Camp Potlatch Application – Source Code Roadmap

This document presents the roadmap to the Camp Potlatch source code. In order to easy the source code analysis, to each major requirement is presented the lines of the code or the class which are responsible and collaborate to the requirement fulfillment.

When one class is referenced in this document, the text can refers to its instances, objects, not the class exactly. This convention is maintained here to simplify the document, without loosing the precision necessary to understand the code, its behavior and states.

The application is divided in two parts: client side, Android application part; and server side, the Java Spring service part, hosted in the Amazon Web Service cloud.

The client side was implemented using the Android Studio IDE, due to the resources provided by it to create user interface and better support for the Gradle configuration and build system. The server side was implemented using the Eclipse IDE because of the Spring Framework and AWS Framework support.

Even tough the client side was developed in the Android Studio IDE, it is easy to create an Eclipse project for the client side, compile the source code and install the "apk" application package in Android devices using the Android SDK plug-in.

Package Structure

All the packages references in this text begins with org.coursera.camppotlatch.

The Android client part, and the network interface classes, of the code resides below the org.coursera.camppotlatch.client package. The server part, Spring service, resides below the org.coursera.camppotlatch.service package.

Client Side

The client part has classes (and interfaces) for Android user interaction (activities, fragments, list adapters, views), background thread handlers, Android service, models, Refrofit JSON proxies, JSON messages exchange, and client side Oauth 2 authentication, which are also stored in separated packages.

- client.view – Android user interaction classes, such as activities, fragments, list adapters and views;

- client.handler – Android handlers in background threads used to refresh contents in the screen;

- client.androidservice – Android service which notifies the current activity to update its contents;

- client.model – models, as known as data objects, which are the primary structured data handled by the application, such as gifts, users, like marks and inappropriate marks – these are the models as seen by the client side, with several common fields which are used to exchange data with the server side;

- client.serviceproxy – Retrofit proxies and factories used to interact with the service controllers, located in the server side, using JSON over HTTPS;

- client.json – JSON extension classes used to serialize special objects and configure GSON library;

- client.auth – adapted classes, from the Jules and Mitchell code, for the client side authentication with the OAuth 2 protocol;

- client.commons – utility and helper classes for common uses throughout the client code and a class to store the application context data of client part.

Server Side

The service part has classes (and interfaces) for the Spring controllers, models, DynamoDB repository interaction, S3 file interaction, OAuth 2 authentication, and JSON messages exchange, which are stored in separated packages. For the client-server interface and JUnit tests, the server side also contains some client packages.

- service – initialization and configuration of the Java Spring-based web service;

- service.controller – controllers to receive RESTful HTTP requests with multipart files and JSON contents, control the request processing and respond to the client – they are the main components of the server side;

- service.model - models, as known as data objects, which are the primary structured data handled by the application, such as gifts, users, like marks and inappropriate marks – these are the models as seen by the server side, with several common fields which are used to exchange data with the client side;

- service.repository – classes that interface with the DynamoBD in order to create, load, update and remove gifts, users, like marks and inappropriate marks;

- service.file – classes that interface with the S3 distributed file system to store and retrieve image files for gifts and users;

- service.auth – classes that interface with the OAuth 2 authentication protocol to recover user information from repository and configure the OAuth 2 protocol for user and client authentication;

- service.json – classes to configure Jackson JSON library and serialization of special objects;

- service.commons - utility and helper classes for common uses throughout the server code and a class to store the application context data of the server part.

The client code in the server part:

- client.serviceproxy – Retrofit proxies and factories used to interact with the service controllers, located in the server side, using JSON over HTTPS;

- client.json – JSON extension classes used to serialize special objects and configure GSON library;

- client.auth – adapted classes, from the Jules and Mitchell code, for the client side authentication with the OAuth 2 protocol.

Basic Project Requirements

1. App supports multiple users via individual user accounts

The user accounts are stored in the Amazon DynamoBD.

In the server side:

- service.repository - **UserRepository** class is responsible to interact with the DynamoBD interface to create, update, select and remove users.

- service.model - **User** class, contains user data.

- service.auth - **OAuth2SecurityConfiguration** class configures the OAuth 2 authentication protocol in the server side. Other classes from this package are used to access the users repository.

In the client side:

- client.auth – **SecuredRestBuilder** class construct the Refrofit service proxy configured to authenticate client and users through password and attaching authentication token that bears user authentication in the OAuth 2 authentication phase for the subsequent message exchanges.

- client.serviceproxy – **PotlatchServiceConn** uses the SecuredRestBuilder to create and maintain Retrofit service proxies.

- client.view – **LoginActivity** class presents an interface for the user insert his user name and password for authentication.

- client.view – **CreateUserAccountActivity** and **EditUserAccountActivity** classes are used to show interfaces to create and edit user account, respectively.

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

All the functions contained in the user interfaces can only be used by authenticated users, except the create user account function, which every person can use to create an account.

- client.view - Android activities and fragments which presents the UI.

The **CreateUserAccountActivity** is used to create user accounts and does not depend of authentication.

The first screen presented is the login screen, in **LoginActivity** class, which requires user authentication for the rest of the user interactions. **LoginActivity** calls the **PotlachServiceConn** with the user name and password to create an authentication token and this is used throughout the application to call the application web service functionalities, which are necessary in all activities and commands.

The **CreateUserAccountActivity**, the exception, uses a special user to interact with the server to create the user account.

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

- Activity
- BroadcastReceiver
- Service
- ContentProvider

This application is composed of several Android activities and one Android service.

- client.view – all Android activities and the used fragments and view classes.

- client.androidservice – the Android service used to notify gift list activities to refresh their contents.

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

All the basic operations for this application is made in the server side, in an application web service, where resides the business logic.

The application is based on the Spring framework.

- service – **Application** and **ApplicationServletInitializer** classes show that the application uses the **SpringApplicationBuilder** and **SpringApplication** classes to create and run the Spring framework and this Spring-based application.

5. App interacts over the network via HTTP

The client side interacts with the server side using a secure version of HTTP, the HTTPS.

The protocol used and the address of the server side and can be seen in the 22 of the **PorlatchServiceConn.java** file, on the client.serviceproxy package. The elasticbeanstalk.com domain pertains to the Amazon Web Service. The service is hosted in EC3 machines and controlled by the Elastic Beanstalk, both from the AWS.

- client.auth – **EasyHttpClient** is a client implemented by third parts, used under the Apache Licence 2.0, which provides automatic HTTP and HTTPS communication clients. This class also supports compacted HTTP messages exchanges, if the web server also have this mechanism implemented.

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

At runtime, the user can navigate through more than 3 screens, for example, in the client.view package the application has some classes which present screens for the UI:

- **LoginActivity** - login screen.

- **CreateUserAccountActivity** – user account creation screen.

- **EditUserAccountActivity** – user account edition screen.

- **ChangePasswordUserAccountActivity** – password changing.

- **CreateGiftActivity** – gift creation screen.

- **EditGiftActivity** – gift edition screen.

- **MainActivity** shows the below listed fragments when the user selects itens in the navigation menu:

    - **NewGiftsFragment** – list of the recent gifts.

    - **TopGiftsFragment** – list of the top gifts.

    - **UserGiftsFragment** – list of the user gifts.

    - **UsersFragment** – list of the users.

    - **TopGiftGiversFragment** – list of the top gift givers.

- **ViewUserAccountFragment** – shows the user account.

- **RelatedGiftsActivity** use the **RelatedGriftsFragment** to show the related gifts (in the gift chain).

- **ViewGiftImageActivity** – shows the gift image.

All the **GiftListFragments** extend the **AbstractGiftsFragment**.

All the **UserListFragments** extend the **AbstractUsersFragment**.

*7. 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.

This application uses two advanced capabilities of Android:

1) Touch gestures – to resize and move gift images in the gift image view screen

- client.view – **GiftImageView** class contains methods to listen to gestures and modify the image viewing accordingly.

2) Multimedia capture – to capture images from the device camera

- client.view – the **CreateUserAccountActivity**, **EditUserAccountActivity** and **CreateGiftActivity** classes:

- create an MediaStore.ACTION_IMAGE_CAPTURE intent to invoke the device camera to capture the image from the real world;

- pass the path to store the image file;

- invoke the startActivityForResult Activity method to start a camera activity

- receive on the onActivityResult method the result code notifying that the image is already saved in the file.

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

All the operations involving gifts and users creation, loading, updating and removal and also the associated images loading are done in background threads. This avoid to use the UI Thread for long duration operations, and therefore lets the application more responsive for the user interactions.

The more quick operations, like a single gift creation, updating or removal are done using the Android AsyncTask. Below are listed classes which uses background threads in AsyncTasks.

- client.view:

- **CreateGiftActivity**, **CreateUserAccountActivity**,

- **AbstractGiftsFragment**, **AbstractUsersFragment**, **ViewGiftImageActivity**

- **EditGiftActivity**, **EditUserAccountActivity**, **ChangePasswordUserAccountActivity**

The gift list view updating and user list view updating, which can take more time, these tasks are

done using Android Handlers which handle a pool of messages in specially created background threads. Semaphores are used to control the concurrency among the list updating requests, if necessary, as when the scroll try to request next gifts from the servers.

- client.view:

- **AbstractGiftsFragment** and **AbstractUsersFragment** classes use Android handlers, passing messages to reload a list of gifts (or users) from a certain page and using a certain searching text. Also, they can use the Android handler to retrieve a single gift (or user) to update a gift (or user) list view, in the case of the refresh necessary after a gift edition or a liking operation, for example.

- client.handler:

- **GiftsRefreshThread** and **UsersRefreshThread** classes are background thread types which contains a message pool associated and a message handler. They refresh the associated list (gifts or users) upon message requests, for a list update, beginning in a certain page of gifts in the server and using a search text that is sent and processed in the server. They can also be used to refresh a single object in the list, as talked before.

Functional Description and App Requirements

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

A gift contains id, title, an image id, and a comment as its main attributes. Also, the image contains the creator user name, creator name, creation time, likes count, like indicator, inappropriate count, inappropriate indicator, caption gift id and related gifts count.

The image id identifies the an image globally in this application. The caption gift id identifies what is the caption gift of a related gift, in a gift chain.

The gift are model, data object, both used in the client and server side, as expected. They differ a little in the client and server sides, because of uses difference in each side. For example, the client gift has list index, image path, image bitmap, thumbnail image path, thumbnail bitmap that the server doesn't must to have. The server side, in another hand, have the S3 link which refers to the S3 file which stores the image file in the distributed file system in the AWS cloud.

- client.model – **Gift** class is the gift model in the client side;

- server.model – **Gift** class is the gift model in the server side.

- server.model – **GiftRepository** class is used to store and retrieve gifts in the server side.

2. 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.

A user can use the application to create a gift taking a picture or selecting an image stored on the device. Then he must add a title to the gift and optionally a comment.

-client.view

- **CreateGiftActivity** class shows the user interface for this operation. There are two button in this screen: one to capture a photo using the camera and other to choose an image from the device gallery or from other directories.

- **EditGiftActivity** class shows a screen which allows a user to edit a gift, which he posted,

modifying its title and comments.

- **AbstractGiftsFragment** class implements a method, used by the derived classes, which allows them to present a pop up menu in their screens to select the remove gift operation. This class also provides a contextual menu to remove some gifts in a single UI operation.

3. Gifts creation

3a. Once the Gift is complete the User can post the Gift to a *Gift Chain* (which is one or more related Gifts).

A related gifts list screen appear after that the user clicks a label below a certain gift in a list, called a caption gift. In this screen, the user can see and create related gifts to the caption gift, aiming, for example, to post gifts which have the same theme or are created from images captured in the same place of the caption gift. The caption gift and its related gifts form a gift chain, with the caption gift as the first and main gift.

- client.view – **RelatedGiftsActivity** class uses the **RelatedGiftsFragment** class to present the related gift list screen.

- client.model – **Gift** class contains the caption gift id field in which related gifts maintain the id of the associated caption gift. The caption gifts uses the related count field to maintain the count of the gifts which are related to itself.

- server.model – **Gift** class in the server side maintain the same field with the same meaning.

3b. Gift data is stored to and retrieved from a web-based service accessible in the cloud.

When created or updated, a gift is sent to the server using a JSON body in requests over the HTTPS protocol. The gift controller in the Spring web service then handle this request and uses the gift repository class to store and update the gift in the DynamoDB.

The gift image is sent using a multipart file in a HTTPS request, which the gift controller handle and calls the gift repository which invokes helper classes to store image files in the S3 file system. The image id is stored in the DynamoDB.

The gift retrieval occurs in the reverse order using JSON body and stream over HTTPS responses to retrieve, respectively, the gift data and image.

The service is hosted in EC2 Linux machines with a Tomcat web server. The Spring framework container is initialized and runs over these web servers. The loading balancing is enabled in the AWS, so more EC2 machines can be started and be stopped on demand, according to the application use. The Elastic Beanstalk manages the deployment of web services in isolated environment, monitors the application use and alert the application administrator in case of the occurrence of certain events, such as application deployment and application health problems.

The HTTPS is enable in AWS using test certificates generated for this application.

- client.model – Gift class, contains and exchanges gift data from the client side.

- client.serviceproxy – GiftServiceApi interface is used to create a Refrofit class to send and retrieve Gifts using JSON bodies over HTTPS. Also this generated Retrofit class is used to send and retrieve the gift images through multipart files and streams, respectively.

- client.json – DateJsonSerializer class used to marshal and unmarshal date and time to JSON objects using GSON library in the client side.

- service.model – Gift class, contains and exchanges gift data from the server side.

- service.controler – GiftController class implements the Spring MVC controller to handle requests and send responses to the clients.

- service.repository – GiftRepository class interface with the DynamoDB NoSQL scalable database, directly, without using the Spring Data JPA, due to some limitation found in its current implementation.

- service.file – GiftImageFileHelper class uses the S3FileManagerHelper to store and retrieve gift image to and from the S3 distributed file system, based on the S3 link stored for the associated gift in the DynamoDB, which makes reference to each image file in the cloud.

- service.json – DateJsonSerializer and DateJsonDeserializer classes, used to marshal and unmarshal date and time to JSON objects using Jackson library in the server side.

3c. The post operation used to store Gift data requires an authenticated user account

Only authenticate users can post Gifts, and also perform the other operation, excluding the user creation operation.

The login screen only allows users to enter in the application if the user has an account and authenticate himself to the application using his user name and password.

Moreover, the OAuth 2 protocol in the server side requires that each client request carries a token that bears the user authentication data. Because all operations which modifies or read gifts and users data depends of the application web service, then only authenticate users can use the client part in the mobile device.

Authenticated user can only perform authorized operations. For example, a user can create and post gift, can like gifts of other users and mark them as inappropriate, but can't edit or remove gifts from other users.

client.view – LoginActivity class shows the login screen.

client.serviceproxy – PotlatchServiceConn class passes the user name and password to the SecureRestBuilder class in order to get a Retrofit client instance that can generated messages with the authenticate token.

client.auth – SecureRestBuilder class implements the client side interaction of the OAuth 2 protocol to authenticate the client and users.

service.auth – OAuth2SecurityConfiguration – configure the Spring Security OAuth 2 authentication protocol on the server side.

4. Users can view Gifts that have been posted.

After a gift posting, all authenticated users can view this gift. The gift can appear in the:

- new gifts list, which presents the gifts in the creation order, the most recently created first;

- top gifts list, in the order of likes it received from users;

- user gifts list of user which created this gift;

- related gifts list, if the gift is a related gift, in a gift chain, not a caption (or main) gift.

The code for this functionality is in the following classes:

- client.view

- AbstractGiftsFragment class which implements the basic gifts list behavior for all the gifts lists views;

- NewGiftsFragment class which extends the AbstractGiftsFragment class to present the gifts in the creation descending order;

- TopGiftsFragment class which extends the AbstractGiftsFragment class to present the gifts in the likes descending order first and creation descending order next;

- UserGiftsFragment class which extends the AbstractGiftsFragment class to present the gifts create by a user in the creation descending order;

- RelatedGiftsFragment class which extends the AbstractGiftsFragment class to present the gifts, in a gift chain, related to a caption (or main) gift, in the creation descending order;

- MainActivity class uses a fragment manager to replace one of these 3 first derived fragment (NewGiftsFragment, TopGiftsFragment, UserGiftsFragment) and other fragments, and then to show the view corresponding view to the user;

- UserGiftsActivity class that uses the UserGiftsFragment class to show the gifts list of a certain user in a separated activity;

- RelatedGiftsActivity class that uses the RelatedGiftsFragment class to show the gifts list related to a certain caption gift in a separated activity.

5. Gifts searching

5a. Users can do text searches for Gifts performed only on the Gift's title.

Users can do gift searching using typed texts in the action bar, in the search view, to obtain all gifts in which title contains the typed text, in a case insensitive matching.

- client.view

- AbstractGiftsFragment class presents the search view in the action bar and listen to the search submit. The code there then pass the search to the GiftsRefreshThread handler.

- client.handler

- GiftsRefreshThread class creates a backgroup thread with a message pool and a message handler. The handler receives the gift list refresh request messages in this background thread and asynchronously to the UI thread, and calls the appropriated method in the associated gift list fragment class in order to obtain the gifts from the server.

- client.view

- NewGiftsFragment, TopGiftsFragment, UserGiftsFragment or RelatedGiftsFragment class, asynchronously to the UI thread, using the handler background thread, calls appropriated method in the Refrofit client to access the server to perform the search.

- client.serviceproxy

- using the dynamically generated class based on the GiftServiceApi calls the server to perform the search, depending of the gifts list being viewed by the user.

- service.controller

- GiftController class has one of its method called to retrieved searched gifts, according to the list been viewed, the offset of the first gift of this list (in case of new queries, equals to 0) and the search text.

-service.repository

- GiftRepository class has one of its method called to retrieved searched gifts, according to the list been viewed, the offset of the first gift of this list (in case of new queries, equals to 0) and the search text. It, then, interacts with the DynamoDB API to construct dynamically a query (or scan), to invoke this query (or scan) on this NoSQL cloud distributed database, and to treat the result to send back to the client.

5b. Gifts matching the search criterion are returned for user viewing.

The reverse from the (5a) Item is followed to get and presents the new lists contents, in the ListView, using the GiftListAdapter (client.view package), in the appropriate gifts list Android UI fragment.

The thumbnail image, if it is not load in the client from a previous call, are load after performing calls for each thumbnail image against the server. The cache control to see if the thumbnail image is already loaded is made based in the thumbnail image id.

The gift full image is only loaded from the server if the user wants to view this image using the ViewGiftImageActivity. Also, the imaged is cached in the client and the cache verification is controller by the image id.

6. Gifts tags

6a. Users can indicate that they were *touched* by a Gift, at most once per Gift (i.e., double touching is not allowed)

When a user is touched by a Gift, he can express this using the like icon below the Gift. The server maintain information about which user liked which gift to avoid double liking. Instead, if the user click the like icon again, the like information is removed in the server, indicating that the user changed his mind.

- client.view

- AbstractGiftsFragment class contains method to listen to the like click from the user.

- GiftListAdapter class show the like count, for each gift, and changes the like icon to blue or gray if the current user, respectively, liked or not the associated gift.

- client.model

- Gift class in the client side maintains likes count and if the current users liked or not the gift object.

- service.controller

- GiftController receives like and unlike operation invocations from the client side and invokes the corresponding methods in the LikeMarkRepository class, GiftRepository class, and UserRepository class in order to create the like mark association between a user and a gift, update the likes count of the gifts, and update the likes count of the users. Also, GiftController return gifts

to the client with the correct count updated and if the current user liked or not this gift.

6b. 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.

The same control for the like mark in done for the inappropriate mark for the gifts.

In the client side, if application sees that the settings of the current user asks to filter inappropriate gifts, then in each request to retrieve gifts list from the server the client passes this information to the server, that then filters inappropriate gifts in the server side.

- client.model

    - User class objects contains information about the user settings. The current user object is held by the AppContext class in the (client.commons package). The client side knows the current user settings this way.

- service.model

    - User class maintains and persists user settings in the server side. So, in the subsequent logons this information is passed to the client.

- service.controller

    - The GiftController class receives requests to filter inappropriate gifts in gifts list request and pass then to the GiftRepository classe.

-service.repository

    - The GiftRespository receives request to filter inappropriate gifts and exclude them from the result returned.

7. Touched counts display and refresh

7a. Touched counts are displayed with each Gift

As already mentioned in the item (6a), the likes count in maintained by the persistence layer for gifts and users. The GiftListAdapter class (client.view package) is responsible to show the likes count besides the likes icon.

This application also maintains related gifts count and show this appropriately in gifts list through the same GiftListAdapter class.

7b. 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.

The user settings is presented in the user account screen and can be modified in the user account edition screen. This preference, like the inappropriate gifts filter preference, is maintained in the server side and can be recovered in the next application accesses by the user.

The update period is informed to the Android service, responsible to notify the current activity to refresh its contents, every time the activity (or fragment), which uses the Android service, resumes its execution and binds to its interface (AIDL interface). When the activity (or fragment) is paused, the timer is removed and the service is unbinded.

Then, after that the user modify the update period, in the user account edition screen, when the

user reenter the screen of one gift list, the corresponding activity (or fragment) resumes, binds to the Android service, and informs the new update period to it. The Android service then creates a new timer to begins the refresh notifications using the new informed period.

- client.view

- AbstractGiftsFragment class binds the Android service, DataRefreshService class, when resumes, and unbind when it pauses. The communication between the Android service and the AbstractGiftsFragment, which contains the base behavior for the gifts lists, is done using one AIDL interface for the service receive requests, and other for the client receive the Android service callback invocation, informing that it is time for the gifts list refreshing.

- client.androidservice

- DataRefreshService class that contains the Android service implementation for periodic gifts list refresh, and maintain a timer to perform callback invocations on the bound clients to inform that it is time for refreshing.

- IDataRefreshService.aidl (main/aidl directory) – AIDL interface definition to requests services from the DataRefreshService.

- IdataRefreshServiceCallback.aidl (main/aidl directory) – AIDL interface definition to be called back from the service when it is time to refresh the corresponding gifts list screen.

- client.model

- User class maintains user preferences in the client side for the updating gifts period.

- service.model

- User class maintains user preferences in the server side, persisted also between client side application uses.

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

The application maintain information about how much is sum of likes the gifts posted by a user received. This likes count, as shown in the item (6a.), is maintained in the User class, both of the client side and of the server side.

The TopGiftGiversFragments shows a screen in which the users are sorted by the number of likes theirs gifts received from all the users, the user which received more likes first.

- client.view

- TopGiftsFragments, that extends the AbstractUsersFragment, shows the users in the likes count descending order first and creation date descending order next.

- client.model

- User class maintains the likes count that his posted gifts received from all users, in the client side.

- service.model

- User class maintains the likes count that his posted gifts received from all users, in the server side.