**Test Plan Version 1.0**

**Team: Shortcuts**

**Team Members: Stephen Bowder and John Hibbert**

**CSC305 Spring 2015**

**University of Rhode Island**

**Dr. Mello-Stark**

Table of Contents

[1: Introduction 3](#__RefHeading__1036_561267090)

[1.1: Overview and Test Philosophy 3](#__RefHeading__1038_561267090)

[1.2: Test Environment 3](#__RefHeading__1040_561267090)

[2: Test Phase Description 4](#__RefHeading__1042_561267090)

[2.1 Unit Testing 4](#__RefHeading__1044_561267090)

[2.2 Integration Testing 4](#__RefHeading__1046_561267090)

[2.3 System Testing 5](#__RefHeading__1048_561267090)

[2.4 Regression Testing 5](#__RefHeading__1050_561267090)

[3: Responsibilities 6](#__RefHeading__1052_561267090)

[4: Test Case Description 7](#__RefHeading__1054_561267090)

[5: Test Schedule 13](#__RefHeading__1056_561267090)

[6: Error Resolution Management 14](#__RefHeading__1058_561267090)

[7: Issues / Risks / Assumptions 15](#__RefHeading__1060_561267090)

[Appendix 1: Test Plan Gannt Chart 16](#__RefHeading__1062_561267090)

# 1: Introduction

## 1.1: Overview and Test Philosophy

The test philosophy that we are using for this product is based on software it is being written with. As Android API is an object-oriented language, the natural cohesion of well designed objects allows the project to be divided into manageable pieces. Once those pieces are in place, we need to check that they work together by building the classes that are manipulating the objects.

## 1.2: Test Environment

This project has a low budget so this software will be primarily tested on the laptops that it will be programmed on. The tests will use the Android Virtual Device Manager incorporated in the Android Studio. Later in the process, if time allows, the program may be tested on a second generation Nexus 7.

# 

# 

# 2: Test Phase Description

This project will be tested with a bottom-up scheme: individual units will be tested first for their core functionality. Following that, the integration between the main modules will be tested. Once it is properly integrated, the overall system can be tested to confirm everything is still working correctly. After the first release, the response from clients will provide a starting point for our regression testing.

## 2.1 Unit Testing

* State all units and their corresponding functions
* State all possible input (including errors) and the corresponding output for each function
* State how the procedures handle exceptions
* Describe any drivers or stub codes

Testing Objectives:

* Ensure Unit stability for Integration testing
* Create a pool of reusable test data for the next test phases and future testing.

## 2.2 Integration Testing

* Test each module with associated/dependent modules
* Test input and information flow between modules
* Ensure returned data is consistent with expected output when passed between modules
* Bottom-up testing

Testing Objectives:

* To confirm proper communication and coupling between application modules.

Integration Testing Task Overview:

* Main\_Activity sends an account to PhotoView\_Activity.
* PhotoView\_Activity invokes Camera\_Wrapper to call the integrated camera.
* PhotoView\_Activity invokes Photo\_Manager to delete pictures.

## 2.3 System Testing

* Test top-down interaction between system modules
* Test all functionality of each module (including exception handling)

Testing Objectives:

* To produce a product-ready application.

System Testing Task Overview:

* Create one hundred accounts to test if application runs slowly.
* Create one thousand accounts to test if application runs at all.
* Add one hundred pictures to an account to test if application runs slowly.
* Add one thousand pictures across all applications to test if application runs at all.

## 2.4 Regression Testing

* Retest any affected modules after bugfix / alterations to code

Testing Objectives:

* Confirm continued application functionality after future software changes and updates.

Regression Testing Task Overview:

* Main\_Activity displays all accounts on device.
* Main\_Activity sorts accounts alphabetically by last name.
* Main\_Activity sends an account to PhotoView\_Activity.
* Create one hundred accounts to test if application runs slowly.
* PhotoView\_Activity invokes Camera\_Wrapper to call the integrated camera.

# 3: Responsibilities

Stephen will be working on testing preparation, test activities and issue resolution for errors involving the PhotoView\_Activity, Photo\_Manager, and Camera Wrapper. For these errors, John will assess and accept the test results.

John will be working on testing preparation, test activities and issue resolution for errors involving the Account, Main\_Activity, Account\_Mananger, and Data\_Manager. For these errors, Stephen will assess and accept the test results.

# 4: Test Case Description

|  |  |  |
| --- | --- | --- |
| **Unit Testing** | | |
| **Main\_Activity** | | |
| **Testing** | **Input** | **Output** |
| listAccounts():List<Account> | No input. Calls from Account\_Manager instance. | A list of Accounts. Each Account in list should be populated with data. (Minimally AccountId) |
| displayAccounts():void | No input. Method call to create/display main screen’s ListView. | A new ListView should be created. Console will display if ListView was successfully created. |
| searchAccounts(String):List<Account> | User input read from textbox. Can be any set of characters, max length will be 25. Strings greater than 25 will be trimmed to proper length. | A returned list containing all accounts containing all or some matches to input. |
| addAccount(Array[String]):void | Passes an Array of Strings gathered from user input within a textbox. Strings greater than 25 will be trimmed to proper length. | Console output confirms addition. |
| editAccount():void | User input within textbox is collected by the function. | Console output will confirm update is complete. |
| deleteAccount(Account):void | Account to be deleted is passed to Account\_Manager. Null objects will throw a warning message and cancel the process. Accounts missing an ID will throw a warning and cancel the operation. | Console output confirms the Account was passed to Account\_Manager. |
| viewAccount(Account):void | Account is passed to the created PhotoView\_Activity. Null objects will throw a warning message and cancel the process. Accounts missing an ID will throw a warning and cancel the operation. | A new PhotoView\_Activity is created upon function call. |
| **Account\_Manager** | | |
| **Testing** | **Input** | **Output** |
| getAccounts():List<Account> | No input. Calls to Data\_Manager. | Returns a list of Account objects, populated with data from Data\_Manager. |
| addAccount(Array[String]):void | Passes Array of Strings to Data\_Manager. Strings will be trimmed to 25 characters. | No output. Console will confirm function call. |
| searchNames(String):List<Account> | A String passed from Main\_Activity is passed to the function. Strings greater than 25 will be trimmed to proper length. | Returns all matching results within a List of Accounts. |
| updateAccount(Account):void | Account passed with updated information for device storage. Null objects will throw a warning message and cancel the process. Account object missing properties for any fields (except ID) will be processed. Accounts missing an ID will throw a warning and cancel the operation. | Console output confirms Account was updated. |
| removeAccount(Account):void | Passed Account object flagged for deletion from Main\_Activity. Null objects will throw a warning message and cancel the process. Accounts missing an ID will throw a warning and cancel the operation. | Console output confirms deletion. |
| **Account** | | |
| **Testing** | **Input** | **Output** |
| setID(int):void | Integer for AccountId is passed from Account\_Manager. | Console confirmation. |
| getID():int | No input. | Returns the AccountId as an integer. |
| setFirstName(String):void | Passed String to set the new Account FirstName. | Console confirmation. |
| getFirstName():String | No input. | Returns the Account FirstName as a String. |
| setLastName(String):void | Passed String to set the new Account LastName. | Console confirmation. |
| getLastName():String | No input. | Returns the Account LastName as a String. |
| setEmail(String):void | Passed String to set the new Account Email. | Console confirmation. |
| getEmail():String | No input. | Returns the Account Email as a String. |
| **Data\_Manager** | | |
| **Testing** | **Input** | **Output** |
| getAccounts():List<Array[String]> | No input. | Returns a List of Arrays containing 3 Strings. Missing data file will create a new, empty file. |
| updateAccount(Account):void | Account passed with updated information for device storage. Null objects will throw a warning message and cancel the process. Account object missing properties for any fields (except ID) will be processed. Accounts missing an ID will throw a warning and cancel the operation. | Console output confirms Account was updated. |
| **PhotoView\_Activity** | | |
| **Testing** | **Input** | **Output** |
| loadPhotos(Account):List<Object> | Function uses passed Account to associate ID for photos. Null objects will throw a warning message and cancel the process. Accounts missing an ID will throw a warning and cancel the operation. | Returns a List of all associated photo objects on the device. |
| displayPhotos():void | No input. | Console output confirms GridViewLayout is created and populated. |
| addPhoto():void | No input. | New Camera\_Wrapper instance is created. Console output will confirm instance creation. |
| deletePhoto():void | Selected photo object passed to Photo\_Manager is collected by this function. | Console output will confirm photo object flagged for deletion is passed. |
| returnToMenu():void | Test system’s secondary UI class. | New activity with organized photos. |
| **Photo\_Manager** | | |
| **Testing** | **Input** | **Output** |
| add(Object):void | Stores picture object to device with specific naming convention. | Console output will confirm storage. |
| delete(int, int):void | Account ID and Photo ID passed to locate specific photo object to delete. | Console output will confirm object deletion. |
| getPhotos(int):List<Object> | AccountId as an integer used to associate photos on device. | Returns all associated photos on the device as a List. |
| **Camera\_Wrapper** | | |
| **Testing** | **Input** | **Output** |
| takePicture():Object | No input. | Returns an object (Picture) to Photo\_Manager. |

|  |  |  |
| --- | --- | --- |
| **Integration Phase Testing Summary** | | |
| **Units** | **Test Summary** | **Expected Results** |
| Main\_Activity / Account\_Manager | Load the main UI class of the system with any available accounts. | Loads a new activity with a ListView containing stored Accounts. |
| Account\_Manager / Data\_Manager | Create and manage Accounts via functions, including:   * Load * Add * Update * Remove | Depending on tested functionality:   * Load: Return a list of Accounts from system * Add: Save a passed Account to system file * Update: Update passed Account to system file * Remove: Remove passed Account from system file |
| Account\_Manager / Account | Create a list of Accounts with each storing data members. | A List of Account objects, each with associated data members. |
| Main\_Activity / PhotoView\_Activity | \*\* System Testing \*\* |  |
| PhotoView\_Activity / Photo\_Manager | Load system’s secondary UI class with stored photos associated with predetermined account. | New activity with organized photos associated to the argument Account. |
| PhotoView\_Activity / Camera\_Wrapper | Call device hardware to take a new photo. | New view with camera preview to take a new photo. |
| Photo\_Manager / Camera\_Wrapper | Take a new photo and store it to the associated Account. | A photo stored on the system with unique identifier to associate it with the associated Account. |

|  |  |  |
| --- | --- | --- |
| **System Phase Testing Summary** | | |
| **Modules** | **Test Summary** | **Expected Results** |
| Main\_Activity / PhotoView\_Activity | Load the main UI class of the system with any available accounts. | Loads a new activity with a ListView containing stored Accounts. |

# 5: Test Schedule

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Task Name | Start | Finish | Effort | Comments |
| 3.1 Add Account | 4/20/15 | 4/21/15 | 2d | Stephen builds, John Tests |
| 3.2 Edit Account | 4/20/15 | 4/21/15 | 2d | Stephen builds, John Tests |
| 3.3 Delete Account | 4/22/15 | 4/23/15 | 2d | Stephen builds, John Tests |
| 3.4 Add Photograph to Account | 4/20/15 | 4/21/15 | 2d | John builds, Stephen Tests |
| 3.5 Delete Photograph | 4/22/15 | 4/23/15 | 2d | John builds, Stephen Tests |
| 3.6 Search Accounts | 4/29/15 | 4/30/15 | 2d | Stephen builds, John Tests |
| 3.7 List Accounts | 4/27/15 | 4/28/15 | 2d | Stephen builds, John Tests |
| 3.8 View Account | 4/23/15 | 4/24/15 | 2d | John builds, Stephen Tests |
| 3.9 Backup Data | 4/29/15 | 5/1/15 | 3d | John builds, Stephen Tests |
| 3.10 Restore to Backup | 5/1/15 | 5/4/15 | 3d | John builds, Stephen Tests |

# 6: Error Resolution Management

Bugs will be classified into three classifications: *Critical*, *Normal*, and *Trivial*. *Critical* refers to all bugs that affect the core functionality of the app as described in the requirements document. These bugs will be given top priority and must be resolved before the app ships. *Normal* refers to bugs that are causing the app to fail to meet the concepts included in the design document. These bugs are important, but the app can ship with them if necessary. These bugs should he handled as often as is practicable. *Trivial* bugs are bugs that involve features not involving any features included in previous documentation. These bugs are given the lowest priority, and should only be worked on if the estimated person-hours is less than two.

Once the application has been launched, two more categories of bugs will be added for regression testing. *Reported*, and *Cannot Reproduce*. New bugs sent in from customers will be initially given the placeholder heading *Reported*. The test engineers will work to duplicate the bugs. If confirmed the lead test engineer will assign it one of the three original classifications. If the test engineers cannot reproduce the bug, it will be assigned *Cannot Reproduce* and will be ignored.

# 

# 

# 7: Issues / Risks / Assumptions

This test plan is operating under the following assumptions:

|  |  |
| --- | --- |
| Project Deadline | The project deadline is May 4th, so testing must be finished before that date. |
| Available Resources | The project needs to be completed with two team members. |
| Documentation | For the tests, the contents of the requirements document and the design document should be considered a precondition. |

This test plan incurs the following risks:

|  |  |
| --- | --- |
| Team Member Loss | If the team should lose one of its two members, they will have to be replaced or the entire project will have to be delayed. |
| Emulation Problems | As the team will not be doing much, if any, testing on physical devices, any unknown differences between the Android Virtual Device and the target physical device could cause problems during testing, or cause unanticipated bugs to appear in the software after release. |

# Appendix 1: Test Plan Gannt Chart

