PotlatchApp application is divided in Client and Server components, but sources are structured in three main folders: Client (the Java Android Client application), Common (the classes that are used by both Client and Server) and Server (the Java Spring-base Server application).
As indicated in project requisites a main application package for application classes has been defined, and it is named org.coursera.androidcapstone.

Basic Project components

1. Basic Project Requirement:
App supports multiple users via individual user accounts.

This requirement is met in two ways:

1. In database there is a table (Users) that contains user information. At the moment it contains only few information like Username field (the key of the table) and Avatar image, but application could be improved adding to Users repository also fields like Password (encrypted for security reasons) and EmailAddress (to activate a new User and to recover lost password).
2. Access to application is allowed only to known users using OAuth 2.0. OAuth2SecurityConfiguration class and other utility classes for authorization are taken from assignment VideoLike of Programming Cloud Services for Android Handheld Systems MOOC. At the moment there are only three hard-coded users that can use full features of the application: “alice” (hard-coded password “alicepass”), “bob” (hard-coded password “bobpass”) and “carol” (hard-coded password “carolpass”).

Unfortunately I didn’t have time to write the code to link the repository of users (class User) that can actively use the application (that is create Gifts, vote them, etc.), with the one that is used for OAuth security (class InMemoryUserDetailsManager).
I created also a read-only user (name “readonlyuser”, password “readonlyuserpass”) that is used to access top Gift givers list from application even if no user is logged in, but that user cannot enter the application to view or create Gifts, etc.
User repository is defined using the class User and interface UserRepository in files:

- Potlatch/Common/src/main/java/org/coursera/androidcapstone/common/repository/User.java.
- Potlatch/Server/src/main/java/org/coursera/androidcapstone/server/repository/UserRepository.java.

Hard-coded users for OAuth security are defined in file:

- Potlatch/Server/src/main/java/org/coursera/androidcapstone/server/oauth/OAuth2SecurityConfiguration.java, line 147.

Even if the two repositories are not linked, I realized a check so that users that pass the OAuth security phase are searched in Users repository to see if they are known users for the system.

The hard-coded well-known users are inserted in Server application initialization code, in particular in file:

- Potlatch/Server/src/main/java/org/coursera/androidcapstone/server/Application.java, lines 101 and 221.

The presence of the user in Users repository is checked in Android Client application through the method checkUsername() defined in Server application in ApplicationController class, accessible by Client through REST (for REST details implementation see later in this document, for example point 3b. Functional Description and App Requirement):

- Potlatch/Server/src/main/java/org/coursera/androidcapstone/server/ApplicationController.java, line 55.

2. Basic Project Requirement:

App contains at least one user facing function available only to authenticated users.

When Android Client application is started the LoginScreenActivity is launched (this is the main activity). It requires user credentials and only if a valid username/password pair is provided the user is able to access to list of Gift chains (note that a single Gift is a Gift chain without other linked gifts).
The mentioned classes are defined in the following files:

- Potlatch/Client/app/src/main/java/org/coursera/androidcapstone/client/ui/login/LoginScreenActivity.java.
- Potlatch/Client/app/src/main/java/org/coursera/androidcapstone/client/ui/login/LoginScreenFragment.java.
- Potlatch/Client/app/src/main/java/org/coursera/androidcapstone/client/ui/gift/ViewGiftChainsActivity.java.
- Potlatch/Client/app/src/main/java/org/coursera/androidcapstone/client/ui/gift/ViewGiftListFragment.java.
- Potlatch/Client/app/src/main/java/org/coursera/androidcapstone/client/ui/common/UIActivityBase.java.

3. Basic Project Requirement:

App comprises at least 1 instance of each of at least 2 of the following 4 fundamental Android components:

- Activity
- BroadcastReceiver
- Service
- ContentProvider

In Android Client application there will be several activities and fragments:

- LoginScreenActivity: login panel to access to application content.
- ViewGiftChainsActivity: the list of Gift chains.
- ViewGiftChainActivity: the list of Gifts in one chain.
- ViewGiftActivity: view details of a Gift.

- CreateGiftChainActivity: create a new Gift as a starter of new chain.
- CreateGiftChainActivity: create a new Gift, attached to an existing chain.
- EditPreferencesActivity: as the name suggests, it allows a user to configure some settings like default server address and port, if user wants to filter flagged gifts, and how often touched counts are updated.
- ViewTopGiftGiversActivity: it shows information about the top Gift givers, the users that have posted Gifts that have touched the greatest number of other users.

Regarding this point, see classes defined in *Activity.java and *Fragment.java files contained in folders:

- Potlatch/Client/app/src/main/java/org/coursera/androidcapstone/client/ui/login.
- Potlatch/Client/app/src/main/java/org/coursera/androidcapstone/client/ui/gift.
- Potlatch/Client/app/src/main/java/org/coursera/androidcapstone/client/ui/settings.

To fulfill the 7b. Functional Description and App Requirement ("Touched counts can be periodically updated in accordance with a user-specified preference") I used the AlarmManager service to schedule update of Gift counters periodically, as configured using EditPreferencesActivity. I implemented a BroadcastReceiver that wait alarm notification, named GiftUpdater. In onReceive() method, it uses LocalBroadcastManager to send a new Intent "FragmentUpdater", that is handled by ViewGiftFragment and ViewTopGiftGiversFragment fragments through a BroadcastReceiver data member in the classes. When notification is received a new request is sent to Server application to update counters (only if fragment is active and visible to avoid unnecessary network traffic).

AlarmManager is initialized in the following file:

- Potlatch/Client/app/src/main/java/org/coursera/androidcapstone/client/globals/PotlatchApp.java, lines 22, 60 and 76.

BrodacastReceiver that handles alarm notification in defined in the following file:

- Potlatch/Client/app/src/main/java/org/coursera/androidcapstone/client/ui/updater/GiftUpdater.java.

The intent "FragmentUpdater" is handled in the following files:

- Potlatch/Client/app/src/main/java/org/coursera/androidcapstone/client/ui/gift/ViewGiftFragment.java, lines 62 and 136.
- Potlatch/Client/app/src/main/java/org/coursera/androidcapstone/client/ui/gift/ViewTopGiftGiversFragment.java, lines 36 and 53.

4. Basic Project Requirement:

App interacts with at least one remotely-hosted Java Spring-based service.

Android Client application interacts with the remotely-hosted Java Spring-based services provided by Server application. For example it interacts with Server part of the application for authentication / authorization (see 1. Basic Project Requirement) to obtain OAuth 2.0 tokens and through REST APIs accesses to Gift and User information stored in CRUD repositories using a Spring-based application controller.

For this point see also point 5. Basic Project Requirement, anyway as main references it is possible to have a look at the following files:

- Potlatch/Client/app/src/main/java/org/coursera/androidcapstone/client/rest/GiftSvc.java.
- Potlatch/Common/src/main/java/org/coursera/androidcapstone/common/oauth/SecuredRestBuilder.java.
- Potlatch/Common/src/main/java/org/coursera/androidcapstone/common/client/GiftSvcApi.java.
- Potlatch/Server/src/main/java/org/coursera/androidcapstone/server/oauth/OAuth2SecurityConfiguration.java.
- Potlatch/Server/src/main/java/org/coursera/androidcapstone/server/ApplicationController.java.

5. Basic Project Requirement:

App interacts over the network via HTTP.

As exposed in 4. Basic Project Requirement, Android Client application interacts to remotely-hosted Java Spring-based services and in particular over the network via HTTP. For example Gift CRUD repository is accessed though REST APIs and data object are marshalled and un-marshalled using JSON. More details will be exposed later, but as an example it is possible to view (among the files listed in the previous point) the Server application main file:

- Potlatch/Server/src/main/java/org/coursera/androidcapstone/server/Application.java, lines from 57 to 159.

6. Basic Project Requirement:

App allows users to navigate between 3 or more user interface screens at runtime.

As exposed in 3. Basic Project Requirement, there are several activities that corresponds to fragments and so user interface screens:

1. LoginScreen: login panel to access to application content. If credentials are correct user can see the list of Gift chains (through ViewGiftChainActivity activity).

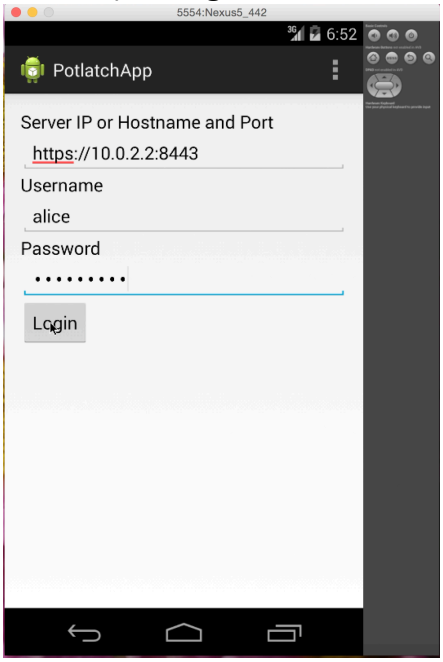

Figure 1 LoginScreen

Reference files for this screen are:

- Potlatch/Client/app/src/main/java/org/coursera/androidcapstone/client/ui/login/LoginScreenActivity.java.
- Potlatch/Client/app/src/main/java/org/coursera/androidcapstone/client/ui/login/LoginScreenFragment.java.
- Potlatch/Client/app/src/main/res/layout/login_screen_activity.xml.
- Potlatch/Client/app/src/main/res/layout/login_screen_fragment.xml.

2. ViewGiftChains: the list of Gift chains. User sees the title of the gift that is at the head of the chain, and a thumbnail of the Gift content near the title. In this view there is textbox that allows filtering Gift chains by title (to clear filter, delete content of textbox and press again "Filter" button), and there is a button to create a new Gift (that creates a chain with only one element). If the user chooses one element in the list, application shows the list of Gifts in the chain (through ViewGiftChainActivity activity). A Gift chain has at least one Gift.

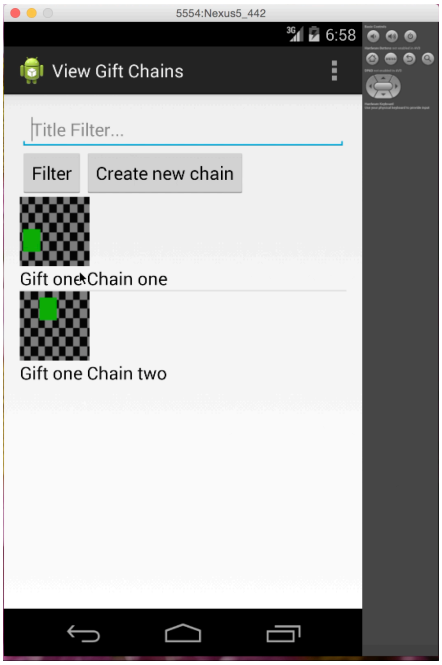

Figure 2 ViewGiftChains

ViewGiftChainsActivity uses ViewGiftListFragment and GiftItemCollectionAdapter classes that are used also for ViewGiftChain screen.

Reference files for this screen are:

- Potlatch/Client/app/src/main/java/org/coursera/androidcapstone/client/ui/gift/ViewGiftChainsActivity.java.
- Potlatch/Client/app/src/main/java/org/coursera/androidcapstone/client/ui/gift/ViewGiftListFragment.java.
- Potlatch/Client/app/src/main/java/org/coursera/androidcapstone/client/ui/gift/GiftItemCollectionAdapter.java.
- Potlatch/Client/app/src/main/res/layout/view_giftchains_activity.xml.
- Potlatch/Client/app/src/main/res/layout/view_gift_listview_fragment.xml.
- Potlatch/Client/app/src/main/res/layout/view_gift_listview_row.xml.

3. ViewGiftChain: the list of Gifts in one chain. User sees the titles of the Gifts that belong to this chain, and near it a thumbnail of the Gift content. Also in this view there is textbox that allows filtering Gifts by title (functionalities are the same exposed for

ViewGiftChains screen), and there is a button to create a new Gift (new Gift is added to current chain). If the user chooses one element in the chain, application shows the details of the selected Gift (through ViewGiftActivity activity).

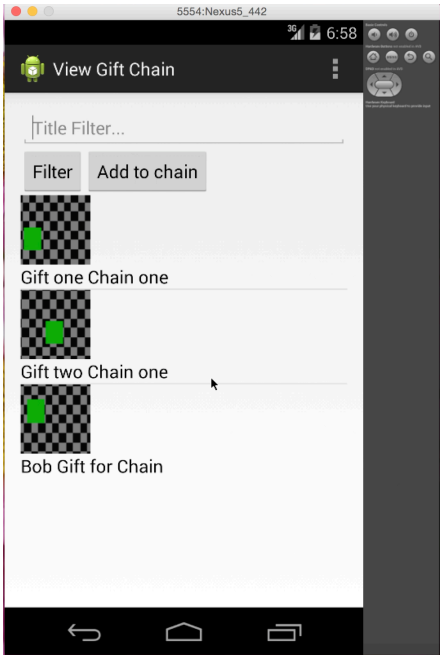

Figure 3 ViewGiftChain

Reference files for this screen are (many of these are the same of ViewGiftChains):

- Potlatch/Client/app/src/main/java/org/coursera/androidcapstone/client/ui/gift/ViewGiftChainActivity.java.
- Potlatch/Client/app/src/main/java/org/coursera/androidcapstone/client/ui/gift/ViewGiftListFragment.java.
- Potlatch/Client/app/src/main/java/org/coursera/androidcapstone/client/ui/gift/GiftItemCollectionAdapter.java.
- Potlatch/Client/app/src/main/res/layout/view_giftchain_activity.xml.
- Potlatch/Client/app/src/main/res/layout/view_gift_listview_fragment.xml.
- Potlatch/Client/app/src/main/res/layout/view_gift_listview_row.xml.

4. ViewGift: view with details of a Gift, in particular Title, Text, Image, CreationDate, Username (of the creator), Number of touches, Number of flags. In this view there are two toggle radio buttons to allow Gift voting (touch/flag), and two buttons to confirm or cancel the vote choice.

   Reference files for this screen are:

   - Potlatch/Client/app/src/main/java/org/coursera/androidcapstone/client/ui/gift/ViewGiftActivity.java.
   - Potlatch/Client/app/src/main/java/org/coursera/androidcapstone/client/ui/gift/ViewGiftFragment.java.
   - Potlatch/Client/app/src/main/res/layout/view_gift_activity.xml.
   - Potlatch/Client/app/src/main/res/layout/view_gift_fragment.xml.

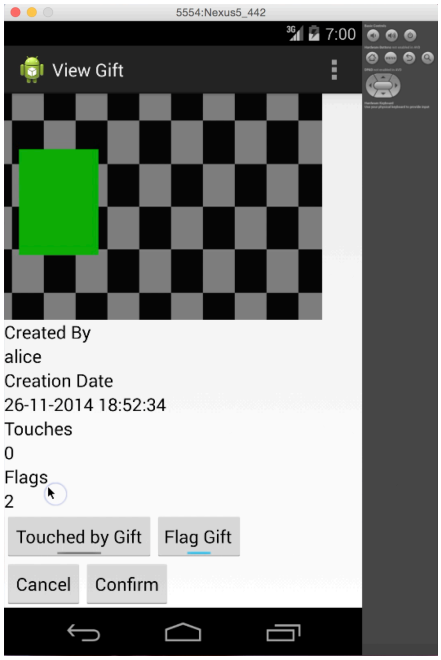

Figure 4 ViewGift

5. CreateGift: it allows to create a new Gift, either attached to a chain or as a starter of new chain, depending on context. In this screen there are two textbox (one for Title and one for Text), one box to preview picture that can be captured using camera or can be chosen from stored pictures (there are two buttons: “Select Image” and “Capture Image”), and two buttons “Confirm” and “Cancel” to confirm or cancel Gift creation. Since this screen is used by ViewGiftChainsFragment and ViewGiftChainFragment to create a new Gift chain or a new Gift in an existing chain, the two activities CreateGiftChainActivity and CreateGiftInChainActivity create CreateGiftFragment using different parameters.

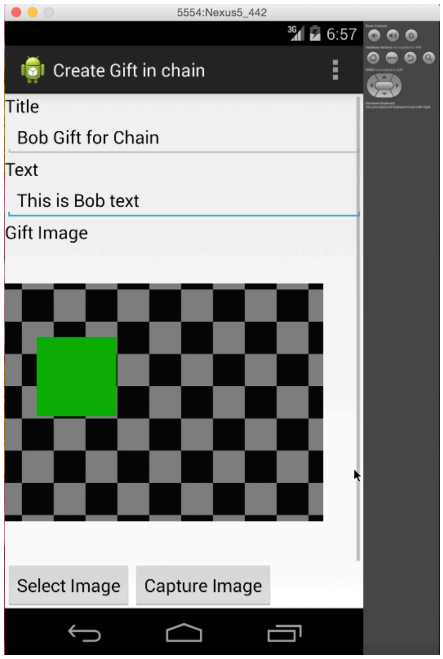

Figure 5 CreateGift

Reference files for this screen are:

- Potlatch/Client/app/src/main/java/org/coursera/androidcapstone/client/ui/gift/CreateGiftChainActivity.java.
- Potlatch/Client/app/src/main/java/org/coursera/androidcapstone/client/ui/gift/Crea

teGiftInChainActivity.java

- Potlatch/Client/app/src/main/java/org/coursera/androidcapstone/client/ui/gift/CreateGiftFragment.java.
- Potlatch/Client/app/src/main/res/layout/create_giftchain_activity.xml.
- Potlatch/Client/app/src/main/res/layout/create_gift_in_chain_activity.xml.
- Potlatch/Client/app/src/main/res/layout/create_gift_fragment.xml.

6. EditPreferences: it allows to configure default server address and port, if user wants to filter flagged gifts (Gift is filtered if contains at least one flag by one User), and how often touched counts are updated (possible choices: 1 minute, 5 minutes or 60 minutes).

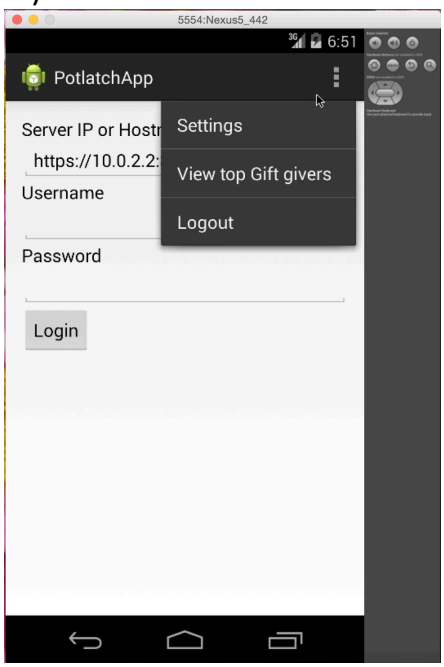

Figure 6 Settings Menu

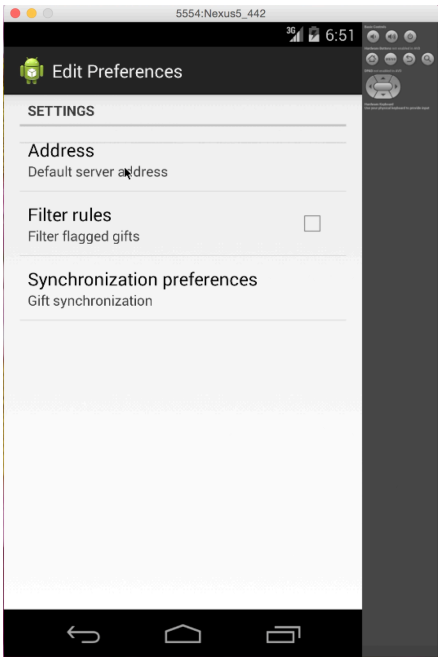

Figure 7 EditPreferences

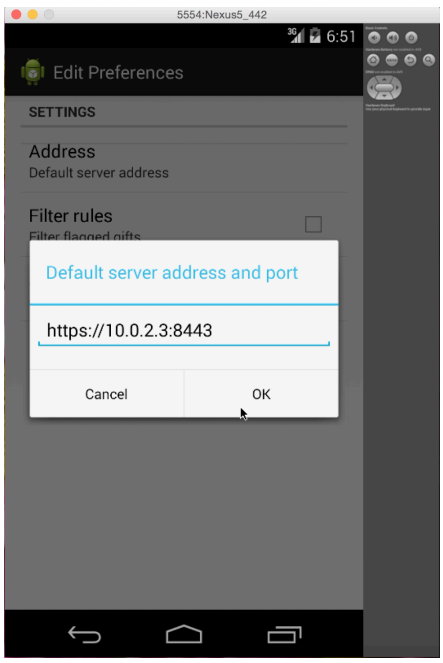

Figure 8 Default Server

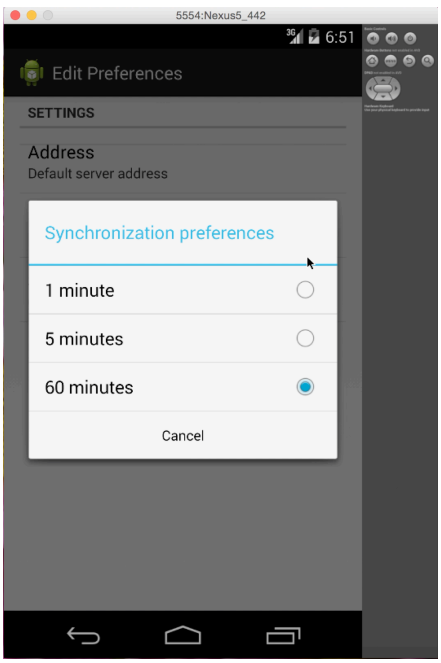

Figure 9 Counters Synchronization

Reference files for this screen are:

- Potlatch/Client/app/src/main/java/org/coursera/androidcapstone/client/ui/settings/EditPreferencesActivity.java.
- Potlatch/Client/app/src/main/java/org/coursera/androidcapstone/client/ui/settings/EditPreferencesFragment.java.
- Potlatch/Client/app/src/main/res/layout/edit_preferences_activity.xml.
- Potlatch/Client/app/src/main/res/xml/edit_preferences_fragment.xml.
- Potlatch/Client/app/src/main/res/xml/preference_headers.xml.
- Potlatch/Client/app/src/main/res/values/arrays.xml.
- Potlatch/Client/app/src/main/res/menu/gift_client_menu.xml.

7. ViewTopGiftGivers: it shows information about the top Gift givers, the Users that have posted Gifts that have touched the greatest number of other Users. This view displays a list in which each item contains username, the User's avatar and total number of touches. This screen is visible to logged and not logged in users and it is reachable through option "View top Gift givers" in options menu (shown in Figure 6 Settings Menu).

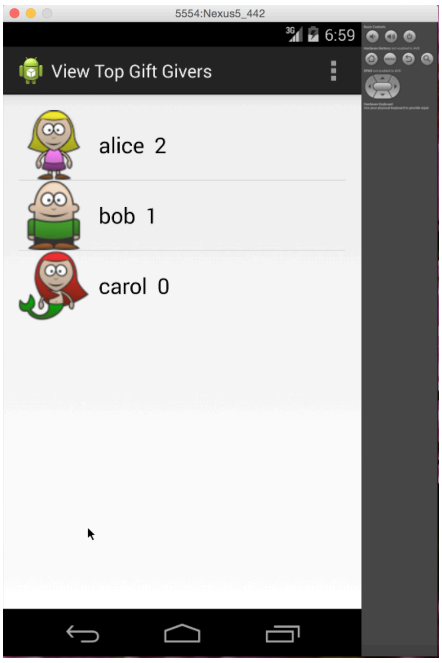

Figure 10 ViewTopGiftGivers

Reference files for this screen are:

- Potlatch/Client/app/src/main/java/org/coursera/androidcapstone/client/ui/gift/ViewTopGiftGiversActivity.java.
- Potlatch/Client/app/src/main/java/org/coursera/androidcapstone/client/ui/gift/ViewTopGiftGiversFragment.java.
- Potlatch/Client/app/src/main/java/org/coursera/androidcapstone/client/ui/gift/TopGiftGiversCollectionAdapter.java.
- Potlatch/Client/app/src/main/res/layout/view_top_gift_givers_activity.xml.
- Potlatch/Client/app/src/main/res/layout/view_top_gift_givers_fragment.xml.
- Potlatch/Client/app/src/main/res/layout/top_gift_giver_listview_row.xml.

7. Basic Project Requirement:

App uses at least one advanced capability or API from the following list (covered in the MoCCA Specialization): multimedia capture, multimedia playback, touch

gestures, sensors, animation.**
**Learners are welcome to use ADDITIONAL other advanced capabilities (e.g., BlueTooth, Wifi-Direct networking, push notifications, search), but must also use at least one from the MoCCA list.

As exposed in 6. Basic Project Requirement Android Client application allows the user to capture a picture that can be inserted in a Gift. To accomplish this it uses the MediaStore Image capture function (MediaStore.ACTION_IMAGE_CAPTURE), see the following link http://developer.android.com/reference/android/provider/MediaStore.html for additional details. This feature was used in iRemember project.
See the following file:

- Potlatch/Client/app/src/main/java/org/coursera/androidcapstone/client/ui/gift/CreateGiftFragment.java, line 327.

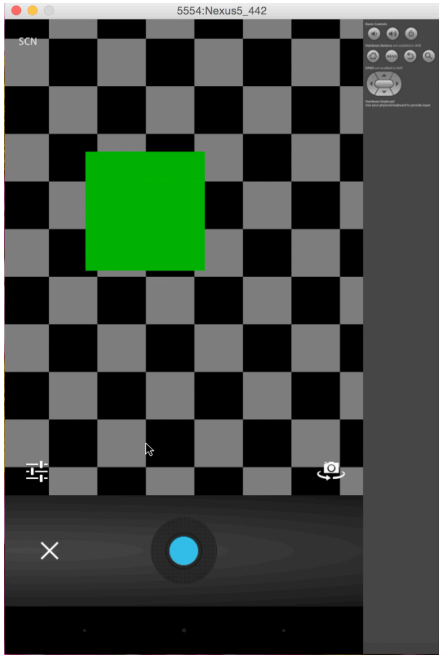

Figure 11 Capture Image

8. Basic Project Requirement:

App supports at least one operation that is performed off the UI Thread in one or more background Threads of Thread pool.

Since images could be very big and could require a certain amount of time to download, it is necessary to execute operations that require network access (REST APIs for example) in a background worker thread and then handle results in the main UI Thread. To accomplish this point I used the class CallableTask (that extends AsyncTask) and interface TaskCallback written by Dr. Jules Write for his mobilecloud-template project.
These files are contained in Client "async" subfolder:

- Potlatch/Client/app/src/main/java/org/coursera/androidcapstone/client/async/CallableTask.java.
- Potlatch/Client/app/src/main/java/org/coursera/androidcapstone/client/async/TaskCallback.java.

AsyncTasks are used in several points in Client application, for example:

- Potlatch/Client/app/src/main/java/org/coursera/androidcapstone/client/ui/login/LoginScreenFragment.java, line 124.
- Potlatch/Client/app/src/main/java/org/coursera/androidcapstone/client/ui/gift/CreateGiftFragment.java, line 189.
- Potlatch/Client/app/src/main/java/org/coursera/androidcapstone/client/ui/gift/ViewGiftFragment.java, lines 244, 298, 339 and 419.
- Potlatch/Client/app/src/main/java/org/coursera/androidcapstone/client/ui/gift/ViewGiftListFragment.java, line 199.
- Potlatch/Client/app/src/main/java/org/coursera/androidcapstone/client/ui/gift/ViewTopGiftGiversFragment.java, line 113.

Functional Description and App components

1. Functional Description and App Requirement:

App defines a *Gift* as a unit of data containing an image, a title, and optional accompanying text.

To accomplish this functional requirement a serializable class Gift is created. Image content is stored as a clob (that is a string that contains the array of bytes corresponding to image encoded as base64 stream) in a CRUD repository named GiftRepository.
I defined additional fields, like for example Thumbnail that is another clob that contains a thumbnail of the Gift image content. Thumbnail is used in lists of Gifts and Gift chains.

```
@Entity
@Table(name="Gifts")
//Define a sequence - might also be in another class
@SequenceGenerator(name="GiftSequenceGenerator",initialValue=1,allocationSize=100)
public class Gift implements Serializable {
…
    @Id
    @Column(name="Id",updatable=false,nullable=false)
    // Use the sequence that is defined above
  @GeneratedValue(strategy=GenerationType.SEQUENCE,generator="GiftSequenceGenerator")
    private long id;

    @Basic
    @Column(name="Title",updatable=false,nullable=false)
    private String title;

    @Basic
    @Column(name="Text",updatable=false,nullable=false,length=1024)
    private String text;

    @Column(name="Content",updatable=false,nullable=false)
    @Lob @Basic(fetch=FetchType.LAZY)
    private String content;

    @Column(name="Thumbnail",updatable=false,nullable=false)
    @Lob @Basic(fetch=FetchType.LAZY)
    private String thumbnail;
…
}
```

Class Gift is defined in file:

- Potlatch/Common/src/main/java/org/coursera/androidcapstone/common/repository/Gift.java.

GiftRepository is an interface for a Spring-based CRUD repository and it is defined in file:

- Potlatch/Server/src/main/java/org/coursera/androidcapstone/server/repository/GiftRepository.java.

2. Functional Description and App Requirement:

A *User* can create a Gift by taking a picture (or optionally by selecting an image already stored on the device), entering a title, and optionally typing in accompanying text.

This functional requirement is fulfilled by activities CreateGiftChainActivity and CreateGiftInChainActivity, through fragment CreateGiftFragment as exposed in 6. Basic Project Requirement (item list number 5. CreateGift).

3a. Functional Description and App Requirement:

Once the Gift is complete the User can post the Gift to a *Gift Chain* (which is one or more related Gifts).

```
@Entity
@Table(name="Gifts")
//Define a sequence – might also be in another class
@SequenceGenerator(name="GiftSequenceGenerator",initialValue=1,allocationSize=100)
public class Gift implements Serializable {
…
     @Id
     @Column(name="Id",updatable=false,nullable=false)
     // Use the sequence that is defined above
   @GeneratedValue(strategy=GenerationType.SEQUENCE,generator="GiftSequenceGenerator")
     private long id;
…
     @JsonIgnore
     @ManyToOne(fetch=FetchType.EAGER,cascade=CascadeType.PERSIST)
     @JoinColumn(name="GiftChain",updatable=false,nullable=false)
     private Gift giftChain;
…
}
```

Class Gift has a data member named giftChain. This field contains a reference to the Gift that is the head Gift of the chain if current Gift belongs to a chain, or it contains a reference to itself value if current Gift started a new chain.

Gift and User objects are stored in CRUD repositories. As a consequence, the field GiftChain will contain in Gifts table the Id of the head if Gift is part of a chain, and Id is equal to GiftChain for head Gift in the chain.

Schema of the database is contained in schema.sql file:

- Potlatch/Server/sql/schema.sql.

```
create table FlaggedByUsers (GiftId bigint not null, Username varchar(32) not null,
primary key (GiftId, Username))
create table Gifts (Id bigint not null, Content clob not null, CreationTimestamp
bigint not null, FlagsCount integer not null, Text varchar(1024) not null, Thumbnail
```

```
clob not null, Title varchar(255) not null, TouchesCount integer not null, CreatedBy
varchar(32) not null, GiftChain bigint not null, primary key (Id))
create table TouchedUsers (GiftId bigint not null, Username varchar(32) not null,
primary key (GiftId, Username))
create table Users (Username varchar(32) not null, Avatar clob, TotalTouchesCount
bigint not null, primary key (Username))
alter table FlaggedByUsers add constraint FK_rsv7c08ixr821oxnghnstpw5h foreign key
(Username) references Users
alter table FlaggedByUsers add constraint FK_qookqbec9prlg78qbt0cfv82g foreign key
(GiftId) references Gifts
alter table Gifts add constraint FK_bk66ll5p36bad2cssw77wv3hd foreign key (CreatedBy)
references Users
alter table Gifts add constraint FK_mwyn4vi1gy5vah4y5v6be4pfj foreign key (GiftChain)
references Gifts
alter table TouchedUsers add constraint FK_l8fqht6ij7mjpoff3uerqgl60 foreign key
(Username) references Users
alter table TouchedUsers add constraint FK_bjo2h9b05d5owsm4iakko605 foreign key
(GiftId) references Gifts
create sequence hibernate_sequence
```

Figure 12 schema.sql for CRUD repositories

File references for Gift and GiftRepository are the same listed in 1. Functional Description and App Requirement.
For references related to Gift chain creation in Client application make reference to what has been mentioned in 2. Functional Description and App Requirement.

3b. Functional Description and App Requirement:

Gift data is stored to and retrieved from a web-based service accessible in the cloud.

This requirement is fulfilled defining the annotated interface GiftRepository that extends CrudRepository<Gift, Long>.

```
@Repository
public interface GiftRepository extends CrudRepository<Gift, Long> {
…
    public Collection<Gift> findByTitle(
           @Param(GiftSvcApi.TITLE_PARAMETER) String title);
…
}
```

Extending GiftRepository interface we are able to add new data management features to repository and so to application.

```
@Repository
public interface GiftRepository extends CrudRepository<Gift, Long> {
…
    public Collection<GiftItem> findGiftChainsItems(
                @Param(GiftSvcApi.FILTER_FLAGGED_PARAMETER) boolean filter_flagged);

    public Collection<GiftItem> findGiftChainItemsById(
                @Param(GiftSvcApi.ID_PARAMETER) long id,
                @Param(GiftSvcApi.FILTER_FLAGGED_PARAMETER) boolean filter_flagged);

   public Collection<GiftItem> findGiftChainByTitle(
                @Param(GiftSvcApi.TITLE_PARAMETER)String title,

                @Param(GiftSvcApi.FILTER_FLAGGED_PARAMETER) boolean filter_flagged);
```

```
    public Collection<GiftItem> findGiftInChainByTitle(
                @Param(GiftSvcApi.ID_PARAMETER) long id,
                @Param(GiftSvcApi.TITLE_PARAMETER) String title,
                @Param(GiftSvcApi.FILTER_FLAGGED_PARAMETER) boolean filter_flagged);
…
}
```

where:

- GiftSvcApi.TITLE_PARAMETER is "title".
- GiftSvcApi.ID_PARAMETER is "id".
- GiftSvcApi.FILTER_FLAGGED_PARAMETER is "filter_flagged".

File reference:

- Potlatch/Server/src/main/java/org/coursera/androidcapstone/server/repository/GiftRepository.java.

Using JPA queries it is possible to return custom results for search queries as collection tuples of Objects, or as a collection of ad-hoc created objects.
GiftItem is a class that contains Gift Id, Title and Thumbnail fields, and can be used to populate listviews of Gift chains and Gifts. Note that we can use the same class to store information for both ViewGiftChainsActivity and ViewGiftChainActivity (they use the same fragment and collection adapter).

```
public class GiftItem implements Serializable {
…
    public long id;
    public String title;
    @Lob @Basic(fetch=FetchType.LAZY)
    public String thumbnail;
…
}
```

File reference:

- Potlatch/Common/src/main/java/org/coursera/androidcapstone/common/client/GiftItem.java.

The Gift repository is available to Client application through GiftSvcApi interface (that is through REST APIs). Requests declared in that interface are then mapped and handled by ApplicationController class in Server application.

```
public interface GiftSvcApi {
    public static final String TOKEN_PATH = "/oauth/token";

    public static final String ID_PARAMETER = "id";
    public static final String CHAIN_ID_PARAMETER = "chain_id";
    public static final String TITLE_PARAMETER = "title";
    public static final String USERNAME_PARAMETER = "username";
    public static final String FILTER_FLAGGED_PARAMETER = "filter_flagged";

    // The path where we expect the GiftSvc to live
    public static final String GIFT_SVC_PATH = "/gift";
    public static final String GIFT_USERNAME_PATH = GiftSvcApi.GIFT_SVC_PATH + "/username";
    public static final String GIFT_ITEM_CHAIN_PATH = GiftSvcApi.GIFT_SVC_PATH + "/chain";
    public static final String GIFT_ID_PATH = GiftSvcApi.GIFT_SVC_PATH + "/{id}";
    public static final String GIFT_DETAIL_ID_PATH = GiftSvcApi.GIFT_SVC_PATH + "/{id}/detail";
    public static final String GIFT_COUNTERS_ID_PATH = GiftSvcApi.GIFT_SVC_PATH + "/{id}/counters";
    public static final String GIFT_CHAIN_ID_PATH = GiftSvcApi.GIFT_SVC_PATH + "/{id}/chain";
    public static final String GIFT_TOUCHEDBY_PATH = GiftSvcApi.GIFT_SVC_PATH + "/{id}/touchedby";
    public static final String GIFT_UNTOUCHEDBY_PATH = GiftSvcApi.GIFT_SVC_PATH + "/{id}/untouchedby";
    public static final String GIFT_TOUCHED_PATH = GiftSvcApi.GIFT_SVC_PATH + "/{id}/touched";
    public static final String GIFT_FLAGGED_PATH = GiftSvcApi.GIFT_SVC_PATH + "/{id}/flagged";
```

```
    public static final String GIFT_UNFLAGGED_PATH = GiftSvcApi.GIFT_SVC_PATH + "/{id}/unflagged";
    public static final String GIFT_FLAGGEDBY_PATH = GiftSvcApi.GIFT_SVC_PATH + "/{id}/flaggedby";

    // The path to search gifts by title
    public static final String GIFT_TITLE_SEARCH_PATH = GIFT_SVC_PATH + "/search/findByTitle";
    public static final String GIFT_CHAIN_TITLE_SEARCH_PATH = GIFT_SVC_PATH + "/chain/search/findByTitle";
    public static final String GIFT_IN_CHAIN_TITLE_SEARCH_PATH = GIFT_SVC_PATH +
                                                        "/{id}/chain/search/findByTitle";
    // The path to search gifts by username
    public static final String GIFT_USERNAME_SEARCH_PATH = GIFT_SVC_PATH + "/search/findByUsername";
    // Find users that have the highest rated gifts
    public static final String GIFT_TOP_GIVERS_PATH = GIFT_SVC_PATH + "/search/findTopGivers";
…
    @GET(GIFT_ITEM_CHAIN_PATH)
    public Collection<GiftItem> getGiftChainsList(
                        @Query(FILTER_FLAGGED_PARAMETER) boolean filter_flagged);

    @GET(GIFT_CHAIN_ID_PATH)
    public Collection<GiftItem> getGiftChainById(
                        @Path(ID_PARAMETER) long id,
                        @Query(FILTER_FLAGGED_PARAMETER) boolean filter_flagged);
…
    @GET(GIFT_CHAIN_TITLE_SEARCH_PATH)
    public Collection<GiftItem> findGiftChainByTitle(
                        @Query(TITLE_PARAMETER) String title,
                        @Query(FILTER_FLAGGED_PARAMETER) boolean filter_flagged);

    @GET(GIFT_IN_CHAIN_TITLE_SEARCH_PATH)
    public Collection<GiftItem> findGiftInChainByTitle(
                        @Path(ID_PARAMETER) long id,
                        @Query(TITLE_PARAMETER) String title,
                        @Query(FILTER_FLAGGED_PARAMETER) boolean filter_flagged);
…
}
```

File reference:

- Potlatch/Common/src/main/java/org/coursera/androidcapstone/common/client/GiftSvcApi.java.

```
@Controller
public class ApplicationController {
    @Autowired
    private GiftRepository gifts;
…
    @RequestMapping(value=GiftSvcApi.GIFT_ITEM_CHAIN_PATH, method=RequestMethod.GET)
    public @ResponseBody Collection<GiftItem> getGiftChainsList(
            @RequestParam(GiftSvcApi.FILTER_FLAGGED_PARAMETER) boolean filter_flagged)
      { … }

    @RequestMapping(value=GiftSvcApi.GIFT_CHAIN_ID_PATH, method=RequestMethod.GET)
    public @ResponseBody Collection<GiftItem> getGiftChainById(
            @PathVariable(GiftSvcApi.ID_PARAMETER) long id,
            @RequestParam(GiftSvcApi.FILTER_FLAGGED_PARAMETER) boolean filter_flagged,
            HttpServletResponse response)
      { … }
…
    @RequestMapping(value=GiftSvcApi.GIFT_CHAIN_TITLE_SEARCH_PATH,
                        method=RequestMethod.GET)
    public @ResponseBody Collection<GiftItem> findGiftChainByTitle(
            @RequestParam(GiftSvcApi.TITLE_PARAMETER) String title,
            @RequestParam(GiftSvcApi.FILTER_FLAGGED_PARAMETER) boolean filter_flagged)
      { … }

    @RequestMapping(value=GiftSvcApi.GIFT_IN_CHAIN_TITLE_SEARCH_PATH,
                        method=RequestMethod.GET)
    public @ResponseBody Collection<GiftItem> findGiftInChainByTitle(
            @PathVariable(GiftSvcApi.ID_PARAMETER) long id,
```

```
            @RequestParam(GiftSvcApi.TITLE_PARAMETER) String title,
            @RequestParam(GiftSvcApi.FILTER_FLAGGED_PARAMETER) boolean filter_flagged)
        { … }
…
}
```

File reference:

- Potlatch/Server/src/main/java/org/coursera/androidcapstone/server/ApplicationController.java.

Let resume the data flow. When Client application (through an HTTPS client created using SecuredRestBuilder class, as it will be shown in 3c. Functional Description and App Requirement) invokes REST APIs, they will be mapped to queries to Gift CRUD repository.
In detail:

- Client request "/gift/chain" is mapped by getGiftChainsList() method in GiftSvcApi interface, that is mapped in ApplicationController class by getGiftChainsList() method, that calls gifts.findGiftChainsItems() method of GiftRepository interface. it returns the list of the head Gift for all Gift chains, that is Id, Title and Thumbnail (using GiftItem) of the Gift that started a chain. Note that when a Gift is created as standalone (not inside a chain), it creates automatically a chain in which it is the only element.
- Request "/gift/{id}/chain" is mapped by getGiftChainById() method in GiftSvcApi interface, that is mapped in ApplicationController class by getGiftChainById() method, that calls gifts.findGiftChainItemsById() method of GiftRepository interface. It returns Id, Title and Thumbnail (using GiftItem) of all the Gifts that belong to a chain with Id equal to the id parameter. Note that GiftChain field is equal to the Gift Id of the Gift that started that chain.
- Request "/gift/chain/search/findByTitle?title={title}" is mapped by findGiftChainByTitle() method in GiftSvcApi interface, that is mapped in ApplicationController class by findGiftChainByTitle() method, that calls gifts. findGiftChainByTitle() method of GiftRepository interface. It returns the list of Gift (using GiftItem) chains that have a title that is “like” the one passed as parameter. Note that the title of a Gift chain is the same of the head Gift, that is the Gift that started the chain.
- Request "/gift/{id}/chain/search/findByTitle?title={title}" is mapped by findGiftInChainByTitle() method in GiftSvcApi interface, that is mapped in ApplicationController class by findGiftInChainByTitle() method, that calls gifts. findGiftInChainByTitle() method of GiftRepository interface. It returns the list of Gifts (using GiftItem) in current chain that have a title that is “like” the one passed as parameter.

3c. Functional Description and App Requirement:

The post operation used to store Gift data requires an authenticated user account.

As exposed in 6. Basic Project Requirement only authenticated users can access to application screens that allow to create new Gift. Moreover, Android Client application exchanges information with Gift repository throw a secured REST adapter. The class

SecuredRestBuilder, taken from VideoLike assignment of previous courses, is a builder class for a Retrofit REST Adapter that extends the default implementation by providing logic to handle an OAuth 2.0 password grant login flow. The RestAdapter that it produces uses an interceptor to automatically obtain a bearer token from the authorization server and insert it into all client requests (see also 1. Basic Project Requirement).
A GiftSvcApi service used by Client application can be created in the following manner.

```
public static synchronized GiftSvcApi init(String server, String user, String pass) {
     giftSvc = new SecuredRestBuilder()
                    .setLoginEndpoint(server + GiftSvcApi.TOKEN_PATH)
                    .setUsername(user)
                    .setPassword(pass)
                    .setClientId(CLIENT_ID)
                    .setClient(new ApacheClient(new EasyHttpClient()))
                    .setEndpoint(server).setLogLevel(LogLevel.FULL).build()
                    .create(GiftSvcApi.class);
       return giftSvc;
}
```

Not that GiftSvcApi is an interface that exposes a method to post a new Gift in GiftRepository.

```
public interface GiftSvcApi {
…
     @POST(GIFT_SVC_PATH)
     public Gift addGift(@Body Gift gift);
…
}
```

where:

- GIFT_SVC_PATH is "/gift".

In the same way as exposed in 3b. Functional Description and App Requirement, the request exposed by GiftSvcApi interface is mapped by a method in ApplicationController calls and so new Gift is stored through a method of GiftRepository interface.
File references are the same as 3b. Functional Description and App Requirement.

4. Functional Description and App Requirement:

Users can view Gifts that have been posted.

This requirement is fulfilled in:

- ViewGiftChainsActivity (it uses ViewGiftListFragment).
- ViewGiftChainActivity (it uses ViewGiftListFragment).
- ViewGiftActivity (it uses ViewGiftFragment).

For additional details about that screens see 6. Basic Project Requirement.
ViewGiftListFragment shows information about GiftItem objects retrieved by Server application. For additional information about GiftItem, GiftSvcApi, ApplicationController and GiftRepository see 3b. Functional Description and App Requirement.
ViewGiftFragment instead shows information contained in an object instance of GiftDetail class.

```
public class GiftDetail implements Serializable {
```

```
    public long id;
    public String title;
    public String text;
    @Lob @Basic(fetch=FetchType.LAZY)
    public String content;
    public String createdByUsername;
    public long creationTimestamp;
    public int touches;
    public int flags;
    public boolean userTouchedBy;
    public boolean userFlagged;
…
}
```

File reference:

- Potlatch/Common/src/main/java/org/coursera/androidcapstone/common/client/GiftDetail.java.

GiftDetail contains Id, Title, Text and Content fields already exposed previously, but it has other fields:

- createdByUsername: it contains the username of the User that created the Gift.
- creationTimestamp: it contains the date / time of Gift creation.
- touches: it contains the number of users that indicated they were touched by this Gift.
- flags: it contains the number of users that flagged this Gift as offensive or inappropriate.
- userTouchedBy: it tells if current user indicated he has been touched by this Gift. It is used to initialize toggle radio button “Touched By Gift” in ViewGiftFragment.
- userFlagged: it tells if current user has flagged this Gift as offensive or inappropriate. It is used to initialize toggle radio button “Flag Gift” in ViewGiftFragment.

ViewGiftFragment uses getGiftDetailById() method of GiftSvcApi interface, that is mapped by getGiftDetailById() method of ApplicationController class.

```
public interface GiftSvcApi {
…
    @GET(GIFT_DETAIL_ID_PATH)
    public GiftDetail getGiftDetailById(@Path(ID_PARAMETER) long id);
…}

@Controller
public class ApplicationController {
…
    @RequestMapping(value=GiftSvcApi.GIFT_DETAIL_ID_PATH, method=RequestMethod.GET)
    public @ResponseBody GiftDetail getGiftDetailById(
                @PathVariable(GiftSvcApi.ID_PARAMETER) long id,
                HttpServletResponse response,
                Principal p)
      { … }
…
}
```

5a. Functional Description and App Requirement:

Users can do text searches for Gifts performed only on the Gift's title.

As exposed in 3b. and 4. Functional Description and App Requirement, GiftRepository

exposes query methods to search Gifts and Gift chains by Title. These methods are accessible to Client application through GiftSvcApi and ApplicationController (in Server application).
As exposed in 6. Basic Project Requirement and 4. Functional Description and App Requirement, ViewGiftListFragment contains a textbox field that allows to execute search on Gifts based on Title. To execute this search Android Client application uses GiftSvcApi interface.

5b. Functional Description and App Requirement:

Gifts matching the search criterion are returned for user viewing.

By default, Spring Data Rest uses a format called HATEOAS (for additional details see the link http://en.wikipedia.org/wiki/HATEOAS) to output the data returned from a Repository (in this case GiftRepository that extends a CrudRepository). The results from methods like findAll() and findByTitle() are wrapped in an Object called Resources. When this Resources object is converted to JSON, it adds additional fields to the JSON so that we don't just get back a list of Gifts objects. The extra HATEOAS "_embedded" and "_links" formatting for the top-level JSON add extra complexity. Because of the format, we can't just directly un-marshall this response into a list of Gift or GiftItem objects. To make it possible to directly un-marshall the responses as a list of Gift or GiftItem objects, it is used a class ResourcesMapper, that extends ObjectMapper and that overrides the default JSON marshalling of Spring Data Rest so that it outputs in an alternative format that allows to directly un-marshall the HTTP response bodies from GiftRepository into a list of Gift or GiftItem (or some other custom type) objects.
In this way, to populate the listview object shown in ViewGiftListFragment it is possible to use the collections returned by GiftSvcApi interface and in particular by methods like getGiftChainsList(), getGiftChainById(), findGiftChainByTitle() and findGiftInChainByTitle().
The same approach is used to populate the list showed by ViewTopGiftGiversFragment.
Class ResourcesMapper is defined in file:

- Potlatch/Server/src/main/java/org/coursera/androidcapstone/server/json/ResourcesMapper.java.

It is used in Server Application class to create the halObjectMapper that must be used:

- Potlatch/Server/src/main/java/org/coursera/androidcapstone/server/Application.java, line 169.

6a. Functional Description and App Requirement:

Users can indicate that they were *touched* by a Gift, at most once per Gift (i.e., double touching is not allowed).

Gift class contains the touchedUsers data member, a collection of Users that have been touched by the Gift (that is Users voted for it). Since information is stored about Users that have been touched by Gift in the past, it is possible to avoid that a User votes for a Gift twice.
Field touchedUsers creates a ManyToMany relationship between User and Gift since a User can vote many Gifts and a Gift can be voted by many Users.

```
@Entity
@Table(name="Gifts")
…
public class Gift implements Serializable {
…
    @JsonIgnore
    @ManyToMany
    @JoinTable(
            name="TouchedUsers",
            inverseJoinColumns=@JoinColumn(name="Username",
                                           referencedColumnName="Username"),
            joinColumns=@JoinColumn(name="GiftId", referencedColumnName="Id")
    )
    private Set<User> touchedUsers;
…
}
```

File reference:

- Potlatch/Common/src/main/java/org/coursera/androidcapstone/common/repository/Gift.java, line 119.

```
@Entity
@Table(name="Users")
public class User implements Serializable {
…
    @Id
    @Basic
    @Column(name="Username",updatable=false,nullable=false,length=32)
    private String username;

    @Column(name="Avatar")
    @Lob @Basic(fetch=FetchType.LAZY)
    private String avatar;

    @JsonIgnore
    @OneToMany(mappedBy="createdBy",fetch=FetchType.LAZY)
    private Set<Gift> gifts;
…
}
```

File reference:

- Potlatch/Common/src/main/java/org/coursera/androidcapstone/common/repository/User.java.

GiftSvcApi interface exposes methods that allow a User to indicate preferences for a particular Gift.

```
public interface GiftSvcApi {
…
    @POST(GIFT_TOUCHEDBY_PATH)
    public Void touchedByGift(@Path(ID_PARAMETER) long id);

    @POST(GIFT_UNTOUCHEDBY_PATH)
    public Void untouchedByGift(@Path(ID_PARAMETER) long id);

    @GET(GIFT_TOUCHED_PATH)
    public Collection<String> getUsersTouchedByGift(@Path(ID_PARAMETER) long id);
…
}
```

where:

- GIFT_TOUCHEDBY_PATH is "/gift/{id}/touchedby": User has been touched by Gift.
- GIFT_UNTOUCHEDBY_PATH is "/gift/{id}/untouchedby": User is no more touched by Gift.
- GIFT_TOUCHED_PATH is "/gift/{id}/touched": list of string Usernames of the Users that have indicated they have been touched by the specified Gift.
- ID_PARAMETER is "id".

6b. Functional Description and App Requirement:

Users can flag Gifts as being obscene or inappropriate. Users can set a preference that prevents the display of Gifts flagged as obscene or inappropriate.

Gift class contains the flaggedByUsers data member, a collection of Users that have flagged the Gift as obscene or inappropriate. Since information is stored about Users that flagged Gift in the past, it is possible to avoid that a User flags a Gift twice. Moreover, using EditPreferencesActivity it is possible to enable filtering of Gifts that current User, or other Users, have flagged in the past (that is filter condition is that flaggedByUsers collection is not empty).
Field flaggedByUsers creates a ManyToMany relationship between User and Gift since a User can flag many Gifts and a Gift can be flagged by many Users.

```
@Entity
@Table(name="Gifts")
public class Gift implements Serializable {
…
    @JsonIgnore
    @ManyToMany
    @JoinTable(
            name="FlaggedByUsers",
            inverseJoinColumns=@JoinColumn(name="Username",
                                           referencedColumnName="Username"),
            joinColumns=@JoinColumn(name="GiftId", referencedColumnName="Id")
    )
    private Set<User> flaggedByUsers;
…
}
```

File reference:

- Potlatch/Common/src/main/java/org/coursera/androidcapstone/common/repository/Gift.java, line 144.

For details about User class see 6a. Functional Description and App Requirement.

GiftSvcApi interface exposes methods that allow a User to flag a particular Gift.

```
public interface GiftSvcApi {
…
    @POST(GIFT_FLAGGED_PATH)
    public Void flagGift(@Path(ID_PARAMETER) long id);

    @POST(GIFT_UNFLAGGED_PATH)
    public Void unflagGift(@Path(ID_PARAMETER) long id);

    @GET(GIFT_FLAGGEDBY_PATH)
```

```
    public Collection<String> getUsersWhoFlaggedGift(@Path(ID_PARAMETER) long id);
…
}
```

where:

- GIFT_FLAGGED_PATH is "/gift/{id}/flagged": User has flagged Gift as obscene or inappropriate.
- GIFT_UNFLAGGED_PATH is "/gift/{id}/unflagged": User has removed flag for Gift.
- GIFT_FLAGGEDBY_PATH is "/gift/{id}/flaggedby": list of the string Usernames of the Users that have flagged the specified Gift as offensive or inappropriate.
- ID_PARAMETER is "id".

7a. Functional Description and App Requirement:

Touched counts are displayed with each Gift.

ViewGiftFragment shows the current touches (and flags) count for selected Gift. Value is contained in object of type GiftDetail returned by getGiftDetailById() method declared in GiftSvcApi interface. For additional details about ViewGift screen see 6. Basic Project Requirement. Touched counts is stored in Gift object in GiftRepository to have fast access to this information. For additional details about touches see also 6a. Functional Description and App Requirement.

7b. Functional Description and App Requirement:

Touched counts can be periodically updated in accordance with a user-specified preference (e.g., Touched counts are updated every 1, 5 or 60 minutes) or updated via push notifications for continuous updates.

As exposed in 3. Basic Project Requirement, in Client application AlarmManager service is used to schedule a task that periodically (based on user preferences set using EditPreferencesActivity) queries repository to update touched counts of Gift showed in ViewGiftFragment, or information showed in ViewTopGiftGiversFragment.
Class GiftUpdater extends BroadcastReceiver and waits AlarmManager notification. When notification arrives, GiftUpdater sends Intent “FragmentUpdater” to interested local BroadcastReceivers (that is registered receivers inside application context) using LocalBroadcastManager service.
ViewGiftFragment and ViewTopGiftGiversFragment have a BroadcastReceiver data member that registers for Intent “FragmentUpdater” notifications:

- When ViewGiftFrament receives notification, it calls getGiftCountersById() method declared in GiftSvcApi interface. That method returns an object instance of class GiftCounters, a small container for information related to Gift counters. In this way only the minimal amount of information is sent from Server to Client to update view.
- When ViewTopGiftGiversFragment received notification, it calls findTopGiftGivers() method declared in GiftSvcApi interface to refresh listview.

```
public class GiftUpdater extends BroadcastReceiver {
    public final static String LOG_TAG = GiftUpdater.class.getCanonicalName();

    @Override
```

```
    public void onReceive(Context context, Intent intent) {
        Log.d(LOG_TAG, "onReceive()");

        LocalBroadcastManager localBroadcastManager =
            LocalBroadcastManager.getInstance(context);
        localBroadcastManager.sendBroadcast(new Intent("FragmentUpdater"));
    }
}
```

File reference:

- Potlatch/Client/app/src/main/java/org/coursera/androidcapstone/client/ui/updater/GiftUpdater.java.

```
public class GiftCounters implements Serializable {
…
    public long id;
    public int touches;
    public int flags;
…
}
```

File reference:

- Potlatch/Common/src/main/java/org/coursera/androidcapstone/common/client/GiftCounters.java.

Note that update requests are sent to Server application only is fragment is active and visible to avoid unnecessary network traffic.
In the following lines a portion of LogCat output is reported to show update of ViewGiftFragment using GiftUpdater as exposed previously.

```
11-26 19:00:01.032: D/org.coursera.androidcapstone.client.ui.updater.GiftUpdater(1194): onReceive()
11-26 19:00:01.082: D/org.coursera.androidcapstone.client.ui.gift.ViewGiftFragment(1194):
BroadcastReceiver.onReceive()
11-26 19:00:01.082: D/org.coursera.androidcapstone.client.ui.gift.ViewGiftFragment(1194):
updateGiftCounters(100)
11-26 19:00:01.112: D/Retrofit(1194): ---> HTTP GET https://10.0.2.2:8443/gift/100/counters
11-26 19:00:01.112: D/Retrofit(1194): Authorization: Bearer dbac9441-0d2a-4c44-8095-123e9af888a3
11-26 19:00:01.112: D/Retrofit(1194): ---> END HTTP (no body)
11-26 19:00:01.172: D/Retrofit(1194): <--- HTTP 200 https://10.0.2.2:8443/gift/100/counters (43ms)
11-26 19:00:01.172: D/Retrofit(1194): Server: Apache-Coyote/1.1
11-26 19:00:01.172: D/Retrofit(1194): X-Content-Type-Options: nosniff
11-26 19:00:01.172: D/Retrofit(1194): X-XSS-Protection: 1; mode=block
11-26 19:00:01.172: D/Retrofit(1194): Cache-Control: no-cache, no-store, max-age=0, must-revalidate
11-26 19:00:01.182: D/Retrofit(1194): Pragma: no-cache
11-26 19:00:01.202: D/Retrofit(1194): Expires: 0
11-26 19:00:01.202: D/Retrofit(1194): Strict-Transport-Security: max-age=31536000 ; includeSubDomains
11-26 19:00:01.202: D/Retrofit(1194): X-Frame-Options: DENY
11-26 19:00:01.202: D/Retrofit(1194): X-Application-Context: application
11-26 19:00:01.202: D/Retrofit(1194): Content-Type: application/json;charset=UTF-8
11-26 19:00:01.212: D/Retrofit(1194): Transfer-Encoding: chunked
11-26 19:00:01.212: D/Retrofit(1194): Date: Thu, 27 Nov 2014 00:00:01 GMT
11-26 19:00:01.212: D/Retrofit(1194): {"id":100,"touches":1,"flags":1}
11-26 19:00:01.212: D/Retrofit(1194): <--- END HTTP (32-byte body)
```

8. Functional Description and App Requirement:

App can display information about the top “Gift givers,” i.e., those whose Gifts have touched the most people.

This requirement is fulfilled using ViewTopGiftGiversFragment as exposed in 6. Basic Project Requirement.
To populate the list, GiftSvcApi interface exposes findTopGiftGivers() method that returns a collection of TopGiftGivers objects.

```
public interface GiftSvcApi {
…
    @GET(GIFT_TOP_GIVERS_PATH)
    public Collection<TopGiftGiver> findTopGiftGivers();
…
}
```

where:

- GIFT_TOP_GIVERS_PATH is "/gift/search/findTopGivers".

TopGiftGiver is a serializable class that contains information about User's username, avatar and total number of touches.

```
public class TopGiftGiver implements Serializable {
…
    public String username;
    @Lob @Basic(fetch=FetchType.LAZY)
    public String avatar;
    public long touches;
…
}
```

File reference:

- Potlatch/Common/src/main/java/org/coursera/androidcapstone/common/client/TopGiftGiver.java.

Collection is descending sorted on number of "touches". If two Users have the same touches value, they are sorted using username ascending lexicographic order.