1. Introduction

As smartphones are going to be much cheaper than before, many people own one and familiar to use it. In this project, elderly users are the main service target. The proposed system will allow the elderly users to use a large screen to read text and smartphone as the input peripheral for text without any complicated installation and physical limitation. The name of the Application is called **RemotePen**.

Initiative:
Difficulties of elderly users when using smartphone and desktop to read articles.

1. For smartphone, screen size is too small for users
2. For desktop, except English and number, they usually do not know any other text input method, such as Chinese characters
3. Price of electronic board is attractive and the installation is complicated for users
4. They need to install the server side program in the desktop every time if they move to another desktop for some remote control solution using smartphone.
5. Handwriting using mouse is a very difficult task for elderly user due to hand tremor

Since elderly users are searching internet for news article through browser most of the time, we want to target on the most popular browser which is Google Chrome to be platform of our proposed solution in the desktop side.

It is known that if users login their chrome account and then they just need to install the Chrome App and Chrome Extension once, they will be able to use them in any other workstation with chrome installed after login. For simplicity, it only supports the device and desktop under the same access point.

To receive the texts sent from mobile phone, there is a server running in the Chrome App. Texts will be passed to Chrome extension by Chrome App using Message Passing API[1]. Then, content script in Chrome extension will responsible to fill in the selected text box on the webpage.

According to the article[2], the Android application will display larger font size and keep the App flow simple.

1 https://developer.chrome.com/extensions/messaging
2 K.Y. “Larger Icons and Keyboards for a Phone Mom Would Love: Android Can Make the Elderly Tech Savvy” July,2013. [Online] http://www.makeuseof.com/tag/larger-icons-and-keyboards-for-a-phone-mom-would-love-android-can-make-the-elderly-tech-savvy/

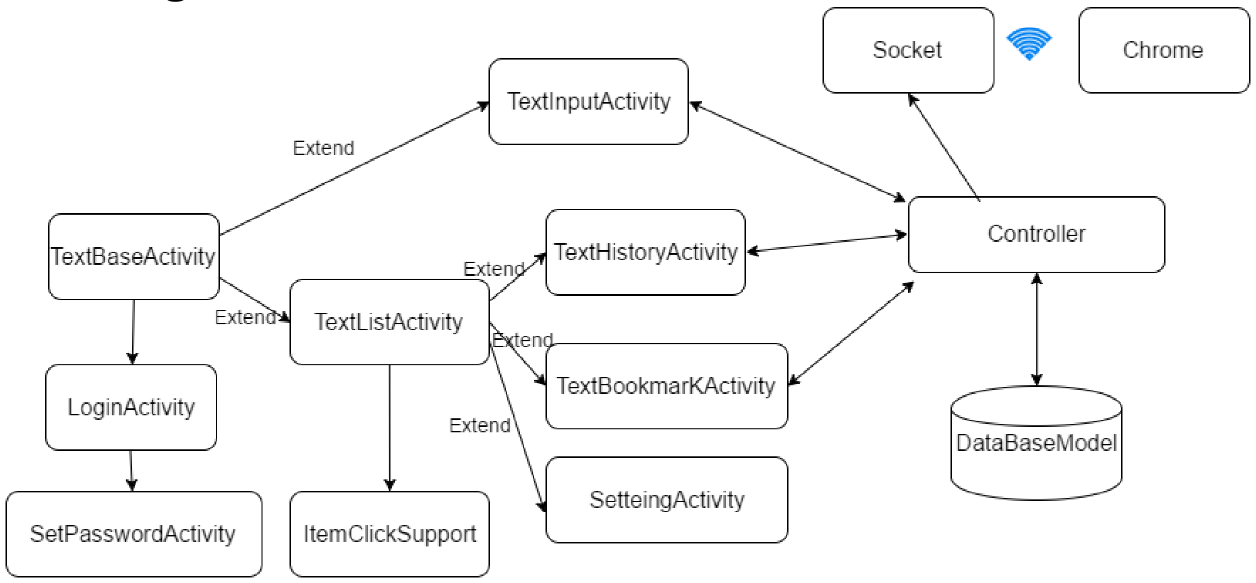
2. Design and Implementation

2.1 Requirement Anaylsis

This application is targeting any user who are using Logogram but not Latin alphabet in the typing method, since Logogram is more reasonable to be inputted using handwriting.

1. User use mouse to select the text box in Chrome and then use their fingers to input text in smart phone and then press confirm.
2. The text in smart phone will be sent to the selected text box in Chrome.
3. The sent text input will be recorded as history and user can select the record to send again conveniently.
4. Simple login is required for protecting the privacy
5. The history record can be bookmarked in order to provide convenience and fast way to send frequently used text.
6. Making the delete function no easy to activate since user may accidentally activate the function which is not reversible
7. Provide a function to delete all history records but not deleting the bookmarked records since there can be larger number of records as time pass by and many records with duplicated content can appear.

2.2 Design



2.2.1 Login Page

Since the User's messages send to Chrome will be recorded inside the Android App which can be private information, the login page can protect any unauthorized access to the records. The user just need to enter 4 numbers password in the Login page for simplicity. The login page will be shown every time when the App is put into background.

Since the login action can be very often, an embedded number pad will be used to enter the password. Once the password is correct, the Login page will disappear and user can continue to use the App from the last page when they put the App into background.

2.2.2 Password Setting Page

Login Page will change if the App contains existing password, if not exist any password, the password setting page will be called. The user just need to enter 4 numbers in password editable text field twice. There will be error message displayed if the two passwords are match or empty.

2.2.3 Action Bar

This bar will provide a convenient way for user to navigate around different pages.

2.2.4 Input Page

This page will allow the user to input and send their text. It contains a very big editable text field for user input text with lager font size and a Button for confirmation of text sending to Chrome App as shown in Error: Reference source not found. If the text is sent successfully, the text will be recorded in the database. The text input in the last time will be always keep in the editable text field.

2.2.5 History Page

This page shows a list of history record which the user sent. User can scroll to read the record. Each list item contains the sent text and time. User can click the item and then choose come options in a dialog box. The option includes sending the text again, add the record to bookmark and cancel. Since delete is unrecoverable, the delete action is set to not convenience to activate. If the user press the item for long time, the delete confirmation dialog box will display. If the item is bookmarked, then there will be a paper clip image icon shown in the top right corner.

2.2.6 Bookmark Page

This page shows a list of bookmarked records and provides a fast access for users if they need those text frequently. User can scroll to read the bookmarked record. Each list item contains the sent text and time. User can click the item and then choose come options in a dialog box. The option includes sending the text again, removing the record from bookmark and cancel. Since delete is unrecoverable, the delete action is set to not convenience to activate. If the user press the item for long time, the delete confirmation dialog box will display. If the item is bookmarked, then there will be a paper clip image icon shown in the top right corner.

2.2.7 Controller Object

This class will do the implementation of sending text to Chrome App and do the connection between DataBaseModel and Activity. It will do the calling method to read, write and delete records inside database

2.2.8 DataBase Model

History Table	
int	Row ID
string	Time
string	Content
Boolean	isBookmark

Table 1 HIstory Table structure in Database

Bookmark Table	
int	Row ID in History Table

Table 2 Bookmark Table structure in Database

The History Table have data and data type as shown in Table 12. Time is the time when user send the text and the time is will be generated inside DataBaseModel when text is prepared to write into table. Content is the text which user input and sent out. isBookmark is indicating whether the record belong to bookmark. SQLite API will be used for database management. The DataBase Model provides all methods of read, write and delete single record, and retrieve a list of history records and bookmark records.

2.2.9 Setting Page

This activity contains the Change Password Button and Clear All History Button. The confirmation dialog will display if the user want to delete all history.

2.3 Implementation

2.3.1 LoginActivity

This activity is responsible for the implementation of Login page with **login_activity.xml** file. Password is stored using **SharedPereferences**. **SetPasswordActivity** will be shown if the retrieved password value is null. 11 buttons are used to create the number pad which includes 0~9 and clear button. All buttons have circle border using **num_button_border.xml** file. Each time user click one button, the corresponding number will be appended into the editable text field. If the user input 4 number, the activity will automatically check the correctness of the input using **attemptLogin().** If the password is wrong, text “Wrong!!!!” will be shown below the password field for 1 second. **postDelayed()** in **Handler** will be used to do the delay.

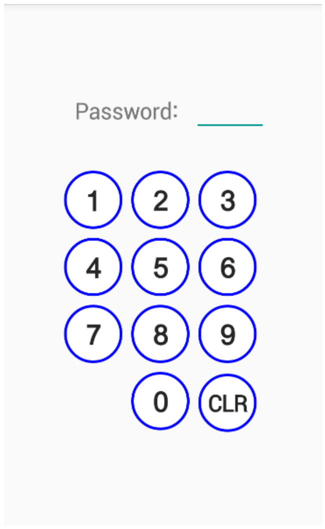
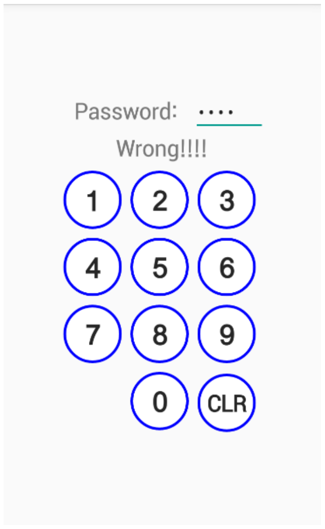


Figure 1 UI of LoginPage


Password: ····

Wrong!!!!

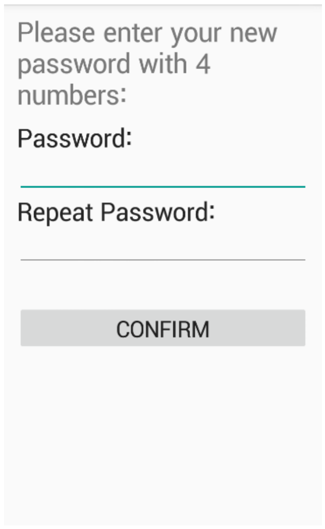
Figure 2 Wrong password in Login

2.3.2 SetPasswordActivity

It is responsible for user to set the password used in Login with **set_password_layout.xml file**. It uses the same key as **SharedPreferences** in **LoginActivity** for password to store the new password. The length of number input is limited to 4. Error message will be display above the confirm buttons. The field only allow to input number. The cancel button will only appear if the **SetPasswordActivity** is start from **SettingActivity** using **intent.putExtra()** and **intent.getBooleanExtra().**

Error Message:

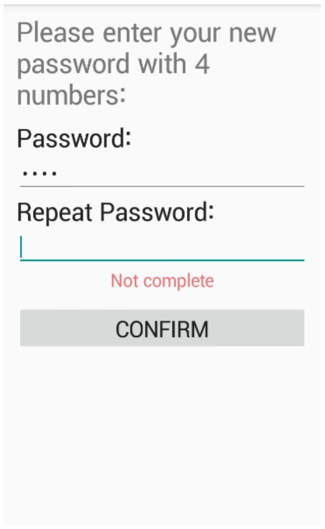
Condition	Output
Either one of the field has length <4	Not complete!
The password in the two field not match	Not match!


Please enter your new password with 4 numbers:

Password:

Repeat Password:

Figure 3 Set Password Page


Please enter your new password with 4 numbers:

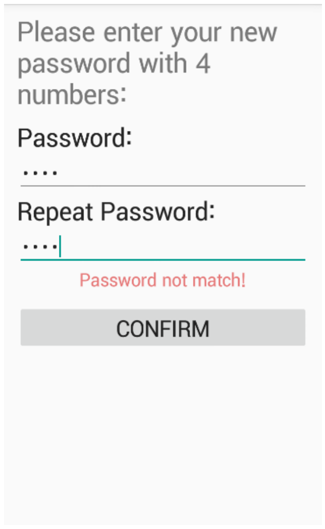
Password:

····

Repeat Password:

Not complete

Figure 4 Display "Not complete" error message


Please enter your new password with 4 numbers:

Password:

....

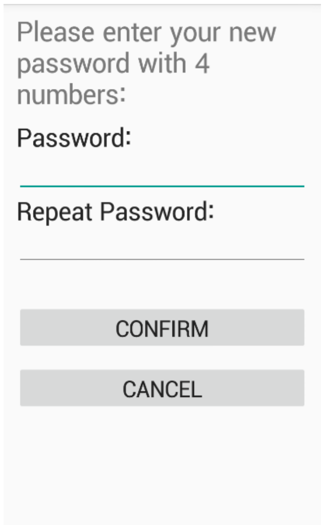
Repeat Password:

....

Password not match!

CONFIRM

Figure 5 Display "Password not match" error message


Please enter your new password with 4 numbers:

Password:

Repeat Password:

CONFIRM

CANCEL

Figure 6 Cancel button appear if the previous page is setting page

2.3.3 ActionBar and TextBaseActivity

The Action Bar will be added with three drawable icons for starting the **TextInputActivity**, **TextHistoryActivity** and **TextBookmarkActivity** by editing the **res/menu** with attribute **app:showAsAction=”always”**. The handling method is **onOptionItemSelected()** and put in **TextBaseActivity**. The Action Bar is shown according to **actionbar.xml**. While the menu is shown according to the **menu_main.xml**. Any Layout require this ActionBar will just need to include **layout=”@layour/actionbar”** in the resource file.

The **TextBaseActivity** is responsible to keep checking whether the App is put into Background and call **LoginActivity** after the App resume from background. This is challenging since Android provide no dedicated API to check this situation. Therefore, there is an Action Bar for navigation in order to assume any Activity switching is putting the App into background. This is also reason most Activity need to extend from **TextBaseActivity** so as to obtain the detection of putting App in background.

There are two Booleans inside the Activity responsible for the detection: **firstLogin** and **isClickInsideApp** . When the icons in Actionbar is click to switch activity, the **isClickInsideApp** will set to true. The detection is implemented in **callLoginActivity()** which will be called every time when **onPause()** is called. **onBackPressed()** is override with empty content in order to prohibit user clicking back key.

The **TextBaseActivity** provide some abstract method for child class to implement: **getLayoutResourse(), getActivityName()**, **preOnResume()** and **postOnResume().**

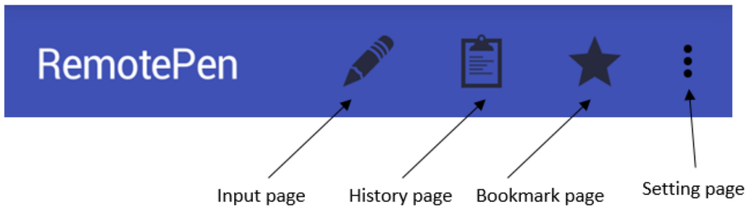


Figure 7. Action bar

2.3.4 TextInputActivity

It is responsible for the logic control of Input Page with UI display shown according to **input_activity.xml**. It implemented the abstract method **preOnResume()** and **postOnResume()** from the extended **TextBaseActivity** to save the user input in the editable field into **SharedPreferences** in order to keep the user input although they switch to other Activities. The Send button will **getText()** from the editable field and then call the **Controller.send()** function in order to send the text to chrome and save the text into database.

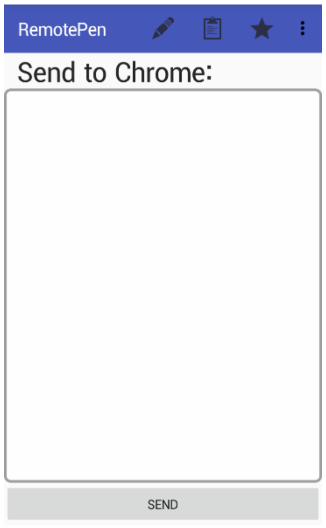


Figure 8 UI of input page

2.3.5 HistoryItem

HistoryItem is an Object to store the information of each record, including RowID, time, content and isBookmark boolean. Time is in format

2.3.6 TextListActivity

It is extended from **TextBaseActivity** and responsible to implement the **RecyclerView** to display a list of records staring with the most recent record. **MyAdaptor** class is implemented by extending **RecyclerView.Adapter<MyAdapter.ViewHolder>.** Inside the **MyAdaptor** class, the view of each list item is contained inside class **ViewHolder** extending **RecyclerView.ViewHolder** and using the resource from **history_item.xml**. **android.support.v7.widget.RecyclerView** is put inside the resource file in order to show the **RecyclerView** List. **LinearLayoutManager** will also be required. **MyAdaptor** is initialized by passing the **historyItemList** of records and then put into **mRecyclerView.setAdapter()**.

View of each list item contains two **TextView** for content and time and one **ImageView** for isBookmark icon. **setText()** according to the content and time in **HistoryItem** in the **historyItemList**. The isBookmark icon will be **setVisibility(View.VISIBLE)** if it is true and else **setVisibility(View.GONE)**. The background of the item is **dialog_bubble.9.png**. The change of content of each view is implemented inside **onBindViewHolder()**.

Abstract method **getList()**, **buildMyOptionDialog()** and **buildMyDeleteDialog()** will be provide to child class to implement. There will be an option dialog box display when list item is click. A delete confirmation dialog will be shown when the list item is long clicked. When the item is click, a global variable **cur_position** will be set as the position of the item in the

historyItemList. An opaque layer will cover the selected item using a **LinearLayout** with background using **drawable/hist_item_gb_color.xml** as a **selector**. The selector is set to true inside the click callback function and reset inside the **onBindViewHolder()** using **MyAdaptor.notifyItemChanged()**.

2.3.7 ItemClickSupport

There are two user interaction for selecting the list item by using the **ItemClickSupport** class. There are one **View.OnClickListener** and one **View.OnLongClickListner** created in the class. They use the passing in parameter View to retrieve the **RecyclerView.Viewholder** and then call the **onItemClicked()** in interface **OnItemClickListenser** or **onItemLongClicked()** in the interface **OnItemLongClickListener**. **TextListActivity** can implement the methods in the interface for its own purpose by using **ItemClickSupport.SetOnItemClickListener()** and **ItemClickSupport.SetOnItemLongClickListener().**

ItemClickSupport use **setTag()** and add **addOnChildAttachStateChangeListener** into the **RecyclerView**. The **addOnChildAttachStateChangeListener** contains **OnClickListener** and **OnLongClickListenser**. As a result, each list item as a **View** will be attached with the click listener. Once the item is click or long click, the callback function **onItemClicked()** or **onItemLongClicked()** will be called with passing in useful parameter as the position and view of the item.

2.3.8 TextHistoryActivity

It extends from **TextListActivity** and using resource **history_activity.xml** for layout. The **historyItemList** is retrieved by calling **Controller.readHistory()**. The option dialog box contains “Send”, “Add to Bookmark” and “Cancel”. “Send” will call the **Controller.send()** without save. “Add to Bookmark” will call **Controller.saveBookmark()** by passing the **HistoryItem.getID()** which is retrieved by using **List.get(cur_position).** All these three option will call **RecyclerView.Adaptor<>.notifyItemChanged()** to reset the selector. Long Click the item will display a dialog to confirm to delete the item. To delete the item, **Controller.deleteHistory()** will be called to delete single history record in database.

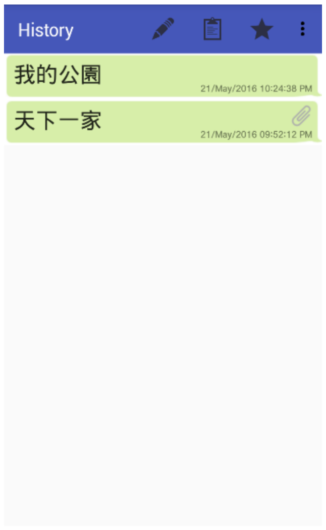


Figure 9 UI of History page

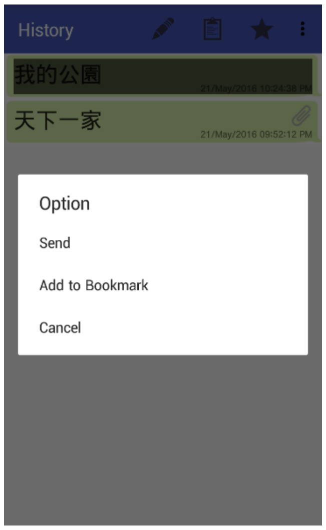


Figure 10 UI of option dialog box in history page

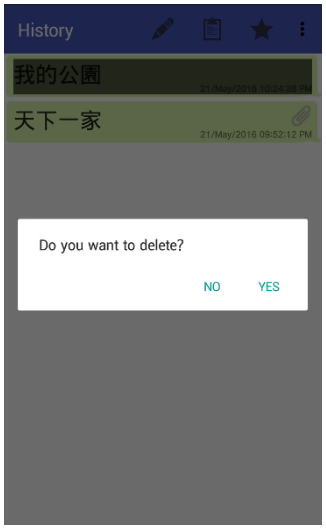


Figure 11 The delete confirmation dialog box in history page

2.3.9 TextBookmarkActivity

It extends from **TextListActivity** and using resource **bookmark_activity.xml** for layout. The **historyItemList** is retrieved by calling **Controller.readBookmark()**. The option dialog box contains “Send”, “Remove Bookmark” and “Cancel”. “Send” will call the **Controller.send()** without save. “Remove Bookmark” will call **Controller.deleteBookmark()** by passing the **HistoryItem.getID()** which is retrieved by using **List.get(cur_position).** . All these three option will call **RecyclerView.Adaptor<>.notifyItemChanged()** to reset the selector. Long Click the item will display a dialog to confirm to delete the item. To delete the item, **Controller.deleteHistory()** will be called to delete single history record in database.

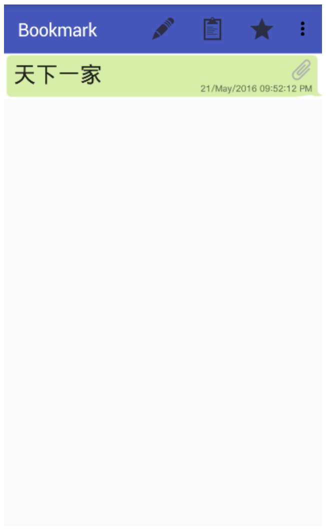


Figure 12 UI of the Bookmark page

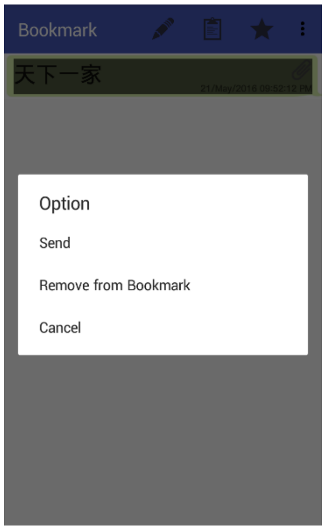


Figure 13 UI of the option dialog in Bookmark page

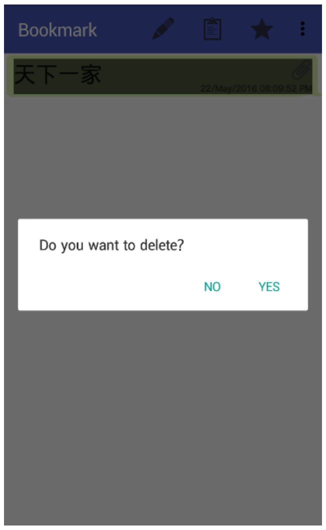


Figure 14 Delete confirmation dialog in Bookmark page

2.3.10 Controller

The methods in controller are just call the methods in DatabaseModel. It also create the socket to connect to Chrome. It checks the internet connection by using ConnectivityManager and NetworkInfo.

2.3.11 DataBaseModel

There are DatabaseHelper and SQLiteDatabase Object inside this class to manage the record in two table, SQL_LITE_HISTORY_TABLE and SQL_LITE_BOOKMARK_TABLE. The default of isBookmark is false and the only column in SQL_LITE_BOOKMARK_TABLE is just the HISTORY_ROWID in SQL_LITE_HISTORY_TABLE.

The SQL to create SQL_LITE_HISTORY_TABLE:

```
"CREATE TABLE if not exists " + SQL_LITE_HISTORY_TABLE + " (" +
        HISTORY_ROWID + " INTEGER PRIMARY KEY  AUTOINCREMENT"+ "," +
        HISTORY_TIME +" TEXT  NOT NULL "+ "," +
        HISTORY_CONTEXT +" TEXT NOT NULL"+ "," +
        IS_BOOKMARK +" INTEGER DEFAULT 0 );";
```

The SQL to create SQL_LITE_BOOKMARK_TABLE:

```
"CREATE TABLE if not exists " + SQL_LITE_BOOKMARK_TABLE + " (" +
       HISTORY_ROWID +" INTEGER  PRIMARY KEY"+");";
```

To retrieve the whole list of History record from SQL_LITE_HISTORY_TABLE in **readHistory(). Cursor** is used obtain by **rawQuery()** the SQL

```
"SELECT  * FROM " + SQL_LITE_HISTORY_TABLE
```

and then put the ID, text content, time and isBookmark Boolean into **List<HistoryItem>** using **cursor.moveToLast()** and **cursor.moveToPrevious()** in a do..while loop.

In **writeHistory(),** time is obtained using **SimpleDataFormat("dd/MMM/yyyy hh:mm:ss aa")** and **Calendar.getInstance().** The sent text and time will be put into **ContentValues** and **insert()** into SQL_LITE_HISTORY_TABLE

The updateHistory_col_Bookmark() use update() to update the isBookmark according to the passing in ID. While the deleteHistory() allow to delete one row in table according to passing in ID.

The removeAllHistory() use the following SQL

```
"DELETE FROM "+SQL_LITE_HISTORY_TABLE
        +" WHERE "+HISTORY_ROWID
        +" NOT IN (SELECT " + HISTORY_ROWID+" FROM "+SQL_LITE_BOOKMARK_TABLE+")";
```

to deleted the all non-bookmarked record.

To retrieve the whole list of bookmarked History record from SQL_LITE_ BOOKMARK _TABLE in **readBookmark(). Cursor** is used obtain by **rawQuery()** the SQL

```
"SELECT  * FROM " + SQL_LITE_HISTORY_TABLE+","+ SQL_LITE_BOOKMARK_TABLE +" WHERE
"+SQL_LITE_HISTORY_TABLE+"."+HISTORY_ROWID+"="+SQL_LITE_BOOKMARK_TABLE+"."+HISTORY_ROWID;
```

and then put the ID, text content, time and isBookmark Boolean into **List<HistoryItem>** using **cursor.moveToLast()** and **cursor.moveToPrevious()** in a do..while loop.

2.3.12 ChromeSocket

Socket object is used to create the socket. PrintWriter is used to output the data. The text is encoded with Base64 before sending.

2.3.13 SettingActivity

It extends from **TextBaseActivity** and using the resource file **setting_activity.xml** for layout design. It contains three buttons. One of the buttons is used for switching to **SetPasswordActivity** for changing the password and the other button is used for switching to SetIPActivity for seting the IP and Port of Chrome Server. The final one is used to delete all records by calling **Controller.removeAllHistory()**, however, only the History without bookmarked will be deleted.

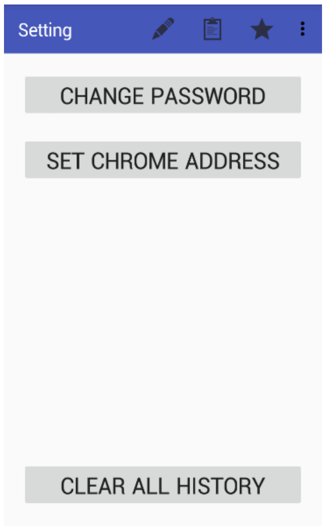


Figure 15 UI for Setting page

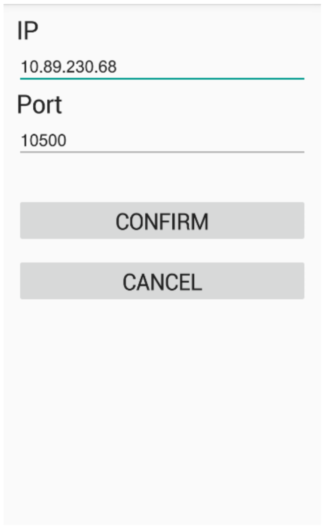

IP
10.89.230.68
Port
10500

Figure 16 UI for IP setting page

2.3.14 Chrome App

Javascript is used in the Chrome App. In background.js, tcpServer is get using chrome.sockets.tcpServer and tcpSocket is get using chrome.socket.tcp. tcpServer is created and listen to the Local IP address of the PC with port 10500. Add the typical OnAccept variable to tcpServer.onAccept.addListener(). The onReceive variable is implemented to send the message to Chrome extension. In the onReceive variable, sent data from the Android App is retrieved using the arrayBufferToString() from receiveInfo.data and is decoded using decodeURIComponent(). Then, packaged with JSON format and then call chrome.runtime.sendMessage() to send the data to Chrome extension with dedicated Chrome extension ID.

The Local IP address is retrieved using RTCPeerConnection() the use the implemented WebRTC by Chrome to allow request to Session Traversal Utilities for NAT server to return the local address. In other word, there is a dummy connection send to STUN server and then get back out local address in the returned packet.

2.3.15 Chrome Extension

Javascript is used in the Chrome Extension. In background.js, chrome.runtime.onMessageExternal.addListener is used to add the function to receive the text passed from Chrome App as the request in the incoming parameter. Then, the text will be set to the current fill in the current tab by using chrome.tabs.query() and chrome.tabs.sendMessage().

3. Testing

Since the App is not complex, most of the logic can be tested manually and simple situation.

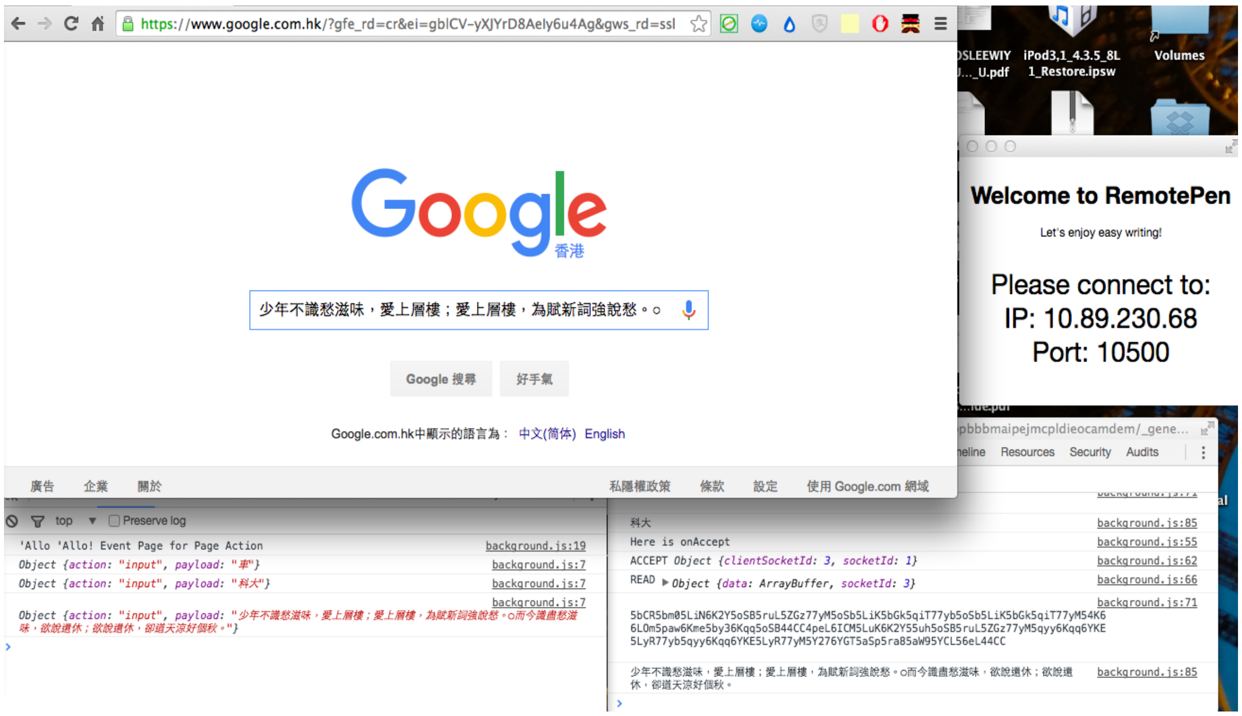
1. Clear the App data to see whether SetPassword page display correctly and force the user to input the set the password before use the APP

2. For input page, try to input and then switch to other page and then switch back to see whether the text is kept. Press send button and then go to History page to check whether the sent text shown in the List.
3. For history page, try click and long click to see whether the dialog shown up correctly. Try the option and see the corresponding result.
4. For bookmark page, check whether bookmarked item shown in the List if the item is added to bookmark in history page. Delete any item to see whether the item disappear in history page.
5. For setting page, click the change password button to see whether the cancel button shown in the page. Enter a new password and then confirm. Test the new password with login page.
6. For login page, try to put the App into background or switch off the screen and then go back to the App to see whether the Login page shown up.
7. For the Chrome part, different websites' text box will be tested for receiving the text.
8. Different platform is tested for connection with Chrome App server and the android APP.

Item 1 to 6 are tested with positive result.

Item 7, the positive result obtained by testing with Google search and yahoo search. Whatsapp and Facebook cannot display the sent text.

Item 8, the existing project work only on Mac platform.

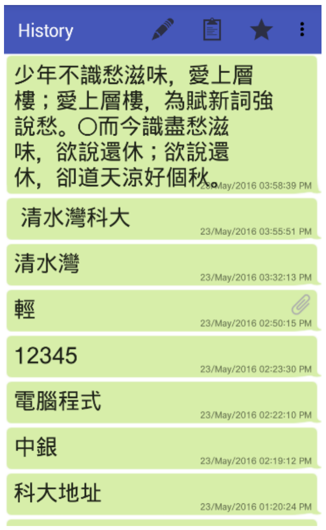

https://www.google.com.hk/?gfe_rd=cr&ei=gblCV-yXJYrD8Aely6u4Ag&gws_rd=ssl
DSLEEWIY …_U.pdf
iPod3,1_4.3.5_8L 1_Restore.ipsw
Volumes
Welcome to RemotePen
Let's enjoy easy writing!
Please connect to:
IP: 10.89.230.68
Port: 10500
少年不識愁滋味，愛上層樓；愛上層樓，為賦新詞強說愁。○
Google 搜尋
好手氣
Google.com.hk中顯示的語言為： 中文(简体) English
廣告
企業
關於
私隱權政策
條款
設定
使用 Google.com 網域
top
Preserve log
pbbbmaipejmcpldieocamdem/_gene…
Timeline
Resources
Security
Audits

```
'Allo 'Allo! Event Page for Page Action                background.js:19
Object {action: "input", payload: "車"}                  background.js:7
Object {action: "input", payload: "科大"}                background.js:7
                                                        background.js:7
Object {action: "input", payload: "少年不識愁滋味，愛上層樓；愛上層樓，為賦新詞強說愁。○而今識盡愁滋
味，欲說還休；欲說還休，卻道天涼好個秋。"}
>
```

```
科大                                                     background.js:85
Here is onAccept                                        background.js:55
ACCEPT Object {clientSocketId: 3, socketId: 1}          background.js:62
READ ▶Object {data: ArrayBuffer, socketId: 3}           background.js:66
                                                        background.js:71
5bCR5bm05LiN6K2Y5oSB5ruL5ZGz77yM5oSb5LiK5bGk5qiT77yb5oSb5LiK5bGk5qiT77yM54K6
6LOm5paw6Kme5by36Kqq5oSB44CC4peL6ICM5LuK6K2Y55uh5oSB5ruL5ZGz77yM5qyy6Kqq6YKE
5LyR77yb5qyy6Kqq6YKE5LyR77yM5Y276YGT5aSp5ra85aW95YCL56eL44CC

少年不識愁滋味，愛上層樓；愛上層樓，為賦新詞強說愁。○而今識盡愁滋味，欲說還休；欲說還   background.js:85
休，卻道天涼好個秋。
>
```

Figure 17 The right hand side is console log for Chrome extension and left hand side is the Chrome App



少年不識愁滋味，愛上層樓；愛上層樓，為賦新詞強說愁。〇而今識盡愁滋味，欲說還休；欲說還休，卻道天涼好個秋。 23/May/2016 03:58:39 PM

清水灣科大 23/May/2016 03:55:51 PM

清水灣 23/May/2016 03:32:13 PM

輕 23/May/2016 02:50:15 PM

12345 23/May/2016 02:23:30 PM

電腦程式 23/May/2016 02:22:10 PM

中銀 23/May/2016 02:19:12 PM

科大地址 23/May/2016 01:20:24 PM

Figure 18 The text sent to Chrome was the top item in the History page

4. Conclusion

A simple and easy to use Android App is designed and implemented for recording the sent text history. However, the function to send the text and receive in the Chrome App is still under progress. Except sending the text, the app can also be a note for user since the App allow user send without connection, then they can save and bookmark the text.

5. Requirement

- Android 4.0 or above
- Recommend screen size larger than 640dp x 480dp
- Support Library:
    - ‘com.android.support:appcompat-v7’
    - ‘com.android.support:recycleview-v7:+’

6. References

[1] w3school-SQL [Online]. Available: http://www.w3schools.com/sql/

[2] Android Developer-Createing List and Card[Online]. Available: https://developer.android.com/training/material/lists-cards.html

[3] Find the local IP using javaScript in Chrome App[Online].Available: http://stackoverflow.com/questions/20194722/can-you-get-a-users-local-lan-ip-address-via-javascript

[4]Decode Base64[Online]. Available: https://developer.mozilla.org/en/docs/Web/API/WindowBase64/Base64_encoding_and_decoding

[5]Click listener in RecyclerView list [Online]. Available: http://www.littlerobots.nl/blog/Handle-Android-RecyclerView-Clicks/