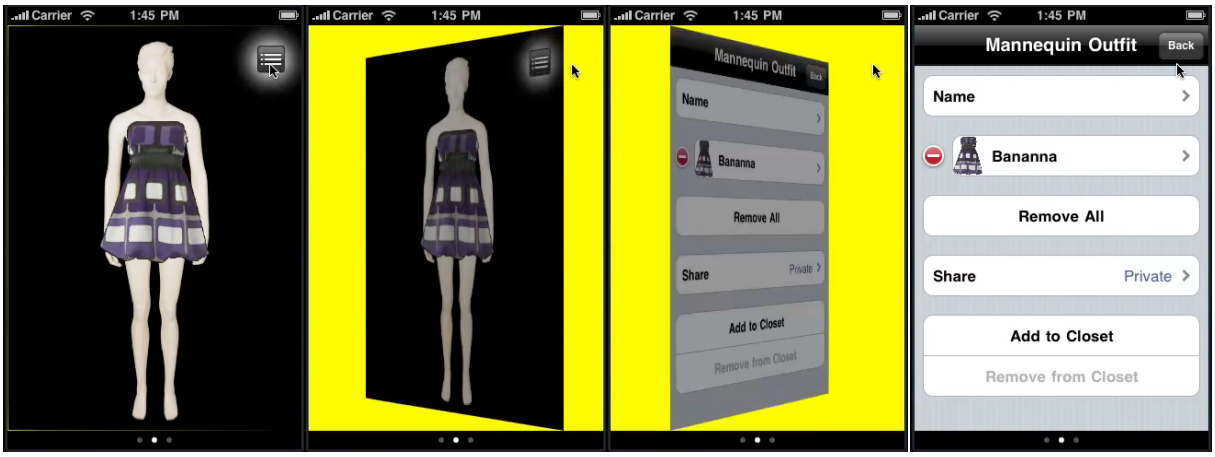
Code 5.3: Calling the removeGarmentAt method of MannequinViewController with the touched location in MultiTouchView.

Note: The discussion for Closet is delayed in the next section 5.2.2 Closet.

5.2.1.b OutfitViewController

The OutfitViewController class is responsible for displaying the details of an outfit on mannequin: name, garments' details, sharing status and providing interactions with the information presented.

The following screenshots illustrate the transitions (by touching the  button) between the mannequin view and the outfit view:

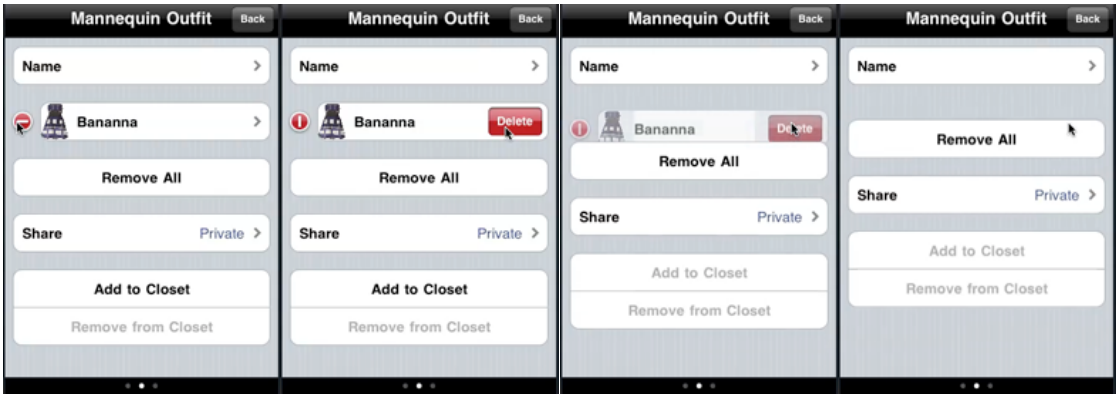


View Transition 5.3: Flipping from Mannequin view to Outfit view

NOTE: The background is set yellow for the illustration purposes.

The outfit view, the left-most view in *View Transition 5.3*, provides interactions with and details of the outfit on mannequin. The following looks at each of the interaction available in the outfit mannequin view.

In the previous section, the removal of a garment off mannequin by touching an appropriate position on the mannequin view was described. The same result can be achieved in the outfit view. The following illustrates the process:



STEP 1. selecting the button for deletion

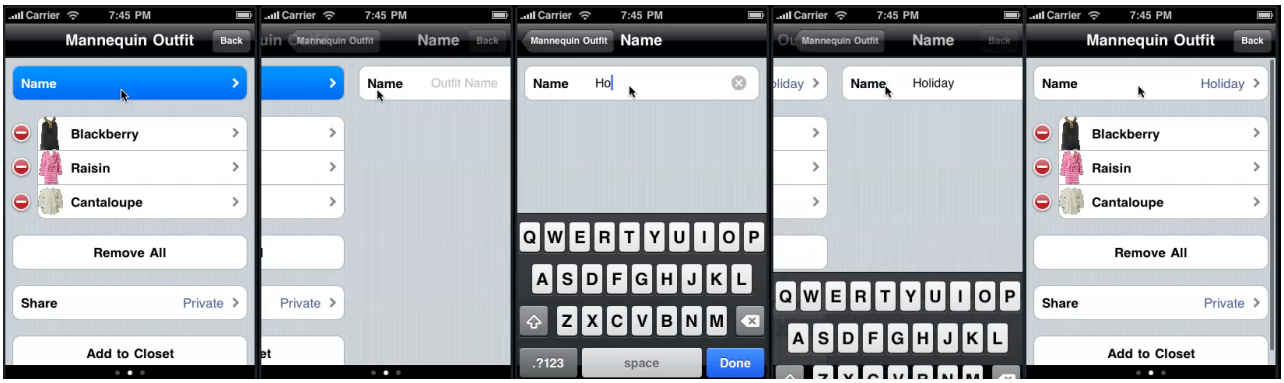
STEP 2. confirming the deletion with the Delete button

Action 5.3: Deleting a garment off mannequin in the outfit view.

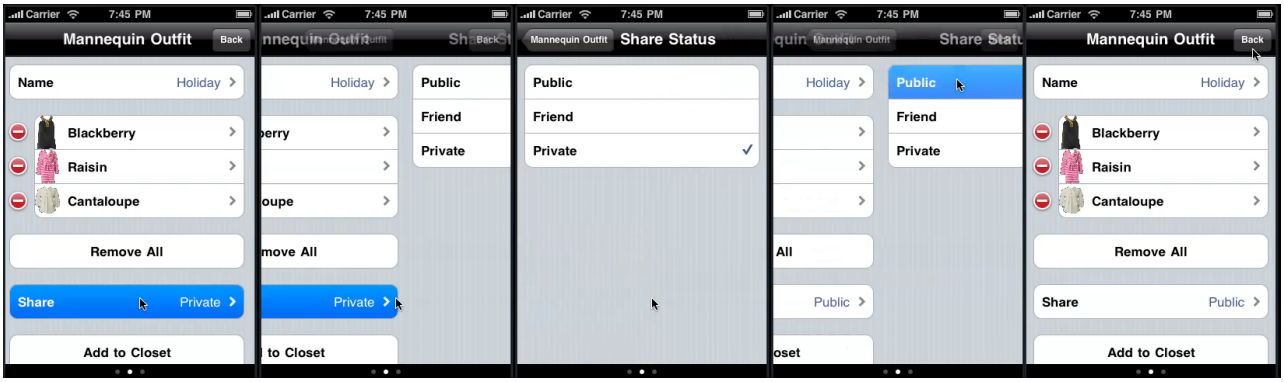
The **Remove All** is to remove all the garments on mannequin stored in an array, NSMutableArray.

Both **Name** and **Share** will slide the outfit view to NameViewController and SharingStatusViewController for setting the sharing status:Public, Private and Friends.

The following illustrates transitions from the outfit view to the name view (*View Transition 1.4*) and to the sharing status view (*View Transition 1.5*):



View Transition 5.4: Slide to the name view, change the name to “Holiday” and slide back to the outfit view.



View Transition 5.5: Slide to the sharing status view, change the sharing status to “Public” and slide back to the outfit view.

The logic behind the sliding is done by UINavigationViewController which supports the navigation i.e. sliding between more than one views. The OutfitViewController is sitting on top of the UINavigationViewController for achieving the transitions i.e. the underneath navigation controller will take charge of the views put on top. The following code is to illustrate the relationship between navigation controller presents and the sharing status view:

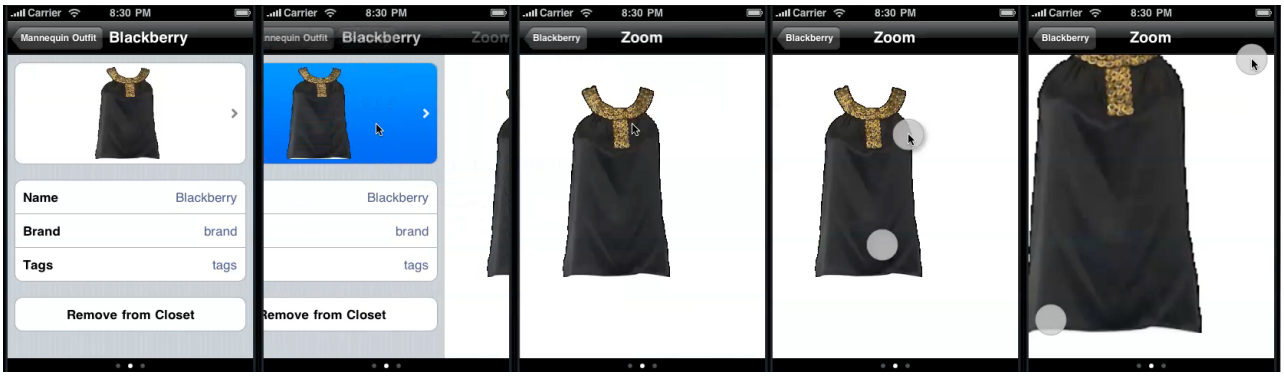
```
// on selecting "Share",
// initialize shareStatusView and push it onto navigationController (transition happens):
SharingStatusViewController *shareStatusView =
    [[SharingStatusViewController alloc] initWithNibName:@"ShareStatusView" bundle:nil];

shareStatusView.title = @"Share Status";
[self.navigationController pushViewController:shareStatusView animated:YES];
[shareStatusView release];
```

Code 5.4: The navigation controller presents (push) the sharing status view controller

The **Add to Closet**, **Save to Closet** and **Remove to Closet** are dynamically enabled and disabled according to the status of the user's closets. e.g. **Add to Closet** if the outfit is not in the user's outfit closet. The details of the actions are discussed in section *5.2.2 Closet.*

5.2.1.c GarmentViewController

The GarmentViewController is responsible for displaying details of a garment. The only interaction for the view is to slide to the zoom view for the zooming feature. The following illustrates the transition and the feature:



View Transition 5.6: Slide to the zoom view from the garment view and zoom the garment by pinching out.

The zooming was the iPhone-specific feature. The iPhone's support for multi-touch, detecting more than one finger movements, suggests us to allow the user to zoom in/out a garment image by pinching in and out.

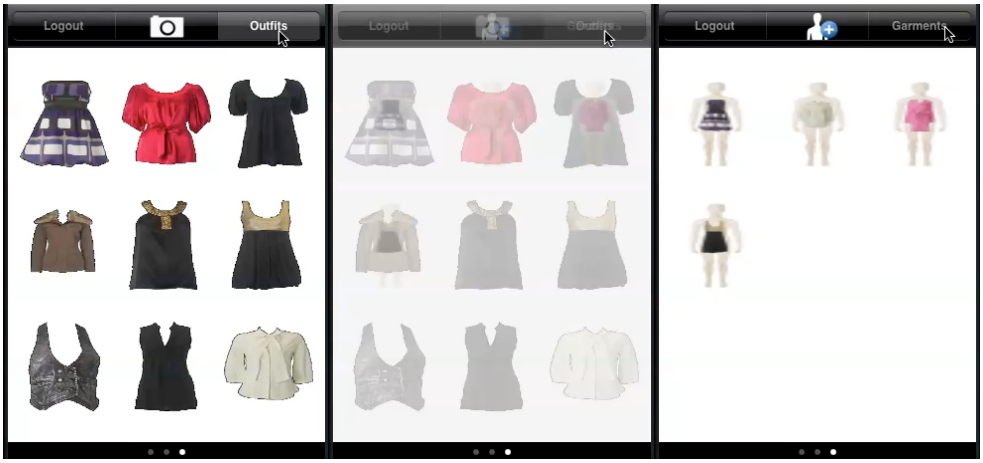
The GarmentZoomViewController class is for providing the functionality. It is essentially a multi-touch enabled UIScrollView (a scrollable view), a view that implements horizontal and vertical scroll.

5.2.2 Closet

There are two categories of closets in FashionEye, one for garment and the other for outfit. This model is directly applied to the iPhone application. The GarmentClosetViewController class is responsible for the garment closet view. The OutfitClosetViewController class is the outfit version with similar functionalities to GarmentClosetViewController.

The mix and match from closets to mannequin is one of the main purposes of having the garment and outfit closet view as illustrated in *View Transitions 1.1* and *1.2*. Also, this is the most appropriate place to put the "Logout" button. Additionally, the garment closet view has the  button for image capturing and uploading discussed in section *5.2.3 Image Uploading* and the outfit closet view has the  button to create a new outfit for editing and adding a outfit from scratch.

The following illustrates transitions between the garment closet and the outfit closet by touching the Outfits button to go to the outfit garment view, and vice versa:



View Transition 5.7: Clicking the "Outfits" button on the garment closet view to present the outfit closet view.

The GarmentClosetViewController class invokes the `presentModalViewController` method to present the outfit closet view on top of itself and the outfit view controller can go back to the garment view controller by calling the GarmentClosetViewController's `dismissModalViewControllerAnimated` method.

Each of the closet is initialized on login as part of login initialization actions discussed in section *5.2.5 Login, Register and Password Reminder*. Initially, each closet is filled with at most 9 items. The limitation is to minimize the overhead of communicating and retrieving information from the server as images must be downloaded form it. The *load more...* button at bottom allows the user to subsequently retrieve more garments and outfits into their closets.

View 5.2: The *load more...* button on the garment closet view

It is essential to keep the closets up-to-date at all times. For every action that is associated with the state changes of garments and outfits i.e. **Add to Closet**, **Save to Closet** and **Remove to Closet**, it is necessary to update the closets as well as the User instance to keep the current user's state up-to-date.

To illustrate the complexities of such updates, I present the code comment (Code 1.5) showing steps for removing an outfit and the code (Code 1.6) for adding a garment to closet:

```
/*
    1. take off all the outfit's garment(s) from mannequin
    2. remove the outfit's garment(s) from the garment closet
    3. refresh the garment closet view
    4. remove the outfit's garment id(s) from the User's garmentIds
    5. remove the outfit from the outfit closet
    6. refresh the outfit closet view
    7. remove the outfit id from the User's outfitIds
    8. set the outfit's outfitId and name to initial values i.e. 0 and "untitled" respectively
    9. flip back to the mannequin view
*/
```

Code 5.6: Removing an outfit from the outfit closet.

For the full code, please refer to the source code in the attached CD.

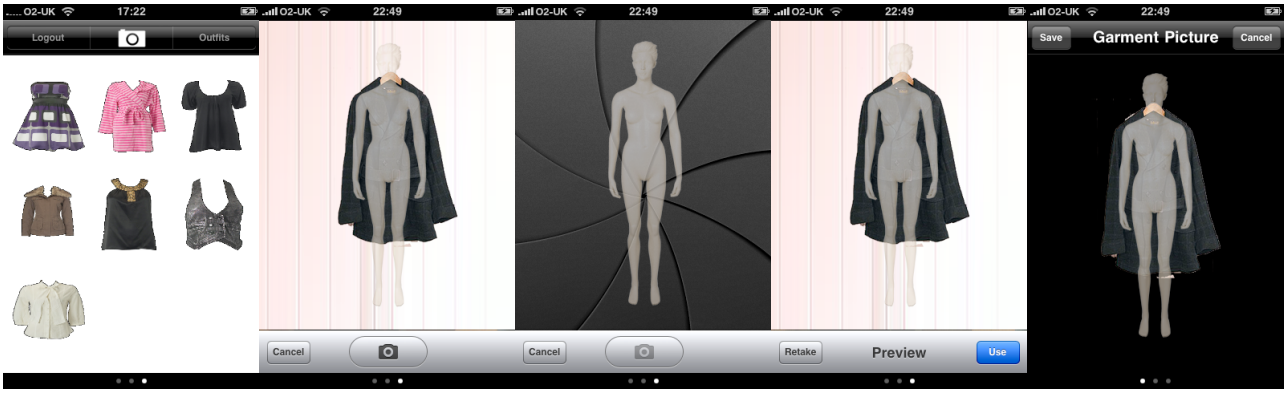
```objc
/*  STEPS for adding a garment
    1. add the garment to the User's garmentsInCloset
    2. refresh the garment closet view
    3. add the garment id to the User's garmentIds
 */

// encoding the input NSString into the JSON object:
NSString *add_garment_json_format = @"{\"username\":\"%@\", \"password\":\"%@\", \"id\":\"%d\"}";
NSString *jsonObject = [[NSString alloc] initWithFormat:add_garment_json_format,
                        [fe_application getUserUsername],
                        [fe_application getUserPassword],
                        self.garment.garmentId
                        ];
[add_garment_json_format release];
// URL of the API
NSURL *url = [NSURL URLWithString:@"http://fashioneye.com/fash1on/api/api.php"];
// sending the request to the API with the JSON object with the "addGarment" action
ASIFormDataRequest *request = [ASIFormDataRequest requestWithURL:url];
[request setPostValue:@"addGarment" forKey:@"action"];
[request setPostValue:jsonObject forKey:@"jsonObject"];
[request startSynchronous];
// on return from the request, if there is no error then take the STEPS for adding a garment
if ([request error]) {
    UIAlertView *alert =
    [[UIAlertView alloc] initWithTitle:nil
                  message:@"Unable to connect to Internet."
                 delegate:self cancelButtonTitle:@"OK" otherButtonTitles:nil, nil];
    [alert show];
    [alert release];
} else {
    NSString *response = [request responseString];
    NSDictionary *response_object = [response JSONValue];
    // 1. add the garment to the User's garmentsInCloset
    [[fe_application getUserGarmentsInCloset] addObject:self.garment];
    // 2. refresh the garment closet view
    [fe_application refreshClosetGarmentView];
    // 3. add the garment id to the User's garmentIds
    [[fe_application getUserGarmentIds] addObject:[NSDecimalNumber numberWithUnsignedInteger:self.garment.garmentId]];
}
```

Code 5.7: Adding a garment to the garment closet.

5.2.3 Image Uploading

The iPhone's built-in camera support opens up many possible use cases for the iPhone version of FashionEye. The client insisted on quick and easy way to allow users to take pictures of their garments and upload it to their account on the server.

The following is the screenshots for the capture-upload process showing the iPhone's camera view (a built-in view) and the garment upload view:



View Transition 5.8: Select the camera button, take a picture, upload it for processing and position it for uploading.

The following steps will be taken to achieve the image capturing and uploading:

STEP 1. User touches the camera button on the garment closet view
STEP 2. The camera view will be initialized and presented
STEP 3. Take a picture
STEP 4. Upload it to the server for the background removal
STEP 5. Bring up another view (GarmentUploadView) for positioning the image
STEP 6. Save it to the user's garment closet on server
STEP 7. On success, apply the change locally

Action 5.4: Uploading an image

5.2.3.1 Implementation Details

UIImagePickerController is the standard class included in UIKit.framework. The class has the system-supplied user interface for manipulating pictures on iPhone including using camera and possible actions after pictures are taken.

The GarmentClosetViewController has an instance of the UIImagePickerController for image uploading. The class will only be instantiated and subsequently used only for a device with a camera, namely iPhone. The static method, `isSourceTypeAvailable`, defined in UIImagePickerController is responsible for the check:

```
[UIImagePickerController isSourceTypeAvailable:UIImagePickerControllerSourceTypeCamera]
```

Code 5.8: Checking if the camera is available on the device.

The check is necessary since the application itself is not limiting to only iPhone users.
The camera view presented by touching the camera button on the garment closet view is shown in the second image in *View Transition 5.8*. The translucent mannequin image is added on top of the

camera view to aid users to capture garment images. The following is the code for achieving the effect:

```
// instantiate an image view with the translucent mannequin image:
UIImageView *imageView = [[UIImageView alloc]
    initWithImage:[UIImage imageNamed:@"mannequin_camera_trans.png"]];
imageView.contentMode = UIViewContentModeCenter;
[self.imagePickerController.view addSubview:imageView];
```

Code 5.9: Adding a translucent mannequin image on the build-in camera view.

After taking a picture, users will have an option of retaking or uploading it to the server for background removal. The background removal will be done by the FashionEye server. If the Internet connection is not available, the process can be skipped by the user. After the image is returned from the server, the the UIImagePickerController is dismissed and the GarmentUploadViewController is presented.

The GarmentUploadViewController view is presented programmatically (not by an event triggered by the user) with the following code:

```
[self performSelector:@selector(bringUpGarmentUploadView)
           withObject:nil
           afterDelay:0.5];
```

Code 5.10: Presenting the garment upload view after 0.5 seconds

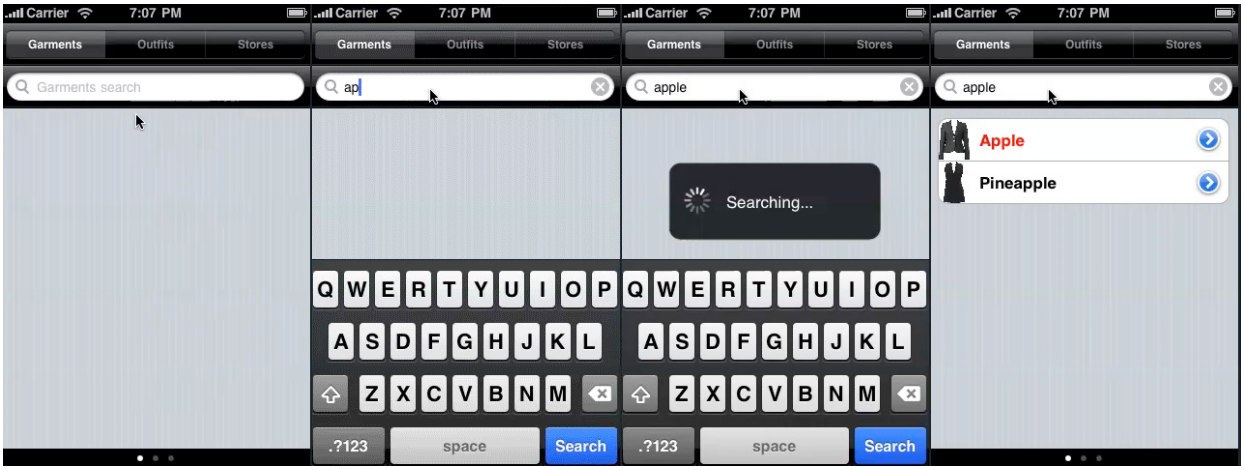
The work-round solution was used since there is no obvious way to invoke present/dismiss views one after another. The `performSelector` method defined in the NSRunLoop class is for triggering an action programmatically. For our purpose, it presents the GarmentUploadViewController view after 0.5 seconds when the method is first reached. The delay is important to allow the first view to be dismissed and the second view to be ready to be presented.

The GarmentUploadViewController class was defined for enabling the user to position the image appropriately on the mannequin by dragging the image on the screen with a finger and save it to the user's garment closet on the server. For the dragging, the DragView was defined for enabling the gesture on the UIImageView with the captured image and storing the positions (co-ordinates) for later upload. Finally, the positioned image will be uploaded when save is clicked.

5.2.4. SEARCHING

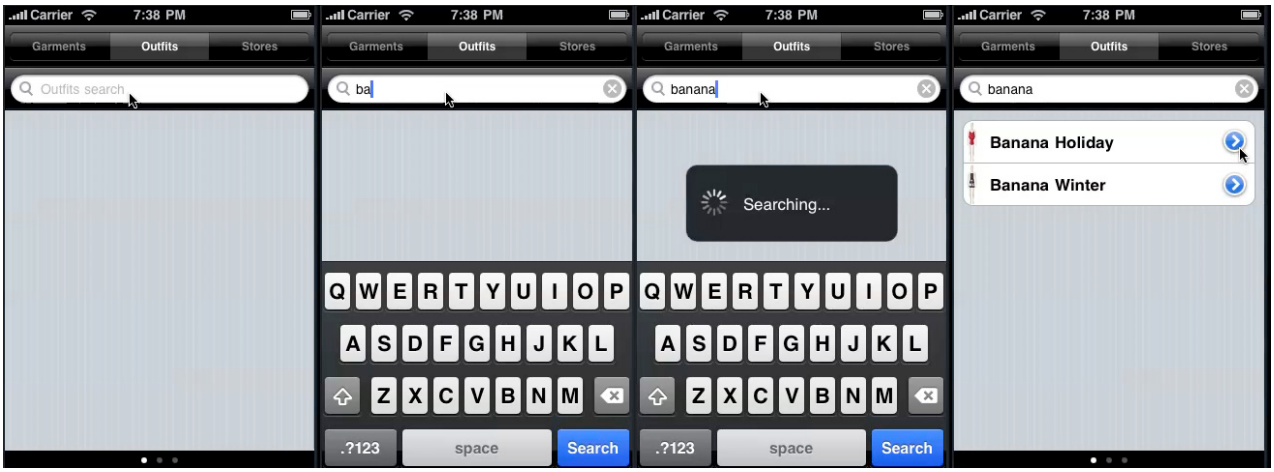
Searching provides a means of accessing the whole range of public garments and outfits available in the entire FashionEye system. This is where the user can mix & match garments and outfits of others and subsequently save them into their own closets. The garment search also searches the garments the user already has in garment closet, in which case the results will be shown differently in red. The search has three options:search garment, search outfit and search stores.

The following illustrates the search garment and its functionalities:



View Transition 5.9: Searching garments with keyword “Apple” and the results displayed.

The red-colored garment names shown in the results in *View Transition 5.9* indicate that the user already owns the garments. The ownership of the garments (and outfits for the outfit results) are checked dynamically against the user’s information so that the appropriate action i.e. add/remove is presented appropriately when the garment view is presented (discussed in the section *5.2.4.1 interactions with search results*). The outfit search works the same way, except the user’s own outfit s are not considered in the search domain. The following illustrates the outfit search:

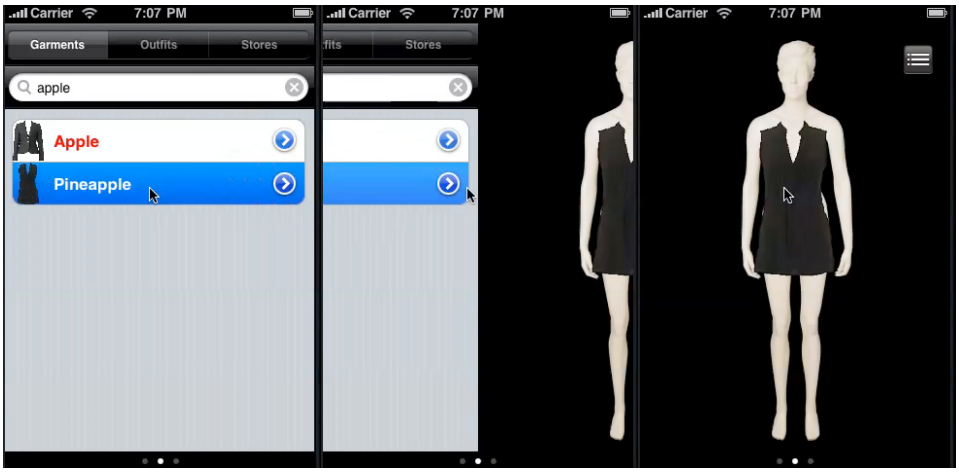


View Transition 5.10: Searching outfits with keyword “Orange” and the results displayed.

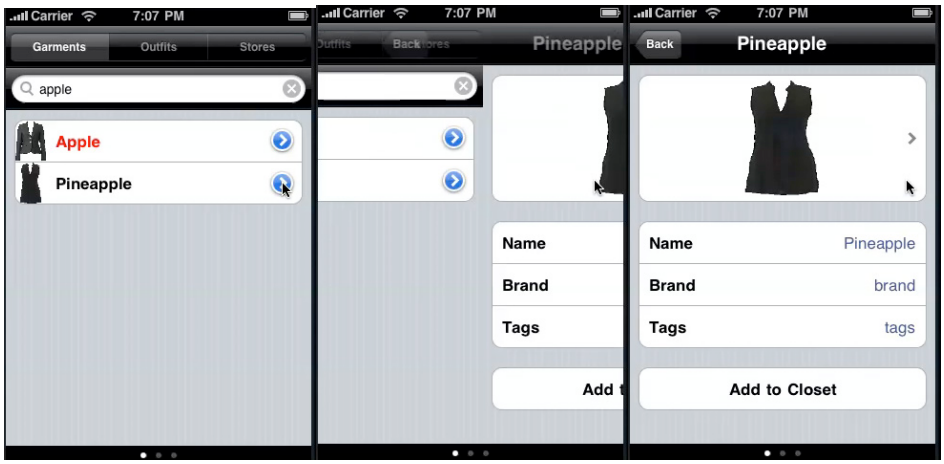
The **search stores** is where the user can search registered stores' clothes on FashionEye. However, the feature is not currently available to the system and the iPhone's implementation is just the skeleton for the up-coming feature and the detailed discussion is given in section *7.4 Future Work*.

5.2.4.1 INTERACTIONS WITH SEARCH RESULTS

There are two interactions supported on each garment shown in the results. The first action is to put on the garment by touching the row, and the second action is to check the details of the garment by touching the  button which presents the garment view for the details of the garment and the the interaction for adding or removing the garment to and from the garment closet. The garment view is essentially the same implementation as the garment view discussed in section *5.2.1.c GarmentViewController*, so zooming is also possible by moving to the garment zoom view. This is an example of the effort to keep the interface uniform both for programing and for usability where familiar views and interactions should be presented to the user for the same functionalities.



View Transition 5.11: Adding a garment on mannequin from the search results.



View Transition 5.12: Presenting the garment view of the selected garment from the search results.

The search outfit works roughly the same way at high-level with the two interactions to put on an outfit on mannequin and to add an outfit to the user's outfit closet.

5.2.5 Login, Register and Password Reminder

The section will look at the usual features of the web application in turn: *Login*, *Register* and *Password Reminder*.

5.2.5.1 Login

View Transitions 5.13: Logging in with Username and Password.

Logging-in is a simple Username-Password pair sent to the server. As it may sound trivial, a number of important initialization is required in the process. The following information is returned by the API and the client's job is to use those information to instantiate objects:

```
{"success" : true, "outfits" : OUTFIT_OBJECT_ARRAY, "outfitIds" : OUTFIT_ID_ARRAY,
"garments" : GARMENT_OBJECT_ARRAY, "garmentIds" : GARMENT_ID_ARRAY}
```

Code 5.6: JSON object returned by the API on success

For more details on API, refer to FashionEye API Reference in Appendix H.

The JSON object contains essential information required to instantiate the object of the User class which is a class to represent the user's information, such as, all the garment's ids (an array of strings) to track of the all garments the user has.

The properties of the User class are as shown in the UML class digram:

User
Properties
garmentIds:NSMutableArray *
garmentsInCloset:NSMutableArray *
outfitIds:NSMutableArray *
outfitOnMannequin:Outfit *
outfitsInCloset:NSMutableArray *
password:NSString *
username:NSString *

Class 5.1: Properties of the User class

The User object is the first class instantiated by the login process. The outfits, outfitIds, garments, and garmentIds are instantiated based on the information of the returned JSON object, and the login view's Username and Password text field values are assigned to the corresponding username and password properties of the User object on success. If the server can not find a matching the username-password pair supplied with the database, an alert will be shown to indicate this.
The next step is to instantiate the garment and outfit objects for the respective closet views. The sequence of HTTP POST requests to the server to instantiate garments and outfits. After all the required instantiations, the garment closet view is presented to the user.
The following Activity Diagram depicts the overall process of the instantiation after successful login:

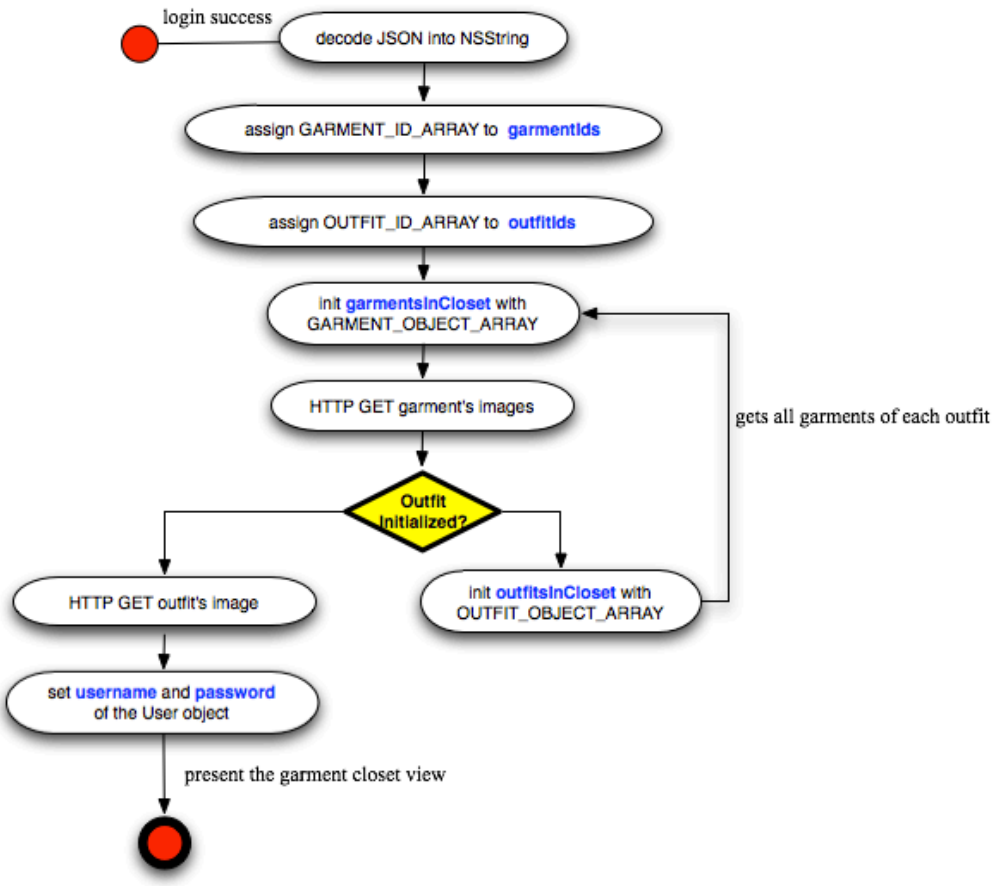


Diagram 5.1: initialization of the User object's properties on login success

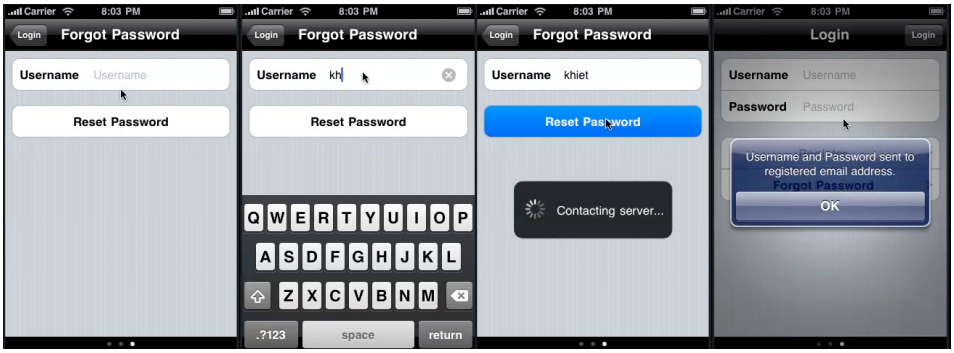
5.2.5.2 Register

The registration requires the basic information along with the information necessary to support FashionEye's future offerings i.e. **Location**, **Gender** and **Birthday** will be used for *Recommender System*, a feature being developed on the website. The registration is made easy for the user to navigate by providing appropriate views for each field. The text fields (UITextField), used from Username to Location illustrated in *Action 1.3*, are added on top of each list's cell as a subview, rather than the iPhone's default text field. The Gender is inputted by selecting between Male and Female in the gender view presented on selection and the Birthday is chosen from the birthday view shown in *Action 1.3*. The information will be sent to the server and API will do the update to the database.

View Transitions 5.14: Registering

5.2.5.3 Password Reminder

The password reminder is also provided in the login view. It is the implementation of the forgot password view with a single feature to send Username to the server and get back the server response. On success, the password is sent to the user's registered email address. The following illustrates the working:



View Transitions 5.15: Requesting Username and Password to be sent to user's registered email address with Username.

5.2.6 API and Server Communication

The communication with the FashionEye server is through an API written by a collaborator, Ralfas Jegorovas. It is written in PHP and uses JSON as a data interchange format. For the details of the work, please refer to his part of the report. The following sequence diagram illustrates the high-level view of the communication:

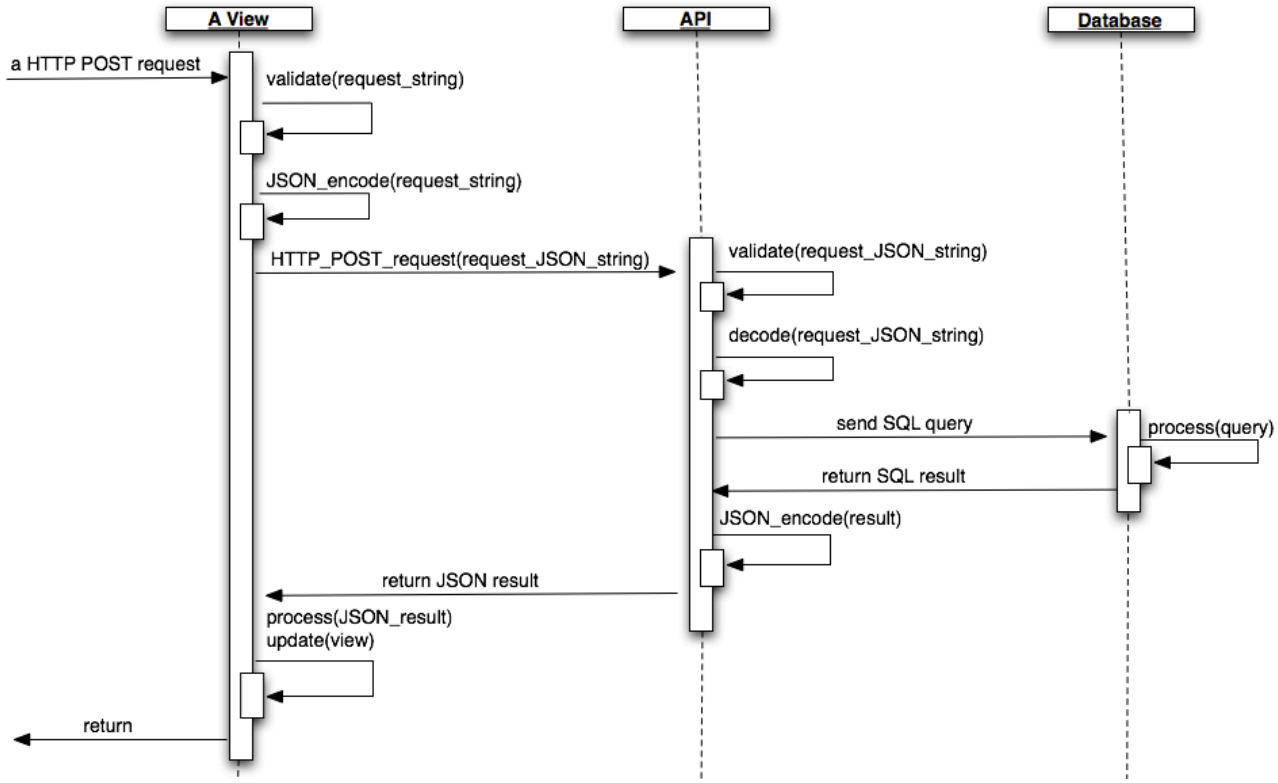


Diagram 5.2: a HTTP POST request

A HTTP POST request from the iPhone application is encoded into a JSON object, and then forwarded to the API with an action specified, such as login. The API decodes the JSON object and, together with the action, translated into the corresponding SQL query in PHP which is then sent to the database for processing. The result of the query is returned form the database to the API, and the API encode the query result into a JSON object which is sent to the iPhone application where, an appropriate action is taken for the returned JSON result.

5.2.6.1 Security

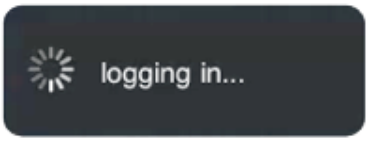
In order to enhance security, the MD5 password hashing is applied to the password sent to the server. To achieve this, the standard NSString class is extended with the following additional method to calculate and produce the hashed value:

```
-(NSString *) md5 {
    const char *cStr = [self UTF8String];
    unsigned char result[16];
    CC_MD5( cStr, strlen(cStr), result ); // md5 call
    return [NSString stringWithFormat:
            @"%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x",
            result[0], result[1], result[2], result[3],
            result[4], result[5], result[6], result[7],
            result[8], result[9], result[10], result[11],
            result[12], result[13], result[14], result[15]
            ];
}
```

Code 5.7: MD5 hash function

5.2.7 Other Details

Other details include the activity indicator, the FashionEye stat-up video (made by Ralfas Jegorovas). The activity indicator (*View 1.3*) is created, rather than using the iPhone's SDK's built-in indicator as part of design choice and for improved flexibilities in using in different places.



View 1.3: Activity indicator for login

The implementation is by adding the default activity indicator (UiActivityIndicatorView) on top of an image view (UIImageView) instantiated with the .png image created by Photoshop. The image view will be added whenever the activity indicator needs showing and removed when done.

```
-(void) addActivityIndicatorOn:(UIView *) v withHeigh:(CGFloat) height andMessage:(NSString *) msg  {
    // initialize the activity indicator with height and msg
    [self initializeActivityIndicatorWithHeight:height andMessage:msg];
    // adding the adcitivy indicator on the view specified, usually the calling view
    [v addSubview:self.activityIndicatorImageView];
}
```

Code 5.8: Method for adding the custom activity indicator

```
-(void) removeActivityIndicatorFrom:(UIView *) v {
    // removing the adcitivy indicator from the view specified, usually the calling view
    [self.activityIndicatorImageView removeFromSuperview];
}
```

Code 5.9: Method for removing the custom activity indicator

The FashionEye stat-up video is included (played) as part of advertising FashionEye on start-up. It also serves as a space holder while instantiating the program's objects. The movie is included in the attached CD.

6

TESTING

This chapter describes the work undertaken to test the application's functionalities by unit testing using an open-source library, debugging and performance tuning tools included in the iPhone SDK.

6.1 UNIT TESTING

Unit tests were written for insuring the critical parts of the code are stable and robust code by detecting problems as early as possible (Roy Osherove 2009). Not only for taking the technique as an important part of building reliable software, it also served to check my understanding against the new Objective-C language at the start of the project by seeing how *units* are working.

For unit testing the application, an open-source library, iPhoneUnitTesting included in Google Toolbox for Mac was used. The Xcode does provide a tool for unit testing but iPhoneUnitTesting was chosen for its ease-of-use.

The *Code 6.1* for testing `testGarmentRemovedbyRemoveGarmentAt` is provided for illustrative purposes. The list of unit tests can be found in the attached CD with the project's source code.

```
@interface OutfitTests : SenTestCase {
    // main fixture
    User *user;
    NSMutableArray *garments;
    Outfit *outfitOnMannequin;
}
@end

@implementation OutfitTests

// seting up fixtures
-(void) setUp {
    // main fixture
    user = [[User alloc] initWithDefaultSetting];
    STAssertNotNULL(user, @"unnable to create User instance by initWithDefaultSetting");
    // fixtures for setting the User's attributes
    garments = [[NSMutableArray alloc] initWithCapacity:9];
    outfitOnMannequin = [[Outfit alloc] initWithDefaultSetting];
    STAssertNotNULL(outfitOnMannequin, @"unnable to create User instance by initWithDefaultSetting");
}

// destroy the fixture
-(void) tearDown {
    STFail(@"come on");
    [user release];
    [garments release];
    [outfitOnMannequin release];
}
```

```
-(void) testGarmentRemovedByRemoveGarmentAt {
    // settingUpOutfitOnMannequin:
    // garments are represented as NSString for testing purposes
    for (NSUInteger i=0; i<10; i++) {
        [garments addObject:[[NSString alloc]
                             initWithFormat:@"Garment#%d", i]];
    }

    // adding the garments on the outfitOnMannequin
    outfitOnMannequin = [[Outfit alloc] initOutfitWithOutfitId:0 andGarments:garments
                                                       andName:nil
                                                   andSharing:nil andImage:nil];
    // making sure the garments were added
    STAssertEquals([outfitOnMannequin.garments count], 10, @"garments NOT added");

    // remove the last garment i.e. Garment#9, simulate the removal of the top layer
    [[fe_application mannequinViewController] takeOffGarmentAt:9];

    // if removal is successful, lastObject should be "Garment#8"
    STAssertEqualStrings(@"Garment#8", [outfitOnMannequin.garments lastObject], @"should NOT fail");
}
```

Code 5.9: Unit Test if a garment is removed form garments in outfitOnMannequin of the User instance

```
Build FEUnitTest
Project FashionEye | Configuration Debug
Check dependencies
  warning: no rule to process file '$(PROJECT_DIR)/MD5Extensions.h' of type sourcecode.c.h for architecture i386
  warning: no rule to process file '$(PROJECT_DIR)/User.h' of type sourcecode.c.h for architecture i386
  warning: no rule to process file '$(PROJECT_DIR)/Garment.h' of type sourcecode.c.h for architecture i386
  warning: no rule to process file '$(PROJECT_DIR)/Outfit.h' of type sourcecode.c.h for architecture i386
Run custom shell script 'Run Script'
  # Controlling environment variables:
  # GTM_DISABLE_ZOMBIES -
  #   Set to a non-zero value to turn on zombie checks. You will probably
  #   want to turn this off if you enable leaks.
  Test Case '-[OutfitTests testGarmentRemovedByRemoveGarmentAt]' started.
  Test Case '-[OutfitTests testGarmentRemovedByRemoveGarmentAt]' passed (0.016 seconds).
```

Figure 6.1: Result for `testGarmentRemovedbyRemoveGarmentAt`

6.2 Debugging and Performance Tuning

The debugging and removing redundancies in the application is crucial to enhance robustness, responsiveness and, therefore, usability of the resulting application. During the development, the tools provided by the iPhone SDK was used to detect any bottlenecks in the program.

The Xcode, an Integrated Development Environment (IDE), was used for the entire application development (only available IDE for iPhone software development), which features syntax highlighting and validation functionalities. One of the Xcode's features is "Build and Analysis" (an option available on compilation) which detects variables that have potential memory leaks (*Code 1.8*).

```
- (NSMutableArray *) getOutfitGarmentsByOutfitId: (NSUInteger) outfitId {

    NSString *get_outfit_garments_json_format = @"{\"username\":\"%@\", \"password\":\"%@\", \"id\":\"%d\"}";
    NSString *jsonObject = [[NSString alloc] initWithFormat:get_outfit_garments_json_format,
                            [fe_application getUserUsername],
                            [fe_application getUserPassword],
                            outfitId
                            ];
    [get_outfit_garments_json_format release];

    NSDictionary *response_object =
        [fe_application httpPostRequestWithAction:@"searchGarment" andJsonObject:jsonObject];
```

Method returns an Objective-C object with a +1 retain count (owning reference)

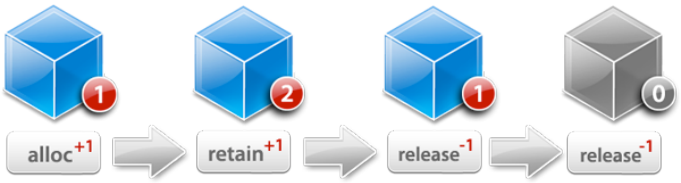
Object allocated on line 40 and stored into 'jsonObject' is no longer referenced after this point and has a retain count of +1 (object leaked)

Code 1.8: A potential memory leak by an instance jsonObject found in "Build and Analysis".

The instance, `jsonObject`, is not released (reference count decremented) anywhere in the program. The solution here is to *autorelease* the instance (because the instance is passed as an argument of a function, `httpPostRequestWithAction:andJsonObject`.

The GDB is also integrated in the IDE and the combination of these features (tools) were used to enhance the build quality of the software.

The most frequent causes of errors and bugs were caused by careless and incorrect memory management. An following illustration is an example of such mistakes occur when releasing a block of memory (an object) that has a reference count of "zero":



Step 1: alloc (allocate a new memory area)
Step 2: retain (increase reference count by one)
Step 3: release (decrement reference count by 1)
Step 4: release (decrement reference count by 1)

Diagram 6.2: Releasing (Decrementing) an object with reference count "zero".

Although the alloc (or retain) - release cycle seems obvious, getting the cycle right all the time everywhere in a program becomes a complex task and certainly a challenge to a programmer.

The iPhone SDK comes with a suit of tools to profile. A tool called "Instruments" included in iPhone SDK is proven to be very helpful to debug the code causing memory related errors. Instruments collects data such as disk, memory, or CPU usage in real time and display these information graphically as tracks over time. The following is an example of running Instrument to detect an error of referencing a freed object, `objc[1867]: FREED(id): message name sent to freed object=0x4464ae0`:

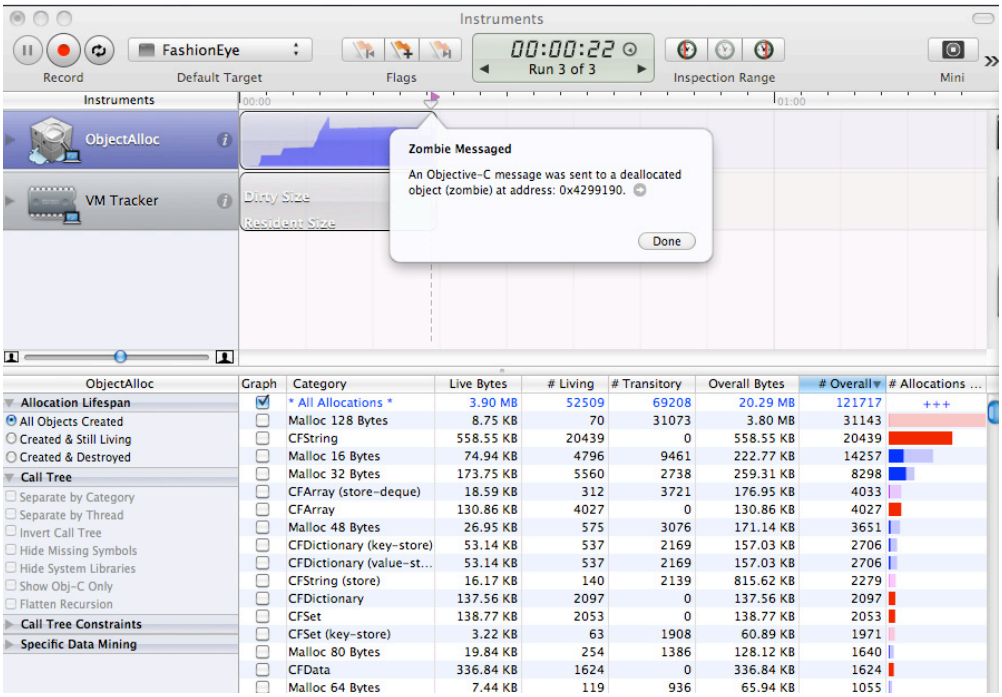


Figure 6.3: Instruments for monitoring and detected an access to a deallocated object 0x4464ae0.

6.3 Use Case Testing

After all the features were tested and performance tuning was completed, the extensive testing of all the features were done as part of the user study conduced by Ralfas Jegorovas. There were minor visual improvement suggestions and trivial bugs were reported and fixed.

7

EVALUATION

This chapter provides the evaluation of the project. The completeness of the project against initial requirements, features that needs more time to be fully tuned, and the future work that should be carried out after the website's project is completed.

7.1 Summary

The implementation covered all the high-level objectives: *Mix & Match*, *Closet*, *Searching*, Sharing, and *Image Uploading* which were identified as critical features of the iPhone version of FashionEye.

7.2 Requirements Completeness

Priority	Count	Completed	Percentage
M (Must-have)	12	12	100%
S (Should-have)	6	6	100%
C (Could-have)	0	0	0%
W (Would-have)	2	0	0%
		Total	90%

List 7.2 Initial requirements completeness in MoSCoW

7.3 Critical Reflection

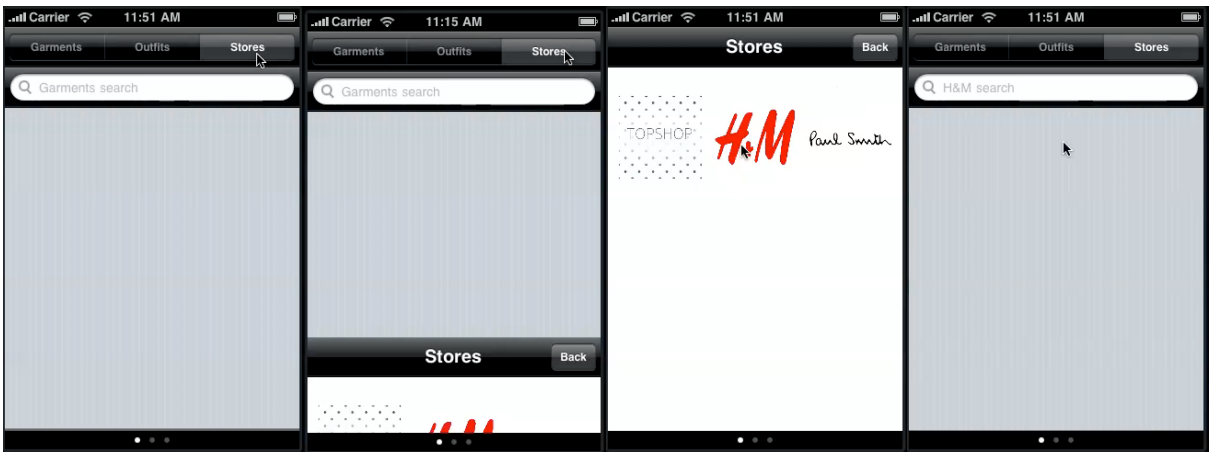
In terms of the requirements agreed in section *3. Requirements Gathering*, the project delivered a product that meats all the core features: "Must-have"s and "Should-have"s. Throughout the development, those features were refined over and over and new features were considered and added, such as, the two interactions for the search results, as part of the iterative process. One non-functional requirement that was met but needs more refinement is minimizing networking. Although, the effort was made in both API and the iPhone program to minimize networking, such as to use the smallest possible image size for thumbnail images, it still has too-much delay when using the application on 3G network. This is one area that needs more time to be fully satisfactory.

7.4 Future Work

A number of features were not implemented fully due to their dependency on the work of the FashionEye website. However, the minimum sets of work, UI and event handlers, were finished so that they are ready to be fully implemented as soon as the FashionEye website has completed their implementation.

7.4.1 Stores

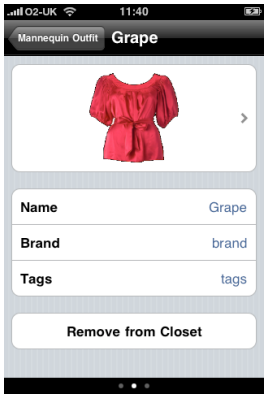
The stores is the feature in which FashionEye can provide a business model to clothing retailers. It is the feature where those retailers can have their garments' photos stored on FashionEye and the user can enjoy all the functionalities with those garments, such as, Mix & Match. Although FashionEye is still an ongoing project and there has not been any chance to advertise the feature to the potential customers i.e. clothing retailers, this is one of the features that the client wanted for FashionEye and the iPhone version went ahead to provide the skeleton of the feature in the search view. The following illustrates the transitions to the stores view is displayed in the search view:



View Transitions 7.1: Presents the store view in the search view.

7.4.2 Tags

The recommender system is one of the FashionEye's features where garments are recommended to the user based on their preferences. It is thought tags that the system will match those items against the user's preferences. Although, the recommender system was not considered to be implemented on the iPhone version (it is not completed for the website version), it was considered reasonable to put a tag field to show tags for garments in the garment view. The picture on the left is to illustrate the **Tags** field.



View 7.1: The garment view

7.4.3 suggestions for next Iteration

The user study conducted by Ralfas Jegorovas at the end of the development cycle revealed a number of suggestions for improvement on top of the current application. These suggestions;both functional and non-functional should serve as a good starting point for the next iteration of the application development. For details of the user study, please refer to a collaborator, Ralfas Jegorovas's report.

7.4.4 Release on Apple's App Store

The final stage of the application development is to release it on Apple's App Store. The application is in a stable state with most core features implemented, the release on the App Store is scheduled to be plant with the client, Prof. Phillip Treleaven after the exam with other collaborators.

8

CONCLUSION

The goal of the dissertation, or project, was to develop a companion iPhone to the ongoing FashionEye.com project. This goal was met by the successful provision of the application to the level described in the previous chapter. However, it is too easy and simple just to mention that a product was delivered. During the 8-month period, there has been a great deal of time and efforts spend on the process of understanding the problem and its context, through to the design, implementation, testing and evaluation of the deliverables which have been detailed in this report. As a final year project, I was satisfied for both the quality of work I could engage in and the learning experience from it. Whatever the outcome, I am extremely graceful for the opportunity.

Bibliography

1. Alexa Internet, Inc. (2010), "The top 500 sites on the web.", http://www.alexa.com/topsites/global.
2. M. Handley (2010), "An Introduction to C Programming for Java Programmers", http://nrg.cs.ucl.ac.uk/mjh/3005/c-intro.pdf.
3. C. Clack (2010), "GC16/3011 Functional Programming
Lecture 18: Automatic Memory Management", http://www.cs.ucl.ac.uk/teaching/3C11/PowerPoint/lecture18.ppt.
4. ISO (1998), "ISO 9241-11: Guidance on usability", ISO.
5. Apple Inc. (2010), "iPhone Developer Program License Agreement", http://www.eff.org/files/20100302_iphone_dev_agr.pdf.
6. Nielsen Online (2009), "A Nielsen report on Social Networking's New Global Footprint", http://blog.nielsen.com/nielsenwire/wp-content/uploads/2009/03/nielsen_globalfaces_mar09.pdf.
7. BBC (2009), "Google unveils Nexus One phone ", http://news.bbc.co.uk/1/hi/8442205.stm.
8. Apple Inc. (2010), "Apple Reports First Quarter Results", http://www.apple.com/pr/library/2010/01/25results.html.
9. Apple Inc. (2010), "Apple Human Interface Guidelines", http://developer.apple.com/mac/library/documentation/UserExperience/Conceptual/AppleHIGuidelines/OSXHIGuidelines.pdf.
10. comScore, Inc. (2010), "UK Leads European Countries in Smartphone Adoption with 70% Growth in Past 12 Months", http://www.comscore.com/jpn/Press_Events/Press_Releases/2010/3/UK_Leads_European_Countries_in_Smartphone_Adoption_with_70_Growth_in_Past_12_Months.
11. Frederick P. Brooks, Jr., "No Silver Bullet: Essence and Accidents of Software Engineering", http://people.eecs.ku.edu/~saiedian/Teaching/Sp08/816/Papers/Background-Papers/no-silver-bullet.pdf.
12. D. Clegg and R. Barker (1994), "Case Method Fast-Track: A RAD Approach", Addison–Wesley.
13. D. Mark and J. LaMarche (2009), "Beginning iPhone Development", Apress.
14. Apple Inc. (2009), "Introduction to The Objective-C Programming Language", http://developer.apple.com/iphone/library/documentation/Cocoa/Conceptual/ObjectiveC/Introduction/introObjectiveC.html.
15. Wikipedia (2010), "JSON", "http://en.wikipedia.org/wiki/JSON".
16. Roy Osherove (2009), "The Art of Unit Testing: With Examples in .Net", Manning Publications.
17. Stephen G. Kochan (2009), "Programming in Objective-C 2.0", Addison-Wesley.
18. E. Sadun(2008), "The iPhone Developer's Cookbook", Addison-Wesley/
19. B. Dudney (2009), "iPhone SDK Development", Pragmatic Bookshelf.
20. Apple Inc. (2009), "Memory Management Programming Guide for Cocoa", http://developer.apple.com/mac/library/documentation/Cocoa/Conceptual/MemoryMgmt/MemoryMgmt.pdf.