DailySelfie Android Application

The app enables users to take pictures of themselves - selfies - over an extended period of time. It periodically reminds the user to take a selfie and presents the selfies in a list that makes it easy to see how the user has changed over time. This extended implementation also allows users to process their selfies to add effects, such as blurring or charcoaling.

Lets take a look at how all the requirements in Daily Selfie Specification have been addressed:

Application Development Environment :

Developed on Android Studiov1.4.

Target API:23, Tested on 21 and 19.

I have tested application on my Lenovo p780 with OS 4.4.2, and genymotion emulator with nexus 6 and OS 5.1.0. It was working fine on both of the devices.

I have used JHLabs for filter processes. Also I reused last MOOCs code in my server and client implementation

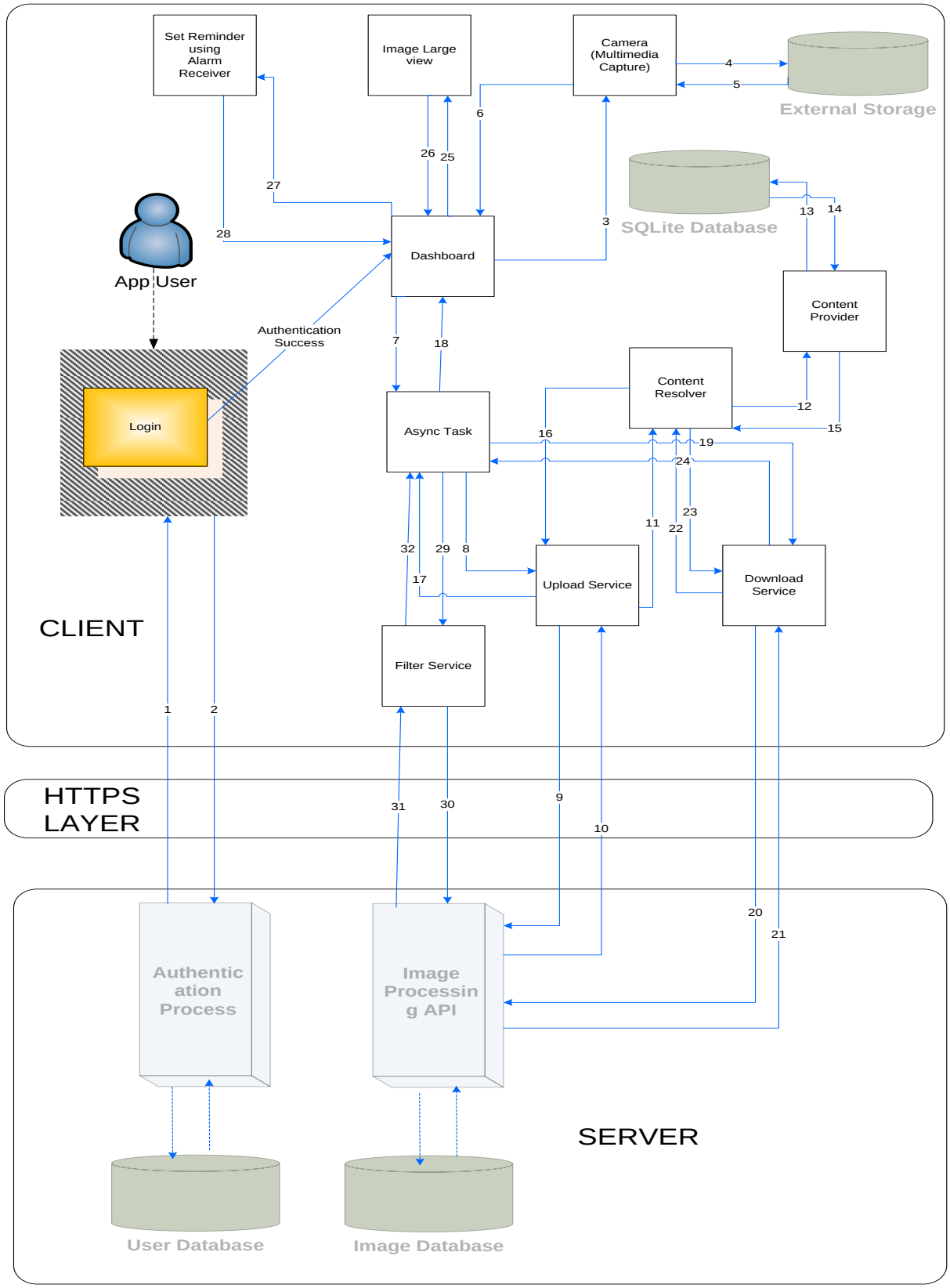


Figure 0.1 Daily Selfie Application Architecture

1. Basic Project Requirement:

App supports multiple users via individual user accounts

Rubric:
0 - Needs Improvement: Does not support multiple users
1 - Partially Complete: Supports multiple users via individual accounts, but cap on number of users is too small
1 - Partially Complete: Supports multiple users, but without individual accounts
2 - Complete: Supports multiple users via individual accounts (cap on number of users is reasonable)

I have hardcoded three users in the OAuth2SecurityConfiguration class on server, using the UserDetailService. Each selfie uploaded is added to SelfieRepository with attached username. So every time a user logs in, only selfies having its username will be displayed. You can add or delete users by adding or deleting them from the list in code not dynamically. I have not done register user or dynamic user functionality.

2. Basic Project Requirement:

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

Rubric:
0 - Needs Improvement: No user facing function requires authenticated user account
1 - Partially Complete
2 - Complete: Contains one or more user facing functions requiring authenticated user account

Status : Done

Dashboard Activity will never open unless and until server authorizes the user. SelfieServiceProxy is initialized in SelfieDataMediator class with the username and password provided by user in LoginActivity. All the methods on server in SelfieController class have @PreAuthorize, and can only be invoked when the user have been granted acces. If the user enters wrong username or password, or tries to access resources on server without authentication, an exception is returned by server. The SecuredServerException which occurs with Login Failure and RetrofitError which

occurs when you try to acces resources without authorization are handled in LoginActivity, which helps in showing the appropriate toast message on screen.

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**

Rubric:
0 - Needs Improvement: Uses no components
1 - Partially Complete: Uses at least one instance of only one component
2 - Complete: Uses at least one instance each of at least 2 components

Status : Done

Activity : DashboardActivity, LoginActivity, Detail Activity, AlarmActivity
BroadcastReceiver : AlarmReceiver, ResultReceiver present in DashboardActivity. Both extends BroadcastReciever
Service : UploadSelfieService, FilterSelfieService, FilterSelfieService all extends FilterService
ContentProvider : SelfieProver and all classes in model/content are used to implement Content Provider.

4. Basic Project Requirement:

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

Rubric:
0 - Needs Improvement: App does not interact with a Java Spring-based service
1 - Partially Complete
2 - Complete: App interacts with 1 or more remotely-hosted Java Spring-based service(s)

Status : Done

Implemented using DS_Serverv2. The application interacts with it over https. OAuth2.0 and retrofit is used to send requests and receive results. SelfieServiceProxy in DailySelfie2 and SelfieServiceApi in DS_Client shows the retrofit request types. SecuredRestBuilder is used to initialize the https connection to the server.

5. Basic Project Requirement:
App interacts over the network via HTTP/HTTPS.

Rubric:
0 - Needs Improvement: App does not interact over the network via HTTP/HTTPS
1 - Partially Compete
2 - Complete: App interacts over the network via HTTP/HTTPS

Status : Done

Implemented using DS_Serverv2. The application interacts with it over https. OAuth2.0 and retrofit is used to send requests and receive results. . SelfieServiceProxy in DailySelfie2 and SelfieServiceApi in DS_Client shows the retrofit request types. SecuredRestBuilder is used to initialize the https connection to the server.

6. Basic Project Requirement:

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

Rubric:
0 - Needs Improvement: Users can only access 1 (or no) interface screens at runtime
1 - Partially Complete: Users can navigate between only 2 user interface screens at runtime
2 - Complete: Users can navigate between 3 or more user interface screens at runtime

Status : Done

Screens include:

1. Login Screen
2. Dashboard which shows all the selfie records
3. Set Reminder to set reminder for notifications to take selfie
4. Detail Screen to show the image in large view

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.**

Rubric:
0 - Needs Improvement: App does not use an advanced capability or API from the MoCCA list
1 - Partially Complete
2 - Complete: App uses at least one advanced capability or API from the MoCCA list above

Status : Done

Animation: LoginActivity uses animation class to show the progress bar as rotating circle which shows when you press Login button, and hides when user gets authenticated.

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.

Rubric:
0 - Needs Improvement: The app does not support any operations off the UI Thread
2 - Complete: The app does support at least one operation off the UI Thread

Status : Done

LoginActivity : LoginTask : When user clicks Login Button, LoginTask is started in attemptLogin() method. This AsynTask creates a background thread, in which authentication is done in doInBackground(), and based on its result, if authenticated go to dashboard and if not authenticated shows authentication failure.

DashboardActivity : SelfieListOps : Contains uploadSelfie, downloadSelfie and applyFilter methods to download a selfie, upload a selfie and apply filters on selfies. All the tasks are done in background thread.

1. Functional Description and App Requirement:

App allows user to take and save a Selfie.
Rubric:
0 - Needs Improvement
2 – Complete

Status : Done

Uses Android Camera Application to take picture and return its URI to upload it on server and save its data in application's content provider .
Look for "//Take Picture using Android's Camera API" in DashboadActivity to check implementation.

2. Functional Description and App Requirement:

App reminds *User* to take a Selfie.

Status : Done

TimePickerDialog and DatePickerDialog to enter time and date of alarm. When user clicks ON Button, Alarm for the selected time is created using alarm manager class. The AlarmReciever class receives the alarm at that time and shows the notification to user to take a selfie. The user can set an alarm for a particular time and even set it on repeating, currently the repeating interval is five minutes and user will get notification after every five minutes starting from the set time. AlarmActivity and AlarmReciever can be checked for implementation

3. Functional Description and App Requirement:

App allows user to view saved Selfies in a ListView.

Rubric:
0 - Needs Improvement
1 - Partially Complete
2 - Complete

Status : Done

RecyclerView is used with records showing as List of Cards. Each card shows the thumbnail of image and its name, plus a checkbox. The layout is defined in activity_dashboard.xml and its content in content_dashboard.xml. The view is populated using SelfieRecordAdapter which extends **RecyclerView.Adapter<SelfieRecordAdapter.SelfieRecordViewHolder>** where SelfieRecordViewHolder holds the view defined in content_dashboard.xml.

4. Functional Description and App Requirement:

If the User closes and then reopens the App, the user has access to all saved Selfies.

Rubric:
0 - Needs Improvement
1 - Partially Complete
2 – Complete

Status : Done

Every time user logs in, a request to server is sent to get the list of selfies for this user by calling SelfieListOps execute method which in turn calls getSelfieList of SelfieDataMediator in background. So every time the user logs in, the selfies associated with his/her account will be displayed.

5. Functional Description and App Requirement:

The App allows the User to select one or more Selfies, select one or more graphic effects, and apply the selected effects to the selected Selfies, and then view the processed Selfies in a ListView.

Rubric:
0 - Completely Missing
1 - Partially Complete: Only some of these requirements are present.
2 - Complete: App allows users to apply graphic effects and view results

Status : Done

DashboardActivity:
User can select multiple selfie records using checkbox and apply filters on them. The apply filter icon on menu, when clicked, displays a chooser dialog where you can select one or multiple filters. When dialog's submit button is clicked, async call to SelfieListOps applyFilter method is done on each image. So if there are three images selected, then three background task will be created. The applyFilter method starts SelfieFilterService, which starts a notification to show filter is being applied. It sends an filter applied message back to activity which has a receiver to receive it and update the results on screen. SelfieFilterService calls SelfieDataMediator applyFilter which in turn calls the SelfieServiceProxy's applyfilters method to send request to server.

6. Functional Description and App Requirement:

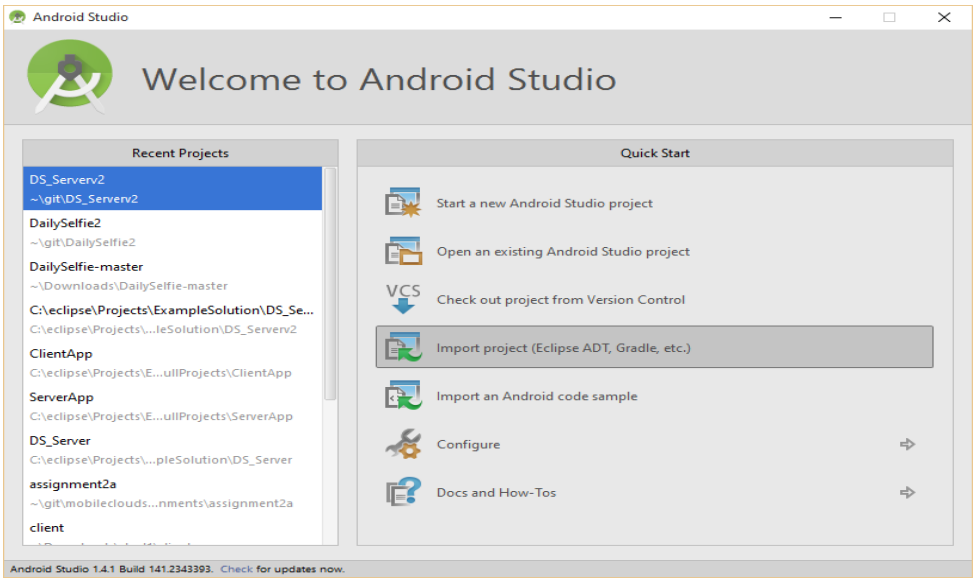
Some part of the operations corresponding to Requirement 5 (Applying Graphics Effects) must be executed concurrently. Therefore, App allows Users to start new a graphic effects operation, while a previous one is still in progress.
Rubric:
0 - Needs Improvement
2 - Complete: Users can execute multiple concurrent operations

Status : Done

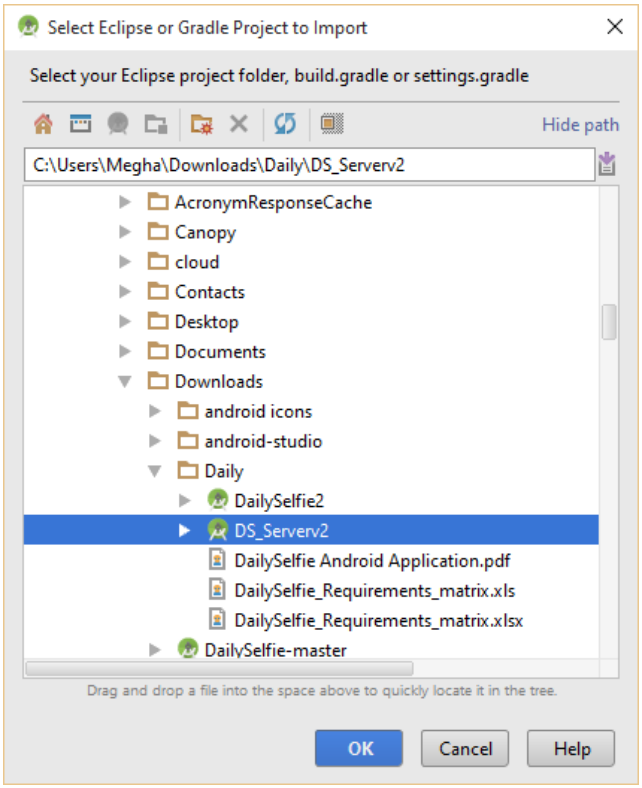
FilterService on server implements the FutureTask<Void> method, where the filters are applied actually on images. The applyFilters method in SelfieController calls this method. So for every set of image and filters list sent, a task will be created in background to handle it. Suppose we send two requests one after the another – Request one with two images and two filter list, Request two with three images and one filter. Request one will generate two tasks in background thread and Request two will generate three tasks in background thread. Still we would be able to send more requests of any type on server.

How to configure Server :

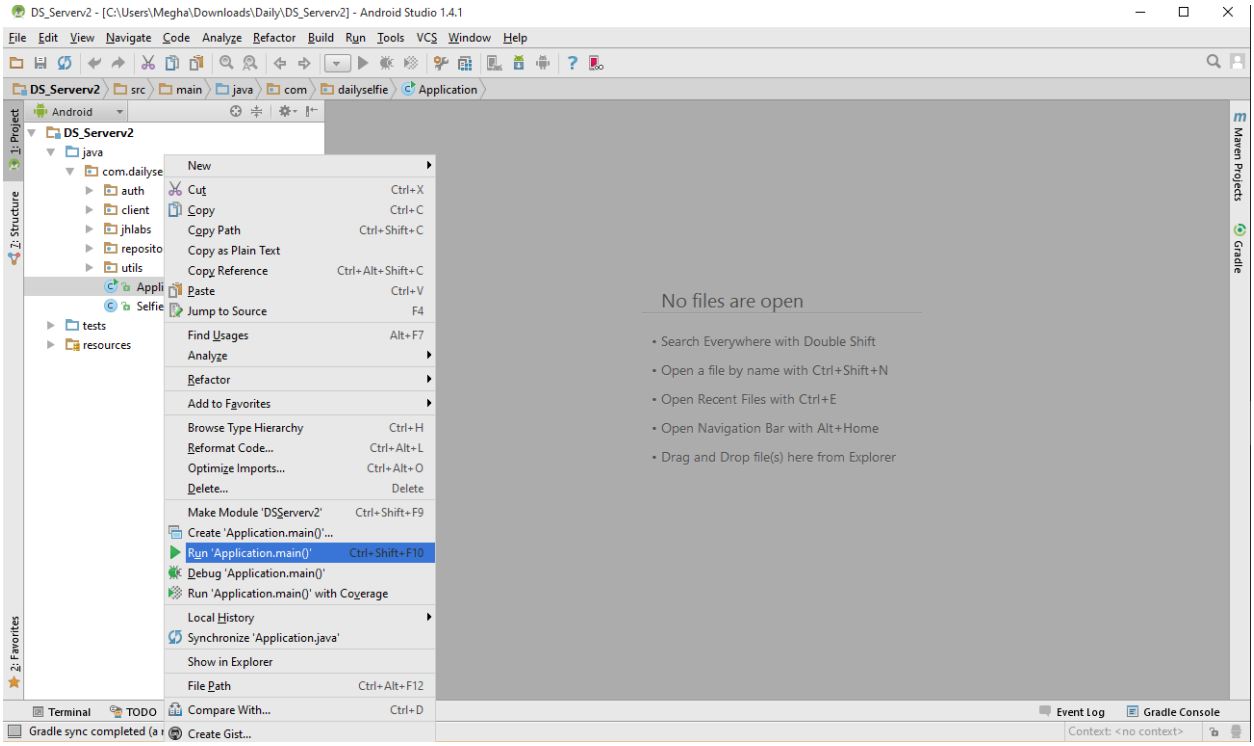
1. Import project



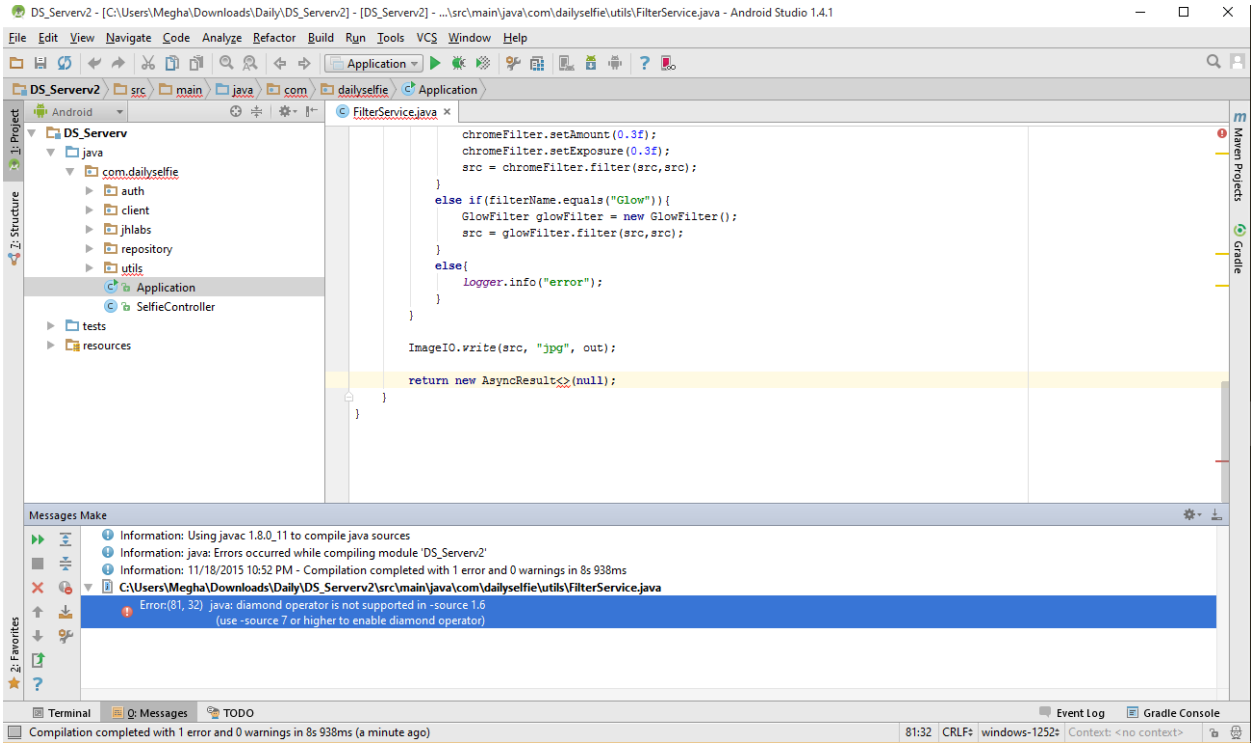
2. Select the Project

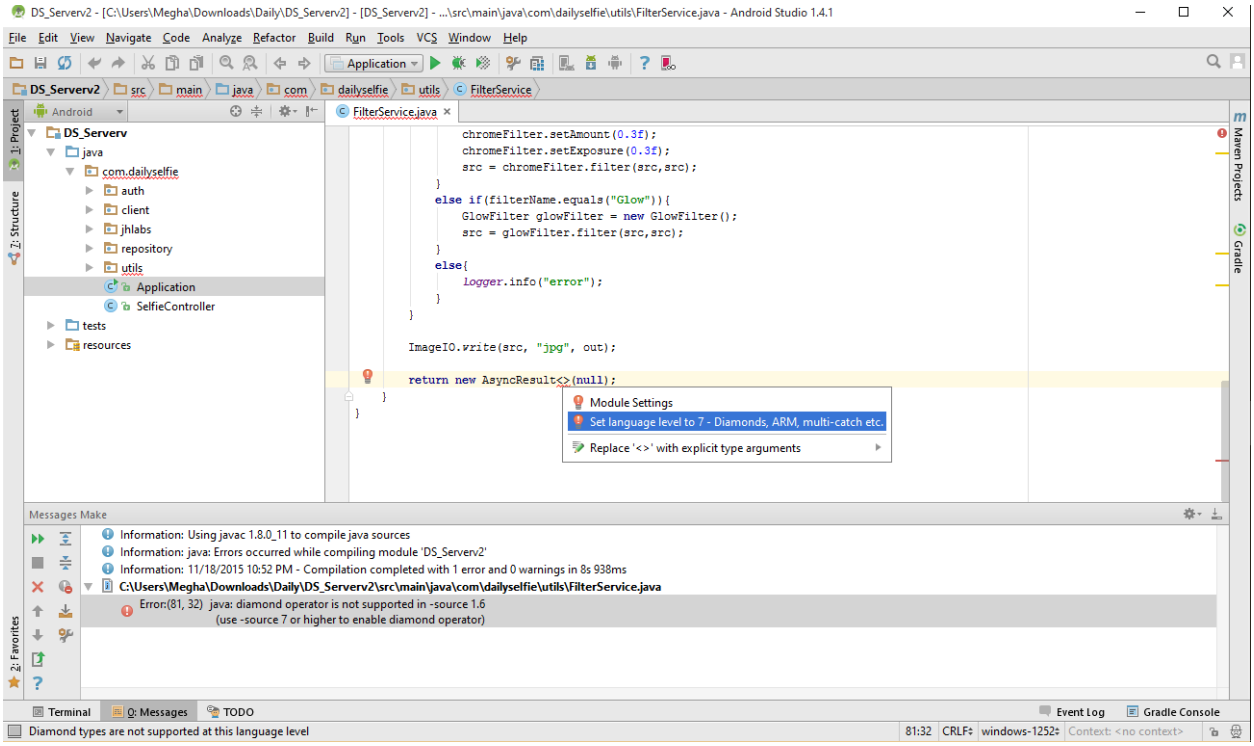


3. Run application.main() by right clicking on it

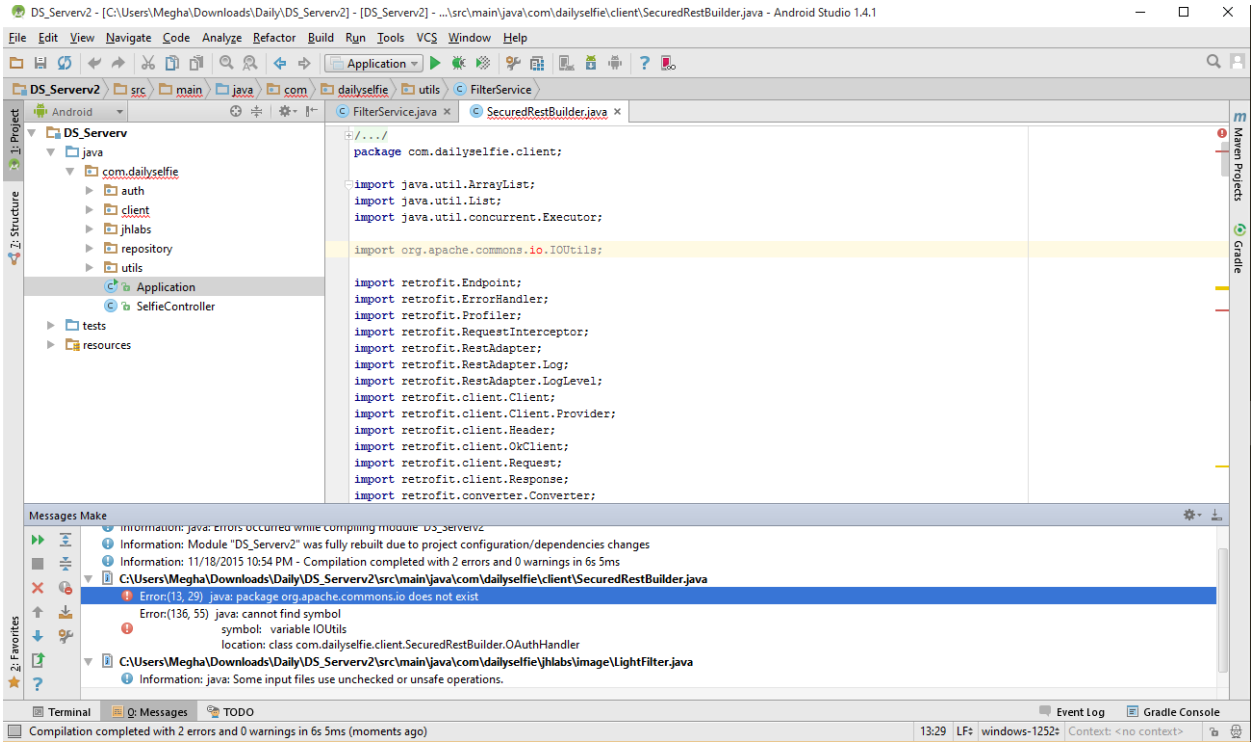


4. You might get this error, so change compliance to 1.7

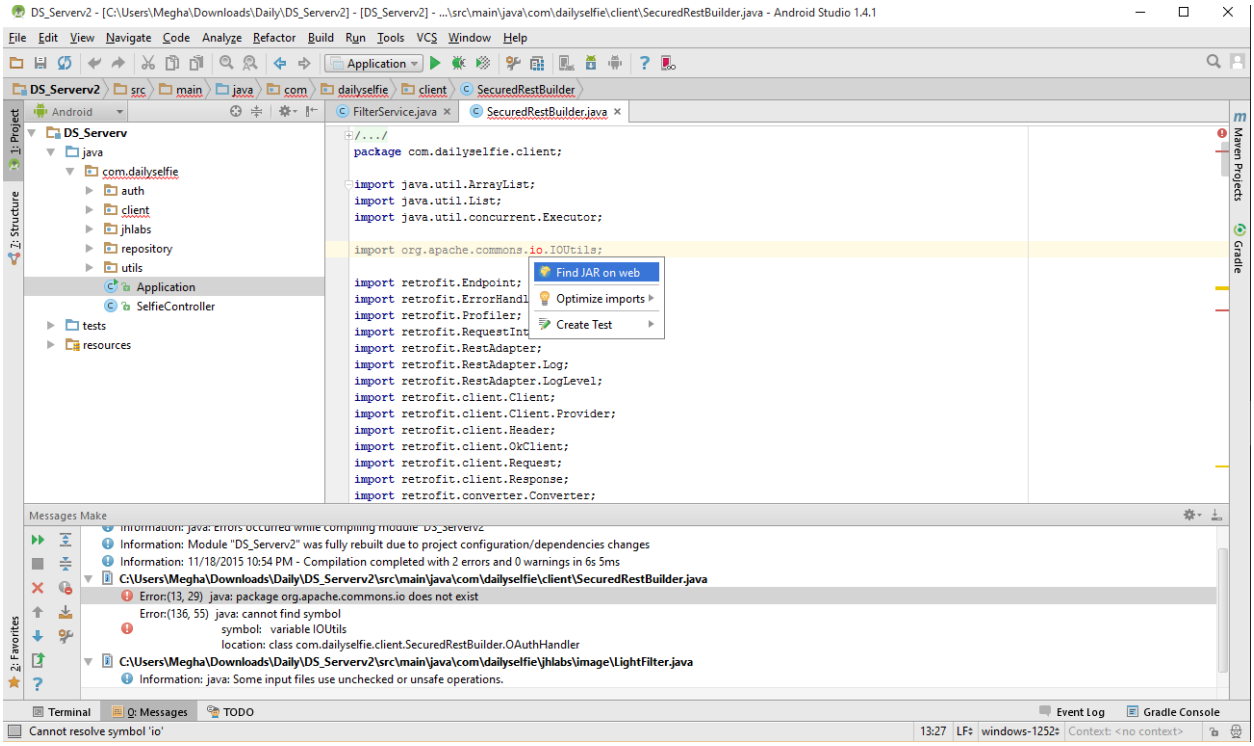


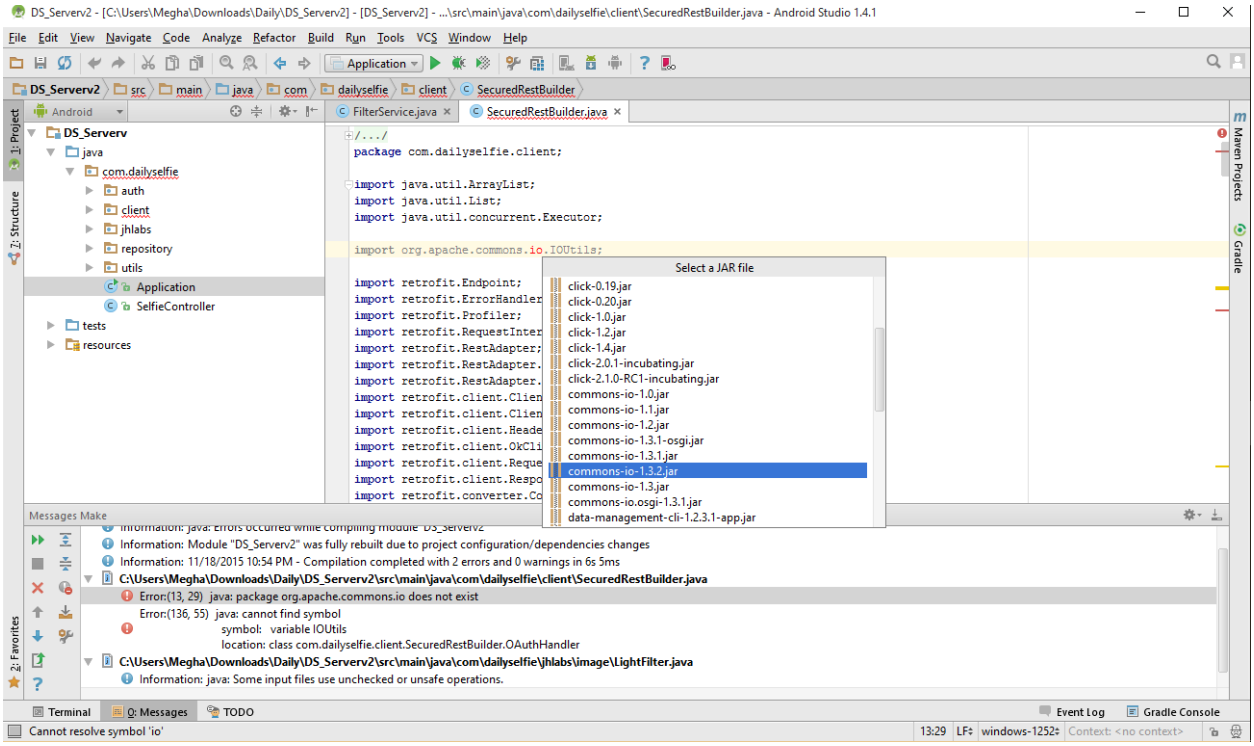


3. Again run, you might get this error.

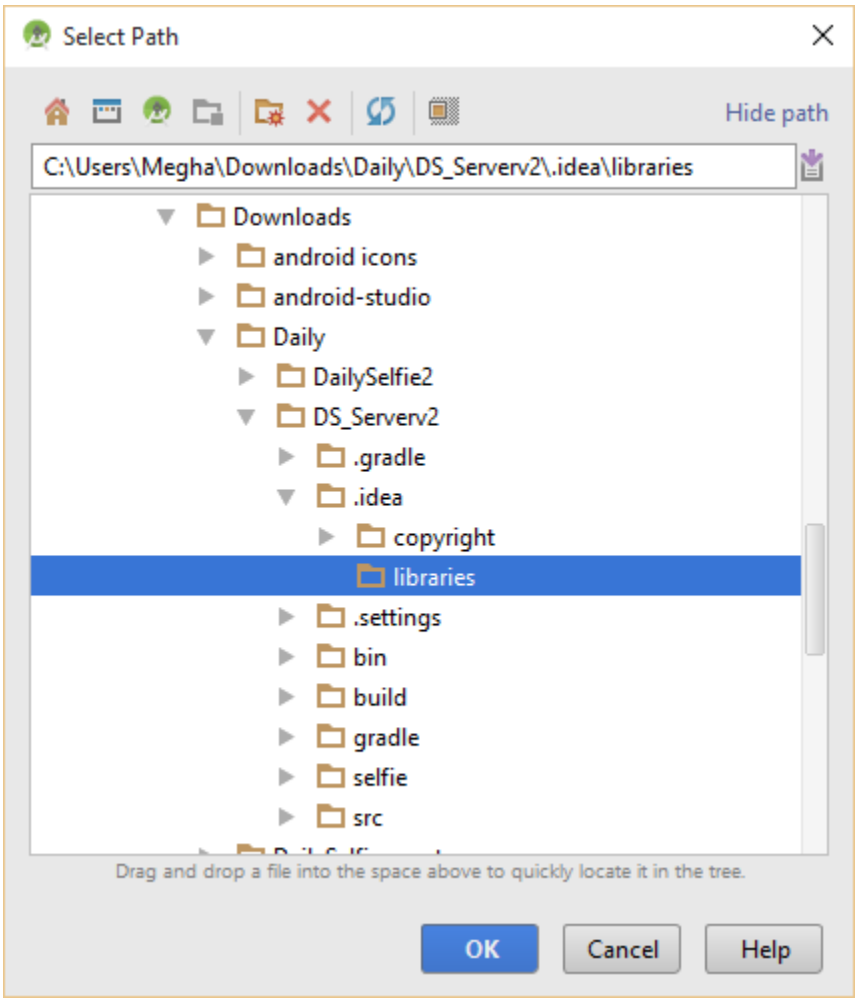


Add jar from web





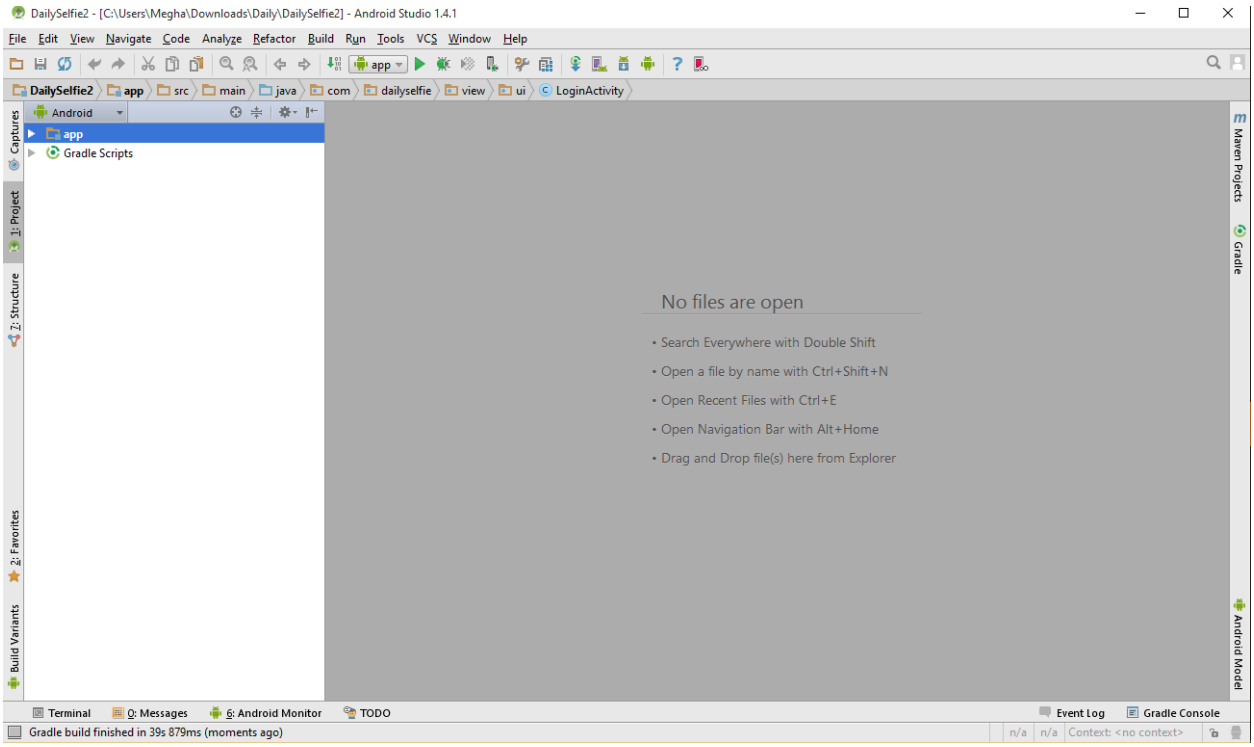
Save it in libraries folder of the same project



Now the server will run fine.

Configuring DailySelfie2

1. File>Import> **DailySelfie2**



2. Build>Rebuild Project

It will go smoothly

3. Change server address to run in different devices.

For running on device: change to address of your machine. You can find address of machine using ip-config in command prompt.

For running on genymotion emulator: https://192.168.56.1:8443

For running on android emulator : http://10.0.2.2:8443