Contents

[Model: 2](#_Toc421602750)

[ Business 2](#_Toc421602751)

[ Comment 3](#_Toc421602752)

[ Friend 4](#_Toc421602753)

[ Post 4](#_Toc421602754)

[ Review 5](#_Toc421602755)

[ Search 6](#_Toc421602756)

[ User 7](#_Toc421602757)

[View 9](#_Toc421602758)

[ Activity 9](#_Toc421602759)

[ Adapter 12](#_Toc421602760)

[ Dialog 13](#_Toc421602761)

[ Fragment 14](#_Toc421602762)

[ Interface\_m 15](#_Toc421602763)

[ Widget\_customized 15](#_Toc421602764)

[Controller 16](#_Toc421602765)

[ Helper 16](#_Toc421602766)

[ image\_loader 17](#_Toc421602767)

[ Object 17](#_Toc421602768)

Layers:

* Model
* View
* Controller

## Model:

* Business:
  + BlockUser: It is an HttpGet webservice. A business owner blocks a user. Blocked user cannot write a comment on a post which belongs to the business.

Input: BUSINESS\_ID: int (business’ id who owns the post)

Output: ResultStatus

* + DeleteBusiness: It is an HttpGet webservice. A business owner can delete his own business.

Input: BUSINESS\_ID: int (the deleting business), USER\_ID: int (the business owner)

Output: ResultStatus

* + GetBlockedUsers: It is an HttpGet webservice. It returns blocked users of a business.

Input: BUSINESS\_ID: int (the business’ id), BEFORE\_THIS\_ID: int (is used to load more data), LIMITATION: int (is used to load more data)

Output: ArrayList<BaseAdapterItem>

* + GetBusinessContactInfo: It is an HttpGet webservice. It returns a business contact info.

Input: BUSINESS\_ID: int (the business’ id)

Output: Business

* + GetBusinessFollowers: It is an HttpGet webservice. It returns the list of a business’ followers.

Input: BUSINESS\_ID: int (the business’s id)

Output: ArrayList<BaseAdapterItem>

* + GetBusinessGategories: It is an HttpGet webservice. It returns categories of the business.

Input: null

Output: ArrayList<Category>

* + GetBusinessHomeInfo: It is an HttpGet webservice. It returns a business’ home information e.g. profile picture id.

Input: BUSINESS\_ID: int (the business’ id), USER\_ID (the business owner’s id)

Output: Business

* + GetBusinessProfileInfo: It is an HttpGet webservice. It returns a business’ profile information e.g. address.

Input: BUSINESS\_ID: int (the business’ id)

Output: Business

* + GetBusinessSubcategories: It is an HttpGet webservice. It returns subcategories of the business.

Input: CATEGROY\_ID: int (the selected category’s id)

Output: ArrayList<SubCategory>

* + RateBusiness: It is an HttpGet webservice. A user can rate a business with an integer value between 1 and 5.

Input: BUSINESS\_ID: int (the business’ id), USER\_ID: int (the user’s id), RATE: int

Output: ResultStatus

* + RegisterBusiness: It is an HttpPost webservice. A user can create a new business.

Input: Business

Output: BUSINESS\_ID: int (registered business’ id)

* + UnblockUser: It is an HttpGet webservice. A business owner can unblock a blocked user.

Input: BUSINESS\_ID: int (the business’ id), USER\_ID: int (the business owner’s id)

Output: ResultStatus

* + UpdateBusinessProfileInfo: It is an HttpPost webservice. A business owner can edit his business profile information.

Input: Business

Output: ResultStatus

## Comment

* + DeleteComment: It is an HttpGet webservice. A business owner can delete a user’s comment on his own business’ posts. The business owner is responsible to control comments on his posts.

Input: BUISNESS\_ID: int (the business’ id), COMMENT\_ID: int (the comment’s id)

Output: ResultStatus

* + GetAllCommentNotifications: It is an HttpGet webservice. It returns all comments of all posts which belongs to a business. If a user have more than one business, this webservice returns all his businesses’ notifications.

Input: USER\_ID: int (the user’s id), BEFORE\_THIS\_ID: int (is used to load more data), LIMITAION: int (is used to load more data)

Output: ArrayList<CommentNotification>

* + GetLastCommentNotification: It is an HttpGet webservice. It returns last comment of all posts which belongs to one of the user’s businesses.

Input: USER\_ID: int (the user’s id)

Output: CommentNotification

* + GetPostAllComments: It is an HttpGet webservice. It returns all comments of a post.

Input: POST\_ID: int (the post’s id), BEFORE\_THIS\_ID: int (is used to load more data), LIMITATION: int (is used to load more data)

Output: ArrayList<Comment>

* + SendComment: It is an HttpGet webservice. A user can comment on a post.

Input: USER\_ID: int (the user’s id), POST\_ID: int (the post’s id), COMMENT: string (the comment’s text)

Output: COMMENT\_ID: int (the sent comment’s id)

* + UpdateComment: It is an HttpGet webservice. A user can edit his own comment

Input: USER\_ID: int (the user’s id), COMMENT\_ID: int (the comment’s id), COMMENT\_TEXT: string (the new comment’s text)

Output: ResultStatus

## Friend

* + AnswerRequestFriendship: It is an HttpGet webservice. A user can answer another user’s friend request with Yes or No.

Input: REQUEST\_USER\_ID: int (the id of the user whom the request is sent for him), APPLICATOR\_USER\_ID: int (the id of the user whom sent the request),

ANSWER: boolean (true means Yes and false means No)

* + GetUserFriendRequests: It is an HttpGet webservice. It returns all the user’s friend request.

Input: USER\_ID: int (the user’s id)

Output: ArrayList<BaseAdapterItem>

* + GetUserFriends: It is an HttpGet webservice. It returns a list of the user’s friends.

Input: USER\_ID: int (the user’s id)

Output: ArrayList<BaseAdapterItem>

* + RequestCancelFriendship: It is an HttpGet webservice. A user can cancel the friendship relation with another user.

Input: APPLICATOR\_USER\_ID: int (the user’s id who wants to cancel the friendship), REQUESTED\_USER\_ID: int (the user’s id who’s the other user wants to cancel his friendship with him)

Output: ResultStatus

* + RequestFriendship: It is an HttpGet webservice. A user can send a friendship request for another user. A user can see his friend shared post in his Time Line or he can see his friend’s following businesses and etc.

Input: APPLICATOR\_USER\_ID: int (the user’s id who wants to send the friendship request), REQUESTED\_USER\_ID: int (the user’s id who’s the other user wants to be friend with)

Output: ResultStatus

## Post

* + AddPost: It is an HttpPost webservice. A business owner can add a new post.

Input: Post

Output: Post (includes the new post information plus post’ id)

* + CancelShare: It is an HttpGet webservice. A user can cancel the sharing of a post.

Input: USER\_ID: int (the user’s id), POST\_ID: int (the post’s id)

Output: ResultStatus

* + DeletePost: It is an HttpGet webservice. A business owner can delete one of his own posts.

Input: BUSINESS\_ID: int (the business’ id), POST\_ID: int (the post’s id)

Output: ResultStatus

* + GetBusinessPosts: It is an HttpGet webservice. It returns all posts of a business.

Input: USER\_ID: int (the id of the business owner), BUSINESS\_ID: int (the id of the business), POST\_ID: int (the post’s id)

Output: ArrayList<Post>

* + GetPost: It is an HttpGet post. It returns a post’s information e.g. post’s title. If the user who sees the post is the owner of the post, he can delete or edit the post or watch the comments and then delete some comments. But if not, he can like or share the post or just watch the comments.

Input: USER\_ID: int (the id of the business owner), POST\_ID: int (the post’s id)

Output: Post

* + GetSharedPosts: It is an HttpGet webservice. It returns posts which a user shared.

Input: BEFORE\_THIS\_ID: int (is used to load more data), LIMITATION: int (is used to load more data)

Output: ArrayList<Post>

* + GetTimeLinePosts: It is an HttpGet webservice. It returns a user’s Time Line posts.

Input: USER\_ID: int (the user’s id), BEFORE\_THIS\_ID: int (is used to load more data), LIMITAION: int (is used to load more data)

Output: ArrayList<Post>

* + Like: It is an HttpGet webservice. A user can like a post.

Input: USER\_ID: int (the id of the business owner), POST\_ID: int (the post’s id)

Output: ResultStatus

* + Report: It is an HttpGet webservice. A user can report a post as inappropriate post.

Input: USER\_ID: int (the id of the business owner), POST\_ID: int (the post’s id)

Output: ResultStatus

* + Share: It is an HttpGet webservice. A user can like a post.

Input: USER\_ID: int (the id of the business owner), POST\_ID: int (the post’s id)

Output: ResultStatus

* + Unlike: It is an HttpGet webservice. A user can unlike a liked post.

Input: USER\_ID: int (the id of the business owner), POST\_ID: int (the post’s id)

Output: ResultStatus

* + UpdatePost: It is an HttpPost webservice. A business owner can edit a post’s information.

Input: Post

Output: Post

## Review

* + DeleteReview: It is an HttpGet webservice. A user how wrote a review for a business, can delete his own review.

Input: REVIEW\_ID: int (the review’s id), USER\_ID: int (the user’s id)

Output: ResultStatus

* + GetBusinessReviews: It is an HttpGet webservice. It returns a list of a business’ reviews.

Input: BUSINESS\_ID: int (the business’ id), BEFORE\_THIS\_ID: int (is used to load more data), LIMITAION: int (is used to load more data)

Output: ArrayList<Review>

* + GetUserReviews: It is an HttpGet webservice. It returns a list of reviews which a user wrote.

Input: USER\_ID: int (the user’s id), BEFORE\_THIS\_ID: int (is used to load more data), LIMITAION: int (is used to load more data)

Output: ArrayList<Review>

* + ReviewBusiness: It is an HttpGet webservice. A user can review a business. He can write a text and rate the business with a value between 1 and 5.

Input: USER\_ID: int (the user’s id), BUSINESS\_ID: int (the business’ id), REVIEW: string (the review’s text), RATE: int (the review’s rate)

Output: ResultStatus

* + UpdateReview: It is an HttpGet webservice. A user can edit his own reviews.

Input: USER\_ID: int (the user’s id), BUSINESS\_ID: int (the business’ id), REVIEW: string (the review’s text), RATE: int (the review’s rate)

Output: ResultStatus

## Search

* + SearchBusinessesLocation: It is an HttpGet webservice. A user can search a business by specify its name, subcategory and a location. The result will be sorted base on the distance of the specified location. The nearest businesses will be on the top.

Input: SEARCH\_TEXT: string (the business’ name), SUB\_CATEGORY\_ID: int (the business’ subcategory id), LOCATION\_LATITUDE: string (the location’s latitude), LOCATION\_LONGITUDE: string (the location’s longitude), BEFORE\_THIS\_ID: int (is used to load more data), LIMITAION: int (is used to load more data)

Output: ArrayList< SearchItemBusiness>

* + SearchBusinessesNearBy: It is an HttpPost webservice. A user can search a business by specify its name, and a place like a city. This webservice has NOT used.

Input: USER\_ID: int (the user’s id), SEARCH\_TEXT: string (the business’ name), NEAR\_BY: string (the place name)

Output: ArrayList< BaseAdapterItem>

* + SearchPost: It is an HttpGet webservice. A user can search a post by its title or hashtags. The hashtags will help the system for better searching.

Input: SEARCH\_TEXT: string (the post’s title or hashtag), BEFORE\_THIS\_ID: int (is used to load more data), LIMITAION: int (is used to load more data)

Output: ArrayList< SearchItemPost>

* + SearchUser: It is an HttpGet webservice. A user can search other users by their identifier. The identifier is an English phrase e.g. “ali124Hooo”

Input: SEARCH\_TEXT: string (the other user’s identifier), BEFORE\_THIS\_ID: int (is used to load more data), LIMITAION: int (is used to load more data)

Output: ArrayList< BaseAdapterItem>

## User

* + FollowBusiness: It is an HttpGet webservice. A user can follow a business. He will see the business’ future posts in his Time Line.

Input: USER\_ID: int (the user’s id), BUSINESS\_ID: int (the business’ id),

Output: ResultStatus

* + ForgetPassword: It is an HttpGet webservice. If a user forget his password, he can recover his password by his email.

Input: EMAIL: string (the user’s email address)

Output: ResultStatus

* + GetFollowingBusinesses: It is an HttpGet webservice. It returns a list of businesses which a user is following.

Input: USER\_ID: int (the user’s id)

Output: ArrayList< BaseAdapterItem>

* + GetUserHomeInfo: It is an HttpGet webservice. It returns a user’s home information. The information will be displayed differently if user watches his own home page or another user.

Input: VISITED\_USER\_ID: int (The id of a user who the user is watching his home), VISITOR\_USER\_ID: int (The id of the user who is watching a user home)

Output: User

* + GetUserProfileInfo: It is an HttpGet webservice. It returns a user’s profile information e.g. “about me”

Input: USER\_ID: int (the user’s id)

Output: User

* + Login: It is an HttpGet webservice.

Input: EMAIL: string (the user’s email address), PASSWORD: string (the user’s account password)

Output: ResultStatus

* + RegisterUser: It is an HttpPost webservice. The user’s identifier and his email address should be unique. After the successful registration, a confirmation request will send to the user’s email address.

Input: User

Output: ResultStatus

* + RequestConfirmation: It is an HttpGet webservice. If a user didn’t receive the confirmation email, he can request to receive confirmation email again. This webservice has not used.

Input: USER\_ID: int (the user’s id)

Output: ResultStatus

* + UnFollowBusiness: It is an HttpGet webservice.

Input: USER\_ID: int (the user’s id), BUSINESS\_ID: int (the business’ id)

Output: ResultStatus

* + UpdatePassword: It is an HttpPost webservice. This webservice has not used. The password will be updated with another webservice.

Input: USER\_ID: int (the user’s id), NEW\_PASSWORD: string (the new password)

Output: ResultStatus

* + UpdateSetting: It is an HttpGet webservice. There is 3 permission for a user which the user can change them:
    - Followed businesses: the user can let his friends be able to see his following businesses.
    - Friends: the user can let his friends be able to see his friends.
    - Reviews: the user can let his friends be able to see his reviews on businesses.

Input: Permission

Output: ResultStatus

* + UpdateUserProfile: It is an HttpPost webservice. A user can edit his own profile info e.g. “name”.

Input: User

Output: ResultStatus

* DownloadCoverImage: this class uses an HttpGet webservice to download a user’s cover image or a business’ cover image. It takes an ImageView and it will displays the download image in the ImageView. After downloading an image, the image will be cropped to make it a triangle shape image. At the end, the image will be saved into local storage. Next time, the class uses the saved image to display cover. Actually it caches images. If the class failed to download an image, it will display a default image depending on it is a user’s cover or business’ cover.

Input: IMAGE\_ID: int (the image’s id), IMAGE\_VIEW: ImageView, IMAGE\_TYPE: Image\_M.ImageType

* DownloadImages: this class woks like DownloadCoverImage without cropping the image. This class can round the image if it is required. This class can download the image in 3 different sizes.

Input: IMAGE\_ID: int (the image’s id), IMAGE\_SIZE: int (1=large, 2=medium, 3= small), IMAGE\_TYPE: Image\_M.ImageType, IMAGE\_VIEW: ImageView, IS\_ROUNDED: Boolean

* WebserviceGET: all upper HttpGet classes use this class to execute HttpGet requests.

Functions:

* + public ServerAnswer execute(Context context);

It executes the HttpGet URL and returns the HttpResponse result as a ServerAnswer object.

* + public ServerAnswer executeList(Context context)

It works like execute () except we use it to get a list of objects e.g. GetUserFriends

* WebservicePOST: all upper HttpPost classes use this class to execute HttpPos requests.

Functions:

* + public void addParam(String paramName, String paramValue)

It adds a parameter to the JSONObject.

* + public ServerAnswer execute(Context context)
  + public ServerAnswer executeList(Context context)
  + private HttpResponse run(HttpPost httpPost)

Two upper functions (execute () and executeList ()) use this function to execute the HttpPost request. It sets entity and adds headers to the HttpResponse before executing it.

# View

## Activity

* + ActivityBusiness: It displays the user’s business home information e.g. name.

onCreate(Bundle savedInstanceState)

If the business owner deletes a post in ActivityPost, a broadcast will be sent to this activity. A BroadcastReceiver (deletePost) will receive the broadcast and notifies the gridView to remove the deleted post.

deletePost = new BroadcastReceiver() {

@Override

public void onReceive(Context context, Intent intent) {

//if the user delete a post from ActivityPost

gridViewBusiness.notifyDeletePost(intent.getIntExtra(Params.POST\_ID\_INT, 0));

}

};

LocalBroadcastManager.getInstance(this).registerReceiver(deletePost, new IntentFilter(Params.DELETE\_POST\_FROM\_ACTIVITY));

getResult (Object result)

This method receives the result of two webservice: GetBusinessHomeInfo and

GetBusinessPosts. If the result is instant of Business, it is the GetBusinessHomeInfo’ result that contains the user’s business’ information e.g. business profile picture:

business = (Business) result;

The result is used to initialize the gridview’s header and the gridview’s body (array list of posts) will be empty:

boolean beThreeColumn = gridViewBusiness == null? true: gridViewBusiness.isThreeColumn;

gridViewBusiness = new GridViewBusiness(this, business, gridView);

gridViewBusiness.InitialGridViewBusiness(new ArrayList<Post>(), beThreeColumn);

If the result is instant of ArrayList, it contains a list of posts that belongs to the business. The posts are used to initialize the gridview’s body.

* + ActivityBusinessBlockedUsers: It displays a business’ blocked users. The business owner can remove a user (unblock) from the blocked users list.
  + ActivityBusinessContactInfo: It displays a business’ contact information e.g. email address.
  + ActivityBusinessFollowers: It displays a business’ follower users.
  + ActivityBusinessOther: The user can sees other user’s businesses’ home information in this activity.
  + ActivityBusinessRegisterEdit: A user can register a new business or edit an existing business in this activity.

onCreate(Bundle savedInstanceState)

If the user is editing his business, the GetBusinessProfileInfo webservice will be executed to get the business’ profile information:

try {

businessId = getIntent().getExtras().getInt(Params.BUSINESS\_ID\_STRING);

businessIdentifier = getIntent().getExtras().getString(Params.BUSINESS\_IDENTIFIER);

} catch (Exception e) {

}

Bundle bundle = new Bundle();

if (businessId != 0) {

((TextViewFontActionBarTitle)v.findViewById(R.id.textView\_title)).setText(getString(R.string.profile\_edit\_business));

new GetBusinessProfileInfo(ActivityBusinessRegisterEdit.this, businessId, ActivityBusinessRegisterEdit.this).execute();

bundle.putBoolean(Params.IS\_EDITTING, true);

progressDialog.show();

}

getResult (Object result)

The RegisterBusiness webservice returns an integer value that is the registered business’ id. Then the user will navigate to FragmentBusinesses and so the businesses list should be updated:

if (result instanceof Integer) {

//RegisterBusiness' result

//result is the registered business' id

((MyApplication) getApplication()).business.id = (Integer) result;

Intent i = getIntent();

setResult(RESULT\_OK, i);

finish();

}

If the user wants to edit his own business, the GetBusinessProfileInfo returns the business’s information:

else if (result instanceof Business) {

//GetBusinessProfileInfo's result

((MyApplication) getApplication()).business = (Business) result;

((MyApplication) getApplication()).business.businessIdentifier = businessIdentifier;

ft.commit();

}

If the user is editing his own business the UpdateBusinessProfileInfo returns a ResultStatus as result. Then the user will naviagte to the ActivityBusiness, so he should be able to see the changes:

else if (result instanceof ResultStatus) {

//UpdateBusinessProfileInfo

Intent i = getIntent();

i.putExtra(Params.PROFILE\_PICTURE, ((MyApplication) getApplication()).business.profilePicture);

i.putExtra(Params.TYPE, Business.ChangeType.EDIT.name());

setResult(RESULT\_OK, i);

finish();

}

notifyDeleteBusiness(int businessId)

If the user deletes his own business, the user will navigate to ActivityBusiness

and since the business id deleted, the user will navigate to the FragmentUserBusinesses. The businesses list should be updated:

Intent i = getIntent();

i.putExtra(Params.BUSINESS\_ID\_STRING, businessId);

i.putExtra(Params.TYPE, Business.ChangeType.DELETE.name());

setResult(RESULT\_OK, i);

finish();

* + ActivityBusinessReviews: It displays a business reviews.
  + ActivityBusinessWorkTime: A user can set a business’ working days and working times in this activity. If the user is editing the business, he can edit the data.
  + ActivityCamera: A user can take a picture from the camera in this activity.

onCreate(Bundle savedInstanceState)

CameraPreview display the camera image as full screen but we need a square image, so a relative layout is placed over the CameraPreview at the bottom to make the CameraPreview looks like a square view.

DisplayMetrics displaymetrics = new DisplayMetrics();

getWindowManager().getDefaultDisplay().getMetrics(displaymetrics);

int width = displaymetrics.widthPixels;

int height = displaymetrics.heightPixels;

int actionBarHeight = 0;

TypedValue tv = new TypedValue();

if (getTheme().resolveAttribute(android.R.attr.actionBarSize, tv, true)) {

actionBarHeight = TypedValue.complexToDimensionPixelSize(tv.data, getResources().getDisplayMetrics());

}

RelativeLayout.LayoutParams lp = new RelativeLayout.LayoutParams(width, height - width - actionBarHeight);

lp.addRule(RelativeLayout.ALIGN\_PARENT\_BOTTOM, RelativeLayout.TRUE);

rl\_camera\_cover.setLayoutParams(lp);

After taking a picture with CustomCamera, the customCamera object returns the picture file path via ICropResult in getResult() method. Then the activity returns the file path to the previous activity:

Intent i = getIntent();

i.putExtra(ActivityCamera.FILE\_PATH, filePath);

setResult(RESULT\_OK, i);

finish();

* + ActivityComments: It displays all comments of a post.
  + ActivityCommentsNotifications. It displays all comments of all posts that belongs to the user’s businesses.
  + ActivityGallery: A user can select an image from his device gallery via this activity.

onActivityResult(int requestCode, int resultCode, Intent data)

This activity opens gallery so the user is able to select an image. After selecting

the image, activity crops the picture and then sends the picture file path to the

previous activity:

if (resultCode == RESULT\_OK) {

if (requestCode == GALLERY\_CAPTURE) {

performCrop(data.getData());

} else if (requestCode == PIC\_CROP) {

String path = data.getStringExtra(CropImage.IMAGE\_PATH);

// if nothing received

if (path == null) {

Intent i = getIntent();

i.putExtra(ActivityGallery.FILE\_PATH, path);

setResult(RESULT\_CANCELED, i);

finish();

return;

}

Intent i = getIntent();

i.putExtra(ActivityGallery.FILE\_PATH, path);

setResult(RESULT\_OK, i);

finish();

}

}

* + ActivityLogin: A user can login to the app via this activity.

getResult (Object result)

Both webservices in this activity, Login and ForgetPassword, returns a ResultStatus object as result. So we differentiate two webservices’ result via getCurrentWebservice() method in MyApplication class:

if (!(result instanceof ResultStatus))

return;

if (myApplication.getCurrentWebservice() == WebservicesHandler.Webservices.LOGIN) {

Intent intent = new Intent(ActivityLogin.this, ActivityMain.class);

startActivity(intent);

myApplication.setCurrentWebservice(WebservicesHandler.Webservices.NONE);

} else if (myApplication.getCurrentWebservice() ==WebservicesHandler.Webservices.FORGET\_PASSWORD) {

new DialogMessage(ActivityLogin.this, getResources().getString(R.string.forgot\_password\_sent)).show();

myApplication.setCurrentWebservice(WebservicesHandler.Webservices.NONE);

}

* + ActivityMain: It the main activity of the app. It handles 4 fragments: FragmentHome, FragmentSearch, FragmentBusinesses and FragmentUser.

setFragment(FragmentTag fragmentTag)

It shows a fragment and hides other framgnet in this activity. It uses setSelection() method.

switch (fragmentTag) {

case HOME:

setSelection(rlHome.getId());

break;

case BUSINESSES:

setSelection(rlBusinesses.getId());

break;

case SEARCH:

setSelection(rlSearch.getId());

break;

case USER:

setSelection(rlUser.getId());

break;

}

addFragment(FragmentTag fragmentTag)

By touching the back key on the device, the user will see the previous fragment

in this activity. In fact, the sequence of watching the framgment is recording. To

do so, by touching any tab, the related fragment will add to the fragmentTagList:

int fragmentPosition = checkFragment(fragmentTag);

//if the fragment is not added before

if (fragmentPosition == -1)

fragmentTagList.add(fragmentTag);

else {

//if the fragment is added before, remove it and add the fragment in the end of //the list

fragmentTagList.remove(fragmentPosition);

fragmentTagList.add(fragmentTag);

}

checkBack()

If there is no fragment in fragmentTagList, by touching the back button, the user will navigate to the home screen.If there is just a fragment in the tracking list, the user will navigate to the FragmentHome. Otherwise, the previous fragment will be shown:

switch (fragmentTagList.size()) {

case 0:

exit();

break;

case 1:

if (fragmentTagList.get(0) == FragmentTag.HOME)

exit();

fragmentTagList.remove(0);

setFragment(FragmentTag.HOME);

break;

default:

fragmentTagList.remove(fragmentTagList.size() - 1);

setFragment(fragmentTagList.get(fragmentTagList.size()-1));

break;

}

setParams(int width)

Each tab in this activity is placed inside a relative layout. This method is used to

set width of all tabs dynamically by using LayoutParams. If there is 3 tabs, the rlBusinesses layout will be gone:

LinearLayout.LayoutParams params = new LinearLayout.LayoutParams(width, LinearLayout.LayoutParams.MATCH\_PARENT);

rlBusinesses.setLayoutParams(params);

rlUser.setLayoutParams(params);

rlHome.setLayoutParams(params);

rlSearch.setLayoutParams(params);

It is supposed to not executing all webservices (GetTimeLinePosts, GetBusinessCategories – in FragmentSearch and GetUserHomeInfo) in this activity together. So by entering this activity, GetTimeLinePosts will be executed first, then GetBusinessCategories and finally, GetUserHomeInfo. To implement this sequential of executing, recursive method are used in fragments to check the previous webservice is done or not. So it is possible, the user touches the user home information tab but the related webservice is not done yet, so the user can’t see any data. To prevent displaying a white screen to the user, some recursive methods in this activity are used to check the related webservice is done or not. If not, check it later but if the webservice is done, so the fragment and its data will be displayed. These methods are:

* + - * recursivelyCallHandlerSearchFragment()
      * recursivelyCallHandlerUserFragment()
      * recursivelyCallHandlerUserBusinessesFragment()

recursivelyCallHandlerSearchFragment() {

handlerSearchFragment.postDelayed(new Runnable() {

@Override

public void run() {

//We don't want to run all webservices together

//first HomeFragment, second SearchFragment and last UserFragment

if (((MyApplication) getApplication()).isSearchCreated) {

if (footerSearch) {

ft = fm.beginTransaction();

ft.show(fm.findFragmentById(R.id.frag\_search));

ft.hide(fm.findFragmentById(R.id.frag\_home));

ft.hide(fm.findFragmentById(R.id.frag\_user));

ft.hide(fm.findFragmentById(R.id.frag\_user\_businesses));

ft.commit();

}

} else

recursivelyCallHandlerSearchFragment();

}

}, 500);

* + ActivityMapChoose: A user can choose a location on the google map via this activity.
  + ActivityMapDisplay: It displays a location on the google map. It is used to display a business location via ActivityBusinessContactInfo
  + ActivityPost: It displays a post’s information e.g. price. By clicking on the post picture while the posts are displaying as the Gridview, the user will navigate to this activity.

notifyUpdateTimeLineShare(int postId)

When a user shares a post, this activity send a broadcast to the FragmentHome

to change the post’s share status to shared:

Intent intent = new Intent(Params.UPATE\_TIME\_LINE);

intent.putExtra(Params.UPDATE\_TIME\_LINE\_TYPE, Params.UPATE\_TIME\_LINE\_TYPE\_SHARE);

intent.putExtra(Params.POST\_ID\_INT, postId);

LocalBroadcastManager.getInstance(ActivityPost.this).sendBroadcast(intent);

* + ActivityPostAddEdit: A business owner can add a new post or edit an existing post.
  + ActivityProfileUser: It displays the user’s profile information e.g. name. The user can edit his profile information.
  + ActivitySearchBusinessResult: The SearchBusinessesLocation webservice runs in this activity and the result will be displayed here.
  + ActivitySearchPostResult: The SearchPost webservice runs in this activity and the result will be displayed here.
  + ActivitySearchUser: The SearchUser webservice runs in this activity and the result will be displayed here.
  + ActivityUserFollowingBusinesses: It displays a list of the user’s following businesses. If the user is watching his own following businesses, he is able to remove a business from the list.
  + ActivityUserFriendRequests: It displays friend requests of the user. The user can answer Yes or No to the request.
  + ActivityUserFriends: It displays a list of the user’s friends. If the user is watching his own friends, he is able to remove a friend from the list.
  + ActivityUserLoginSignUp: It is not implemented.
  + ActivityUserOther: It displays another user’s home information rather than the current user.
  + ActivityUserRegister: A user who wants to user the application should register first.
  + ActivityUserReviews: It displays a list of reviews which the user wrote. The user can edit or delete a review.
  + ActivityUserSetting: There are three setting options:
    - Permission for shared post: with this permission, the friends of a user can see his shared posts.
    - Permission for following businesses: with this permission, the friends of a user can see his following businesses.
    - Permission for reviews: with this permission, the friends of a user can see his reviews.

## Adapter

* + AdapterBlockedUsers: It is used to display the list of a business’ blocked users.
  + AdapterBusinessFollowers: It is used to display the list of users who follow a business.
  + AdapterBusinessReview: It is used to display a business’ reviews.
  + AdapterBusinessSearchResult: It is used to display the result of search about businesses.
  + AdapterCategories: It is used to display the list of the business’ categories.
  + AdapterCommentNotification: It is used to display the list of comment notifications – all comments of a user’s all posts.
  + AdapterPostBusiness: It is used to display the list of a business’ posts.
  + AdapterPostComments: It is used to display the list of all comments of a post.
  + AdapterPostGrid: It is used to display the list of posts’ pictures in a Gridview.
  + AdapterPostShared: It is used to display the list of a user’s shared posts.
  + AdapterPostTimeLine: It is used to display the list of user’s time line posts.
  + AdapterUserBusinesses: It is used to display the list of user’s businesses.
  + AdapterUserFollowingBusinesses: It is used to display the list of user’s following businesses.
  + AdapterUserFriends: It is used to display the list of a user’s friends.
  + AdapterUserFriendshipRequest: It is used to display the list of a user’s friend requests.
  + AdapterUserReview: It is used to display the list of a user’s review.
  + AdapterUserSearchResult: It is used to display the result of the search about users.

## Dialog

* + DialogAddReview: It extends the MyDialogOkCancel. It takes the user’s review which includes text and rate.
  + DialogBlockUserConfirmation: It extends the MyDialogOkCancel. It displays a message to confirm blocking a user.
  + DialogCancelFriendshipConfirmation: It extends the MyDialogOkCancel. It displays a message to confirm cutting the friendship with a user.
  + DialogChangePassword: It extends the MyDialogOkCancel. It takes the user’s current password, new password and confirm password.
  + DialogClearSearchHistoryConfirmation: It extends the MyDialogOkCancel. It displays a message to confirm clearing the search history.
  + DialogDeleteBusinessConfirmation: It extends the MyDialogOkCancel. It displays a message to confirm removing a business.
  + DialogDeleteCommentConfirmation: It extends the MyDialogOkCancel. It displays a message to confirm deleting a comment.
  + DialogDeletePostConfirmation: It extends the MyDialogOkCancel. It displays a message to confirm deleting a post.
  + DialogDeleteReviewConfirmation: It extends the MyDialogOkCancel. It displays a message to confirm deleting a review.
  + DialogEditComment: It extends the MyDialogOkCancel. It displays a user’s written comment. The user can edit the comment.
  + DialogEditReview It extends the MyDialogOkCancel. It displays a user’s review. The user can edit the review’s text and change the review’s rate.
  + DialogExit: It extends The MyDialogOkCancel. It displays a message to confirm deleting a review.
  + DialogForgetPassword: It extends the MyDialogOkCancel. It the user completely forgot his own account’s password, with using his email address, he can retrieves his forgotten password.
  + DialogMessage: It extends the MyDialogOkCancel. It is used to display a message.
  + DialogUnfollowBusinessConfirmation: It extends the MyDialogOkCancel. It displays a message to confirm unfollowing a business.
  + MyDialog: It extends MyPopup. All dialogs which extend this class have a title, body and a footer. This class adds a body and a footer to the MyPopup.
  + MyDialogOk: It extends MyDialog. All dialogs which extend this class have a title, body, and a button (OK) in the bottom of the dialog. With clicking on the button, the dialog will dismiss.
  + The MyDialogOkCancel: It extends MyDialog. All dialogs which extend this class have a title, body, and two buttons (OK and Cancel) in the bottom of the dialog. With clicking on the OK button, a defined action will execute and with clicking on the Cancel button, the dialog will dismiss.
  + MyPopup: It is the parent class for all classes in the Dialog directory. All dialogs which extend this class have a body.
  + PopupBlockUser: It has a button which is used to block a user.
  + PopupCancelFriendship: It has a button which is used to cancel a friendship relation with a user.
  + PopupCancelSharePost: It has a button which is used to cancel sharing a post by the user.
  + PopupCategories: It displays a list of business’ categories. A user can select one category. It is used in Search Business and Register Business.
  + PopupDeleteCommentBlockUser: It has two buttons, delete a comment and block a user. A business owner can delete user’s comment in his own posts. The business owner can block a user either. The blocked user won’t be able to comment on the business owner’s posts again.
  + PopupEditDeleteBusiness: It has two buttons, edit a business profile information and delete a business. A business owner can delete his own business or edit its profile information.
  + PopupEditDeleteComment: It has two buttons, edit a comment and delete the comment. A user can delete his own comment or edit it.
  + PopupEditDeleteReview: It has two buttons, edit a review and delete the review. A user can delete his own review or edit it.
  + PopupReportCancelSharePost: It has a button which is used to cancel sharing a post, or report the post.
  + PopupReportPostActivity: It has a button which is used to report a post inside ActivityPost.
  + PopupReportPostAdapter: It has a button which is used to report a post inside adapters.
  + PopupSelectCameraGallery: It has two buttons, select camera and select gallery. To choose a picture, user can uses camera to take a picture or opens his device gallery to select a picture.
  + PopupSubCategories: It displays a list of business’ sub categories. A user can select one sub category. It is used in Search Business and Register Business.
  + PopupUnfollowBusiness: It has a button which is used to unfollow a following business.

## Fragment

* + FragmentBusinessRegisterBaseInfo: It is first step of registering a business. In this fragment, a user enters business’s base information e.g. identifier, name, etc.
  + FragmentBusinessRegisterContactInfo: It is second step of registering a business. In this fragment, a user enters business’s contact information e.g. work time, cellphone number, email address or website URL. Except work time field, all other fields are optional.
  + FragmentBusinessRegisterLocationInfo: It is second step of registering a business. In this fragment, a user enters business’s location information e.g. province, city, address and business coordinates.
  + FragmentHome: The user’s Time Line will be displayed in this fragment.
  + FragmentSearch: The user can search about businesses, users and post in this fragment. To search businesses, it is necessary to determine business location.
  + FragmentUser: The user’s home information e.g. user’s shared post will be displayed in this fragment.
  + FragmentUserBusinesses: A list of user’s businesses will be displayed in this frgment.
  + FragmentUserLogin: This fragment is not implemented.
  + FragmentUserSignUpeEmailID: This fragment is not implemented.
  + FragmentUserSignUpeNamePassword: This fragment is not implemented.
  + FragmentUserSignUpePicture: This fragment is not implemented.

## Interface\_m

* + IAddReview: It is used to add a new review to a business’ reviews after successfully running the ReviewBusiness webservice.
  + IChangeBusiness: It is used to delete a business from user’s businesses list in FragmentUserBusinesses after successfully running the DeleteBusiness webservice.
  + IChangePassword: It is used to notify ActivityUserProfile if user changed his password.
  + IChangeTabs: It is used to notify ActivityMain to display 4 tabs when the user register a business or 3 tabs when the user delete all his businesses.
  + ICommentChange: It is used to notify AdapterPostComment to display an updated comment or remove a deleted comment.
  + ICropResult: It is used to notify ActivityCamera to return crop result.
  + IDeletePost: It is used to notify ActivityPost to return to the previous activity and it is used to notify AdapterPostBusiness to remove the deleted post.
  + IGetCallForTakePicture: It is used to notify ActivityBusinessRegisterEdit to display PopupSelectCameraGallery dialog.
  + IGetNewTimeLinePost: It is used to notify FragmentHome to update its adapter.
  + IPullToRefresh: It is used to notify all activities and fragments which use pull\_to\_refresh library to refresh their data.
  + IReportPost: It is used to notify ActivityPost, AdapterPostShared and AdapterPostTimeline to hide report option. If the user reports a post, he shouldn’t be able to report a post again.
  + ISelectCategory: It is used to return selected category from PopupCategories or return selected sub category from PopupSubCategories.
  + IUnfollowBusiness: It is used to notify AdapterUserFollowingBusinesses to remove an unfollowed business or notify GridViewBusinessOther to change follow status.
  + UpdateTimeLine: It is used to notify FragmentHome to update a post’s share status.
  + IUpdateUserPraofile: It is used to notify FragmentUser to display the user’s updated information after updating in
  + IWebserviceResponse: It is used to return a webservice’s result.

## Widget\_customized

* + Buttons : (Mr. Khalili should writes this section)
  + Charsoo\_activity: (Mr. Khalili should writes this section
  + checkbox: (Mr. Khalili should writes this section)
  + EditTextFont: It is extends EditText and displays text with a font. It overrides onTouchEvent and onTextChanged to set error as null.
  + EditTextFontPasteDisabled: It is like the EditTextFont but the ability of pasting is disabled.
  + GridViewBusiness: It displays a business home information that belongs to the user in a Gridview. It contains two different adapters to display business posts as a grid with 3 columns or as a grid with one column (which is like a list). It initialize the header of the Gridview.
  + GridViewBusinessOther: It is like the GridViewBusiness but it displays other users’ businesses home information.
  + GridViewUser: It is like the GridViewBusiness but it displays the user’s home information
  + GridViewUserOther: It is like the GridViewBusiness but it displays other users’ home information
  + MaterialProgressBar: (Mr. Khalili should writes this section)
  + TextViewFont: It extends TextView and displays text with a font.
  + TextViewFontActionBarTitle: It is like the TextViewFont which is used to display a title in action bar.

# Controller

## Helper

* + Alarm\_M: It is used to set an alarm to periodically check user’s last notification.
  + AlarmReciever: It is a broadcast receiver that handle an intent that Alarm\_M sends.
  + BaseAdapterItem: It is used in some adapters as an object to keep data e.g. id, image id, title.
  + CameraPreview: It extends SurfaceView and is used to display camera picture.
  + CurrentLocationProvider: It is used to get current location via the Google Play Service to display location on the Google Map.
  + CustomCamera: It is used to take a picture via camera. It saves the picture and returns the saved file path.
  + FriendshipRelation: The friendship status between two users is defined here.
  + Hashtag: It is used to process hashtags. It gets a list of hashtags from a string or it sets a list of hashtags as a string.
  + Image\_M: It is used to process images. It converts an image to a string or vice versa.
  + ImageHelper: It makes corners of an image round.
  + Location\_M: It is an object which is used to hold a coordinate including latitude and longitude.
  + LocationManagerTracker: It is used to get current location without using the Google Play Service
  + LoginInfo: It handles the login process. After login, it stores the user’s id in the Share Preferences and after logout, it removes the user’s id from Share Preferences. Other objects can get the user’s id from the Share Preferences.
  + MyGestureDetector: It extends GestureDetector.SimpleOnGestureListener and it is used to detect a double tab action on a view e.g. liking a post with double tab.
  + MyNotification: It is used to display a notification. It displays a notification by using NotificationCompat.Builder
  + NetworkHandler: It is used to check the network connectivity.
  + Params: All constant variables except URLs are defined here.
  + Permission: All the user’s permissions are defined here. There is three types of permission for the user:
    - FollowedBusiness: If it is true, that means the user’s friends cans see his following businesses.
    - Friends: if it is true, that means the user’s friends cans see his other friends.
    - Review: if it is true, that means the user’s friends cans see his reviews.
  + PersianDate: It is used to display a date in Persian format. Most important method in this class is getCreationDate() that gets a date in DateTime format and compare the date with current date and returns two times difference as hours.
  + PullToRefreshGrid: It is used to implement the pull to refresh function for a grid view. It adds a footer to the grid view to implement load more function.
  + PullToRefreshList: It is like the PullToRefreshGrid but it implement the pull to refresh function for a list view.
  + ResultStatus: Some webservices just return the success status of executing the webservice. The ResultStatus is used to keep the success status and an error code if the success status is false.
  + SearchItemBusiness: It extends BaseAdapterItem. It contains the result of the search about a business. In addition to the BaseAdapterItem variables, this class has a double variable called “distance” which contains the distance between user’s current location and a result business.
  + SearchItemPost: The SearchPost webservice returns result as a list of SearchItemPost object.
  + Sex: It is an object that contains all sex types including male, female and none (is not determined).
  + SolarCalendar: The user’s birthday fields are solar calendar (Shamsi) based. The SolarCalendar is used to validate those fields e.g. the day number of the 12th month shouldn’t be greater than 29.
  + TestUnit: It is used to display static information in the app to test the functionality of every section.
  + TextProcessor: It is used to process a string to display hashtags with a different format (blue text)
  + URLs: All webservices’ URLs defined in this class.
  + Validation: It is used to validate all input fields e.g. email address, phone number and etc.
  + WebservicesHandler: It returns a ServerAnswer object with network connection error. It is used when a webservice is executed without internet connection.
  + WorkTime: It is used to set work time for a business including opening and closing times and days which the business is working.

## image\_loader

Mr. Khalili should writes this section.

## Object

* + Business: It is an object that contains a business information including business’ base information (e.g. identifier, name), contact information (e.g. phone, website, email) and location information (e.g. province, city and coordinate)
  + Category: Every business belongs to a category and a subcategory. A Category object have an id and a name.
  + Comment: A user can writes many comments on a post. There is no limitation for the number of comments which a user can writes for a post. Each comment object contains information about the post, the user and a text.
  + CommentNotification: If a user writes a comment on a post, a CommentNotification will be displayed to a user who owns the business which create the post. Each CommentNotification contains the user’s information (e.g. identifier, profile picture), the post’s information (e.g. post picture) and the comment’s information (e.g. comment’s text).
  + MyApplication: It is an Application class that keeps some global variables. It is accessible from all sections in the app.
  + Post: It is an object that contains a post’s information e.g. title, picture, etc. This class have some methods to extract a post information from a json object.
  + Review: A user can write a review for a business. The user only can write a review. He can edit his review later. A Review object contains the user’s information, the business information, a text and a rate. The user can rate a business with integer values between 1 and 5.
  + SubCategory: Every business belongs to a category and a subcategory. A SubCategory object have an id and a name.
  + User: It is an object that contains a user’s information includes profile information, permissions, friendship relation status, his businesses and etc.

Cached Images

Local Data Base and Share Preferences