Clothing Fashion Style Recommendation System

A Thesis Presented

by

Wei Dai

to

The Department of Electrical and Computer Engineering

in partial fulfillment of the requirements

for the degree of

Master of Science

in

Electrical and Computer Engineering

Northeastern University

Boston, Massachusetts

May 2015

To my family.

Contents

List of Figures

List of Tables

List of Acronyms

CFSRS Clothing Fashion Style Recommendation System

CPU Central Processing Unit

FPM FastCGI Process Manager

FTP File Transfer Protocol

GUI Graphical User Interface

HDD Hard Disk Drive

HSI Hue-Saturation-Intensity

HTML HyperText Markup Language

HTTP Hypertext Transfer Protocol

IPC Interprocess Communication

JSON JavaScript Object Notation

KNN k-Nearest Neighbors

MITM Man-in-the-middle

MVC Model-View-Controller

MVP Model-View-Presenter

NIC Network Interface Controller

OS Operating System

PASCAL Pattern Analysis, Statistical Modelling and Computational Learning

PHP PHP: Hypertext Preprocessor

RHEL Red Hat Enterprise Linux

ROI Region of Interest

SQL Structured Query Language

SSH Secure Shell

TLS Transport Layer Security

URL Uniform Resource Locator

VOC Visual Object Classes

vsftpd very secure FTP daemon

Acknowledgments

Foremost, I would like to express my sincere gratitude to the committee chair and my advisor Dr. Yun Fu for the continuous support of my master study, for his patience and motivation. His enthusiasm, immense knowledge and guidance encouraged me and helped me in the time of researching and writing this thesis.

Furthermore, I would like to sincere thank the rest of my thesis committee, Dr. Yongmin Liu and Dr. Yu Kong for their encouragement and insightful comments.

My sincere thanks also goes to James Roche, Nolan Paduch, James King, Chris Towns, Vahid Kaviani and Andrew Corsini for offering me the summer internship opportunities in their groups at EMC Corporation.

Special thanks to Dr. Dmitry Kit, Ph.D candidate Ming Shao and Ph.D candidate Sheng Li for their help both in academic and life. I would also thank all members of Synergistic Media Learning (SMILE) Lab. SMILE Lab is Prof. Yun Fu's lab at Northeastern University. This thesis is also finished at SMILE Lab.

Last but not least, I owe more than thanks to my family, especially my mother, for their support and encouragement during my graduate study at Northeastern University and their love throughout my life — without them, none of this would be possible.

Abstract of the Thesis

Clothing Fashion Style Recommendation System

by

Wei Dai

Master of Science in Electrical and Computer Engineering

Northeastern University, May 2015

Dr. Yun Fu, Adviser

This thesis proposes a clothing recommendation system that can recommend clothing images based on the fashion style of the provided clothing images. In this work, we focus on the images of upper body clothing and with human model in the images.

In the first part, we present a clothing dataset collected from Internet containing 27,375 men's and women's clothing images of 11 clothing categories. We develop a recommendation system that can differentiate fashion categories of query images. We propose a framework that divides the system into three decoupled and autonomous components in order to provide a highly flexible and extensible system. Then we describe an implementation of this framework on a Linux server. To demonstrate this clothing recommendation system we also develop two user interfaces, including a Web Application and an iOS App. Lastly, we discuss the approaches to secure the system and user privacy.

We set up a Demo of this clothing recommendation system running on iPhone, which can achieve promising results within 5 seconds.

Chapter 1

Introduction

Sometimes we see someone on the street wearing the clothes which looks good and appealing. We may want to find more clothing with similar fashion style or even more to find out where we can purchase them. This thesis provides a clothing recommendation system, people can use their iPhone take a photo of the clothes then search it using the App we developed. The recommendation results including 12 clothing images with similar fashion style and the fashion category of the query image (e.g. elegant). In this system we only focus on upper-body clothing.

When designing this clothing recommendation system, we want to maximize the amount of code can be reused when we extending the current system. For example, when we need the mobile App not only work on iPhone but also on Android phone, we don't want to redesign the whole system and rewrite every piece of code of current system. In addition to code reusability, we also want the system easy to maintain. In order to archive these requirements, the thesis designs the clothing recommendation system using a decoupled architecture (i.e. the Model-View-Presenter architecture pattern).

In this thesis, we first introduce the clothing image dataset we collected from Internet and clothing fashion style categories in Chapter 2. Then Chapter 3 will give an overview of the Clothing Fashion Style Recommendation System. Chapter 4 will discuss the details of how the system is designed using a decoupled architecture. Last Chapter 5 will go through the implement of the back end server of the clothing recommendation system and the security measures applied in it.

Chapter 2

Clothing Database and Clothing Fashion Style Categories

This chapter will first introduce the clothing dataset we collected and how we store it into a database. Then Chapter 2.2 will discuss the 12 fashion categories we defined for classification.

2.1 Clothing Image Dataset and Clothing Database

We collected a clothing dataset from Internet containing 27,375 women's and men's images. The images we collected are from 11 clothing categories, Table 2.1 shows the categories name and the number of images for each category.

Table 2.1: Clothing categories and the number of images for each category

Clothing Category	**Count**	**Clothing Category Name**	**Count**
Women's Dresses	8740	Women's Coats	1193
Women's Tops	6958	Women's Sweaters	2149
Women's Jackets	1478	Men's Suits	564
Men's Casual Button-Down Shirts	2157	Men's Polos	952
Men's T-Shirts	1644	Mens Sweaters & Sweatshirts	852
Men's Coats & Jackets	688		

The tools for collecting images is developed using Python. We use the HTMLParser Python package [1] to parse the source code of the HTML page then extract and store the needed

data (e.g. image Uniform Resource Locator (URL), image title, page URL, clothing category etc.) After getting all the image URLs (i.e. the links), we use bash script to automatic download all the clothing images and put them into specific folder according to the clothing category.

(a) (b) (c) (d) (e) (f)

(g) (h) (i) (j) (k)

Figure 2.1: Sample images for each clothing category. (a)Women's Dresses (b)Women's Coats (c)Women's Tops (d)Women's Sweaters (e)Women's Jackets (f)Men's Suits (g)Men's Casual Shirts (h)Men's Polos (i)Men's T-Shirts (j)Men's Sweaters (k)Men's Coats

We use a relational database — MySQL to store the image file path on the machine and all other textual information we collected along with the image. A relational database is a structured collection of data which stores data in separate but related tables instead of putting all the data in one big storeroom. [2] Figure 2.2 shows the tables and the relation between them.

We should mention that we also stored the feature vector of each image in the database, the details of feature extraction will be discussed in Chapter 4.2.3.2

2.2 Clothing Fashion Categories

Chapter 2.1 describes 11 clothing categories, while this chapter will define 12 fashion categories for clothing image. These 12 fashion categories will be used for fashion style classification.

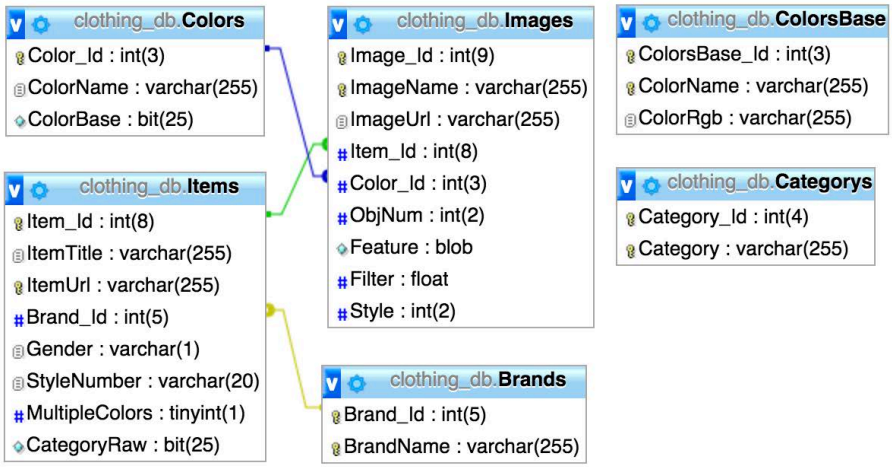


Figure 2.2: Database structure

Details of fashion style classification will be discussed in Chapter 4.2.3.3. Here is the definition for these categories:

Avant-garde: Avant-garde fashions are characterized by their abnormality. Nontraditional cuts, patterns, and/or colors. Lots of drapey, dark clothing. Abnormal proportions. Often leather and other materials with interesting textures are used.

Elegant: Traditional proportions, enhancing a traditionally attractive figure. Refined, sleek, tasteful, often dressier clothing. Often subdued color schemes, though bright colors can be considered more elegant, especially for womenswear. Includes long evening dresses for women and sleek suiting for men. Often expensive looking clothing. Not overtly sexual.

Folk: Clothing influenced by rural life and other cultures. Nature references, often less constricting cuts. Classic patterns such as plaids, simple florals, and stripes. Not urban or modern. Hippie-styled clothing falls within this category. Earth tones and bright colors, usually not subdued color schemes. Flowy layers. Mixed patterns. Worn-in clothing.

Leisurely: Relaxed, casual clothing. Athletic wear falls in this category. Can be any color. Looser cuts, comfortable looking usually. Soft or cozy materials. Not dressy. Soft textures, clothing that will not constrict movement. Clothing with sporting team logos or references usually fall within the leisurely category. Sneakers, sweatpants, sweatshirts, etc.

Modern: Minimal, trend-following clothing. Slim and skinny cuts, intermingling of casual and more formal looks. Sleek. Black and white and rich hues used. Shiny and smooth textures. Clothing can be more sexual, though typically not super suggestive. Urban, classy feel.

Neutral: Clothing that does not fit strongly into any category. Simple clothing, neutral colors, usually not patterned. Simple jeans, tees, shirts, pants, jackets, etc.

Renascent: Clothing reminiscent of earlier times, especially fashions from the first half of the 19th century. Classier clothing, not showing lots of skin, modest, etc. Similar to Romantic style but feels more dated and less embellished. Clothing with conservative cuts, colors, patterns etc. Much classical menswear will fall in this category (non-slim blazers, waistcoats, etc.)

Romantic: Traditionally flattering clothing. Womens clothing accentuates girlish figure, mens clothing emasculating. Not especially modern, more conservative and taking cues from fashions of the past. Wide hems and nipped waists for women. Cutesy/frilly things for women, lace, etc. Pastel colors especially for women, traditional neutral shades more for men.

Sexy: Overtly sexual clothing meant to make the wearer look as desirable as possible. Shows lots of skin, emphasizes legs, chest, back for women, primarily chest and arms for men. Often flashy, shiny, or attention grabbing in some other way. Bright colors or loud patterns common.

Splendid: Clothing that is very embellished, adorned, embroidered, fanciful. Often appears quite expensive. Rich colors, intricate patterns, complex clothing. Cuts can vary, but tend to be traditionally flattering. Over-the-top detailing.

Technology: Fashion influenced by technology. Things that look futuristic. Fashions made of technical materials or that incorporate new technologies into their designs. Clothing that is made to interface well with other technology (cell phones, etc.). Overly functional clothing with excessive zippers, closures, straps, etc.

Young: Clothing made to embrace youthfulness or to make the wearer appear younger. Fun clothing, things with more juvenile, trendy designs. Lots of prints and bright colors. Basic clothing like tee shirts, jeans, simple dresses. Lacks embellishment beyond colors and patterns.

Chapter 3

Clothing Fashion Style Recommendation System Overview

Given a Women or Men's clothing image, the Clothing Fashion Style Recommendation System (CFSRS) can classify the query image into 12 fashion style categories (defined in Chapter 2.2) and recommend clothing with similar fashion style. This chapter will give an overview of the CFSRS. Chapter 3.1 will take a glance at system component at software level. Then Chapter 3.2 will go over the system flowchart at logical level.

3.1 System Component

The CFSRS contains many parts, from front end to back end. In front end, CFSRS has Graphical User Interface (GUI) on both iOS App and Web Application. Because the cross-platform compatibility of Web Application, CFSRS can running on whatever platform supported HTML5, CSS and JavaScript, which means all major Operating System (OS), such as Linux, Unix, Mac OS and Windows along with Android, iOS and Windows Phone can meet the requirements to running this clothing recommendation application. While for the iPhone platform, there is a redesigned iOS App which can provide much more friendly GUI than the Web Application when using CFSRS on iPhone or iPad. Through the GUI, the user can upload query images, select search options and check recommendation results. Figure 3.1 shows the screenshot of Web Application running on Windows 8.1 using Chrome browser. In the back end, CFSRS has the Daemon program, Database server, Web server and PHP. All these programs are running on a Linux server — a physical computer.

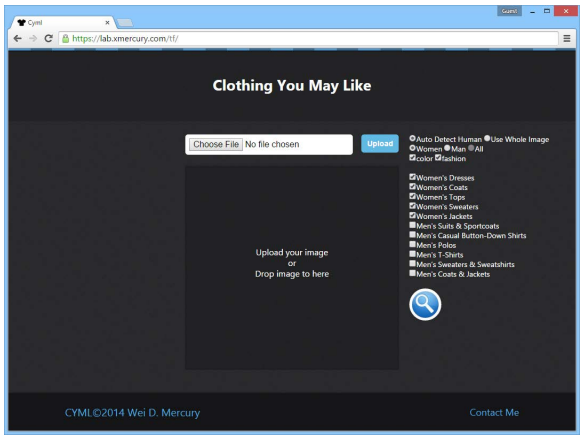


Figure 3.1: Screenshot of Web Application

In general, CFSRS has a client — the GUI and a server — the Linux server. Details of each component will be discussed in Chapter 4 from a different perspective. Then Chapter 5 will cover the Linux server.

3.2 System Flowchart

Flowchart is a diagrammatic representation of algorithm, workflow or process. It is used to design complex program and can help other people understand what is going on.

Figure 3.2 shows the flowchart of CFSRS. The flow starts at the GUI, either Web Application or iOS App, when user starts choosing query image. The query image can be taken from camera or choose from existing photos from local device. If using the iOS App, users can crop the image right after selection. Before submitting the search request, users can set the search options, such as gender, clothing categories and whether auto detect human or not.

When clicking or pressing the search button, the GUI will submit the Hypertext Transfer Protocol (HTTP) request to the PHP and the Web Server on the Linux server. The selected image along with the search options will be uploaded to Linux server via HTTP. PHP code is executed when the HTTP request is received by Web server and forwarded to the PHP. The PHP code will parse the search request from GUI, then initialize a local network socket connection with the Daemon server, search options sent by GUI are translated and enclosed in the socket.

The socket server, the Daemon server on Linux server, keeps listening the incoming socket connection. When gets connected by a socket client (i.e. the PHP code), the Daemon server will

execute the core code for image recommendation and classification. The core code will first attempt to detect human body from the given image, if do detect the human body, the bounding box of clothing area will be used as Region of Interest (ROI) for fashion features extraction, otherwise the whole image will be used. The extracted features from ROI will be used for both image ranking and fashion categories classification. Then the rank and classification results are sent back to PHP code through the same socket connection.

When socket client received, the results from socket server, the socket connection will be closed and the socket server will back to it's listening mode. Eventually the results are sent back to GUI using JavaScript Object Notation (JSON) format as the callback of HTTP request sent by GUI earlier, then the GUI parse the JSON data and show it.

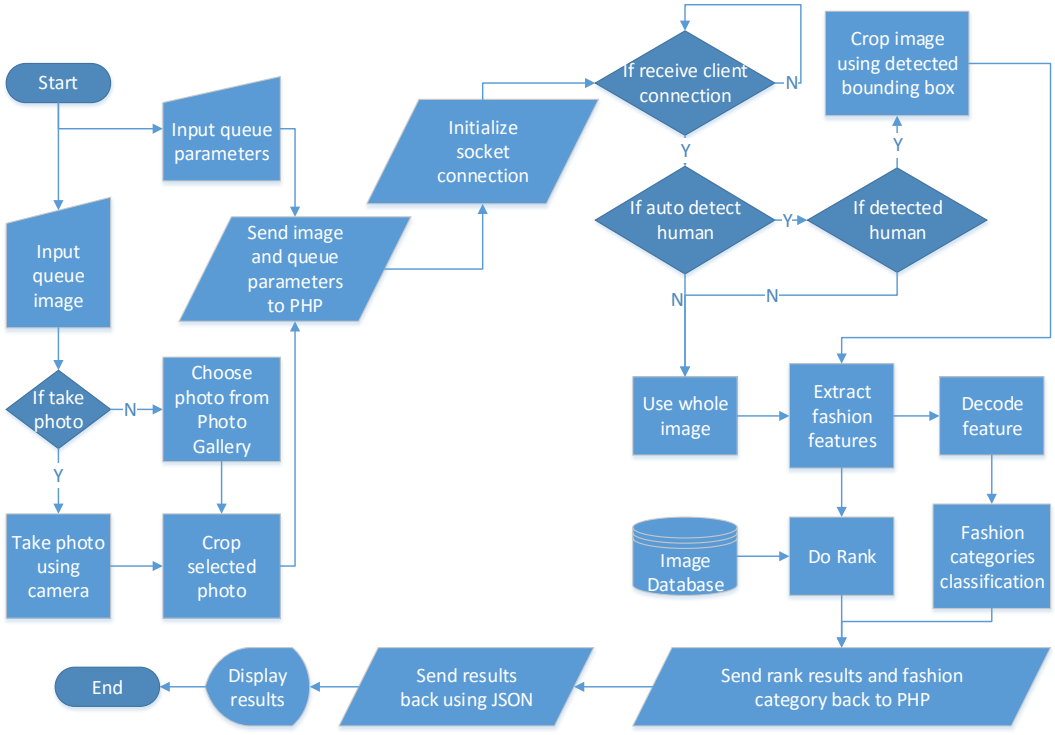


Figure 3.2: System Flowchart

Chapter 4

Model-View-Presenter Architecture Pattern in CFSRS

Last chapter gives an overview of the CFSRS, while this chapter will go over the details of CFSRS at the software architecture level. Software architecture is the high level structure of a software system. Unlike system component only contains software element and system flowchart only represents the software logical, the software architecture is a higher level abstraction of the software system, which can reveal the high level relation between the elements in the system. Chapter 4.1 will discuss the software architecture pattern used in CFSRS.

4.1 Model-View-Controller and Model-View-Presenter

4.1.1 Model-View-Controller Architecture Pattern

MVC is one of the software architecture pattern. It divides the given application into three interconnected components — the model, the view and the controller. The model is the central component MVC[3], it directly accesses the database, manages the data and implements the logic of the application. The view is displaying component to output data and information. Most times, the view is the component that display the application's GUI. The controller handle the user interaction and send command to model and view.

Figure 4.1 shows a typical MVC implementation. The three components in MVC are interconnected and have interactions between each other.[4] The view provides the interface for user input and hands over to controller instead of processing it. The view can also query data from model.

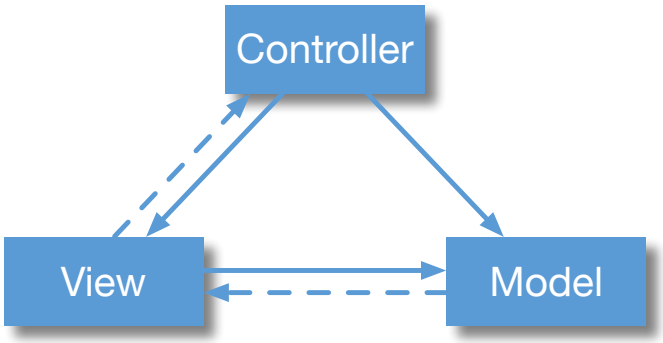


Figure 4.1: Components of MVC and Their Interactions. The solid line represents a direct association, the dashed line represents indirect association.

The controller processes user input then sends commands to the model to change data or the view to update view. The model will response the data query from view and implement the data change command from controller. The model can also notify the view when data is changed.

4.1.2 Model-View-Presenter Architecture Pattern

MVP is another software architecture pattern which is a derivation of traditional MVC. MVP also contains three components — the model, the view and the presenter. The major different between traditional MVC and MVP is that inMVP model and view do not communicate directly, which means model and view is completely decoupled.[5] In MVP, presenter is the component of intermediary between the model and the view. All display logic is handled by presenter.[6]

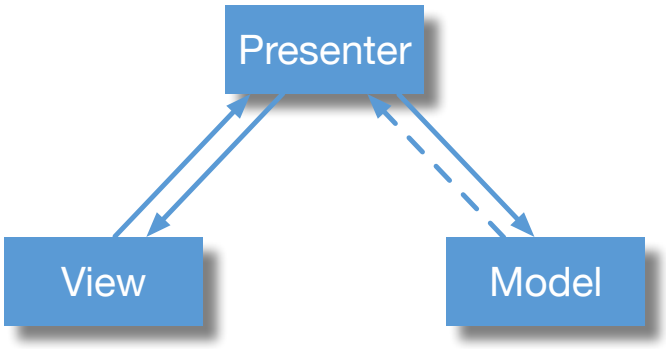


Figure 4.2: Components of MVP and Their Interactions

Figure 4.2 shows the relation between model, view and presenter in MVP. Similar to

MVC, the view hands over the user input to presenter. However, the view can only been updated by presenter. The presenter handles and processes the user input, then sends corresponding commands to to change or require data from model.

The CFSRS is designed using MVP architecture pattern. The model in CFSRS is the Daemon server along with all other supporting services (e.g. Database). CFSRS has two different views: the Web Application GUI and the iOS App GUI. The last part, the presenter, is the PHP code which act as a intermediary between the GUI and the Daemon server. Figure 4.3 shows how MVP applies in CFSRS. As can be seen in the figure, there is no direct connection between the CFSRS GUI and the Daemon server.

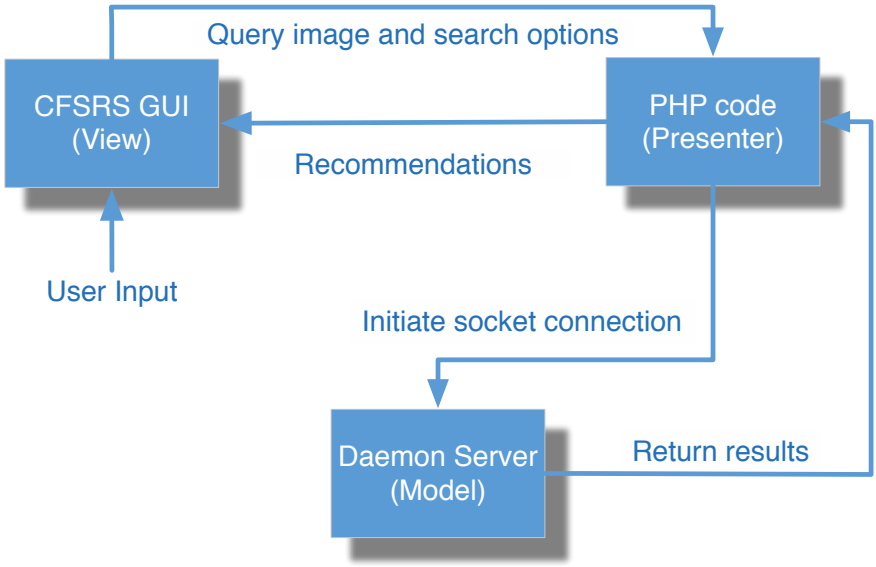


Figure 4.3: MVP Architecture Pattern in CFSRS

There are lots of advantages implementing the MVP architecture pattern in CFSRS. First, MVP can support multiple views while using same model and presenter, and also when update model, views and presenter don't need to change. Thus lots of code can be reused when only modify views or model. Second, because of the model and the view is completely decoupled in MVP, the GUI development and algorithm development can be separate and there is no need to consider the other one. Third, the MVP architecture pattern improves maintainability and make the code easy to test.

The following chapters in Chapter 4 will go through details of the model, the view and the presenter in CFSRS.

4.2 The Model — Daemon Program

In OS, a Daemon is a process (i.e. computer program) that runs in background, rather than being under the direct control of an interactive user.[7] Most of the daemons are started by system at boot time. For example, there is a *sshd* daemon program running on the university Linux sever, which provide services for incoming SSH connections. so student can login via SSH client on their own computer.

4.2.1 Socket Server in Daemon

The CFSRS Daemon program is similar to *sshd* daemon program. It is a C++ program being configured to start at boot time, running at background and listening to incoming socket connections (from PHP code). The concept of running in the background is implemented using network socket.

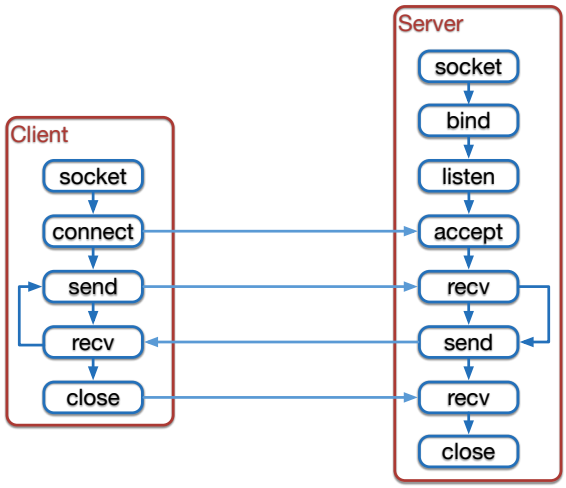


Figure 4.4: Socket Communicating Process

A network socket is an endpoint of a two-way communication link between two programs running on the network. An endpoint is a combination of an IP address and a port number.[8] For the socket server, after bounding to a specific IP address and port number, it just waits, listening to the incoming socket for a client connection request. On the client side, the client using the server's IP address and given port to initiate the connection request. If accepted by server, the socket connection is established, then the client and server can communicate by writing to or reading from their sockets. The connected server and client are called a socket pair. [9] Socket connection is closed when client

send a close message to the server. The communicating process between socket server and socket client is shown in Figure 4.4.

As for the socket connection in CFSRS, the socket server and client are on the same Linux server. So the socket connection between them is a local connection, which means the socket server and client have the same IP address (i.e. *localhost* or 127.0.0.1) but have different port number. It is worth mentioning that CFSRS doesn't use Unix domain socket or Interprocess Communication (IPC) socket, which is an endpoint of communication between two processes running on the same machine.[10] The Unix domain socket seems more suitable for current CFSRS, because the socket client (is also the PHP code, the presenter), and socket server (is also the Daemon program, the model) are on the same machine. However, the model and the presenter are not necessary to be on the same Linux serve. Sometimes the hardware and software requirements for model and presenter may be different and are not suitable and economic to put them on the same machine. For example, the model of CFSRS performs image precessing, which sometimes requires large memory, while the presenter handles the user interaction, which will require multiple CPU cores and high performance Network Interface Controller (NIC) when we have large number of users. Thus, using network socket instead of IPC socket retains the ability to put the PHP code and the Daemon program onto two different Linux servers — one focus on computing, another focus on handling multi-user HTTP request. This will be an important step when taking the number of users and system payload into account.

Table 4.1: Information enclosed in socket communication

Index	**Information**	**Direction**
1	Client IP address	client $\rightarrow$ server
2	Query image	client $\rightarrow$ server
3	Clothing categories	client $\rightarrow$ server
4	Bounding box of clothing area	client $\leftarrow$ server
5	Fashion category of query image	client $\leftarrow$ server
6	Top 12 image Recommendation results	client $\leftarrow$ server

Table 4.1 lists the information transported between Daemon program and PHP code. The transmission take place in the shown order, after received first three rows from PHP code, the Daemon program will pause the socket transmission and execute the image recommendation and classification core code. "Pause" means the socket connection is still there, while the socket server

doesn't send anything back (yet) to client. As soon as the Daemon program get the recommendation and classification results, it will send them back (row 4, 5 and 6 in table) to the PHP code — the socket transmission resumed.

4.2.2 Core Code in Daemon

Chapter 4.2.1 explains how the Daemon program of CFSRS works as a network socket server. This chapter will discuss the details when it "pause" the socket transmission.

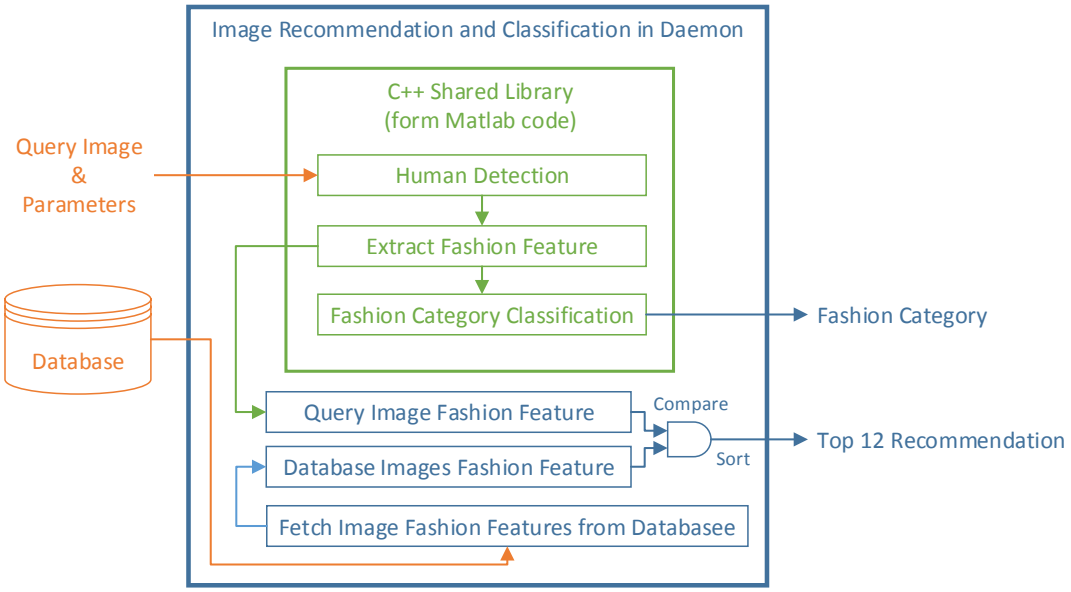


Figure 4.5: Structure of image recommendation and classification core code

Figure 4.5 shows the input, output and internal logical of the image recommendation and classification core code in Daemon program. As shown in figure, input of core code are the query image and parameters received from socket. Another input is the database which stored the fashion features extract from the clothing dataset collected from Internet. This is the only place that can access database, which conform the MVP design in CFSRS.

The green rectangle in the figure represents a C++ shared library (is known as a *.so* file on Linux or a *.dll* file on Windows) generated by Matlab code. Matlab programming language is good for matrix computations and image processing; however, pure Matlab code is hard to run as a Daemon program and access database. So in CFSRS, we first developed Matlab code to do the

image processing, then compile the Matlab code into a C++ shared library which can be called by the Daemon C++ program. In this way, the input query image and parameters is hand over to Matlab C++ shared library directly, after processing and computing, the shared library return the fashion category and the query image fashion features.

The returned query image fashion features will be used to compare with the database image features by the C++ code. The Daemon C++ program will fetch a subset of database image fashion features based on the parameters from the presenter (i.e. clothing categories), then compare each image fashion features from the subset with the query image fashion features.

The compare results is measure by cosine similarity of the query image fashion features and database image fashion features. Let f_q denote the fashion features of the query image, and f_i denote the i-th image fashion features in the database. The similarity between f_q and f_i is:

$$s_{i,q} = \frac{(f_i \cdot f_q)}{||f_i|| \cdot ||f_q||}. \tag{4.1}$$

After sorting the compare results $s_{i,q}$ of each i in the selected subset, the top 12 ranking image is send back to socket client along with the fashion category. Chapter 4.2.3 will cove the Matlab program within the C++ shared library.

As for connection the database, the Daemon program of CFSRS using the MySQL Connector/C++, which is a MySQL database connector for C++ provided by Oracle.[11]

4.2.3 C++ Shared Library in Daemon

Matlab provides a toolbox *MATLAB Compiler* to compile the Matlab program into a standalone application or a C++ shared library. [12] When using *MATLAB Compiler* to generate C++ shared library, it works like a wrapper for Matlab program, which can be called as functions in C++ program. This chapter will discuss the original Matlab program in CFSRS (before compiled into a C++ shared library).

4.2.3.1 Human Detection and Clothing Area Bounding Box Extraction

The human detection and clothing area bounding box extraction is based on the Matlab program[13] developed by the participants of Pattern Analysis, Statistical Modelling and Computational Learning (PASCAL) Visual Object Classes (VOC) [14][15] — discriminatively trained deformable part models. [16]

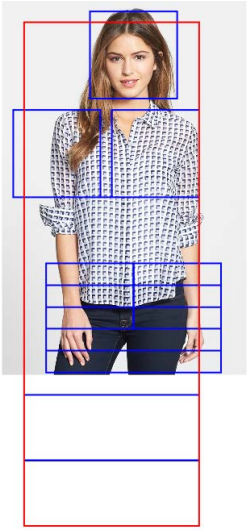


Figure 4.6: Detection results of PASCAL VOC program

Figure 4.6 shows the detection results of this part base human detection program. As shown in the image, there is a large red bounding box indicating the contour of detected human body and lots of small blue bounding boxes indicating the detected parts of the human body (e.g. hand, head, shoulder etc.). There are some or part of bounding boxes out of the image, because the part base human detection program including a bounding boxes prediction algorithm for images with only part of the body. [17] However in CFSRS, the bounding box of clothing area is more important than the bounding box of the whole human body. Thus, we extend the parts base human detection code to output clothing (upper-body) area bounding box using the detected body parts from shoulder to waist. Figure 4.7 shows the clothing area bounding box generated by the extended parts base human detection program. Using the clothing area bounding box instead of the whole human body bounding box can help reduce noise when doing the image ranking for clothing.

4.2.3.2 Fashion Feature Extraction

After getting the clothing area bounding box, we split it into 4×2 boxes (4 rows and 2 columns) evenly. Then for each box, we extract 7 dense features — RGB color value [18], Lab color value, HSI color value, Gabor, MR8 texture response [19], HOG descriptor [20], and the probability of pixels belonging to skin [21].

Now we get eight patches for each image. In each patch, first we use mean-std pooling of these features, then concatenate all pooled features for 8 patches together for whole body model as

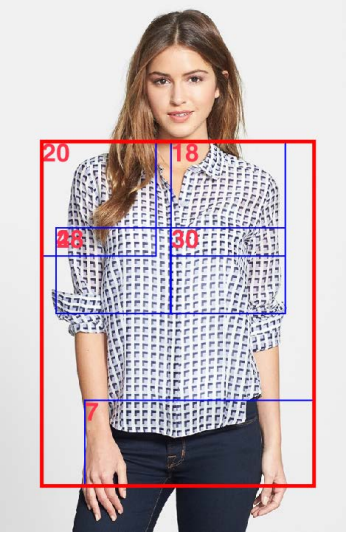


Figure 4.7: Detection results with clothing area bounding box

the style descriptors. The total dimension of the style descriptors of each image is 5312.

4.2.3.3 Fashion Style Classification

From the collected clothing image dataset mentioned in Chapter 2.1, we randomly select 2222 images from 11 clothing categories (number of images for each clothing category is even). Then we invite 7 volunteers to individually label those 2222 images using the 12 fashion categories mentioned in Chapter 2.2.

After labeling we want to merge the labeling results of 7 people into one. For each image i, we counted the number of people labeling this image to category j, $j \in \{1, ..., 12\}$. Let $\phi_i^{(j)}$ denote the number of people labeled image i into category j. We choose $j = \arg\max_j \phi_i^{(j)}$ as the final category for image i.

Then we use these 2222 labeled images as the training data to classify the query image using k-Nearest Neighbors (KNN).

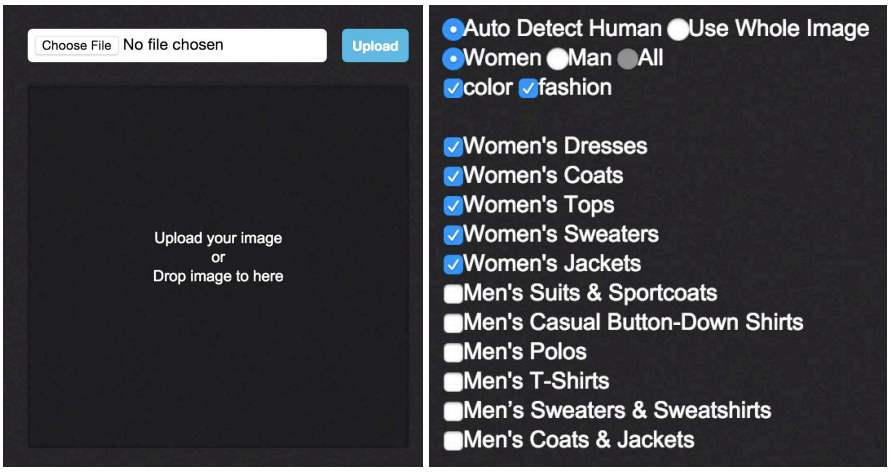
4.3 The View — Web Application and iOS App

As introduced in Chapter 4.1.2, the responsibility of the view is simply provide a GUI to accept input and present output. In CFSRS, we first developed a Web Application which can run on most of Internet platform, such as computer, tablet or even mobile phone. Then for iPhone, we

developed a native iOS App which can provide more user friendly GUI for small size. This chapter will go over the design of Web Application and iOS App.

4.3.1 Web Application

The Web GUI of CFSRS using HTML form [22] to let user upload query image and set search options. For uploading, user can either click the "upload" button to choose image file, or directly drag & drop the query image into the center box. Figure 4.8 shows the HTTP form in the Web Application.



(a) HTML form for image upload (b) HTML form for search options

Figure 4.8: HTML form in Web Application

After clicking "Search" to submit the HTTP form, a popup lightbox will shown with a "Searching..." animation. (Figure 4.9)

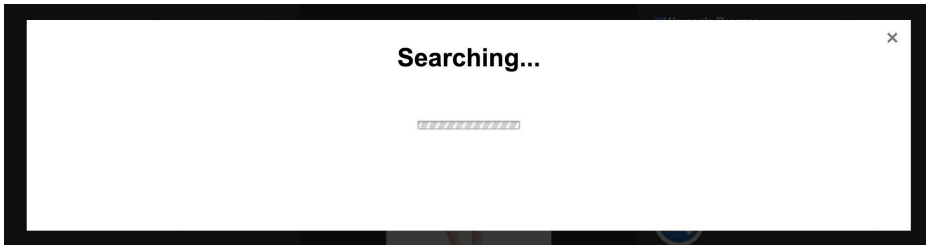


Figure 4.9: The animation during search

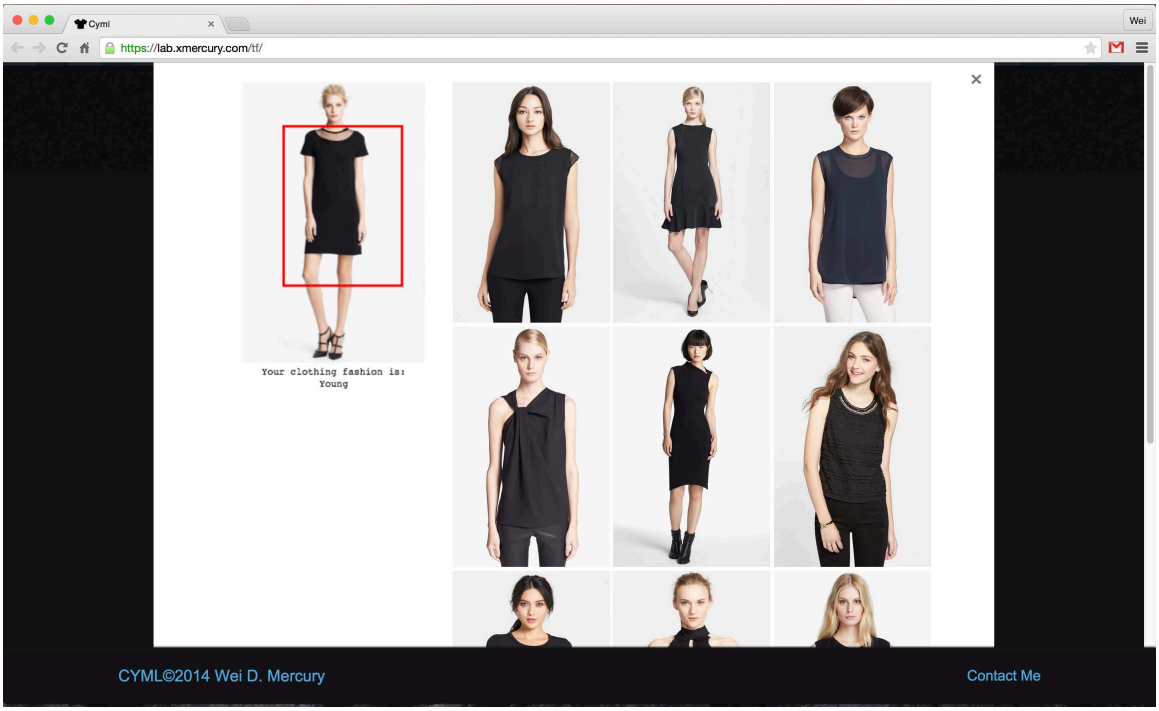


Figure 4.10: Search results in shown popup lightbox

After around 5 seconds, the fashion style recommendation results will be shown in the same popup lightbox and replace the "Searching..." animation. As shown in Figure 4.10, the left image with red bounding box on it is the query image. The bounding box indicate the detected clothing area. The fashion style classification results is shown below the query image. The query image in Figure 4.10 has a *Yang* fashion style. The images on the right side are recommended clothing with similar style.

4.3.2 iOS App

For iPhone (and other mobile phone), because the small screen size, the Web Application present in Chapter 4.3.1 is not quite user friendly. So we develop an iOS App — another view of CFSRS, to provide better user experience on iPhone.

This iOS App provides same functionality with the Web Application, because they are both "the view". It also use the HTTP request to upload query image and submit search options to the PHP code.

4.4 The Presenter — PHP code

The PHP code is a "middle man" between the GUI and the Daemon program. How PHP code communicate with the GUI and Daemon program has already been discussed in Chapter 4.2.1 and Chapter 4.3 respectively (i.e. via HTTP form and via socket). In this section we only focus on what will PHP code do after connected by the GUI and before it initiating the socket connection with the Daemon program.

When received a HTTP form submission from the GUI, it will first validation the input, then rename the query image filename and put it into specific folder. The query image is read and processed by the model, however the PHP code response for provide service to upload the query image from user device to the CFSRS Linux server and the PHP code should provide the model the file path of the query image. The PHP code use the MD5 hash value of current time plus a random integer to rename the uploaded image (e.g. *095455722f4fba8eb7db9a44f98ed34f.jpg*). In this way, we can avoid the issue of duplicate filename. Now the PHP code is ready to initiate the socket connection.

Chapter 5

Back End Linux Server Configuration

This chapter will talk about the back end Linux Server. Chapter 5.1 will first introduce the Linux server environment and configuration in CFSRS, then Chapter 5.2 will discuss the security issue within the system.

5.1 Operating System and Service

5.1.1 Operating System Environment

There are lots of Linux distributions, the one CFSRS used is Red Hat Enterprise Linux (RHEL) [23] developed by Red Hat. Unlike other popular Linux distributions such as Ubuntu, Fedora and Mint, RHEL is famous for its stability and target toward the commercial market, which is well suited for CFSRS back end server. Table 5.1 shows the hardware and software environment of the CFSRS Linux server.

Table 5.1: System specification

OS	Red Hat Enterprise Linux 6 64-bit
CPU	Intel i7-3770 (Quad Core, 3.40GHz, 8MB Cache)
Memory	32GB DDR3 1600MHz
HDD	WD 1TB 7200RPM 64MB Cache SATA 6Gb/s
NIC	Intel Gigabit LAN Controller

From the specification, we can find that most hardware is designed for desktop computer and not ideal for high payload server usage. However for this thesis we only setup a Demo, and not

publish it to thousands of users, so current specification is sufficient for development and demonstrate. If needed, we can easily move it to more powerful server or even cloud server in the future. In addition, within current system specification, the CFSRS can return the search results within 5 seconds. If using more powerful server, the CFSRS recommendation results even faster.

5.1.2 Supporting Service on Linux Server

There are lots of supporting service for CFSRS, for example it need a web server to process HTTP request. "Service" means a Linux program running in background [24], so it is an alias of name Daemon, while because Daemon program provides a <*service*>.service file that contain information about how and when to start the associated daemon [25], people tend to use word "service" to refer a Linux background program instead of "Daemon". In Chapter 4.2, we introduced the Daemon program developed for CFSRS, while this chapter will introduce the supporting Daemons (i.e. services) running on CFSRS which shipped by Linux OS or third-party companies.

Table 5.2: Supporting service on Linux server

Service	**Program**	**Description**
httpd	Nginx	An HTTP and reverse proxy server, as well as a mail proxy server [26]
PHP	PHP-FPM	A simple and robust FastCGI Process Manager for PHP [27]
mysqld	MariaDB	A robust, scalable, and reliable SQL server [28]
sshd	OpenSSH	SSH protocol utilities provide secure network communications [29]
ftpd	vsftpd	A lightweight, stable and secure FTP server for UNIX-like systems [30]

Table 5.2 shows the related services on CFSRS Linux server. First we have the *httpd* service, *httpd* stands for Hypertext Transfer Protocol Daemon (i.e. Web server). There are lots of programs can be used as Web server, the one we use in CFSRS is Nginx. Web server provide service to handle HTTP request, while in CFSRS it provides service for Web Application and the presenter (i.e. PHP code). PHP-FPM is a program to execute PHP code. When Nginx found the HTTP request is ponit to a ".php" file, it will calling PHP-FPM to execute the PHP code in the ".php" file. In CFSRS, PHP-FPM provides service exclusive for the presenter. For Structured Query Language (SQL) server (i.e. database server), we use MariaDB. SQL server provides service to let SQL client to access and query data from database. In CFSRS, the SQL client is the model (i.e. Daemon program), so the SQL server provides service to the model. Secure Shell (SSH) Daemon and File Transfer Protocol (FTP) Daemon are mainly used to maintain the Linux Server.

5.2 Server Security

As long as the server expose to Internet, attacks will come from all over the world every day, every hour or even every minute. This chapter will introduce the security measures in CFSRS. Chapter 5.2.1 will discuss how to secure the OS, then Chapter 5.2.2 will talk about how to secure user data and information.

5.2.1 Firewall and Packet Filtering

Firewall is a hardware and/or software that controls the Internet traffic (incoming and outgoing) based on the predefined rules. [31] In CFSRS, we use *iptables* to setup the firewall rules. *iptables* is a command line program used to configure the Linux kernel packet filtering ruleset, it can filter the network packet using IP address, port number or protocol. [32] In general, there are two paths to setup the firewall rules, one is deny specific packets then allow all others (allow-by-default), another is deny everything but specific packets (deny by default). In CFSRS, we want to maximum the security, so the deny-by-default policy is applied. Table 5.3 shows the allowed port number and corresponding service for incoming connection on CFSRS Linux server. All other incoming connections will be dropped.

Table 5.3: Allowed ports and services

Port	**Service**
21	FTP
22	SSH
80	HTTP
443	HTTPS

It is worth mentioning that the incoming connection for SQL server and socket server are prohibited, because SQL client and socket client is on the same machine, and those connections are local connection and do not go through Internet.

5.2.2 User Privacy and Security

The CFSRS not only consider the security of OS but also the security and privacy of user data and information. HTTP (i.e. *http://*) is vulnerable to Man-in-the-middle (MITM) attack [33], which the attacker insets him/herself between the connection between user and sever, then

controls the entire communication. In CFSRS, user will send images to Linux server, so it is our responsibility to secure user's data during the transmission. CFSRS use HTTP Over TLS (i.e. *https://*) to secure the communications with Linux server. HTTPS layers the HTTP on top of Transport Layer Security (TLS) protocol to enhance the security of standard HTTP communications. [34]

Chapter 6

Conclusion

Using a mobile phone App, people can easily take of photo of the appealing clothes they saw on magazine, web page or even street, then get the recommended clothing with similar fashion and style in seconds. People can even directly link to the online shopping website to purchase if they like it. When people find a clothes they like but don't know where to buy it or how to find more similar clothing, the Clothing Fashion Style Recommendation System provides a convenient way to help find that.

What's more, designed under the concept of Model-View-Presenter, the Clothing Fashion Style Recommendation System provides a highly flexible and extensible framework. For example, the GUI can easily extended to Android or Windows Phone platform and do not need to rewrite the logic and algorithm parts. Also, if we want to significant improve the system in the future, we can let people who good at programming and aesthetic designing work on improve the view (i.e. the GUI), and let others who good at algorithm and researching work on the model (i.e. the Matlab code). In this way, people who develop GUI don't need to know the underlying research and algorithm, and the one who develop the model don't need know the programming for GUI, and also don't need redesign or rewrite their code accordingly if the view changed or added a new view. Thus, this is a well-designed framework for long term maintaining and upgrading.

Bibliography

[1] *HTMLParser Simple HTML and XHTML parser*, Python Software Foundation, 2015. [Online]. Available: https://docs.python.org/2/library/htmlparser.html

[2] *MySQL 5.0 Reference Manual — What is MySQL?*, Oracle Corporation, 2015. [Online]. Available: https://dev.mysql.com/doc/refman/5.0/en/what-is-mysql.html

[3] S. Burbeck, "Applications programming in smalltalk-80(tm): How to use model-view-controller (mvc)," 1987. [Online]. Available: http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html

[4] D. Schmidt, M. Stal, H. Rohnert, and F. Buschmann, *Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects*, ser. Pattern-Oriented Software Architecture. Wiley, 2013. [Online]. Available: https://books.google.com/books?id=rYiKY3mrrswC

[5] K. Roebuck, *Multitenancy: High-impact Strategies - What You Need to Know: Definitions, Adoptions, Impact, Benefits, Maturity, Vendors*. Emereo Publishing, 2012. [Online]. Available: https://books.google.com/books?id=dzEQBwAAQBAJ

[6] M. Potel, "MVP: Model-View-Presenter The Taligent Programming Model for C++ and Java," Taligent, Inc., Tech. Rep., 1996. [Online]. Available: http://www.wildcrest.com/Potel/Portfolio/mvp.pdf

[7] M. Kerrisk, *The Linux Programming Interface*, ser. No Starch Press Series. No Starch Press, 2010. [Online]. Available: http://books.google.com/books?id=5J9wmAEACAAJ

[8] *The Java™ Tutorials: What Is a Socket?*, Oracle, 2015. [Online]. Available: https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html

[9] W. Stevens, B. Fenner, and A. Rudoff, *UNIX Network Programming*, ser. Addison-Wesley professional computing series. Addison-Wesley, 2004, no. v. 1. [Online]. Available: http://books.google.com/books?id=ptSC4LpwGA0C

[10] W. Stevens and S. Rago, *Advanced Programming in the UNIX Environment*, ser. Addison-Wesley Professional Computing Series. Pearson Education, 2013. [Online]. Available: https://books.google.com/books?id=kCTMFpEcIOwC

[11] *MySQL Connector/C++ Developer Guide*, Oracle Corporation, 2015. [Online]. Available: https://dev.mysql.com/doc/connector-cpp/en/index.html

[12] *MATLAB® Compiler™ User's Guide*, The MathWorks, Inc., 2015. [Online]. Available: http://www.mathworks.com/help/releases/R2015a/pdf_doc/compiler/compiler.pdf

[13] R. B. Girshick, P. F. Felzenszwalb, and D. McAllester, "Discriminatively trained deformable part models, release 5." [Online]. Available: http://people.cs.uchicago.edu/~rbg/latent-release5/

[14] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, Jun. 2010.

[15] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn *et al.*, "The PASCAL Visual Object Classes Challenge 2010 (VOC2010) Results." [Online]. Available: http://www.pascal-network.org/challenges/VOC/voc2010/workshop/index.htm

[16] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part based models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 9, pp. 1627–1645, 2010.

[17] R. Girshick, P. Felzenszwalb, and D. McAllester, "Object detection with grammar models," in *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, 2011.

[18] D. Pascale, "A review of RGB color spaces ... from xyY to R'G'B'," The BabelColor Company, Tech. Rep., 2003. [Online]. Available: http://www.babelcolor.com/download/A%20review%20of%20RGB%20color%20spaces.pdf

[19] M. Varma and A. Zisserman, "A statistical approach to texture classification from single images," *International Journal of Computer Vision*, vol. 62, no. 1-2, pp. 61–81, 2005.

[20] N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 1. IEEE, 2005, pp. 886–893.

[21] G. Jain, “Detects skin by producing a map of ”skin-like” pixels within a given image,” 2010. [Online]. Available: http://kr.mathworks.com/matlabcentral/fileexchange/28565-skin-detection

[22] *HTML5 — A vocabulary and associated APIs for HTML and XHTML*, W3C, 2014. [Online]. Available: http://www.w3.org/TR/html5/forms.html#forms

[23] *Red Hat Enterprise Linux 6 — 6.6 Release Notes*, 6th ed., Red Hat, Inc., 2015. [Online]. Available: https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/pdf/6.6_Release_Notes/Red_Hat_Enterprise_Linux-6-6.6_Release_Notes-en-US.pdf

[24] Linux.com, “An introduction to services, runlevels, and rc.d scripts,” 2006. [Online]. Available: https://www.linux.com/news/enterprise/systems-management/8116-an-introduction-to-services-runlevels-and-rcd-scripts

[25] *Arch Linux Documentation — Daemons*, Arch Linux, 2015. [Online]. Available: https://wiki.archlinux.org/index.php/Daemons

[26] *Nginx Documentation*, Nginx, Inc., 2015. [Online]. Available: http://nginx.org/en/docs/

[27] *PHP Manual*, The PHP Group, 2015. [Online]. Available: http://php.net/manual/en/index.php

[28] *MariaDB Documentation*, MariaDB Corporation, 2015. [Online]. Available: https://mariadb.com/kb/en/mariadb/documentation/

[29] *OpenSSH Manual*, OpenBSD, 2015. [Online]. Available: http://www.openssh.com/manual.html

[30] C. Evans, *Very Secure FTP Daemon*, 2012. [Online]. Available: https://wiki.archlinux.org/index.php/Very_Secure_FTP_Daemon

[31] M. Curtin and M. J. Ranum, “Internet firewalls: Frequently asked questions,” 2000. [Online]. Available: http://www.faqs.org/faqs/firewalls-faq/

[32] H. Welte, *netfilter/iptables FAQ*, Netfilter Core Team, 2007. [Online]. Available: http://www.netfilter.org/documentation/FAQ/netfilter-faq.html

[33] A. Ornaghi and M. Valleri, "Blackhat conference — man in the middle attacks deoms," 2003. [Online]. Available: https://www.blackhat.com/presentations/bh-usa-03/bh-us-03-ornaghi-valleri.pdf

[34] E. Rescorla, *RFC 2818 — HTTP Over TLS*, The Internet Society, 2000. [Online]. Available: http://tools.ietf.org/html/rfc2818