Text Blast

Technical Documentation

1. **Executive Summary**

La Salle Computer Society (LSCS) is the only student organization in the College of Computer Studies. LSCS has been striving to develop its members to become well-rounded Lasallians in their chosen field and in the industry. LSCS also supports its members in the academic workload. LSCS provides various on-campus activities such as programming language seminars, tutorials, fund-raising activities, and exhibits, among others. It also provides off-campus activities such as outreach programs, exchange programs with CS students from other universities within the country, career talks and company exams for students aspiring to be computer professionals to name a few.

1. **Overview**
   1. **Existing Business Process**

LSCS’s existing process in promoting its activities and informing its members about these is by manually texting each of the members. Other than manually texting its members, LSCS also share their publication in social media sites such as Facebook to promote upcoming activities. Although, Facebook is a great avenue to promote activities, some of its members may still not be notified about these since they might not be active on social media. Manually texting the members may be an effective way to inform its members but the process is tedious.

* 1. **Solution**

With that said, developing an Android application is proposed to help promoting LSCS’ activities to its members. All it needs is an excel file containing the name, cellphone number, id number, and course of each member and an Android phone where the application will be installed. This may help lessen the time consumed in manually texting each members in disseminating the activities to its members.

1. **System**

* **System Design**

|  |  |
| --- | --- |
| **MainActivity** | **Description** |
| onCreate | Sets the layout of the screen. Initializes the action listener for buttons. |
| onActivityResult | Sets the filename to the shared preference. |
| blastAll | Blast the message to all numbers in the excel file. |
| blastFilter | Calls the BlastActivity class to blast the message to selected members. |
| getPath | Returns the location or path of the excel file. |
| getDataColumn | Converts the URI to the actual file path of the excel file. |
| onResume | Asks the user to enter the password on resume of the application. |
| readExcelFile | Reads the excel file and returns the array list containing all the cell numbers. |
| alwaysSendSMS | Always send the text message to the listed numbers. |
| send | Sends the text message to the array list of cell numbers. |

|  |  |
| --- | --- |
| **BlastActivity** | **Description** |
| onCreate | Sets the layout of the screen. Initializes the action listener for buttons. Displays the list of ID Numbers and Courses found in the excel file. Blast the text message to the chosen filters. |
| initializeIDNumber | Returns an array list of Filters for all ID numbers found in the excel file. |
| initializeCourses | Returns an array list of Filters for all courses found in the excel file. |
| checkIfInList | Returns true if the ID Number already exist in the array list, returns false otherwise. |
| checkIfInList2 | Returns true if the Course already exist in the array list, returns false otherwise. |
| filter | Returns the array list containing the filtered cellphone numbers. |
| getNumber | Returns the correct cellphone number format to be used for sending. |

|  |  |
| --- | --- |
| **BlastDialogFragment** | **Description** |
| onCreateDialog | Returns the blast dialog box. |

|  |  |
| --- | --- |
| **SaveActivity** | **Description** |
| onCreate | Sets the layout of the screen. Calls the Save Dialog Fragment. |
| notSaveAll | List all cellphone numbers vertically allowing the user to select which numbers to save. |
| saveAll | Saves all the numbers to your contact list. |
| readExcelFile | Reads the excel file and returns the array list containing all the cellphone numbers. |
| addToContacts | Save each contact to the phone directory. |
| getGroupId | Returns the group ID if group exists, otherwise create new group and return its group ID. |
| ifGroup | Checks if group already exist, if yes, return group ID. |
| addContactToGroup | Adds each contact to the created group. |

|  |  |
| --- | --- |
| **SaveDialogFragment** | **Description** |
| onCreateDialog | Returns the save dialog box. |

|  |  |
| --- | --- |
| **LoginActivity** | **Description** |
| onCreate | Sets the layout of the screen. Initializes the action listener for buttons. Asks the user to enter the 4-digit passcode. |

|  |  |
| --- | --- |
| **SettingsActivity** | **Description** |
| onCreate | Sets the layout of the screen. Initializes the action listener for buttons. |
| onResume | Asks the user to enter the password on resume of the application. |

|  |  |
| --- | --- |
| **SetupActivity** | **Description** |
| onCreate | Sets the layout of the screen. Initializes the action listener for buttons. Checks if the new passcode matches the passcode in the confirm textbox. |
| onResume | Asks the user to enter the password on resume of the application. |

|  |  |
| --- | --- |
| **Contact** | **Description** |
| Contact | Constructor for Contact, sets the name and number of the contact. |
| getName | Returns the name of the member. |
| setName | Sets the name of the member. |
| getNumber | Returns the cell number of the member. |
| setNumber | Sets the cell number of the member. |
| isSelected | Returns true if the member is selected. |
| setSelected | Sets to true if the contact is selected; sets to false otherwise. |

|  |  |
| --- | --- |
| **Filter** | **Description** |
| Filter | Constructor for Filter, sets the name and kind of the filter. |
| getFilter | Returns the filter name. |
| setFilter | Sets the filter name. |
| isSelected | Returns true if the filter is selected. |
| setSelected | Sets to true if the filter is selected; sets to false otherwise. |
| getKind | Returns type of filter. |
| setKind | Sets the kind of filter (ID Number or Course). |

* 1. **Functionalities**
     1. **Officer**

|  |
| --- |
| **Task #1**: The user will register by entering a valid 4-digit passcode, name of the organization, and name of the user. |
| Pre-condition:  User must have installed the application in his Android phone. |
| Scenario:  1. The user will open the application.  2. The user will to enter his 4-digit passcode, the name of the organization, and his name.  3. The user will save the information.  4. The application saves the information.  5. The system will direct the user to the Main Screen. |
| Post-condition:  The user will be registered to the application and be at the Main Screen. |
| Acceptance criteria:  1. Test that the 4-digit passcode is valid. |
| Technical specifics:  After installing the application, since there is no existing account, it will redirect the user to the SetupActivity.java to let the user register. The user will enter his 4-digit passcode, name of the organization, and his name. The information will be save in the shared preferences for future use. After the registration, from SetupActivity.java, the user will be directed back to the Main Screen. |

|  |
| --- |
| **Task #2**: The user will enter the 4-digit passcode which will give access to the Main Screen. |
| Pre-condition:  User must be registered in the application. |
| Scenario:  1. The user will open the application.  2. The user will to enter his 4-digit passcode.  3. The system authenticates the information.  4. The system will direct the user to the Main Screen. |
| Post-condition:  The user will be logged in to the system and be at the Main Screen. |
| Acceptance criteria:  1. Test that the 4-digit passcode is valid.  2. If it is not valid, the user will not be directed to the Main Screen and will stay at the Login Screen until the correct 4-digit passcode is entered.  3. If it is valid, test that the user will be directed to the Main Screen. |
| Technical specifics:  The application checks if there is an existing user. If yes, redirects to LoginActivity.java. The user enters his 4-digit passcode. The application checks if the entered passcode is similar to the passcode stored in the shared preference. If yes, it redirects to the Main Screen, otherwise, it stays at the Login Screen. In case there is no registered user, the application will redirect the user to the SetupActivity.java to let the user register. |

|  |
| --- |
| **Task #3**: The user will select the excel file containing the members’ information. |
| Pre-condition:  The user must have copied the excel file in the mobile device. The file must follow the specified format for it to be read by the application. |
| Scenario:  1. The user will choose the option “Choose Excel File”.  2. The user will find the location of the excel file.  3. The user will select the excel file.  4. The user will enter the 4-digit passcode for verification.  5. The application will display the location of the excel file in the Main Screen. |
| Post-condition:  The user will see the filename of the excel file at the Main Screen. |
| Acceptance criteria:  1. Test that the filename of the excel file is correct.  2. If it is correct, test that the user will be also to see the correct file location Is displayed.  3. To be able to select the excel file, test that the 4-digit passcode is valid.  4. If the 4-digit passcode is not valid, the excel file will not be chosen and the user will stay at the Login Screen until the correct 4-digit passcode is entered. |
| Technical specifics:  From the Main Screen, the user chooses the option “Choose Excel File”. An action listener in the MainActivity.java will then display a File Chooser where the user will find the excel file. The user then chooses the excel file containing the members’ information. The onActivityResult() will then get the file path thru the method getPath() and displays the file path. This will also save the path in the shared preferences for future uses. Lastly, the application will now call LoginActivity.java to ask the user to enter the 4-digit passcode. If the 4-digit passcode is valid, it will redirect to the Main Screen, else, it will stay in the Login Screen until the correct 4-digit passcode is supplied. |

|  |
| --- |
| **Task #4**: The user will choose the option “Settings” in the Main Screen. |
| Pre-condition:  User must be in the main screen and is logged in. |
| Scenario:  1. The user will choose the option “Settings” in the Main Screen.  2. The system will direct the user to the Setting Screen. |
| Post-condition:  The user will be in the Setting Screen. |
| Acceptance criteria:  1. The user must be directed to the Setting Screen. |
| Technical specifics:  From the Main Screen, the user chooses the option “Setting”. An action listener in the MainActivity.java will then redirects it to SettingActivity.java to display the Settings Screen. |

|  |
| --- |
| **Task #5**: The user will change the current passcode in the Settings Screen. |
| Pre-condition:  User must be in the Settings Screen and is logged in. |
| Scenario:  1. The user will choose the option “Change Passcode” in the Settings Screen.  2. The user will enter the current passcode.  3. The application will direct the user to the Change Passcode Screen.  4. The user will enter the new passcode.  5. The user will confirm the new passcode.  6. The user will save the new passcode. |
| Post-condition:  The user has changed his passcode and the passcode stored in the shared preferences will be updated. |
| Acceptance criteria:  1. To proceed in the Change Passcode Screen, test that the current passcode entered is valid.  2. The new passcode must be the same with the passcode in the confirmation text box. |
| Technical specifics:  From the Settings Screen, the user chooses the option “Change Passcode”. An action listener in the SettingsActivity.java will then redirects it to the LoginActivity.java and let the user enter the current passcode. If the current passcode is correct, it will now be directed to the SetupActivity.java. The user will now enter the new 4-digit passcode and confirm the new passcode. If these 2 are similar, the passcode stored in the shared preferences will be updated with the new one and the user will be redirected to the Settings Screen. If they did not match, it stays in the Setup Screen. |

|  |
| --- |
| **Task #6**: The user will choose the option “Save To Contacts” in the Main Screen. |
| Pre-condition:  The user must be in the Main Screen and is logged in. The excel file containing the members’ information must be selected. |
| Scenario:  1. The user will choose the option “Save To Contacts” in the Main Screen.  2. A pop-up dialog box will appear. |
| Post-condition:  The user will be able to see the pop-up dialog box. |
| Acceptance criteria:  1. The user must be directed to the pop-up dialog box. |
| Technical specifics:  From the Main Screen, the user chooses the option “Save To Contacts”. An action listener in the MainActivity.java will then redirects it to the SaveActivity.java. In SaveActivity.java, it will call SaveDialogFragment.java which will be responsible for creating the pop-up dialog box asking the user if he will save all to contacts or to choose who to save. |

|  |
| --- |
| **Task #7**: The user will choose to save all to contacts to phone directory. |
| Pre-condition:  The user must be see the pop-up dialog box. |
| Scenario:  1. The user will choose the option “Yes” in the pop-up dialog box.  2. The application will create a group and save all contacts to phone directory. |
| Post-condition:  All numbers will be saved. |
| Acceptance criteria:  1. A group is created and the numbers are correctly saved in the contacts.  2. Check that the saved number and the number stored in excel file is the same. |
| Technical specifics:  If the user chooses to save all to contacts, then the method SaveActivity.saveAll() will be executed. In SaveActivity.saveAll(), it will first read the excel file by calling the readExcelFile(filename) method and passing the filename stored in the shared preferences and after that, it will call the AsyncTask. The AsyncTask will then call the addToContacts(contacts). From the method addToContacts(contacts), it will add each contact to a group (default name of group is the name of the organization) by calling addContactToGroup(). In addContactToGroup(), it will first try to get the group ID by calling getGroupId() if there is already an existing group. If there is no group, there a new group will be created and then return its group ID. After successfully saving all to contacts, this will redirect the user back to the Main Screen. |

|  |
| --- |
| **Task #8**: The user will choose to save selected members to phone directory. |
| Pre-condition:  The user must be see the pop-up dialog box. |
| Scenario:  1. The user will choose the option “No” in the pop-up dialog box.  2. The application will direct the user to the Save Screen. |
| Post-condition:  The selected numbers will be saved. |
| Acceptance criteria:  1. A group is created and the selected numbers are correctly saved in the contacts.  2. Check that the saved number and the number stored in excel file is the same. |
| Technical specifics:  If the user chooses to save selected members, then the method SaveActivity.notSaveAll() will be executed. In the SaveActivity.notSaveAll(), it will first list all the names of the members found in the excel file. The user will now choose which of the members he wishes to save. An array list of the selected contacts will be created. Only the members in the array list will be save to contacts by calling the AsyncTask. The AsyncTask will then call the addToContacts(contacts). From the method addToContacts(contacts), it will add each contact to a group (default name of group is the name of the organization) by calling addContactToGroup(). In addContactToGroup(), it will first try to get the group ID by calling getGroupId() if there is already an existing group. If there is no group, there a new group will be created and then return its group ID. After successfully saving to contacts, this will redirect the user back to the Main Screen. |

|  |
| --- |
| **Task #9**: The user will choose the option “Blast” in the Main Screen. |
| Pre-condition:  User must be in the main screen and is logged in. The excel file containing the members’ information must be selected. The text message must be supplied in the Main Screen. |
| Scenario:  1. The user will choose the option “Blast” in the Main Screen.  2. A pop-up dialog box will appear. |
| Post-condition:  The user will be able to see the pop-up dialog box. |
| Acceptance criteria:  1. The user must be directed to the pop-up dialog box. |
| Technical specifics:  From the Main Screen, the user chooses the option “Blast”. In MainActivity.java, it will call BlastDialogFragment.java which will be responsible for creating the pop-up dialog box asking the user if he will blast the message to all or not. |

|  |
| --- |
| **Task #10**: The user will choose to blast the message to all. |
| Pre-condition:  User must be see the pop-up dialog box. |
| Scenario:  1. The user will choose the option “Yes” in the pop-up dialog box.  2. The application will send the message to all members in the excel file. |
| Post-condition:  A text message is sent to all the members in the excel file. |
| Acceptance criteria:  1. The text message is sent to all members.  2. The text message is received by the members. |
| Technical specifics:  If the user chooses to blast the message to all, then the method MainActivity.blastAll() will execute an AsyncTask which allows the application to perform background operations and display the result on the UI thread after successfully accomplishing the task. In the AsyncTask, it will first get the numbers of the members by calling and passing the filename as a parameter to readExcelFile(filename). After returning an array list consisting of the numbers, it will now send the text message by calling the method send(message, number). A pop-up progress bar will be shown while the sending each text message to let the user see the progress. |

|  |
| --- |
| **Task #11**: The user will choose to blast the message to selected members. |
| Pre-condition:  User must be see the pop-up dialog box. |
| Scenario:  1. The user will choose the option “No” in the pop-up dialog box.  2. The application will send the message to the selected members. |
| Post-condition:  A text message is sent to the selected members. |
| Acceptance criteria:  1. The text message is sent to the selected members.  2. The text message is received by the selected members. |
| Technical specifics:  If the user chooses to not blast the message to all, then the method MainActivity.blastFilter() will be executed. In MainActivity.blastFilter(), it will direct the user to the BlastActivity.java. In BlastActivity, two methods namely, initializeIDNumbers() and initializeCourses() will be called to initialize and display the two type of filters: (1) ID Number and (2) Course. The user will choose what type of filter he wants and this will be added to the array list. The numbers of the members based on the selected filter will be return by the method filter(filename, filters, columnname). After acquiring the numbers, it will call the AsyncTask. In the AsyncTask, it will now send the text message by calling the method send(message, number). A pop-up progress bar will be shown while the sending each text message to let the user see the progress. |

* 1. **Development Environment Setup**
* Android Studio (for development)
* GenyMotion or Android Device (for testing)

1. **Proponents**

Robee Khyra Mae J. Te

Assistant Vice President for Research and Technology Development

La Salle Computer Society

BS in Computer Science with Specialization in Software Technology

robee\_te@dlsu.edu.ph

09178766655

1st Term – 2nd Term AY 2016-2017

XGB

Assistant Vice President for Research and Technology Development

La Salle Computer Society

BS in Computer Science with Specialization in Software Technology

<email?>

<cell no?>

<term?>