

# Student ID card Barcode Recognition for Android Mobile Phone Project Report



**Student Name:** Long Long

**Student ID:** C00131028

**Supervisor:** Christophe Meudec

**Date:** 16 Apr 2010

## Contents

Introduction .....	3
1. Problems Encountered and Solutions .....	3
1.1 Problems .....	3
1.2 Solutions.....	3
2. What is achieved.....	7
3. What is not achieved.....	8
3.1 A red scan line in the middle of the camera preview frame .....	8
3.2 Log of previous recognized barcode .....	8
3.3 Management system for web server .....	8
4. What I learned.....	9
5. What would do differently .....	9
6. Updates to earlier reports.....	10
6.1 Research.....	10
6.2 Design Manual .....	12
6.2.1 GUI.....	12
6.2.2 Domain Model .....	15
6.2.3 Class Diagram.....	16
6.2.4 Web server .....	17
7. Module description .....	19
8. Testing .....	22
8.1 Functional testing.....	22
8.2 Reliability testing.....	27
8.3 Recognize time testing .....	28
9. Conclusion.....	29
Reference .....	30

# Introduction

This project report makes a further introduction about the entire project, including problems encountered and solutions, what is achieved and what is not achieved, project modules, testing and some updates to previous manuals.

There are some sample barcode images in testing part, please wait a second if this file loading a bit slowly.

## 1. Problems Encountered and Solutions

### 1.1 Problems

1. Cannot capture image data from camera
2. Auto focus works only for the first time
3. Cannot do grey scaling for the captured image data properly
4. Hard to find a threshold value work for every condition(different luminance condition)
5. Hard to find a perfect edge trimming algorithm to get barcode region
6. Image gridding problem

### 1.2 Solutions

1. Cannot capture image data from camera

Looking up Android develop guide[1] and API documents[2], I found the way to capture image from camera is to add a `Camera.PictureCallback` for camera. Syntax like:

```
mCamera.takePicture(null, pictureCallback, null);
```

(Add pictureCallback for mCamera)

For pictureCallback, we should new a `Camera.PictureCallback` and override the function `onPictureTaken`, and then we can do image pre-processing inside this function.

Syntax:

```

PictureCallback pictureCallback = new PictureCallback() {
    @Override
    public void onPictureTaken(byte[] data, Camera camera) {
        // now we can get image here just by reading the argument (byte[] data)
        // and do image pre-processing below as we wanted
    }
};

```

(Initialize a PictureCallback)

However, this method is for capturing one single image, not for capturing a continuous image data stream. By the guide and API doc, I found the second way to capture images, and it does capture a continuous image data stream, so that I can do scanning and pre-process image at real time instead of capturing just one single image at a time and pre-process it after.

Syntax:

```
mCamera.setOneShotPreviewCallback(previewCallback);
```

(Add previewCallback for mCamera)

```

private final Camera.PreviewCallback previewCallback = new Camera.PreviewCallback()
    public void onPreviewFrame(byte[] data, Camera camera) {
        // do pre-processing and decoding below
    }
};

```

(Initialize a previewCallback)

## 2. Auto focus works only for the first time

The way I did before was adding auto focus one line before my `takePicture` function, so this leads the problem that it works only for the first time. If we request taking picture but cancel it(no picture taking) and request taking picture again, the auto focus will not appear.

Looking at ZXing's project Barcode Scanner[3], I recognized one method to achieve continuous auto focus is to send the request to the system message queue and once the request is implemented we apply another, so in this way it could simulate a continuous auto focus.

I've used this method for requesting previewCallback, so the program could get both auto focus and preview working continuously.

## 3. Cannot do grey scaling for the captured image data properly

Because the pixel format of image data captured from `previewCallback` is not RGB format. The methods I've tried to do grey scaling before are all for RGB format. I checked Android Developer API document and found the pixel format from `previewCallback` is YCbCr defaulting as YCbCr\_420\_SP which uses the NV21 encoding format.[4]

YCbCr\_420\_SP(NV21) is a sub type of YCbCr format. In YCbCr format, Y is the luma component and Cb and Cr are the blue-difference and red-difference chroma components.[5]

YCbCr\_420\_SP(NV21) description:

“These are two-plane versions of the YUV 4:2:0 format. The three components are separated into two sub-images or planes. The Y plane is first. The Y plane has one byte per pixel. For V4L2\_PIX\_FMT\_NV12, a combined CbCr plane immediately follows the Y plane in memory. The CbCr plane is the same width, in bytes, as the Y plane (and of the image), but is half as tall in pixels. Each CbCr pair belongs to four pixels. For example, Cb<sub>0</sub>/Cr<sub>0</sub> belongs to Y'<sub>00</sub>, Y'<sub>01</sub>, Y'<sub>10</sub>, Y'<sub>11</sub>. V4L2\_PIX\_FMT\_NV21 is the same except the Cb and Cr bytes are swapped, the CrCb plane starts with a Cr byte.

If the Y plane has pad bytes after each row, then the CbCr plane has as many pad bytes after its rows.”[6]

So to do grey scaling, we read the first 2/3 of the image data and that's the grey value for this piece of image. Therefore, we could do thresholding afterwards.

4. Hard to find a particular threshold value work for all conditions(different luminance condition)

Different light condition has different average grey value for thresholding. For example, threshold value for image captured outside is generally bigger than the one captured inside; it for image captured in daytime is generally bigger than the one captured in the evening.

So to determine a particular threshold value for all conditions is difficult and would probably receive bad thresholding result. Then I change to build a function called `calThresholdValue` that produce threshold values for each piece of image. It calculates the number of “black” pixels and “white” pixels, and looks for a value between them but a little closer to the “white” pixels. (At the beginning, I use the middle value between them, but sometimes it won't work. So everytime I move it a little bit and finally find a good one. It's the least pixel that next to the “white” value).

5. Hard to find a perfect edge trimming algorithm to get barcode region

I tried many edge trimming algorithms to detect barcode area. Their ideas are more or less the same. They are about looking at black and white pixels. One of them is checking the scale of numbers of black and it of white pixels in the rectangular box. If the scale is between a particular value and another, the area inside the rectangular box is treated as barcode region; another algorithm is that, calculate the number of non-black-or-white pixels, if it is less than a particular value, this area would be treated as barcode region.(Because barcodes are all constructed by black colour and white colour).

But most of them do not work very well. Additionally, it costs too much resource. To mobile devices it is a big problem. For these reasons, I decided to throw this part away from the project.

6. Image gridding problem

What image gridding does is to try to convert curving barcode lines(both black lines and white lines) into straight lines. The idea is that read two continuous pixels above and two continuous pixels below the target pixel. Work out the average y axis, take this average value and set the y axis of five of them to this value, then move to the next.

It relies on edge trimming. If edge trimming is good and could detect the exact barcode region, it might work well. But if edge trimming was not doing well, it will be a useless function. As I decided to throw edge trimming, there is no point to do image gridding any more.

## **2. What is achieved**

- **Scan and recognize barcode in real time(Code 39 only)**
- **Display recognized barcode(implying student id)**
- **Retrieve student details from web server according to the found student id**
- **Display student details when succeed retrieve from web server or an error message otherwise**
- **Built a dummy student web server and a dummy database**
- **Check and response for empty or invalid user name or password inputs and invalid student ids.**

Therefore, all main functionalities are achieved.

## **3. What is not achieved**

### **3.1 A red scan line in the middle of the camera preview frame**

The assist red scan line is mentioned in earlier manual(design manual). The purpose is to assist scanning barcode, like a sighting device for a gun. With this scan line, users could capture barcode image in good quality(not much oblique or rotary). So it improves recognizing efficiency. But I have no idea how to draw it and keep it above the camera preview surface. It would disappear after the surface updates. Although it is a good idea and is used by many existing barcode scanners, I have to abstain it.

### **3.2 Log of previous recognized barcode**

It's another useful function, but I've got no time to finish this function. It helps users to review previous recognized barcodes(student ids).

### **3.3 Management system for web server**

Web management system to manage dummy student database is not achieved. As a result system administrators can only insert, update or delete student details using MySQL command line. As the MBR entire project is mainly a mobile project on Android system rather than a web project and is about to recognize student card barcode, web management system is planned to be an addition region.

However, a web server for receiving http request from both MBR client application and browsers is achieved. It could retrieve student details from accessing MySQL database according to student ids. If the server found the student id, it will return the details of this student. Otherwise, it will return a "student not found" message.



## **4. What I learned**

From this project I have learnt how the Android framework works, its message queue system, knowledge about pixel format and how Android processes image data. And how to use Spring framework to build a web server connected to MySQL database.

## **5. What would do differently**

I would make changes on design model. I found it difficult following a waterfall software process model. When have my design done, I can't say it's correct for sure. There would always be something that haven't been concerned about but should be taken into account. For example, in class design, I think I need five classes to achieve the goal, but later when coding, I find more assisted classes are needed. If starting from scratch, I wish to do a part of design then do a part of coding, and then do the other parts of design then do the other parts of coding. In this way it would be easier do coding and writing design manual I think.

# 6. Updates to earlier reports

## 6.1 Research

Two more significant researches will be updated here. One is the type of barcode used for student ID card, and another one is the pixel format used for camera preview images on Android. They are Code 39 barcode and YCbCr(uses NV21 encoding) pixel format[5]. In the case of this Mobile Barcode Recognition project, we concern about the barcode encoding rule so that we can do barcode decoding, and the pixel format structure so that we are able to do image pre-processing.

### 1. Code 39 encoding rule

Charecter	Encoding Structure	Logical Value	Charecter	Encoding Structure	Logical Value	Charecter	Encoding Structure	Logical Value
0		000110100	F		001011000	U		110000001
1		100100001	G		000001101	V		011000001
2		001100001	H		100001100	W		111000000
3		101100000	I		001001100	X		010010001
4		000110001	J		000011100	Y		110010000
5		100110000	K		100000011	Z		011010000
6		001110000	L		001000011	-		010000101
7		000100101	M		101000010	.		110000100
8		100100100	N		000010011	Space		011000100
9		001100100	O		100010010	*		010010100
A		100001001	P		001010010	\$		010101000
B		001001001	Q		000000111	/		010100010
C		101001000	R		100000110	+		010001010
D		000011001	S		001000110	%		000101010
E		100011000	T		000010110			

Table: Code 39 encoding rule[7]

### 2. YCbCr(uses NV21 encoding) pixel format

YCbCr(uses NV21 encoding) is the default format for camera preview images.[5]

Below are some descriptions of NV21 encoding.

“These are two-plane versions of the YUV 4:2:0 format. The three components are separated into two sub-images or planes. The Y plane is first. The Y plane has one byte per pixel. For V4L2\_PIX\_FMT\_NV12, a combined CbCr plane immediately follows the Y plane in memory. The CbCr plane is the same width, in bytes, as the Y plane (and of the image), but is half as tall in pixels. Each CbCr pair belongs to four pixels. For example, Cb<sub>0</sub>/Cr<sub>0</sub> belongs to Y'<sub>00</sub>, Y'<sub>01</sub>, Y'<sub>10</sub>, Y'<sub>11</sub>. V4L2\_PIX\_FMT\_NV21 is the same except the Cb and Cr bytes are swapped, the CrCb plane starts with a Cr byte.

If the Y plane has pad bytes after each row, then the CbCr plane has as many pad bytes after its rows.”[4]

**Byte Order.** Each cell is one byte.

```
start + 0:  Y'00 Y'01 Y'02 Y'03
start + 4:  Y'10 Y'11 Y'12 Y'13
start + 8:  Y'20 Y'21 Y'22 Y'23
start + 12: Y'30 Y'31 Y'32 Y'33
start + 16: Cb00 Cr00 Cb01 Cr01
start + 20: Cb10 Cr10 Cb11 Cr11
```

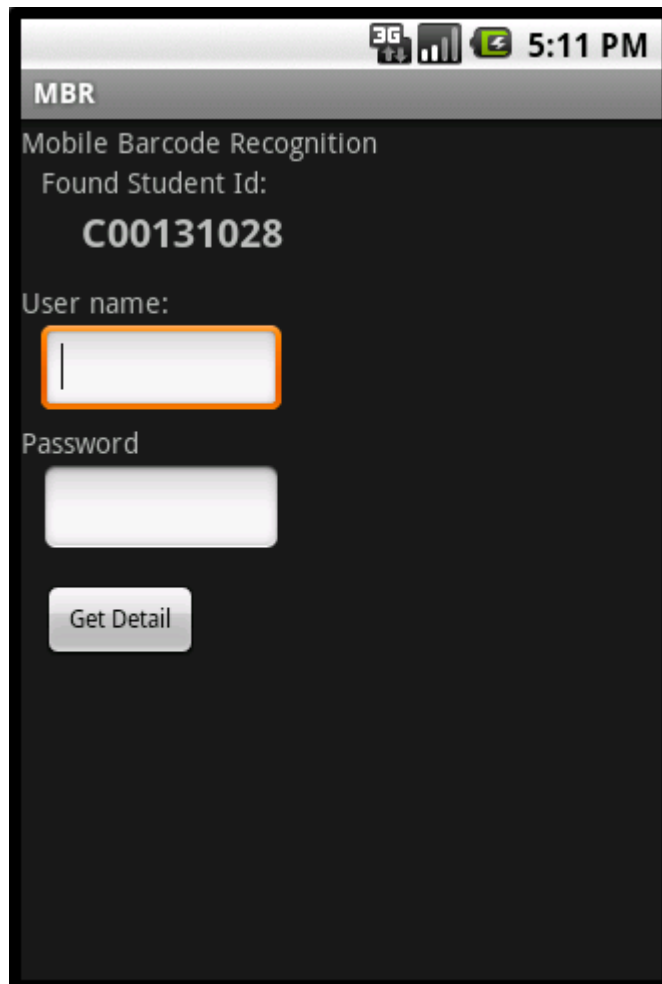
Table: YCbCr byte order[4]

## 6.2 Design Manual

### 6.2.1 GUI

There are some GUI improvements in Result Display UI and Detail Display UI

- Result Display UI:



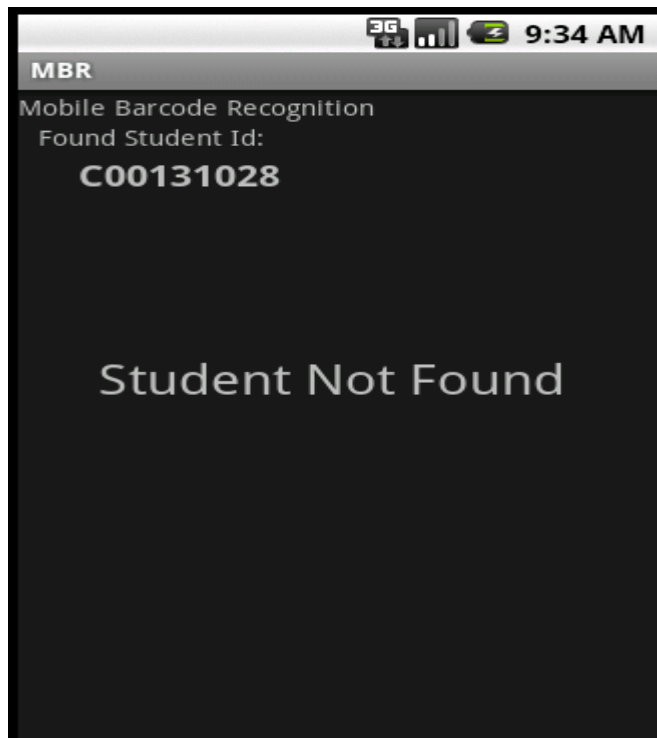
Screenshot: Result display UI

- Detail Display UI:



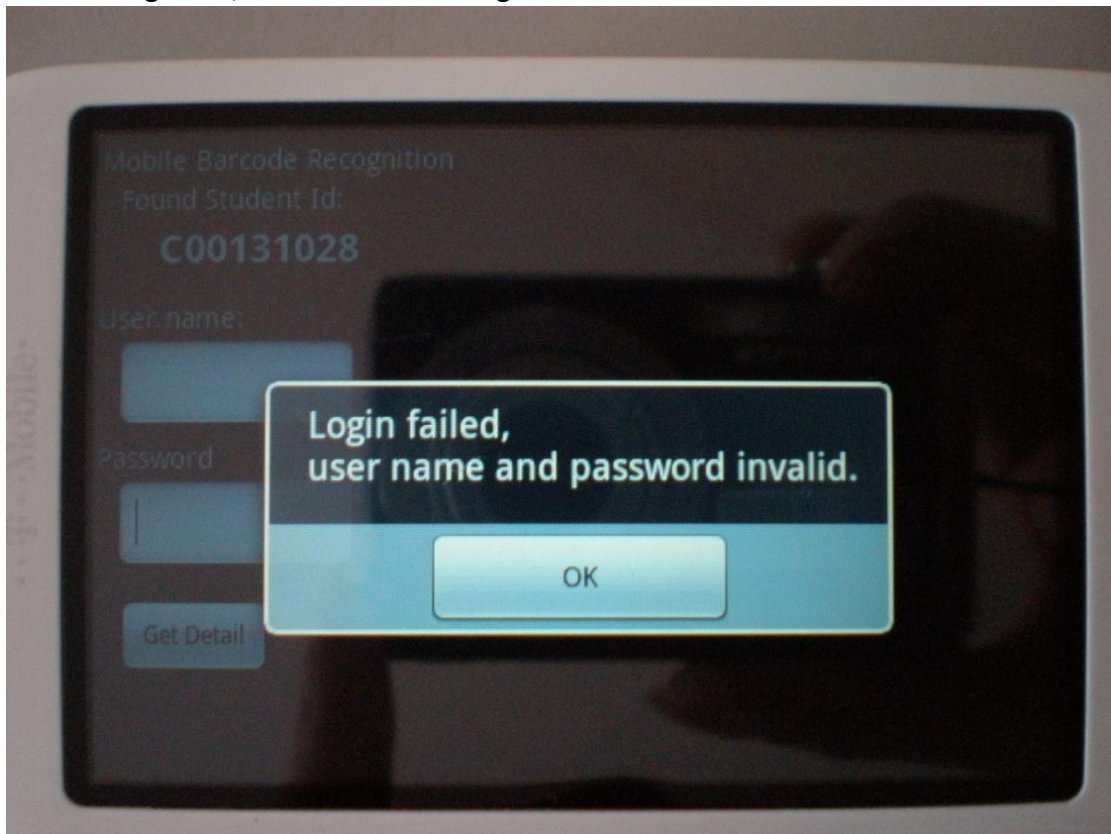
Screenshot: Detail display UI

- Student not found by web server in database



Screenshot: Student not found UI

- Login fail, show a alert message.



Screenshot: Login fail UI

## 6.2.2 Domain Model

The domain model has changed a lot:

- Disable BarcodeDetector and BarcodeSplitter
- ImageSource charges the works of BarcodeProcessor
- Add ResultActivity and ResultActivityHandler to display result and student details
- Add StudentDetail class to encapsulate student details

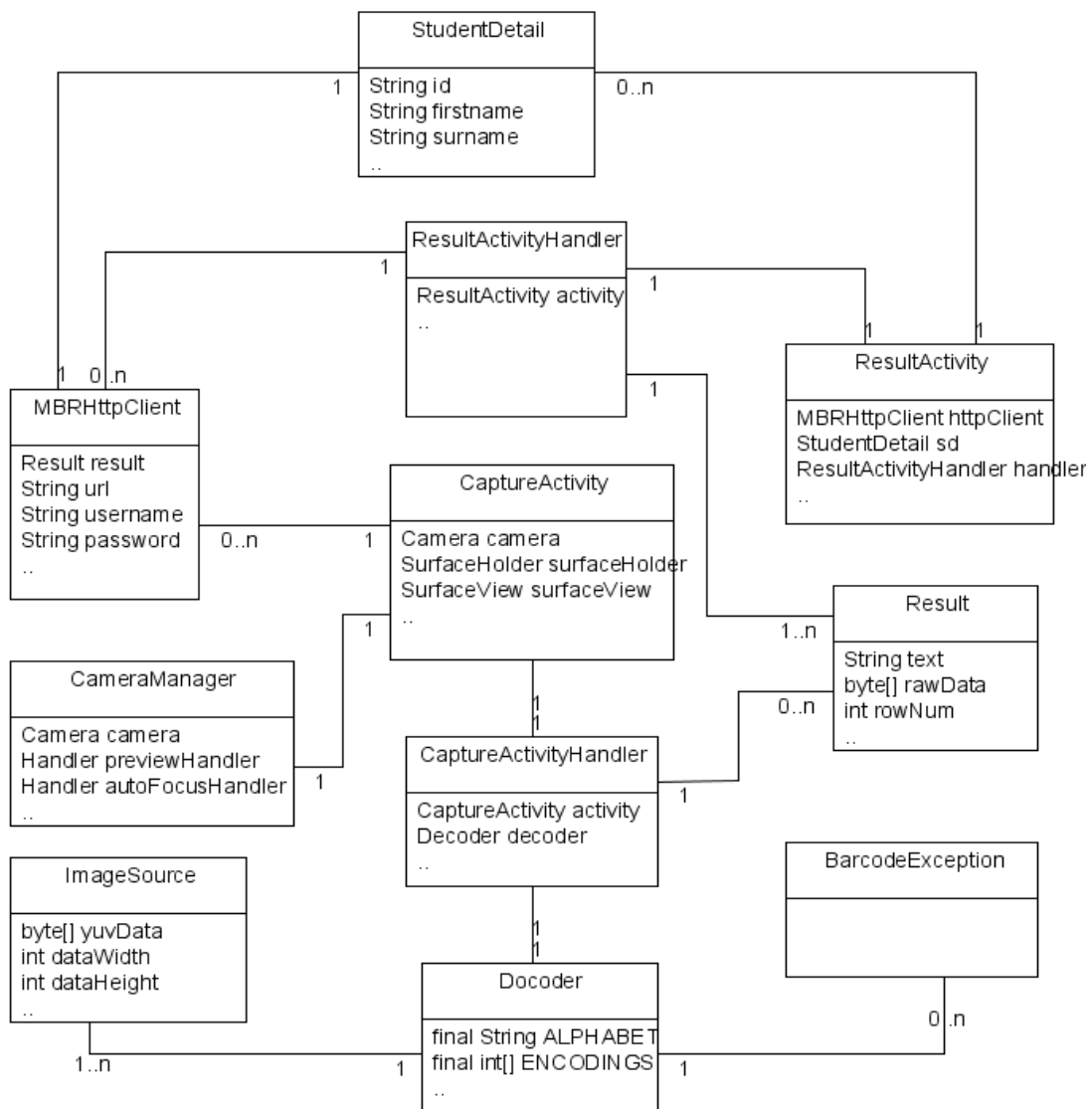


Diagram: Domain model

Class functions please see class diagram below.

## 6.2.3 Class Diagram

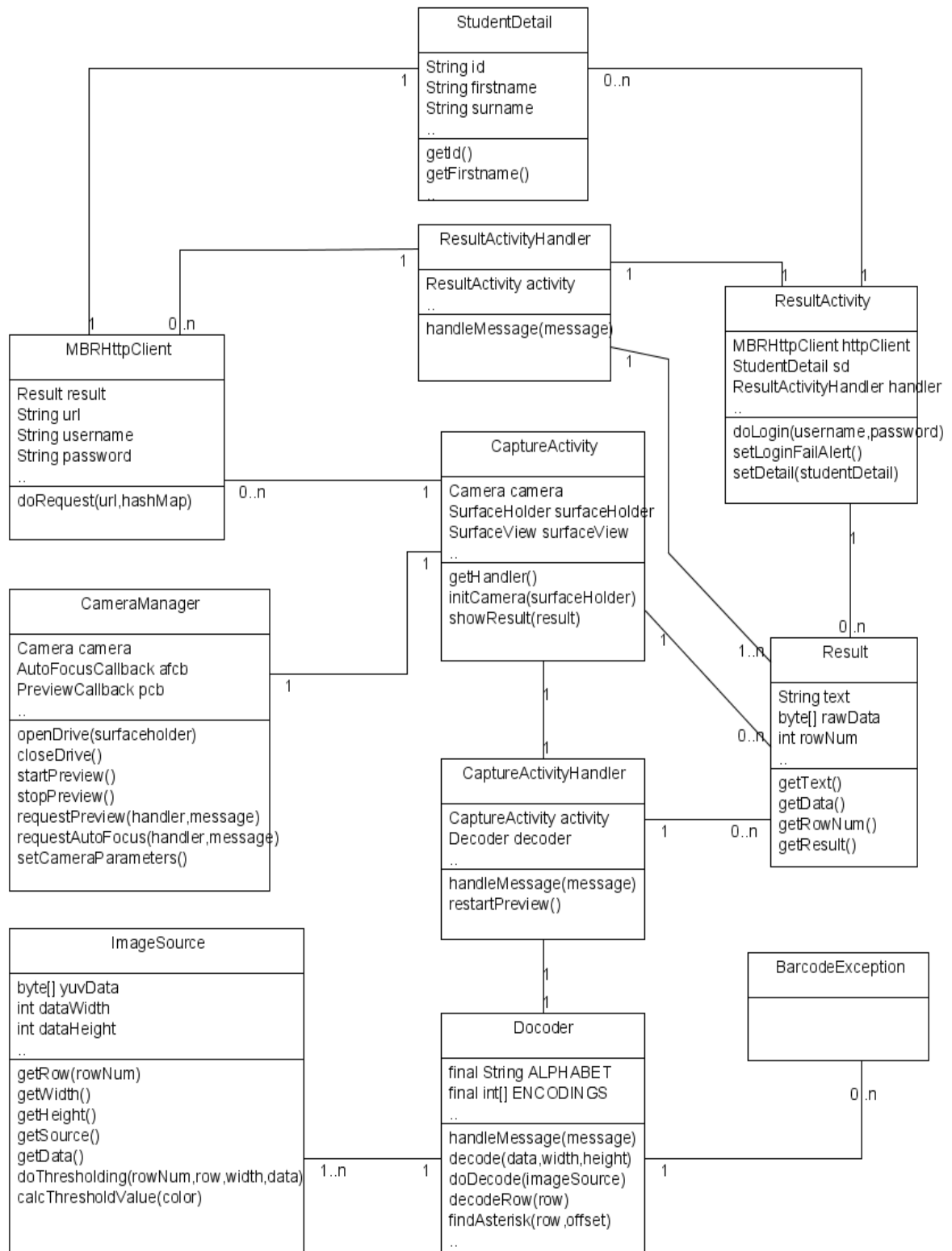


Diagram: Class diagram



## 6.2.4 Web server

The web server uses Spring framework to construct the student web service. Because I find it easier to build a web server this way. The framework will do lots of work for you.

This is StudentServer hierarchy, it includes source code, relied libraries, some configure files and an index page.

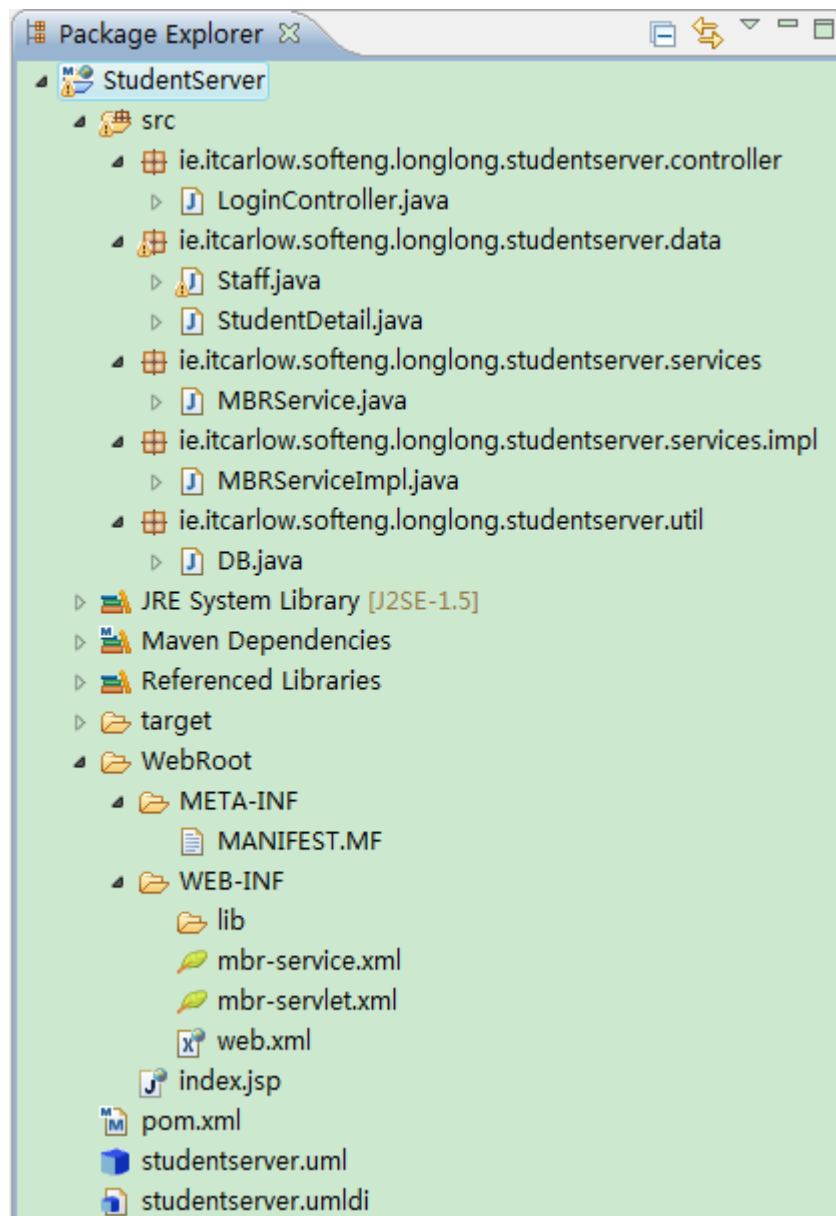


Chart: Web server hierarchy

The server provides one Login service, but this login service actually does both login and retrieving. When client connect to the server, it sends login user name and password with student id together. So once LoginController receive the request, it

checks validity first. If valid, it retrieves database (by calling retrieve(studentId) function) with studentId and it returns a response with retrieve result to the client afterwards. If not valid, it returns a response with a message of "login fail" to the client.

## 7. Module description

(Modules do not include web server.)

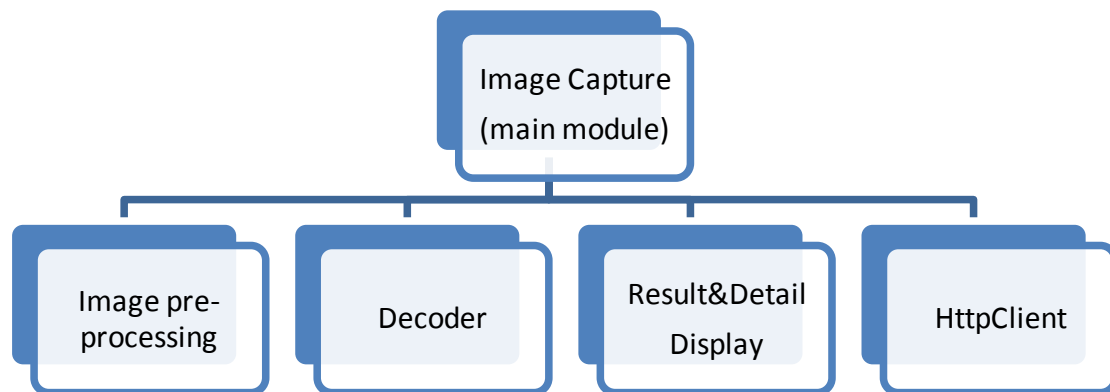


Chart: Modules

### **Image Capture(main module):**

In MBR project, the Image Capture plays a double role which is both image capture module and main module. It firstly is a main module which provides an entry to the entire application, and then is an image capture module which request capture function and auto focus function.

Functionalities include:

- Starting and initializing application
- Connecting camera drive and initializing camera settings
- Request image capturing and auto focus
- Send captured image data to Decoder

### **Image pre-processing:**

This module provides a do thresholding function. And this do thresholding function combines grey scaling, median filter and thresholding together. The purpose for this change is to minimize usage of cpu resource. If we do them separately, cpu need to read the captured image data three times, which for mobile devices is a big expense. Although in function design, it's not a good habit, but reading it one time and do all three functionalities does save lots of resource.

Functionalities include:

- Do thresholding

### **Decoder:**

Decoder does all heavy works about decoding barcodes. A decoding process flow chart(omit exceptions in this chart):

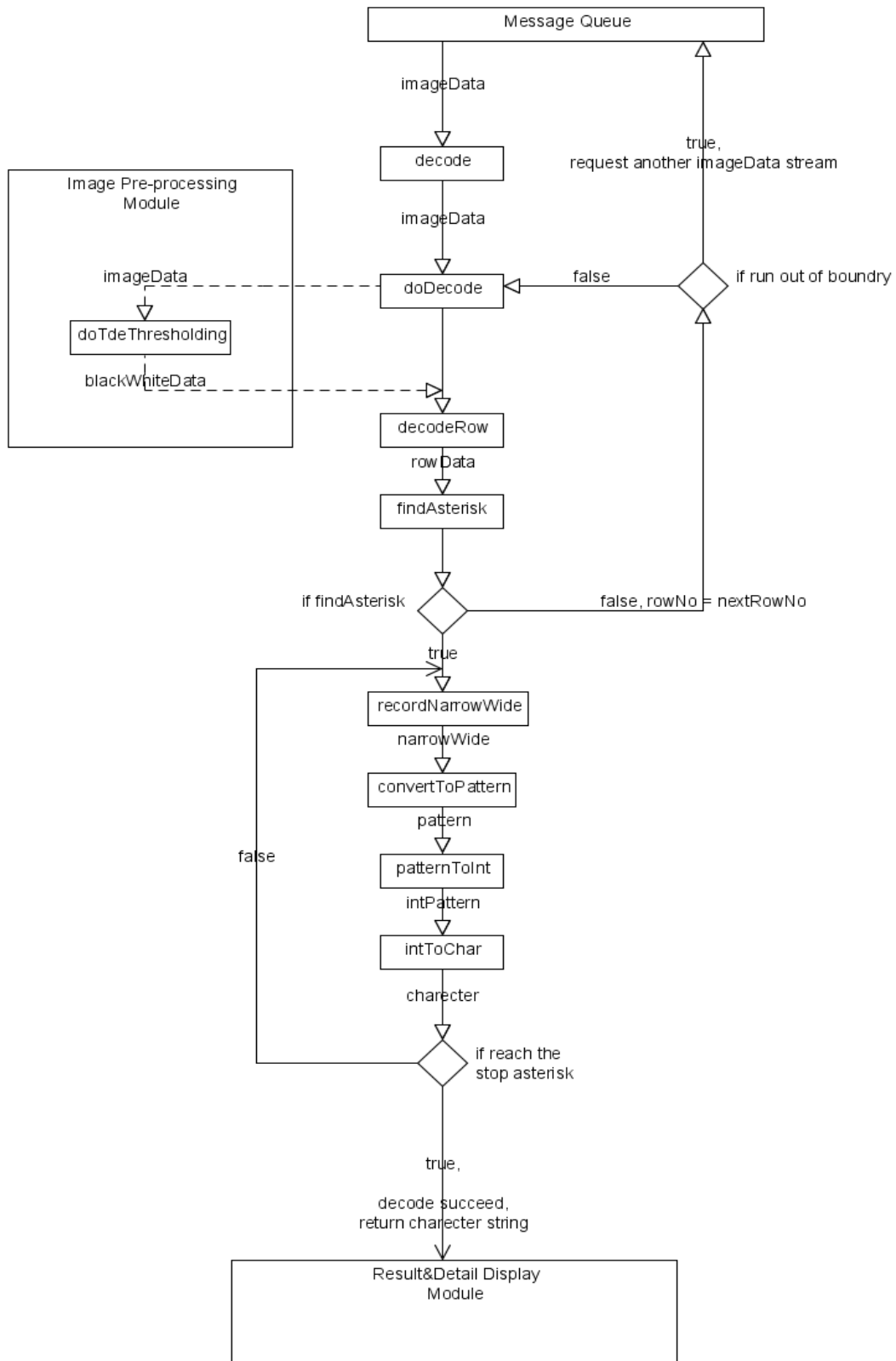


Chart: Decode Process Flow

**Result&Detail Display:**

In this module, once get the recognized barcode(implying student id), a new thread will be started to display result and connect to web server.(Because connecting to web server may cause time delay and application would probably be treated as no response and stopped by system, so start a new thread to do it.)

Functionalities include:

- Display recognized barcode
- Start a HttpClient to retrieve detail from web server
- Display details if succeed, show error message otherwise

#### **HttpClient:**

This module charges sending http request to web server and receiving response from the server. It would send login user name and password with the recognized student id together to web server by POST method, and parses the http response into readable information by using JSON technique.(the web server also uses JSON to encapsulate http response).

Functionalities include:


- Send http request
- Receive http response
- Parse http response into readable information


## 8. Testing

Project has been tested in different light conditions and for up to twenty different barcode images or student cards.

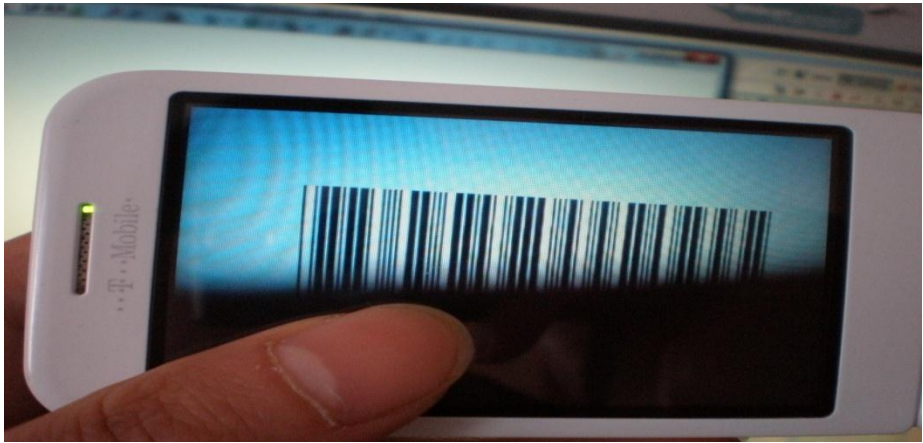
### 8.1 Functional testing

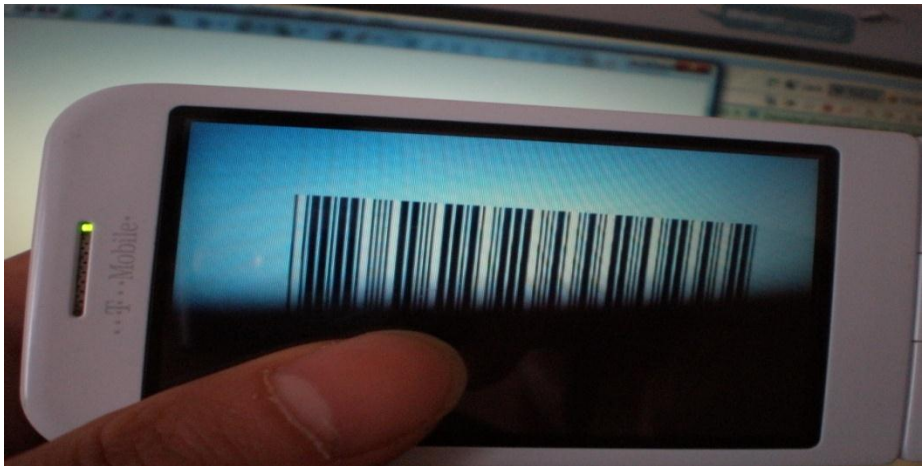
- In different conditions

Light Condition	Good
Image Condition	Complete
Sample Image	
Readable or Not	Yes
Recognized Barcode	C00131028

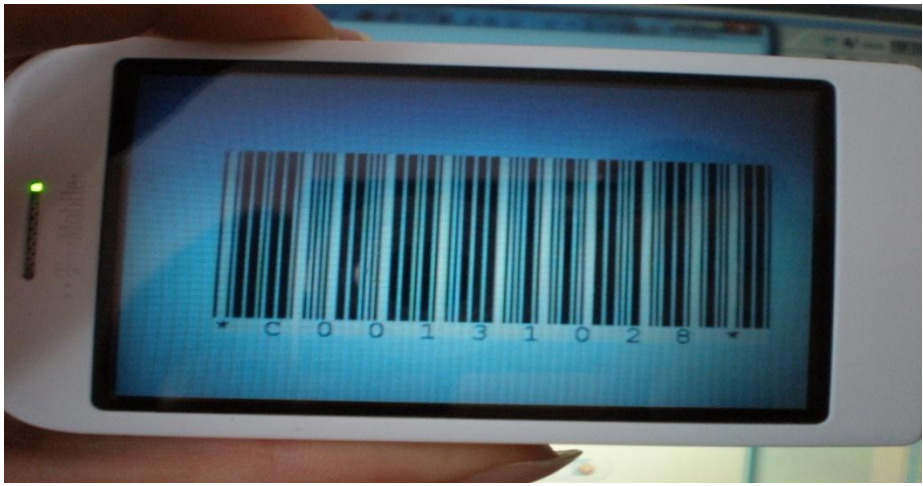
Light Condition	Good
Image Condition	Almost complete
Sample Image	
Readable or Not	Yes
Recognized Barcode	C00131028


Light Condition	Good
Image Condition	Upper part masked
Sample Image	
Readable or Not	Yes
Recognized Barcode	C00131028

Light Condition	Good
Image Condition	Bottom half masked
Sample Image	
Readable or Not	Yes
Recognized Barcode	C00131028


Light Condition	Normal
Image Condition	Bottom half masked
Sample Image	
Readable or Not	Yes
Recognized Barcode	C00131028




Light Condition	Bad
Image Condition	Complete
Sample Image	
Readable or Not	Yes
Recognized Barcode	C00131028


Light Condition	Awesome
Image Condition	Complete
Sample Image	
Readable or Not	No
Recognized Barcode	null

- For different barcodes or student cards

Type	Student card
Sample Image	
Readable or Not	Yes
Recognized Barcode	C00131028

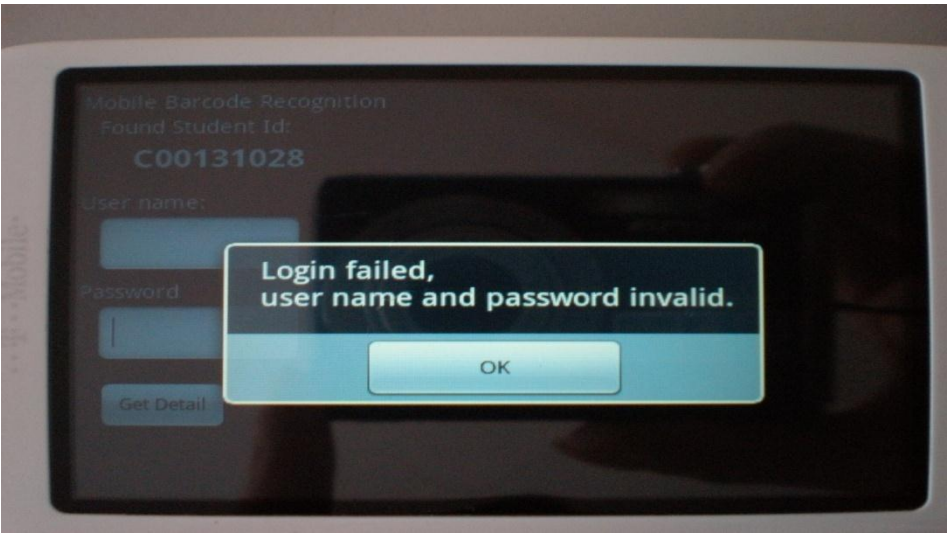
Type	Other barcode
Sample Image	
Readable or Not	Yes
Recognized Barcode	CHIUDW

- Read reversely

Is reversed	Yes
Sample Image	
Readable or Not	Yes
Recognized Barcode	C00131028

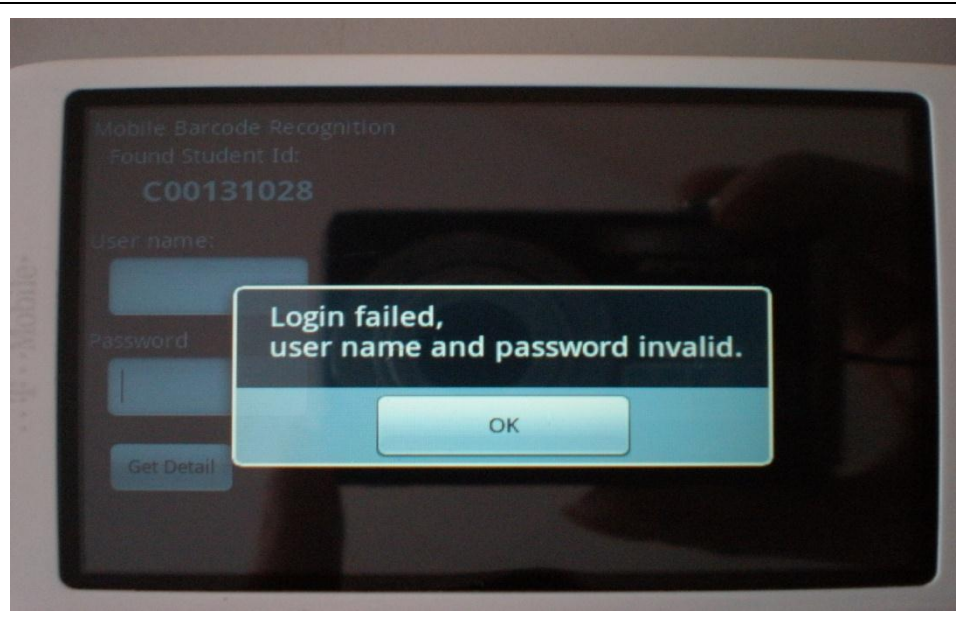
## 8.2 Reliability testing

- Input empty user name or password

Response	
----------	--

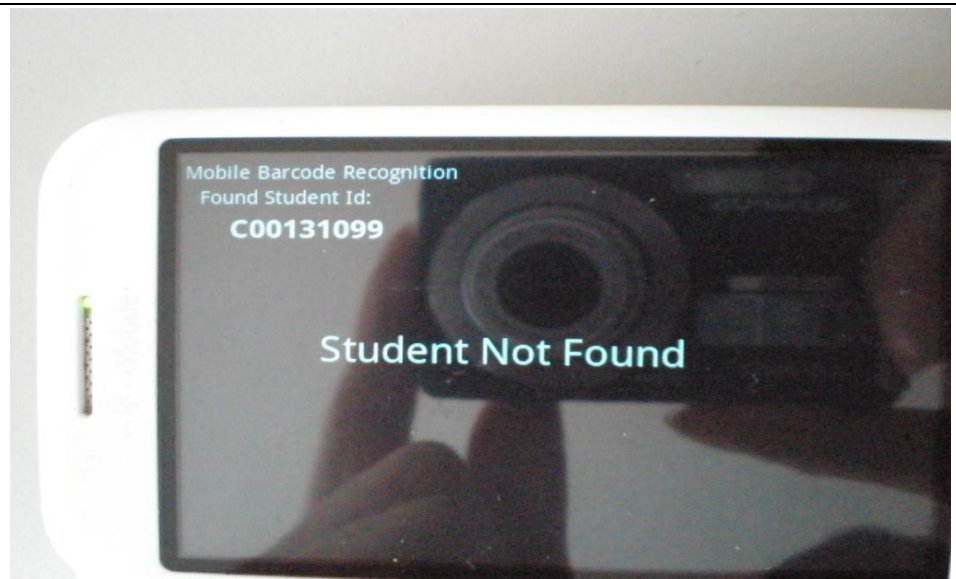
- Input invalid user name or password

Response



- Detect an invalid student id

Response



### 8.3 Recognize time testing

From over 50 sample barcodes, the result shows generally the time for recognize one barcode is about 2-4 seconds.

## 9. Conclusion

From testing, we conclude the recognition rate of MBR application is over 95% and could suit different light condition except some really dark condition and could read in both positive sequence and reserve sequence, and it connecting to server function works well.

## Reference

1. “*user feature*”, Android Developers, web.  
<<http://developer.android.com/guide/topics/manifest/uses-feature-element.html>>
2. “*Camera.PictureCallback*”, Android Developers, web.  
<<http://developer.android.com/reference/android/hardware/Camera.PictureCallback.html>>
3. *ZXing Project Home*, Google Code, web. <<http://code.google.com/p/zxing>>
4. “*V4L2\_PIX\_FMT\_NV12 ('NV12'), V4L2\_PIX\_FMT\_NV21 ('NV21'), YUV Formats*”, KERNEL.org, web. <<http://www.kernel.org/doc/html/docs/media/re18.html>>
5. “*PixelFormat*”, Android Developers, web.  
<<http://androidappdocs.appspot.com/reference/android/graphics/PixelFormat.htm>>
6. “*YCbCr*”, WIKIPEDIA, web. <<http://en.wikipedia.org/wiki/YCbCr>>
7. “*The Encoding Principle and the Design of Code 39 Used in Libraries*”, YANG Jing, GUAN Yanhui and CAO Janhui, Hebei Normal University of Science & Technology, China.