Basic Project components

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

This requirement is met in two ways. In database there is a table (Users) that contains user information. At the moment it contains only Username field, but it could contain also password (encrypted), email address (to activate user and to recover lost password) and an avatar. Moreover access to application is permitted 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 hardcoded users: “alice” (hardcoded password “alicepass”), “bob” (hardcoded password “bobpass”) and “carol” (hardcoded password “carolpass”).

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. 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 (a single Gift is a Gift chain without other linked gifts).

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.
- CreateGiftActivity: create a new Gift, either attached to a chain or as a starter of new chain depending on context.
- EditPreferencesActivity: it allows a user to choose how often touched counts are updated and if user wants to filter flagged gifts.
- ViewTopGiftGiversActivity: it shows information about the top “Gift givers”, the users that have posted Gifts that have touched the greatest number of other users.

To fulfill the 7b. Functional Description and App Requirement ("Touched counts can be periodically updated in accordance with a user-specified preference") I think to use the AlarmManager service to schedule update of touches counters periodically.

4. Basic Project Requirement:
App interacts with at least one remotely-hosted Java Spring-based service

Android Client application interacts with more than one remotely-hosted Java Spring-based service. For example it interacts with Server part of the application for authentication / authorization (see 1. Basic Project Requirement) and through REST APIs accesses to Gift information stored in a CrudRepository.

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 CrudRepository is accessed though REST APIs and data object are marchalled and unmarshalled using JSON.

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 will be several activities that will correspond to fragments and so user interface screens:

- LoginScreenFrament: login panel to access to application content. If credentials are correct user can see the list of Gift chains (ViewGiftChainActivity).
- ViewGiftChainsFragment: the list of Gift chains. User sees the title of the gift that is at the head of the chain (maybe it could be possible to add a thumbnail of the Gift near the title). In this view there is textbox that allows to filter Gift chains by title, 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 (ViewGiftChainActivity). A Gift chain has at least one Gift.
- ViewGiftChainFragment: the list of Gifts in one chain. User sees the titles of the Gifts that belong to this chain (also in this list it could be possible to add a thumbnail of the Gift near the title). In this view there is textbox that allows to filter Gifts by title, 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 (ViewGiftActivity).
- ViewGiftFragment: view details of a Gift, in particular Title, Body, Image, CreationDate, Username (of the creator), Number of touches, Number of flags. In this view there are (toggle) buttons to allow Gift voting (touch/flag).
- CreateGiftFragment: create a new Gift, either attached to a chain or as a starter of new chain depending on context. In this view there are two textbox (one for Title and one for Body), one box to preview picture that can be captured using camera or can

be chosen from stored pictures (there will be two buttons: Select and Snapshot), and two buttons Confirm and Cancel to confirm or cancel Gift creation.

- EditPreferencesFragment: it allows a User to choose how often touched counts are updated and if user wants to filter flagged gifts (Gift is filtered if contains at least one flag by one User).
- ViewTopGiftGiversFragment: 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 and total number of touches.

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.

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 post results to main UI Thread.

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 Blob (array of byte) in CrudRepository.

```
@Entity
@Table(name="Gifts")
```

```
public class Gift implements Serializable {
…
    @Id
    @Column(name="Id")
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long id;

    @Column(name="Title")
    private String title;

    @Column(name="Body")
    private String body;

    @Column(name="Content")
    @Lob @Basic(fetch= FetchType.LAZY)
    private byte[] content;

    @Column(name="ContentType")
    private String contentType;
…
}
```

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 CreateGiftActivity/CreateGiftFragment as exposed in 6. Basic Project Requirement.

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")
public class Gift implements Serializable {

    @Id
    @Column(name="Id")
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long id;
…
    @Column(name="GiftChain")
    private long giftChain;
```

```
…
}
```

Class Gift has a data member named giftChain. This field contains the Id of the head Gift if current Gift belongs to a chain, or it contains the Id value if current Gift started a new chain (that is Id == GiftChain for head Gift in the chain).

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>.

```
@RepositoryRestResource(path = GiftSvcApi.GIFT_SVC_PATH)
public interface GiftRepository extends CrudRepository<Gift, Long> {
…
      // Find all gifts with a matching title (e.g., Gift.title)
      public Collection<Gift> findByTitle(
                  // The @Param annotation tells Spring Data Rest which HTTP request
                  // parameter it should use to fill in the "title" variable used to
                  // search for Gifts
                  @Param(GiftSvcApi.TITLE_PARAMETER) String title);
…
}
```

The @RepositoryRestResource annotation tells Spring Data Rest to expose the GiftRepository through a controller and map it to the GiftSvcApi.GIFT_SVC_PATH (equal to "/gift") path. This automatically enables to do the following:

- List all Gifts by sending a GET request to /gift.
- Add a Gift by sending a POST request to /gift with the JSON for a gift.
- Get a specific Gift by sending a GET request to /gift/{id} (e.g., /gift/1 would return the JSON for the Gift with id = 1).
- Send search requests to our findByXYZ methods to /gift/search/findByXYZ (e.g., /gift/search/findByTitle?title=Foo).

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

```
@RepositoryRestResource(path = GiftSvcApi.GIFT_SVC_PATH)
public interface GiftRepository extends CrudRepository<Gift, Long> {
…
        public Collection<GiftItem> findGiftChainsList();
```

```
    public Collection<GiftItem> findGiftChainById(
                                    @Param(GiftSvcApi.ID_PARAMETER) long id);

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

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

where and TITLE_PARAMETER is "title" and ID_PARAMETER is "id".

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 and Title 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. Also these information will be available through REST APIs.

```
public class GiftItem implements Serializable {
…
    public long id;
    public String title;
…
}
```

The previous methods returns the following information:

- findGiftChainList: it returns the list of the head Gift for all Gift chains, that is Id and Title 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.
- findGiftChainById: it returns Id and Title of all the Gift that belongs to a chain with Id equal to the parameter. Note that chain Id is equal to the Gift Id of the Gift that started that chain.
- findGiftChainByTitle: it returns the list of Gift 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.
- findGiftInChainByTitle: it returns the list of Gifts 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 views 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 GiftService 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".

4. Functional Description and App Requirement:
Users can view Gifts that have been posted.

This requirement is fulfilled in ViewGiftChainsActivity / ViewGiftChainsFragment and ViewGiftChainFragment / ViewGiftChainFragment (see 6. Basic Project Requirement).

GiftSvcApi is an interface that exposes some methods to get the list of Gifts.

```
public interface GiftSvcApi {
…
      @GET(GIFT_SVC_PATH)
```

```
    public Collection<Gift> getGiftsList();

    @GET(GIFT_ITEM_CHAIN_PATH)
    public Collection<GiftItem> getGiftChainsList();

    @GET(GIFT_CHAIN_ID_PATH)
    public Collection<GiftItem> getGiftChainById(@Path(ID_PARAMETER) long id);

    @GET(GIFT_ID_PATH)
    public Gift getGiftById(@Path(ID_PARAMETER) long id);

    @GET(GIFT_TITLE_SEARCH_PATH)
    public Collection<Gift> findByTitle(@Query(TITLE_PARAMETER) String title);

    @GET(GIFT_CHAIN_TITLE_SEARCH_PATH)
    public Collection<GiftItem> findGiftChainByTitle(@Query(TITLE_PARAMETER) String title);

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

where:

- GIFT_SVC_PATH is "/gift".
- GIFT_ITEM_CHAIN_PATH is "/gift/chain".
- GIFT_CHAIN_ID_PATH is "/gift/{id}/chain".
- GIFT_ID_PATH is "/gift/{id}".
- GIFT_TITLE_SEARCH_PATH is "/gift/search/findByTitle".
- GIFT_CHAIN_TITLE_SEARCH_PATH is "/gift/chain/search/findByTitle".
- GIFT_IN_CHAIN_TITLE_SEARCH_PATH is "/gift/{id}/chain/search/findByTitle".
- ID_PARAMETER is "id".
- TITLE_PARAMETER is "title".

The previous methods allow to retrieve the following information:

- getGiftsList: it retrieves the complete collection of Gifts.
- getGiftChainsList: it retrieves the collection of GiftItems containing information of Gift chains.
- getGiftChainById: it retrieves the collection of GiftItems containing the information about Gift chain whose head Gift has Id data member equal to id parameter, that is all the Gifts whose GiftChain data member has value equal to id parameter.
- getGiftById: it retrieves the information about Gift with Id data member equal to id parameter.
- findByTitle: it retrieves the collections of Gifts that have Title that is “like” the one passed as parameter.

- findGiftChainByTitle: it retrieves the collection of GiftItems containing the information about Gift chains whose Title is “like” the one passed as parameter.
- findGiftInChainByTitle: it retrieves the collection of GiftItems containing the information about Gifts in Gift chain with GiftChain data member equal to id parameter and Title that is “like” the one passed as parameter.

For additional information about Gift Item see 3b. Functional Description and App Requirement.

5a. Functional Description and App Requirement:
Users can do text searches for Gifts performed only on the Gift's title.

As exposed in 3b. Functional Description and App Requirement, GiftRepository exposes query methods to search Gifts and Gift chains by Title.
As exposed in 4. Functional Description and App Requirement, the interface GiftSvcApi gives access to query methods that allow to search Gift and Gift chain by Title.
Finally, as exposed in 6. Basic Project Requirement, ViewGiftChainsFragment and ViewGiftChainFragment contain a textbox field that allow to execute search on Gifts based on Title. To execute this searches Android Client application will use 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 is a CrudRepository). The results from methods like findAll() and findByTitle(), in this case, 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 adds extra complexity. Because of the format, we can't just directly unmarshall this response into a list of Gift or GiftItem objects. To make it possible to directly unmarshall 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 unmarshall the HTTP response bodies from GiftRepository into a list of Gift or GiftItem objects.
In this way, to populate the listview objects shown in ViewGiftChainsFragment and ViewGiftChainFragment it is possible to use the collections returned by GiftSvcApi interface and in particular by methods like getGiftChainsList, getGiftChainById, findGiftChainByTitle and findGiftInChainByTitle.

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 are 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 {
…
    @ManyToMany
    @JoinTable(
      name="TouchedUsers",
      joinColumns=@JoinColumn(name="Username", referencedColumnName="Username"),
      inverseJoinColumns=@JoinColumn(name="GiftId", referencedColumnName="Id")
    )
    private Set<User> touchedUsers;
…
    public long getTouches() {
      return this.touchedUsers.size();
    }
…
}
```

At the moment User class contains only Username information, but it could be possible to extend it adding more information (as exposed in 1. Basic Project Requirement).

```
@Entity
@Table(name="Users")
public class User implements Serializable {
…
    @Id
    @Column(name="Username")
    private String username;

    @OneToMany
    @JoinColumn(name="CreatedBy", referencedColumnName="Username")
    private Set<Gift> gifts;
…
}
```

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" (number of touches for Gift) and 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 are 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 as per user base 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 {
…
    @ManyToMany
    @JoinTable(
      name="FlaggedByUsers",
      joinColumns=@JoinColumn(name="Username", referencedColumnName="Username"),
      inverseJoinColumns=@JoinColumn(name="GiftId", referencedColumnName="Id")
    )
    private Set<User> flaggedByUsers;
…
    public long getFlags() {
      return this.flaggedByUsers.size();
```

```
    }
…
}
```

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" (number of flags for Gift) and ID_PARAMETER is "id".

7a. Functional Description and App Requirement:
Touched counts are displayed with each Gift.

ViewGiftFragment shows the current touches count for selected Gift (see also 6. Basic Project Requirement). This information is stored in Gift object since application stores the set of Users that have been touched by current Gift (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 I think to use AlarmManager service to schedule a task that periodically (based on user preferences set using EditPreferencesActivity) queries repository to update touched counts of Gifts contained in application cache. I think that it could be possible to update content only for Gifts that User has viewed in the past and that are stored in cache, used to speed up application if User wants to view again a Gift that has viewed before.

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 a method that returns the list of Gifts.

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

where GIFT_TOP_GIVERS_PATH is "/gift/search/findTopGivers" and TopGiftGiver is a serializable class that contains information about User’s username and total number of touches.

```
public class TopGiftGiver implements Serializable {
…
    public String username;
    public long touches;
…
}
```

Collection is descending sorted on number of “touches”. If two Users have the same touches value, they could be sorted using username ascending lexicographic order.