Self-Health UM Extension

High Level Design Document

Green Team

11/06/2014

Noah Bilgrien
Molly McMahon
Zachary Hardy
Timothy Contois
Arul Garimella
Calvin Mei
Edward Murphy
Paul Edwards
Ryan Ballas
Shaoxi Ma

TABLE OF CONTENTS

INTRODUCTION

The purpose of this document is to give a high level description of how we will add the functionality outlined in the Requirements Specification document to the existing SelfHealth-UM system. Here, we provide a general overview of the structure of an Android application, the architecture of the existing system, and how our new components will fit into this architecture. Our design approaches, additional classes, and potential alterations to the existing system are summarized in the following sections.

GENERAL OVERVIEW AND DESIGN APPROACH

The current SelfHealth-UM system is built to run on the Samsung Galaxy Tab3 tablet, using Java and the Android SDK.

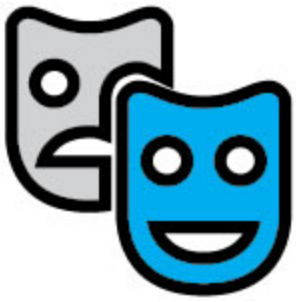
Assumptions

To support wireless configuration, we assume that both the Provider and Client devices are equipped with Bluetooth, and that they have Bluetooth enabled. Additionally, for the Client to make use of Facebook, Skype, etc. via the new app integration features, his or her device must have internet access.

Constraints and Risks

Security: Wireless Import/Export

Given the sensitivity of medical data, we must take precautions to ensure that any data sent between the Client and Provider apps is properly protected. To ensure that malicious devices cannot access the data, the Client and Provider device will be “paired”, in that they know of each other’s existence. This will allow us to add authentication and encryption when sending medical data.



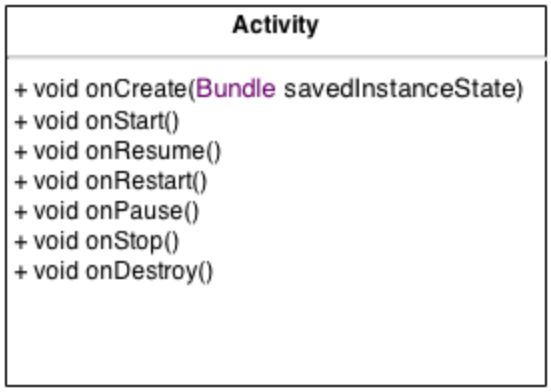
SYSTEM ARCHITECTURE AND DESIGN

Refinements and Restructuring

The existing SelfHealth-UM system contains large portions of duplicate code and behavior between the Provider and Client apps. To minimize this duplication, we may restructure the current package hierarchy by pulling all shared classes and functionality out of the Provider and Client code into a new code library. Ideally, in the new system, the Client and Provider code will reference this library for any behavior or data which is shared between the systems. Examples of shared functionality include database operations, app controller behavior, and survey definitions. This refactoring will take place as time allows - the additions outlined in the Requirements Specification document will take priority.

Additions

In keeping with the existing system, our additions will predominantly follow the **Model-View-Controller** architecture approach. Our *views* handle the display of data (obtained by the controller from the model) on the screens of the Client and Provider applications. In the context of an Android application, these views extend the Activity class, which represents “a single screen with a user interface.”

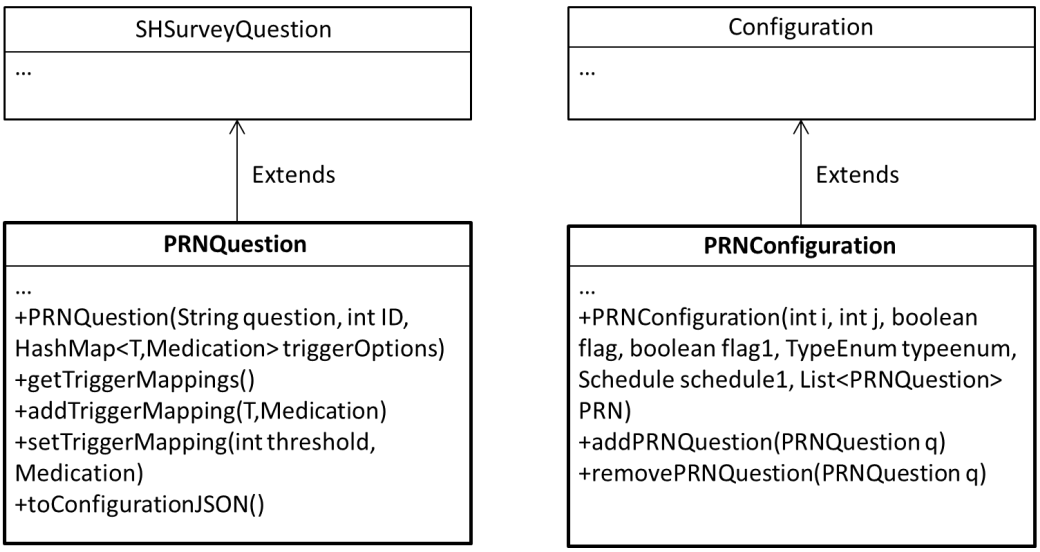


We are adding additional *models* to define the data and behavior associated with the new functionality, and modifying the existing *controller* mechanisms to properly handle storing and retrieving data in/from these new models. We are also adding new *views* to facilitate interaction with the new models. The specific additions and alterations we are making are summarized in the following sections, each corresponding to a new app function.

1. PRN Medications

1.1. Provider Configuration of PRN associations

New functionality needs to be added to the provider app to allow the provider to configure survey questions with PRN triggers. To achieve this, a new PRN configuration view must be created in the provider app. Currently, most of the survey classes are in the Client app. We need to create the same group of classes in the Provider app and also add PRN trigger capability to the configuration. These PRN triggers must be stored with the other data sent over to the client app when exporting a profile.



New Classes:

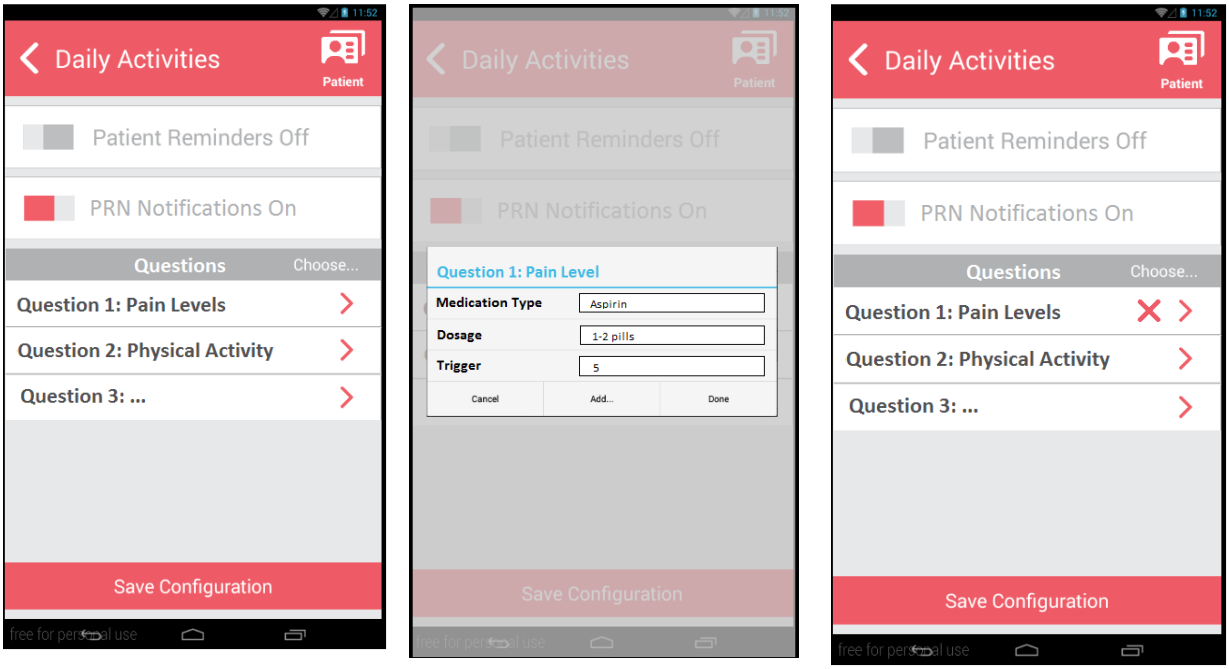
- SHSurvey*
- SHSurveyQuestion*
- SHSurveyResponse*
- PRNQuestion
- PRNConfiguration

Provider app already has Configuration, SurveyConfiguration, SurveyEnum classes.

*These new classes listed above already exist in a package in the Client app, but these classes must also exist in the Provider app to allow PRN configuration.

New Views:

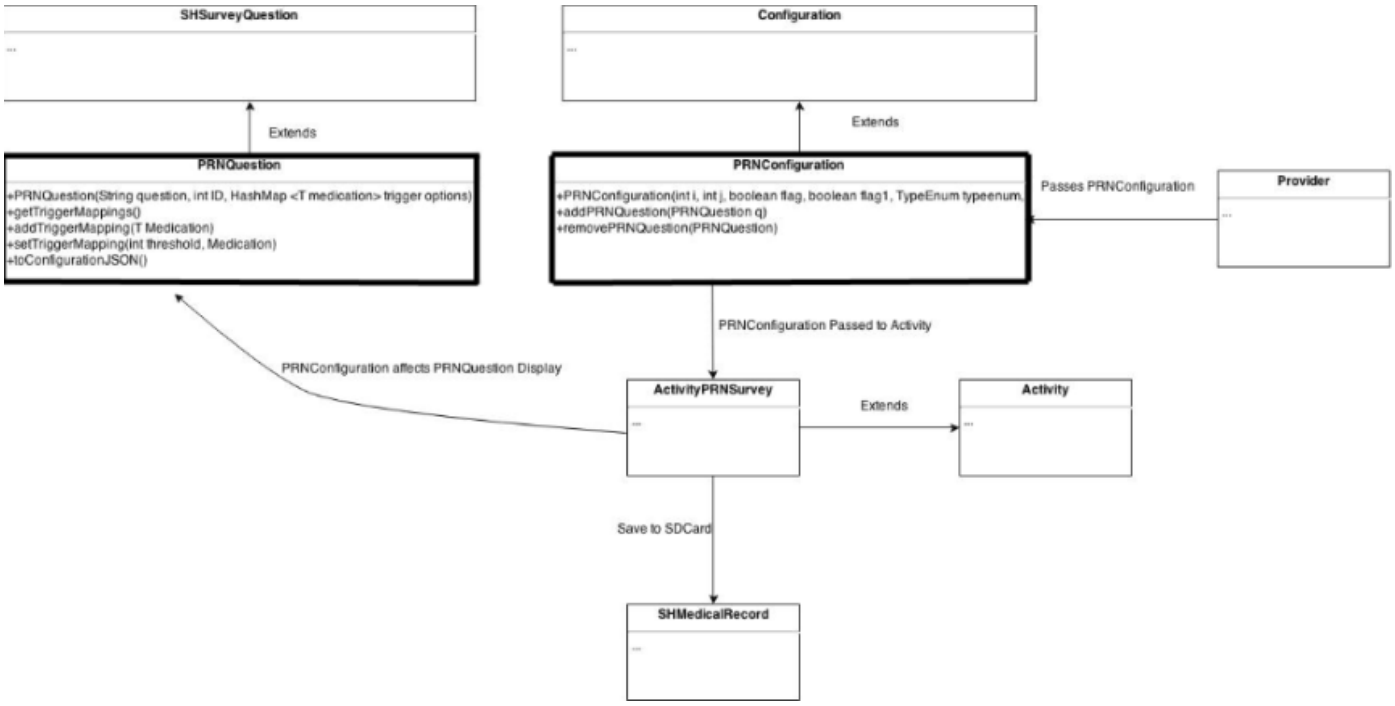
In the Daily Activities section of the Provider app, a new option shall be added allowing the Provider to turn PRN Notifications On/Off. If PRN Notifications are turned on, a list of survey questions shall be displayed. The Provider can choose a survey question, which opens up a new view where the Provider can specify Medication Type, Dosage, and Trigger.



Alterations to Controllers and Existing Classes:

SHSurvey, SHSurveyQuestion, and SHSurvey question are all existing classes in the Client app. When these classes are added to the Provider app, a toConfigurationJSON() method will need to be added. The new classes must also communicate with the DatabaseController.

1.2. Client PRN Survey Prompts

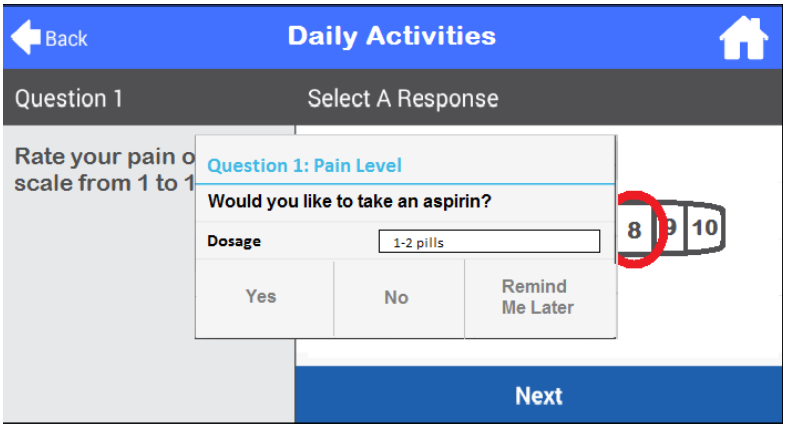


When the client completes a survey, the answers they gave will be checked against the trigger for a PRN. This trigger is handed over to the Client through a class called “SHPRNConfiguration”. The implementation for saving medication information is already set up, therefore, we simply have to make sure the trigger is checked.

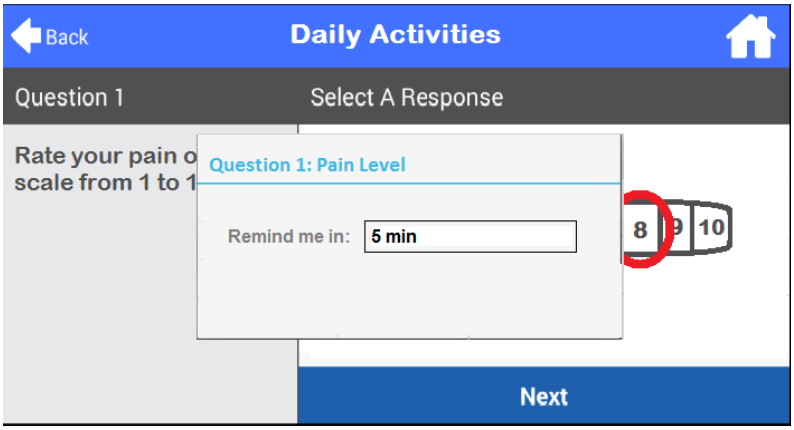
New Classes:

- PRNQuestion
- PRNSurveyConfiguration
- ActivityPRNSurvey

New Views:



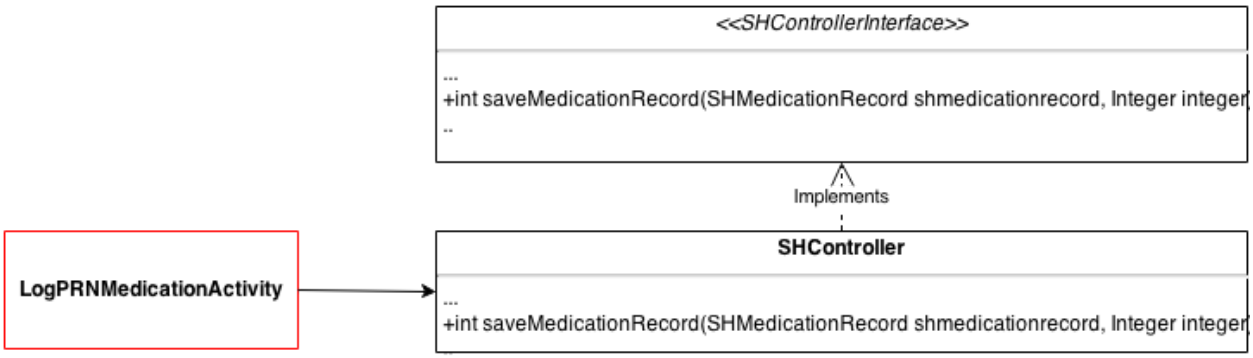
There are no new views in regards to this aspect of the program, but there will be further extension added to the already existing views. This extension will take the form of a dialogue box that the user will be able to interact with to reply to the PRNQuestion triggered by their question to a previous survey.



Alterations to Controllers and Existing Classes:

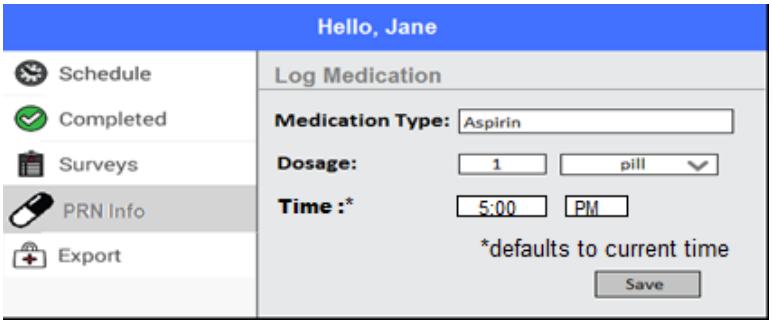
We will have to change SHController to receive the configurations from the provider app and make the appropriate changes to the Client app.

1.3. Client Medication logging



New Views:

We will add a “Log PRN Medication” button to main menu screen of the Client application. When selected, this will lead to a new screen with the header “Log PRN Medication.” This screen will have the appropriate with textboxes, dropdowns, etc. for the patient to enter the medication they took. The information will be retrieved from the view and saved by the controller as a record via a new method called “getPRNInfoRecord().”



Excluding the additional views, all of the behavior associated with saving medication information already exists in the Client application. The medication record will be saved by the controller to the disk in the same manner as before.

2. Wireless Configuration via Bluetooth

The new wireless configuration consists of a Server-Client architecture within overall MVC design. The Client device opens a Bluetooth server socket, the Provider device scans for other Bluetooth devices via a process called “Device Discovery”, and initiates the connection by sending requests to the Client device’s MAC address. Both the Client and Provider Apps can then read and write data via I/O streams opened on the resulting connection. The Client profile configuration will be sent over this connection and saved on the client device.

New Classes and Interfaces:

Client App:

- ImportExportInterface
- WirelessImportExport
- SDImportExport
- ConnectToProvider
- ConnectionThread

Provider App:

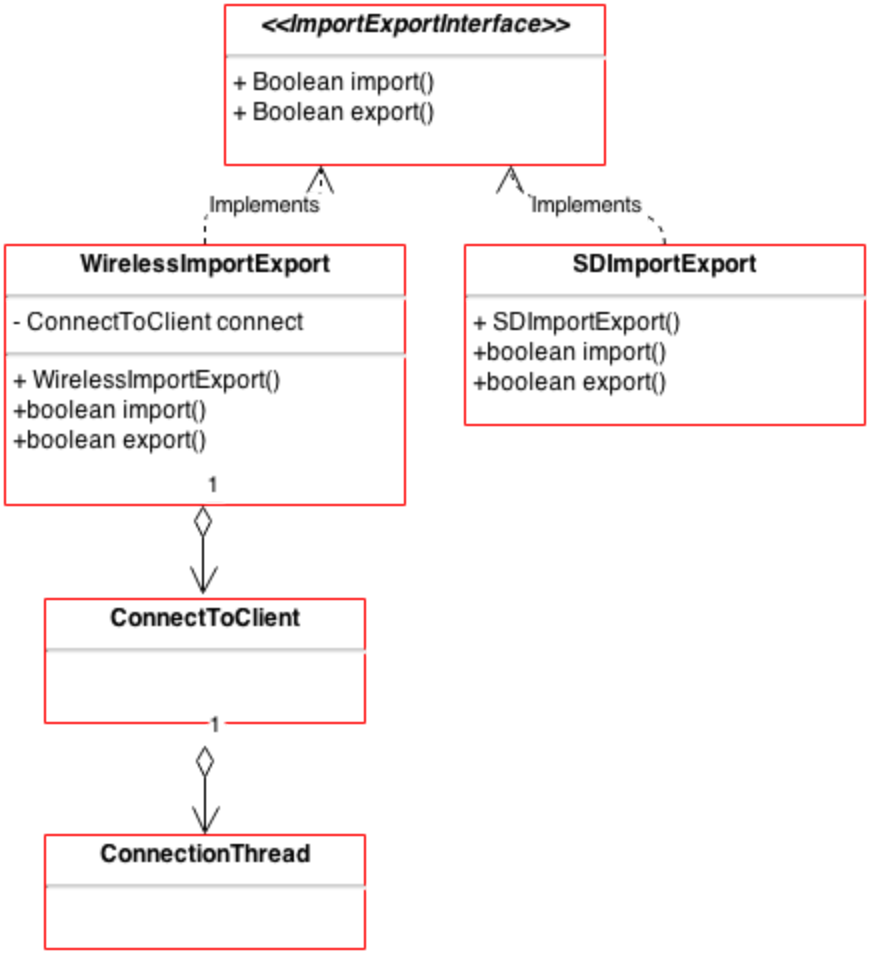
- ImportExportInterface
- WirelessImportExport
- SDImportExport
- ConnectToClient
- ConnectionThread

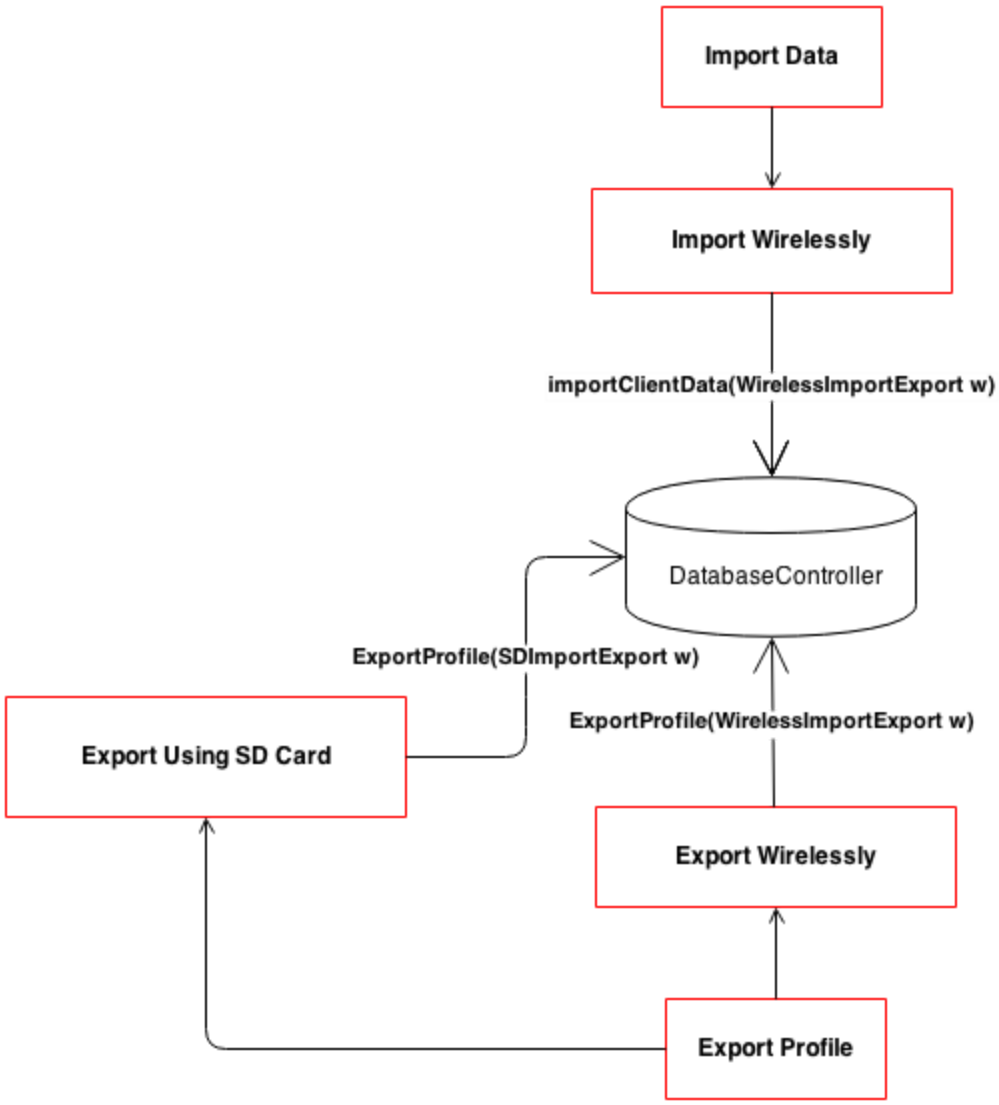
ConnectToClient, ConnectToProvider, and ConnectionThread all extend the Java Thread class, and will all make use of the android.bluetooth API to establish and manage connections.

Alterations to Controllers and Existing Classes:

Provider App:

As shown below, we are extracting the import/export behavior out of the DatabaseController into two new classes, **WirelessImportExport** and **SDImportExport**, both of which implement the interface *ImportExportInterface*. We are adding generic import and export methods to the DatabaseController which take an ImportExportInterface object as a parameter, calling that object’s import or export method. This decouples the import/export behavior from the DatabaseController, leaving it to handle the transferred information.


<<ImportExportInterface>>
+ Boolean import()
+ Boolean export()
Implements
Implements
WirelessImportExport
- ConnectToClient connect
+ WirelessImportExport()
+boolean import()
+boolean export()
SDImportExport
+ SDImportExport()
+boolean import()
+boolean export()
1
ConnectToClient
1
ConnectionThread



Client App:

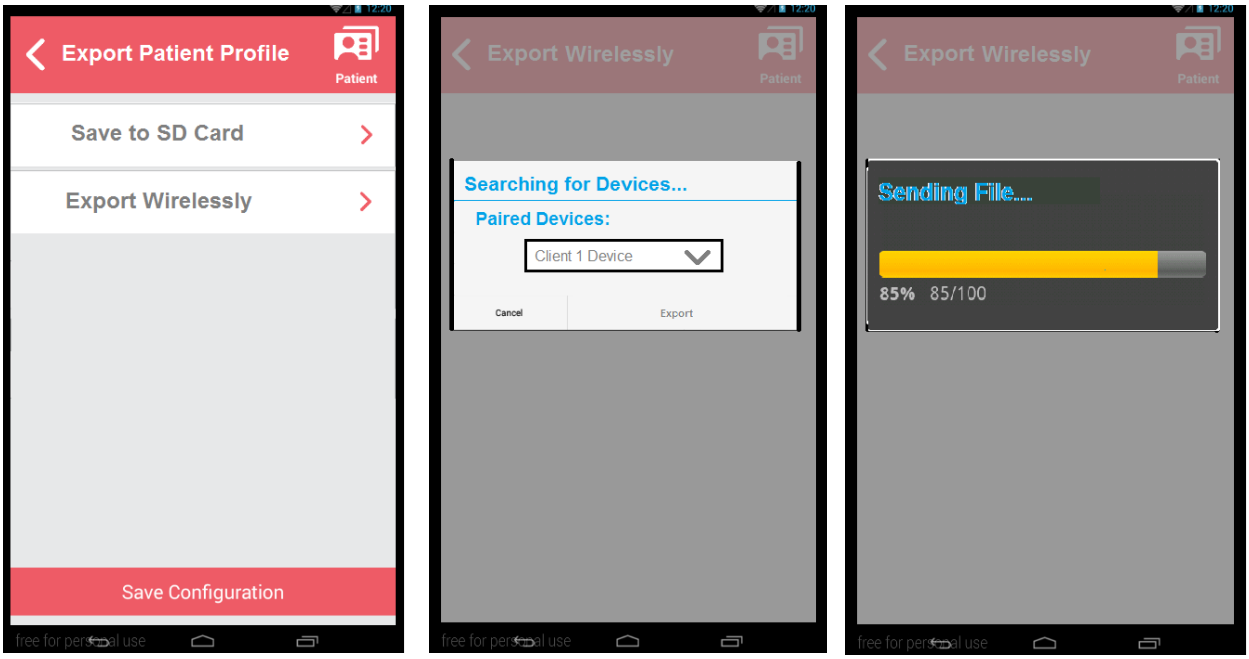
The Client app will have the general same setup (outlined above) as Provider app. The equivalent of the DatabaseController in the Client code is a combination of the SHController and the com.sh.configuration classes. Again, we will extract the import/export behavior into two classes, one for wireless and one for SD, both of which implement a common interface. The controller will then refer to objects of these classes for import/export behavior.

New Views:

Provider App:

In the Provider application, we will add a new "Import Data" option to the main menu, as well as the existing "Export" option. We will add a new intermediary screen with "Save to SD Card" and "Export Wirelessly" options, which is displayed when "Export" is selected. Selecting "Export Wirelessly" will lead to a

new screen for with “Export Wirelessly” header (“Export Wirelessly” rectangle represents this view). “Save to SD Card” simply saves the profile as before. The processes outlined above will take place on these screens, and be tied to the underlying activities.



Client App:
On the main screen of the Client app, we will add a new “Import” option

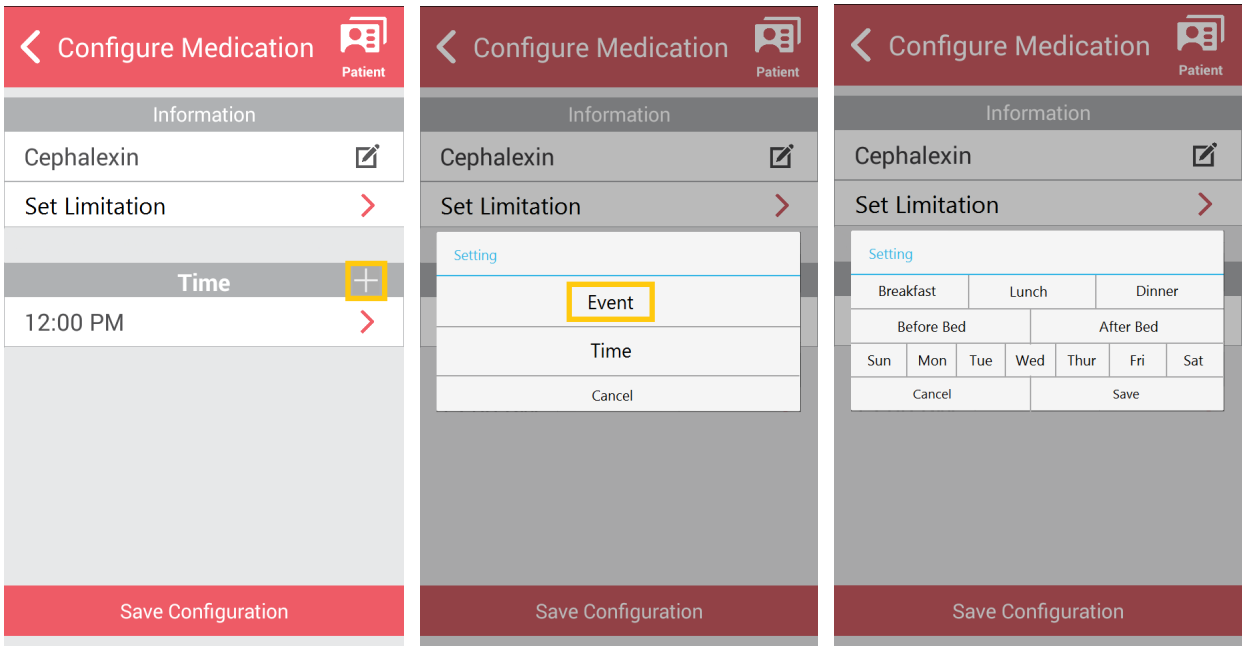
3. Multi-Time Medications

The provider will see an improved user interface in the provider app when adding new medication times. This change reflects the new functionality of adding medications based on “events” and “times,” and will result in the addition of new classes as well as modifications to existing ones. In addition, a new limitation (i.e. total dosage) for each medication may be set under the Information header and this will also require a new class to support the added functionality.

3.1. Event Based Medications

New Views: We will add “Events” that can trigger a medication, instead of a time, allowing for medications to be taken with a meal, before going to bed, and upon waking up in the morning. Now as the provider configures medication times, after clicking the “+” button to add a time they will be prompted with a new screen. There will be 2

buttons on the screen, one labeled “Event”, and one labeled “Time”. Upon clicking the “Event” button, the provider will be prompted with a list of events that can be chosen. There will be 5 different events, Breakfast, Lunch, Supper, Before Bed, and Waking Up. The provider will be able to select one or multiple events for the alarm to trigger. There will also be a list of days that are selected for the chosen events to happen on.

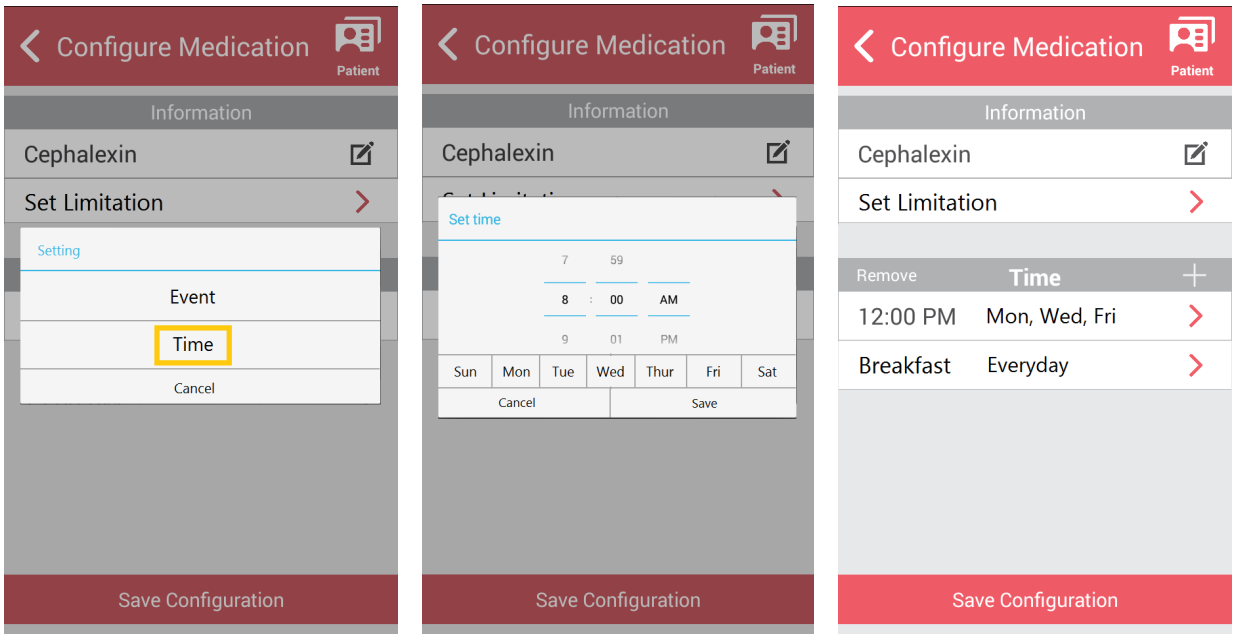


New Classes: There will be a new class added to support the new “Events” functionality. This class will map the events to preconfigured times set by the provider. Although a full new class isn’t required to support the new functionality, this new class will allow new functionality to be added easily and efficiently in the future. This could be the need to receive a message from a person-sensing bed, to alert the user that has entered the bed to take the needed medication before falling asleep.

3.2. Frequency Based Medications

New Views: The method in which the times for medication are set has also been altered. When the provider clicks the “+” button, a new screen will appear with two buttons labeled “Event” and “Time.” Upon clicking the “Time” button, the provider will be able to choose a time of day to take the medication. Once the time is set, the provider may select one or more days of the week at which the medication will be taken.

Alternatively, after selecting the time, the provider may choose to select the “take daily” option which would set up the patient to take the medication every day at the prescribed time of day.

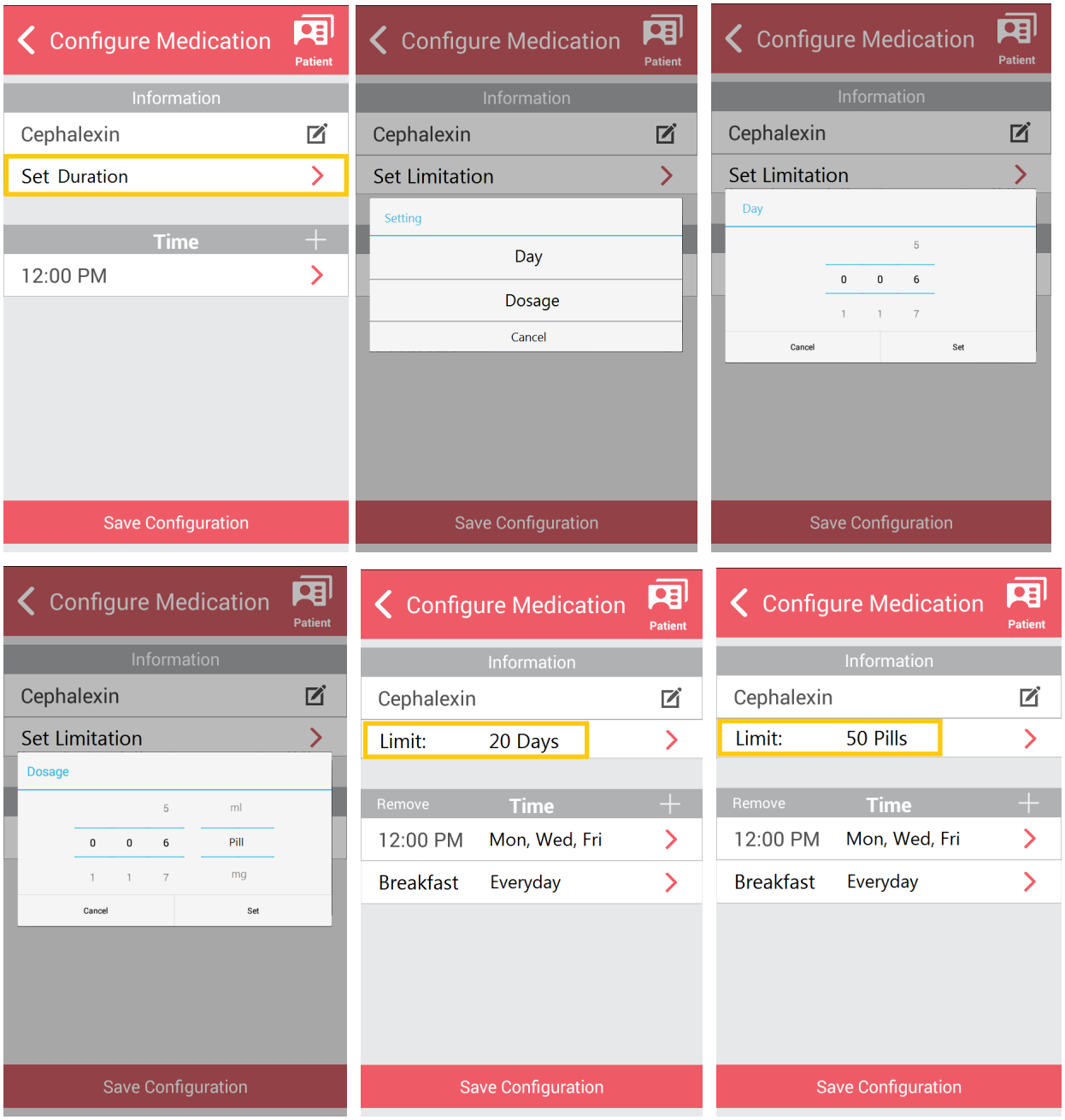


New Classes: There should already be an existing Times class that handles setting a time of day for a medication and so this class will only need to be modified in order to interact with the new sequence of selecting times. However, because there are now two ways of adding medication times (Events and Times), it is possible to add a new interface that will be implemented by both respective classes, as that will also allow new functionality to be added in the future.

3.3. Non-permanent Medications

New Views: We will modify the “Information” section in the medication configuration by replacing the old “Form | Dosage | Unit | Quantity” tab with a new “Duration” tab. Upon clicking the “Duration” tab, the provider can choose one of two new buttons, “Day” or “Dosage”, in the popup box for setting an appropriate duration. The provider can set limiting number of days or set limiting amount of dosages and unit of the medicine in the next popup box. The duration information

will replace the “Duration” tab after setting up. The provider can still change the configuration by clicking the duration information tab. Every popup box contains “Cancel” button to exit the configuration.

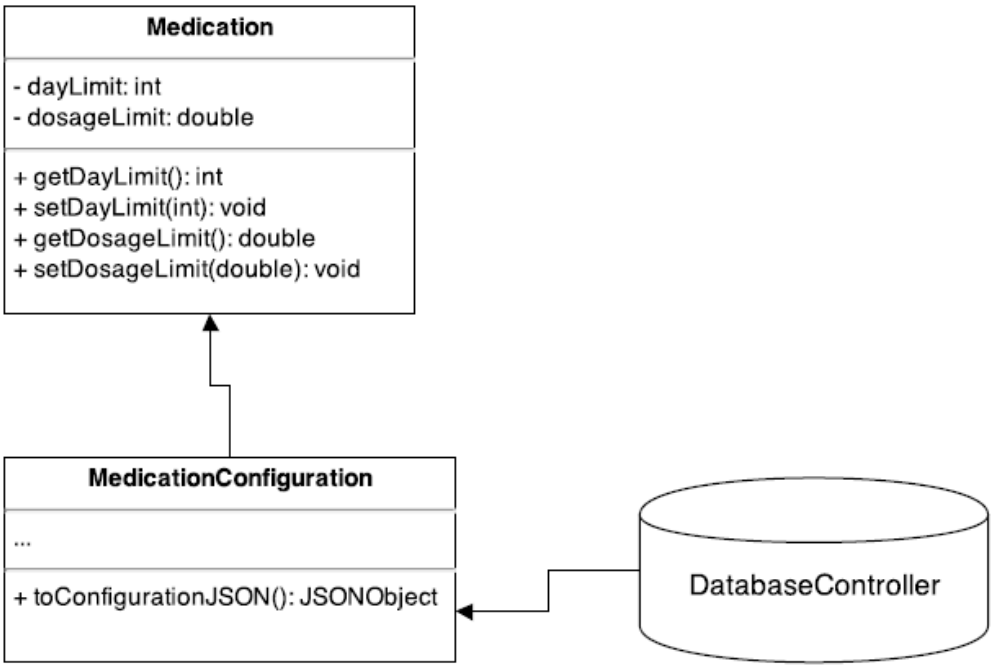


New Classes:

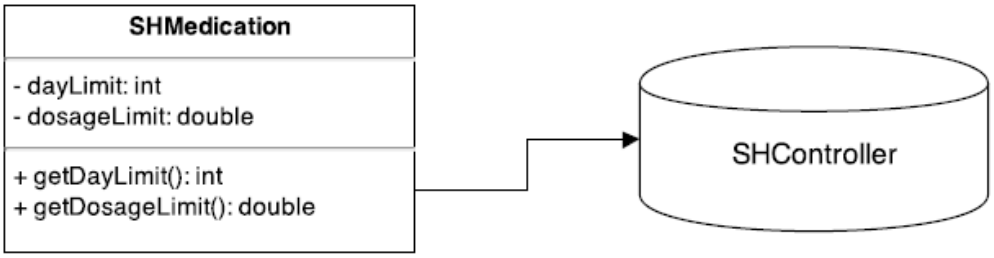
- **Provider app:**
  Duration class may be needed for this function; or we can modify the old class to get and set the limiting numbers.
- **Client app:**

We need to add new variables for saving the limiting number of days or dosages. We also need to modify the logic to trigger notifications according to the duration.

Provider App



Client App



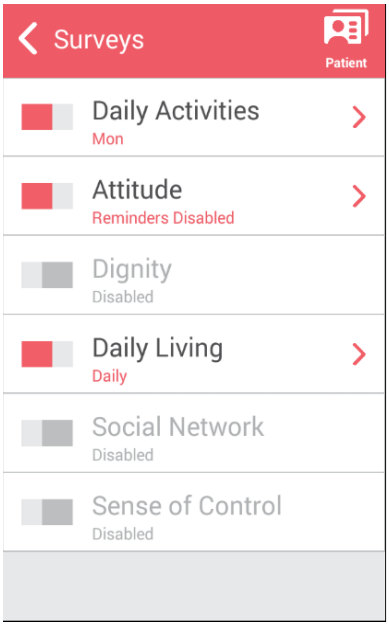
4. App Integration

Our surveys will now include a “Lower” and “Upper” Bound field that will be set by the provider. When the client goes to take a survey, if they answer strays from the threshold (set from the provider), a trigger will occur [Such as a prompt to open the Contacts list].

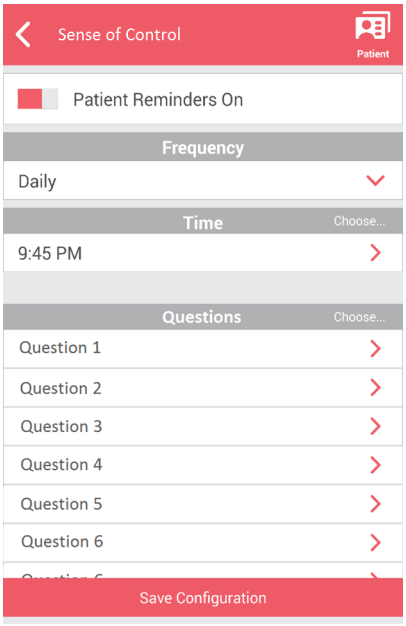
4.1 Configuration of client app's social integration threshold

New Views: We will add an app-integration threshold to all questions of the survey. *(Note: if the provider does not wish for a question to trigger app integration, this can be done by leaving the lower and upper bounds blank.)* When the provider goes to edit a question, they will now see two textboxes to enter lower and upper bounds to trigger app integration. This view will also have a "Back" button to return to the main page of the survey and store the changes until the provider either decides to save or exit without saving.

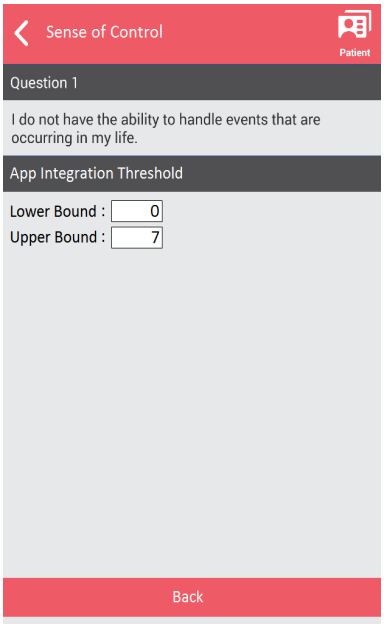
Provider selects a survey

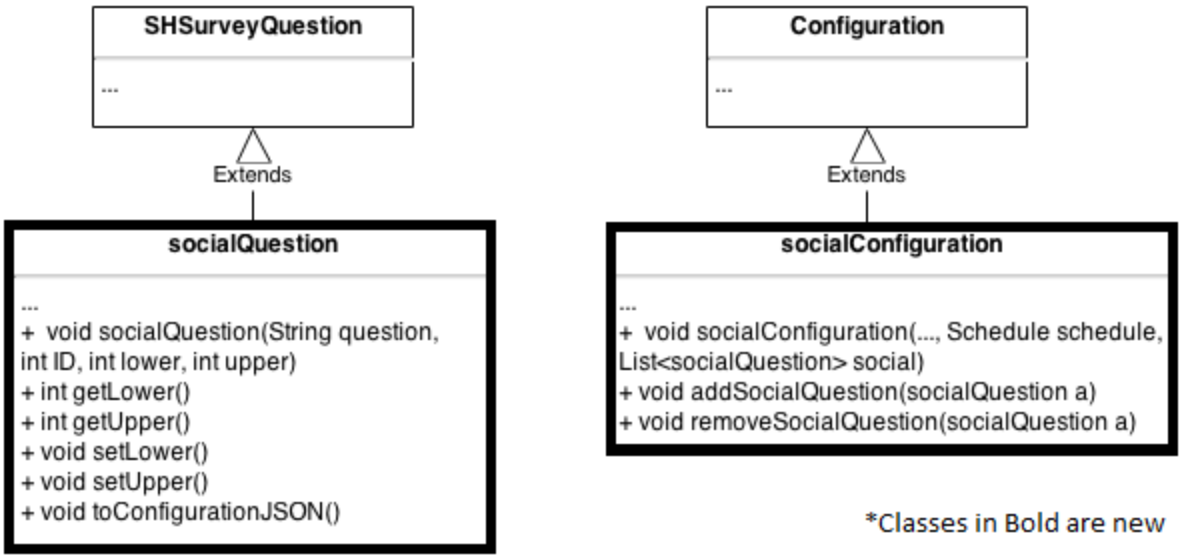


Provider is brought to this view and can configure survey settings



Clicking on a question navigates to this view





New functionality needs to be added to the provider app to allow the provider to configure survey questions with *threshold attributes* that will store the upper and lower bounds of the threshold for that question. To achieve this, a new configuration view called socialConfiguration must be created in the provider app. Presently, most of the survey classes are in the Client app. We need to create the same group of classes in the Provider app and also add the capability to save *thresholds* to the configuration. These *thresholds* must be stored with the other data sent over to the client app when exporting a profile

New Classes:

- SHSurvey*
- SHSurveyQuestion*
- SHSurveyResponse*
- socialQuestion
- socialConfiguration

*These new classes listed above already exist in a package in the Client app, but these classes must also exist in the Provider app to allow social survey response threshold configuration.

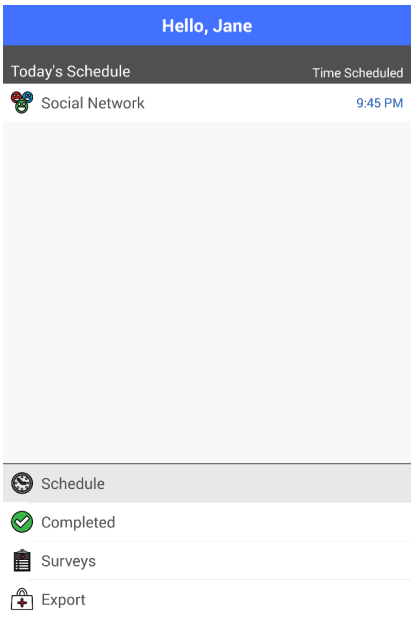
Alterations to Controllers and Existing Classes:

SHSurvey, SHSurveyQuestion, and SHSurvey question are all existing classes in the Client app. When these classes are added to the Provider app, a toConfigurationJSON() method will need to be added. The new classes must also communicate with the DatabaseController.

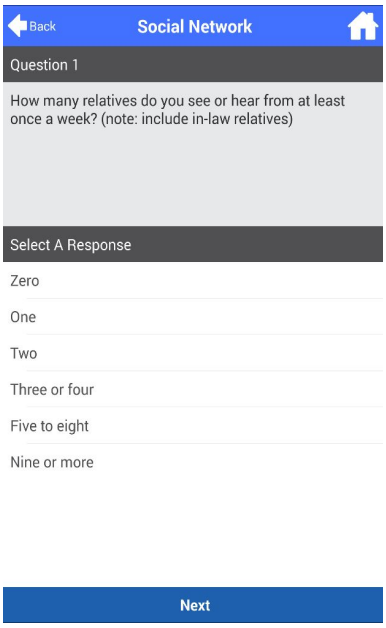
4.2 Client Navigates to a Social Network after taking Survey

New Views: When responses to a survey exceed the app integration threshold, a "Dialog" view will pop up that will ask the client to choose a social network to navigate to. Within the dialog box there will be buttons to navigate to (a) Facebook (b) Skype or (c) Return to Home.

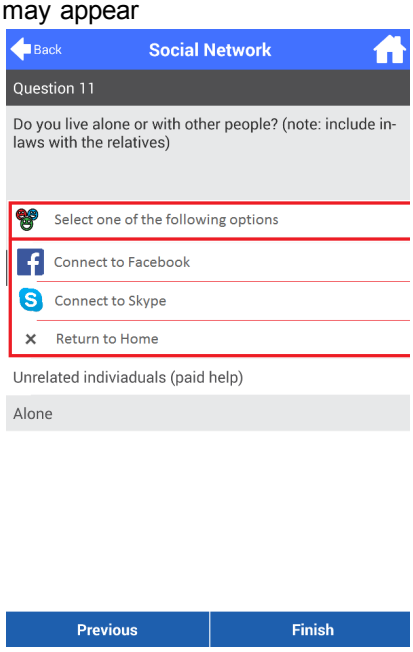
Survey pops up on client app

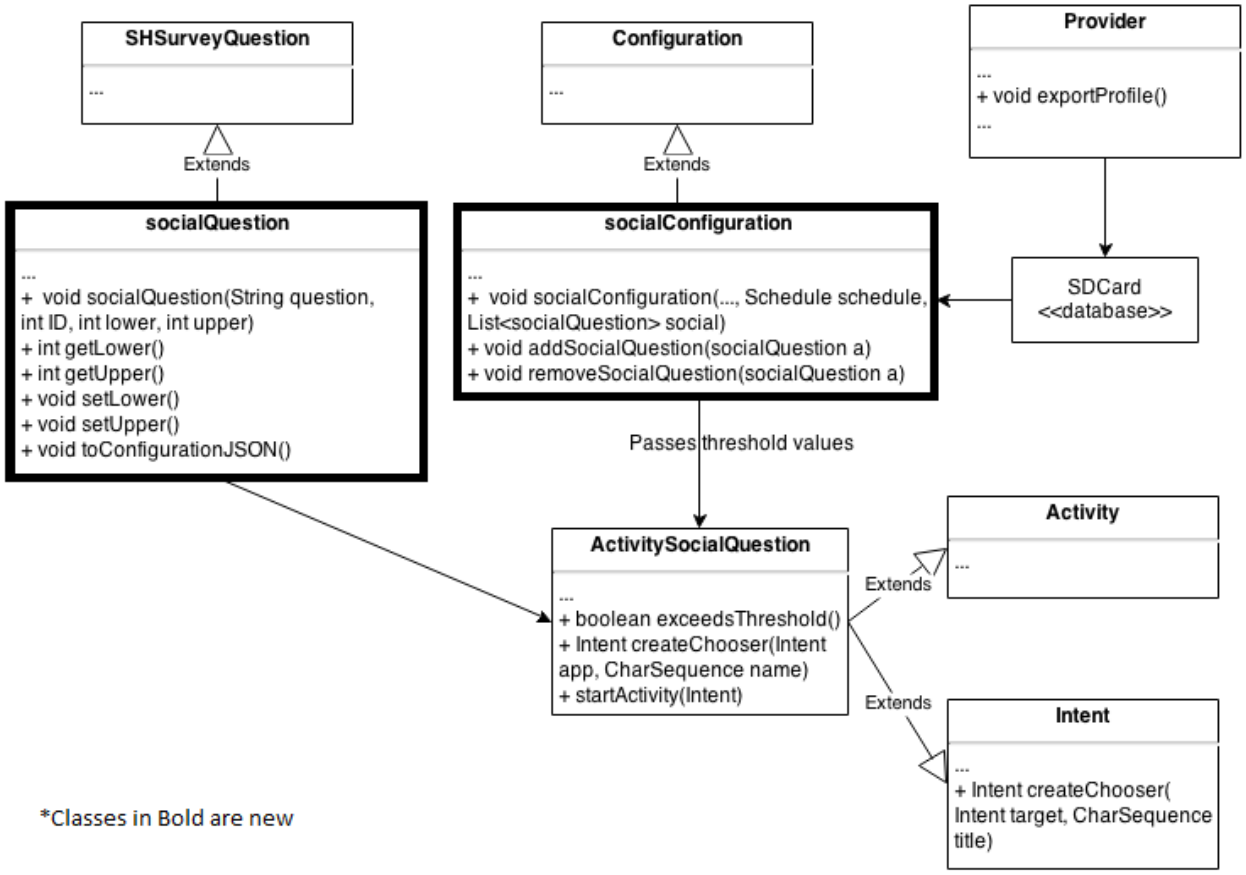


Client opens and fills survey



If answers are within "App Integration" threshold a popup may appear





When the client completes a survey, the answers they gave will be checked against the *threshold values* for App Integration. These *thresholds* are handed over to the Client through a class called “SHSocialConfiguration”. The implementation for surveys is already set up, therefore, we simply have to make sure the trigger is checked when the client completes the survey.In addition to creating new classes we will need to access the *Intent* and *Activity* android classes to provide the user with a list of social networking apps to navigate to. This is done by creating a new intent using the Intent.createChooser() method relaying the list of apps (Skype and Facebook here) as parameters. The chooser displayed will give the user the option between the apps. Once one is selected that app will need to be sent through the Activity.startActivity() method as an Intent object.

New Classes:

- SocialQuestion
- SocialSurveyConfiguration
- ActivitySocialQuestion

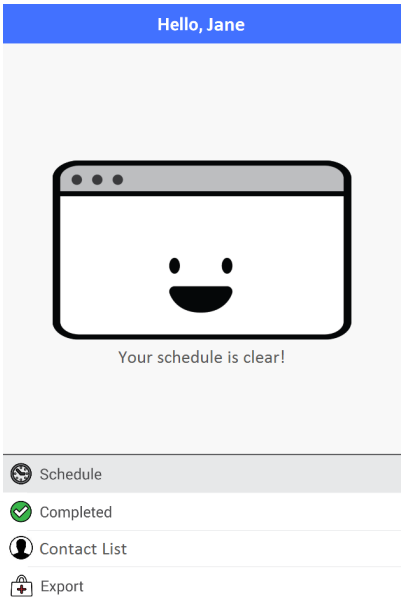
5. Contacts List

We will be adding entirely new classes to implement our new contacts list feature! Our contacts list will be used to promote self-health management from the client. The client can add, edit and remove contacts as they chose via our new *ContactList* class. We will have easy manageability of these contacts to promote the client to be active with their contact list! These contacts will be stored in a List to hold *Contact* objects each having attributes such as name, skype, facebook etc. Editing these contacts is essentially editing the attributes of the existing contact. Creating/Removing a contact is simply creating a new object to insert or delete out of our List!

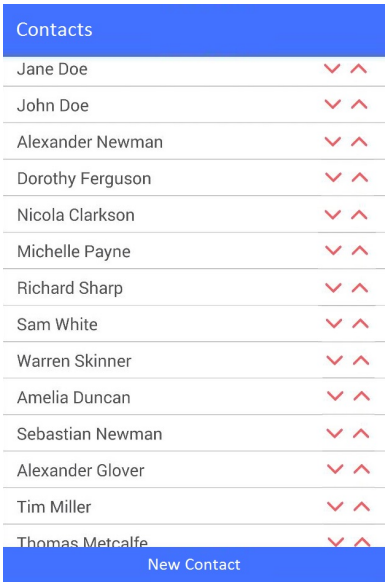
5.1 Client Adds Contacts

New Views: We will add a "Contact List" button to the main screen in the Client application. When selected, this will lead to a new list view of the contacts saved in the app. To create a new contact, the client can click the "New Contact" button which will open up a new page. Here they will be prompted to put in information [name,facebook,skype,etc] about the contact into text boxes. When the user is done filling in the information they will hit the "Save Contact" button which will save the contact into the contacts list.

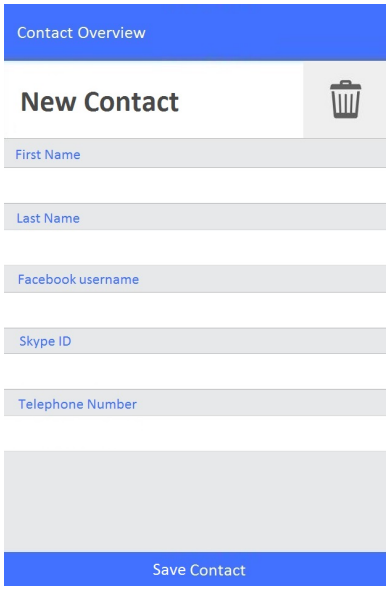
Click Contact List

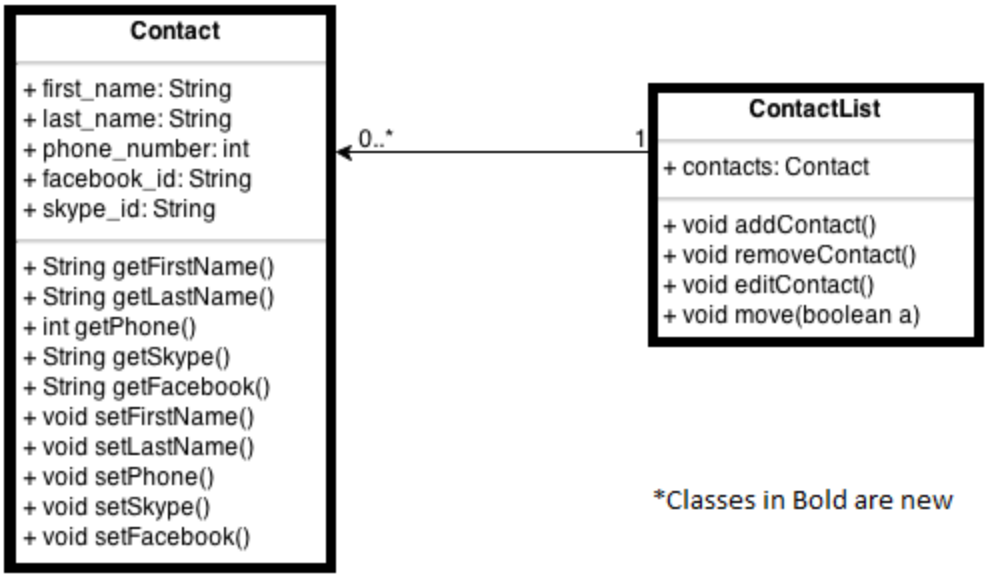


Click New Contact



Enter Contact information and click save





We will add a *Contact* class to store information associated with a particular contact and a *ContactList* class to store a List of *Contact* objects to represent the client's currently saved contacts and the order in which they will be displayed. The *Contact* class will store contact information using the following attributes: first_name, last_name, phone_number, facebook_username, skype_ID, and picture. In particular, when a client adds a contact, a new *Contact* object will be created with the entered contact information, and a reference for it will be placed at the end of the List in the *ContactList* class.

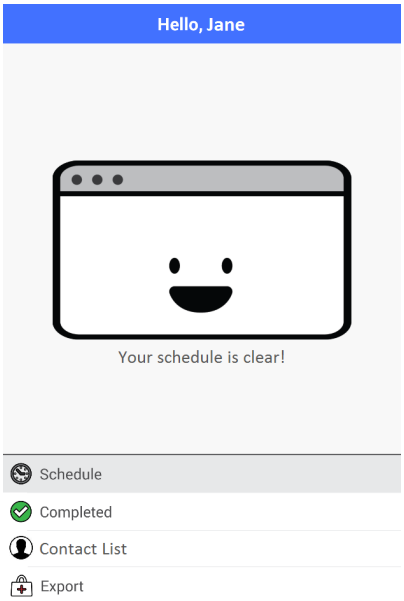
New Classes:

- Contact
- ContactList

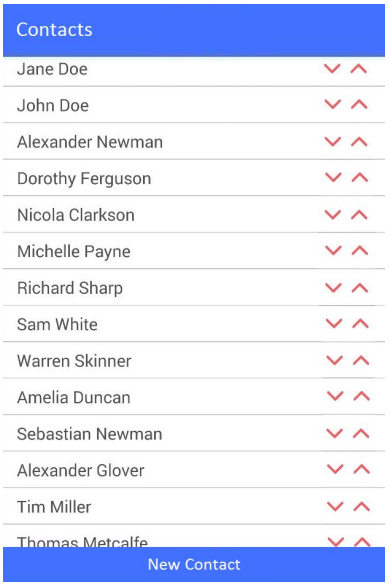
5.2 Client Edits/Removes Contacts

New Views: We will add a “Contacts” button to the main screen in the Client application. When selected, this will lead to a new list view of the the contacts saved in the app. The contacts are selectable. Upon selecting a contact, the user will be brought to another new view that will show various details about the contact in the form of text fields. There will also be a save button and a delete button in this view with which the user can save the edits made to a contact or remove it all together.

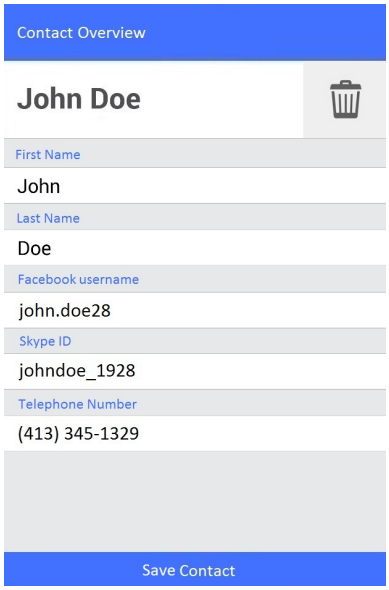
Click Contact List



Click New Contact



Edit contact information and click Save



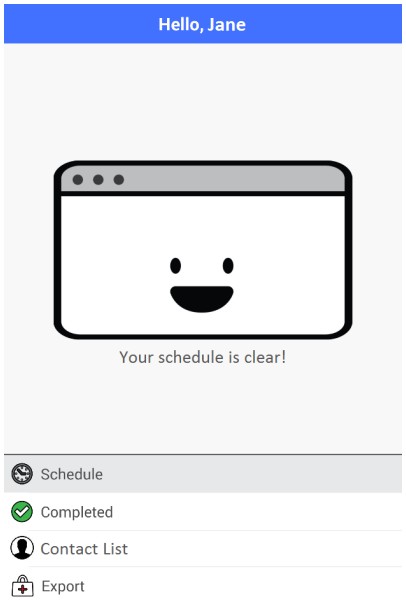
New Classes:

The process of editing/removing a contact will need to access the classes previously explained in section 5.1 under the subheading “New Classes”. In particular, when the client clicks on a contact, the respective *Contact* object will be accessed to retrieve the appropriate contact information to display. Also, when a client removes a contact, the associated *Contact* object will be deleted and its reference will be removed from the List in the *ContactList* class.

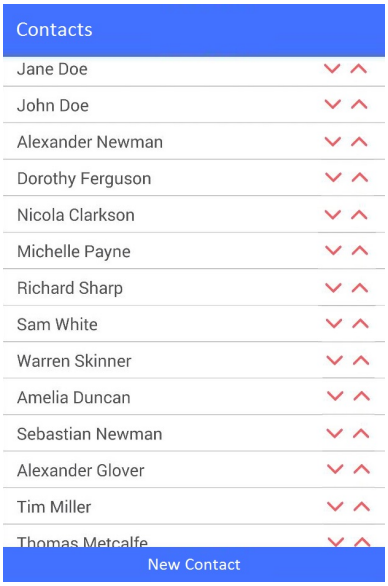
5.3 Client Orders Contacts

New Views: We will add a “Contact List” button to the main screen in the Client application. When selected, this will lead to a new view of the contacts currently saved in the app. Each contact has up and down buttons to reorder their position in the contacts list.

Click Contact List



Click up or down button to edit order of list



New Classes:

The process of ordering contacts will need to access the *ContactList* class described in section 5.1 under the subheading “New Classes”. In particular, when the client clicks the “up arrow” button on a contact, the associated *Contact* object’s position in the List that is located in the *ContactList* class will be swapped with the preceding *Contact* object in the List. Similarly the “down arrow” button will swap the associated *Contact* object’s position in the List with the succeeding *Contact* object. After an automated refresh, the changed order of the List in the *ContactList* class will be reflected in the view seen by the client.