

- Coin Flipping Android App Project -

1) .:PROJECT TASK:.

An android application project about coin flipping (to achieve the highest score) which will work as a multiplayer game playing framework in general and allow basic multiplayer gaming operations such as connecting two players, allowing communication between players, sending/receiving invitations and creating leaderboard functions.

The application can be used for the aforementioned game playing purposes and with a few modifications it could also be possible to further utilize the platform for creating other multiplayer games.

To be able to achieve this project, own written classes and modules or already well structured libraries ought to be utilized.

2) .:UNDERSTANDING THE PROBLEM:.

a. Stakeholders

- End users: To have a good time/ feeling to earn something / interact with other people
- Ads marketers: Putting an ad into a high quality app brings more meaningful ROI numbers
- Online community/tech writers/channels: Unique application means something to mention in their journals and getting more traffic for this group

Key Benefits of Online Multiplier Game Apps

- Creating short time entertainment& relaxation chances to their users.
- Giving chance to players from different parts of the world to communicate with each other, creating teams etc..
- They allow the users to earn something (real/virtual)

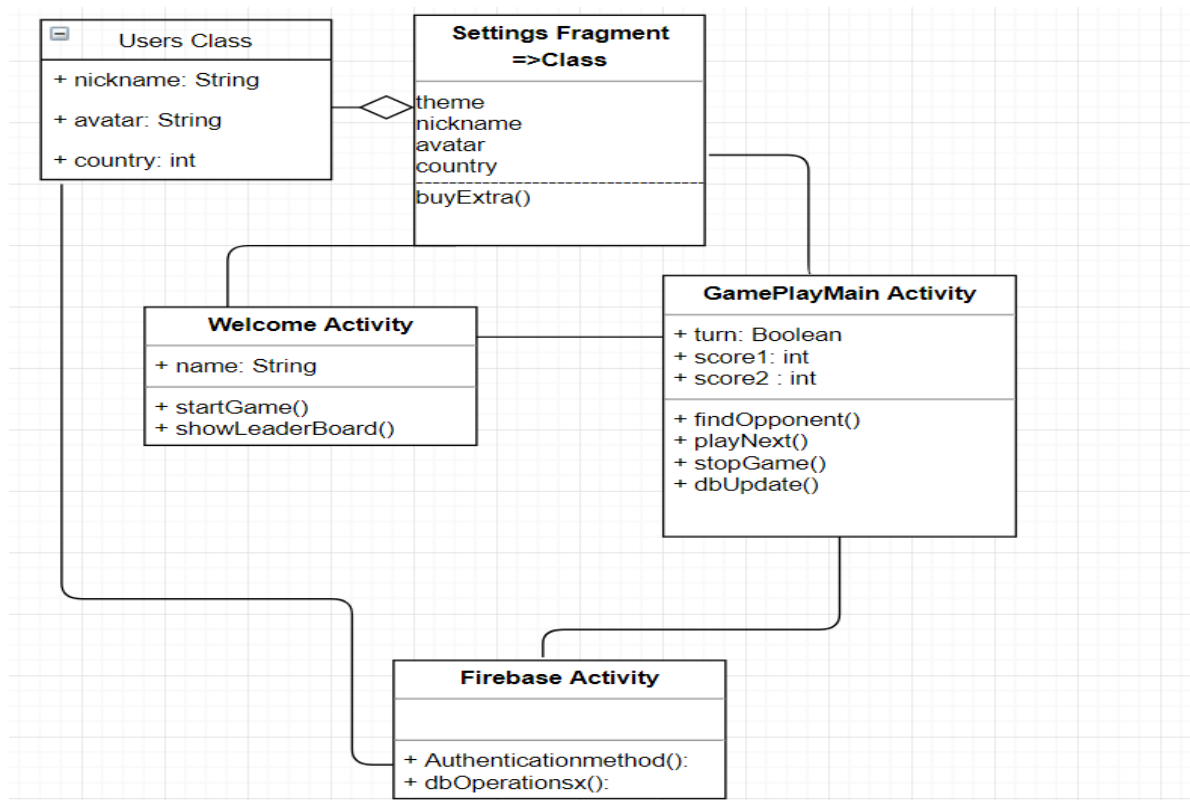
b. Unknowns/Features

- Establishing connection between the players from a pool of players
- Creating/using the database to save game scores/profiles
- MBaaS (Firebase, Parse) for easy database operations.
- Creating game profiles(nickname/avatar/country etc.) for the players
- Having a leaderboard for competition

c. Compartmanization/SubTasks

- Matching two players from online users
- Displaying the actual game on screen
- Updating the database

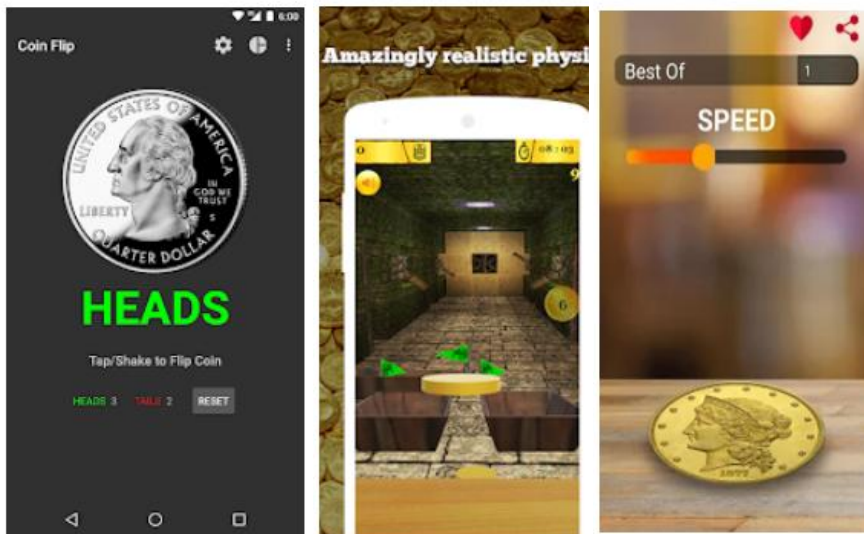
d. Graphics/UML Structure



3) .:PLANNING A SOLUTION:.

a-b. Checking Similar Problems

- There is not a very similar application that I have encountered. Other applications don't have a multiuser-web based functionalities, they have only boolean type inapp coin-flipping function.



- Though not being contextually related, other easy games to be played between users in realtime exists.

c-d. Subproblems/Design Model

- Creating a system that will match online users in realtime
- Using an online database (like Firebase) to save the outcome of the games and calculate rankings
- MVC pattern is classical example to be able to be used in this kind of applications

e. Requirements

Non-Functional Requirements

1. Operational Requirements

- 1.1 Application will operate under Android environment
- 1.2 Application needs Wifi/3G connection for multiplayer mode
- 1.3 Database needs to handle potential user overload by choosing the database manager wisely for scaling(Firebase/Firestone)
- 1.4 Application package size should be kept under 10 mb max at all times. For unexpected package enlargement issues, new bundling technology can be implemented.

2. Performance Requirements

- 2.1 Application needs to find+arrange match with opponents under 30-45 seconds at most
- 2.2 Real time ranking should be fetched under 6 seconds at most
- 2.3 Crash/single user ratio should be less than 1 occurrence/3 months period.

3. Security Requirements

- 3.1 Application needs to have certain protection mechanisms to prevent manipulation of the games' outcome or coin quantity (Authorization, not using phone's memory to save-fetch user data, obfuscation mechanisms).
- 3.2 Children may needed to be prohibited from playing the game according to latest Play Store policies (PG13).
- 3.3 User data should be kept private on database level (counting on security functions of external database)

4. Cultural and Political Requirements

- 4.1 It may be needed to do country specific search for following Google's policy. Other than that, countrys' own coins may needed to be implemented to have more meaningful user acquisitions.

Functional Requirements

1. User capabilities

- 1.1 Playing training match against the computer
- 1.2 Playing real time matches with other users
- 1.3 Sending/getting invitations to/from other users

2. System capabilities

- 2.1 Login function for the users for online matches
- 2.2 Finding opponents automatically
- 2.3 Creating a ranking system including all users in a database
- 2.4 Updating each users' score in realtime

f. Diagrams

SCENARIOS

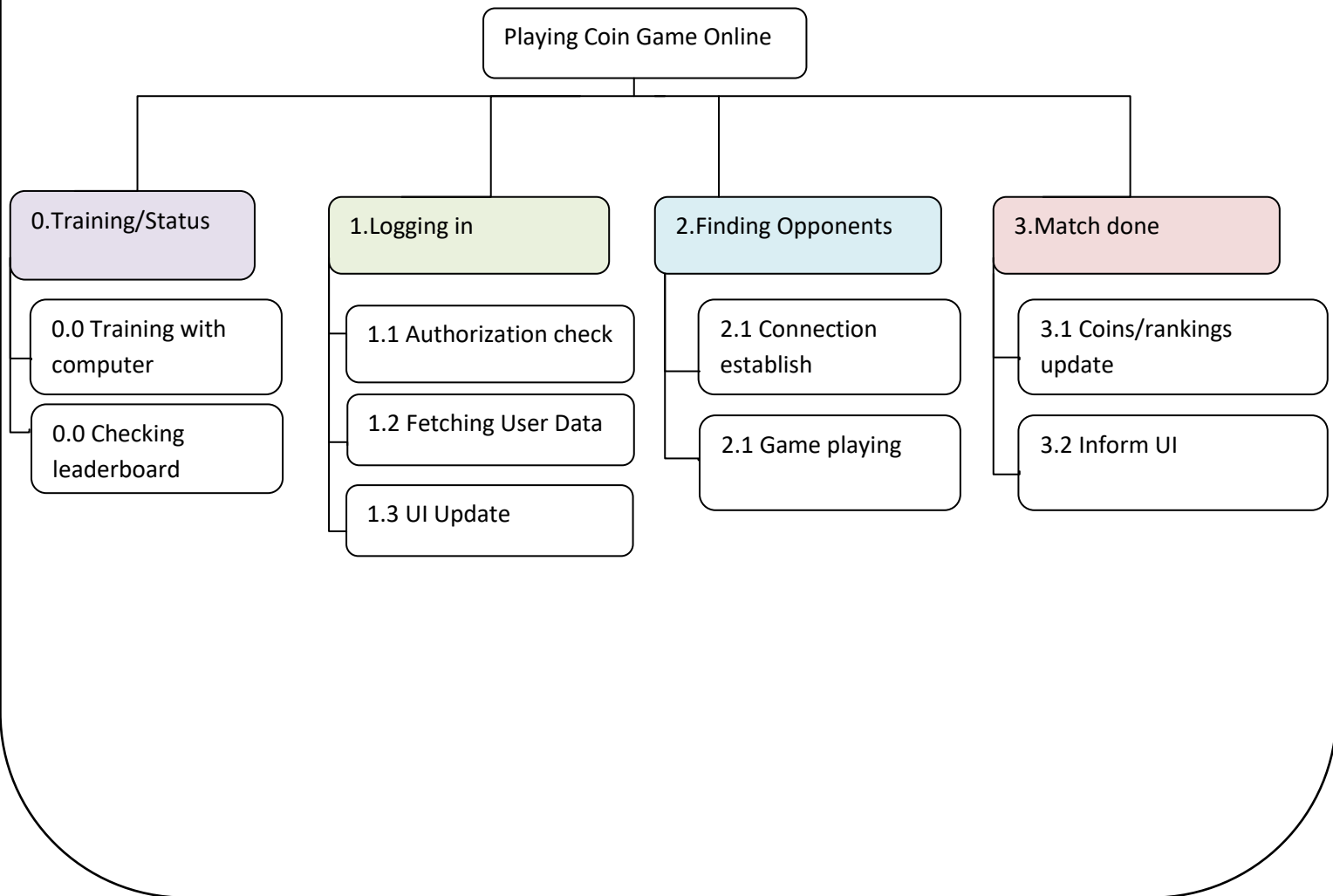
Scenario 1

Ken opens the application and logs in to the server automatically using his pre-saved username & password. He is not warned by the system because his internet is on. When he enters the matching room, he clicks “new match” button for the system to match him with an opponent from any part of the world. When the time comes, Ken wins the match by having more number of valuable clicks, he wins the match and databases are refreshed for the profiles and the leaderboards.

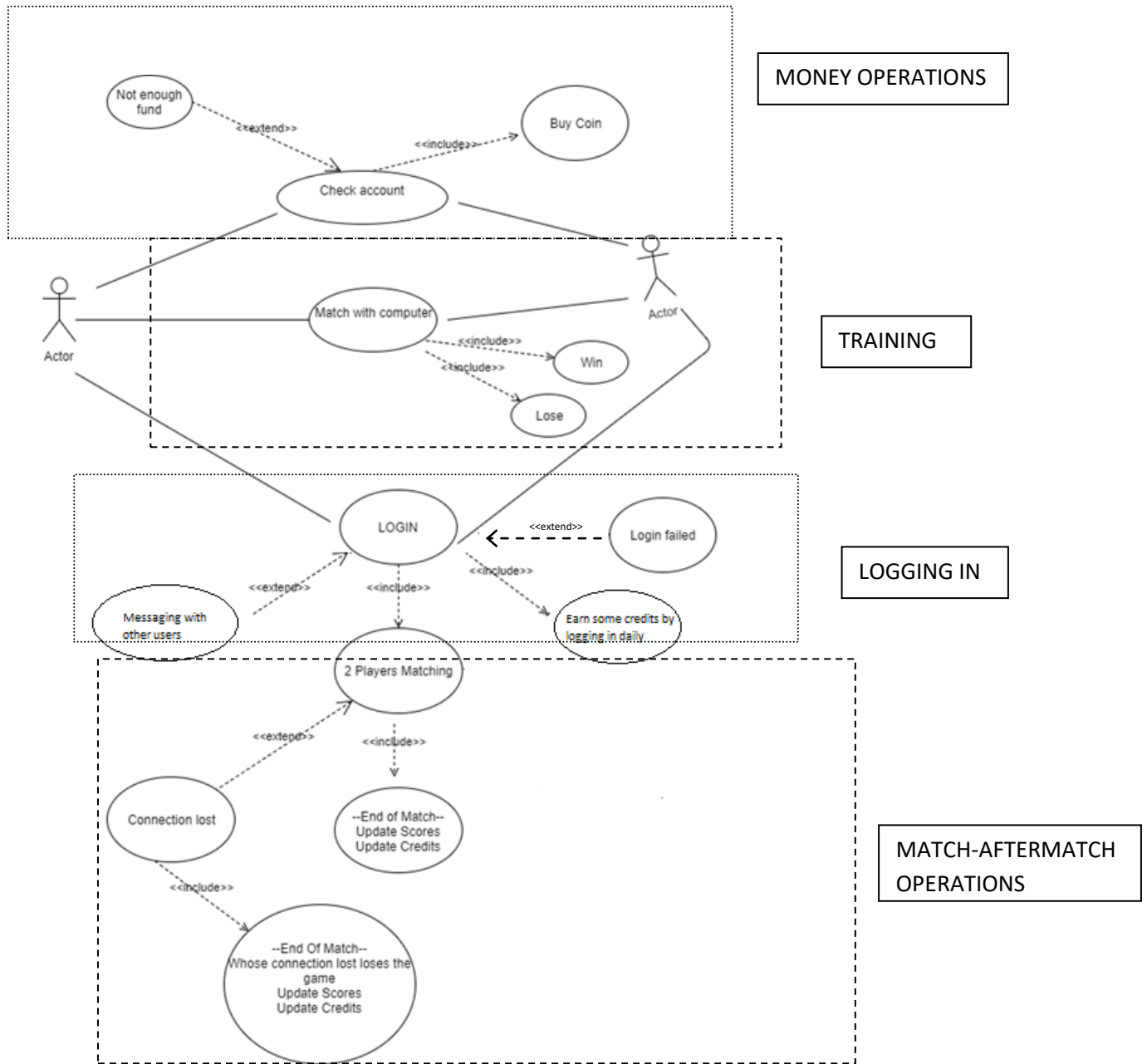
Scenario 2

The guy who lost to Ken sent an invitation back to Ken for a revenge match. After Ken got notified by his phone, he accepted the invitation in his application. This time after losing the match to his opponent, when he checks he finds out his highest score ranking now was lower than his opponent.

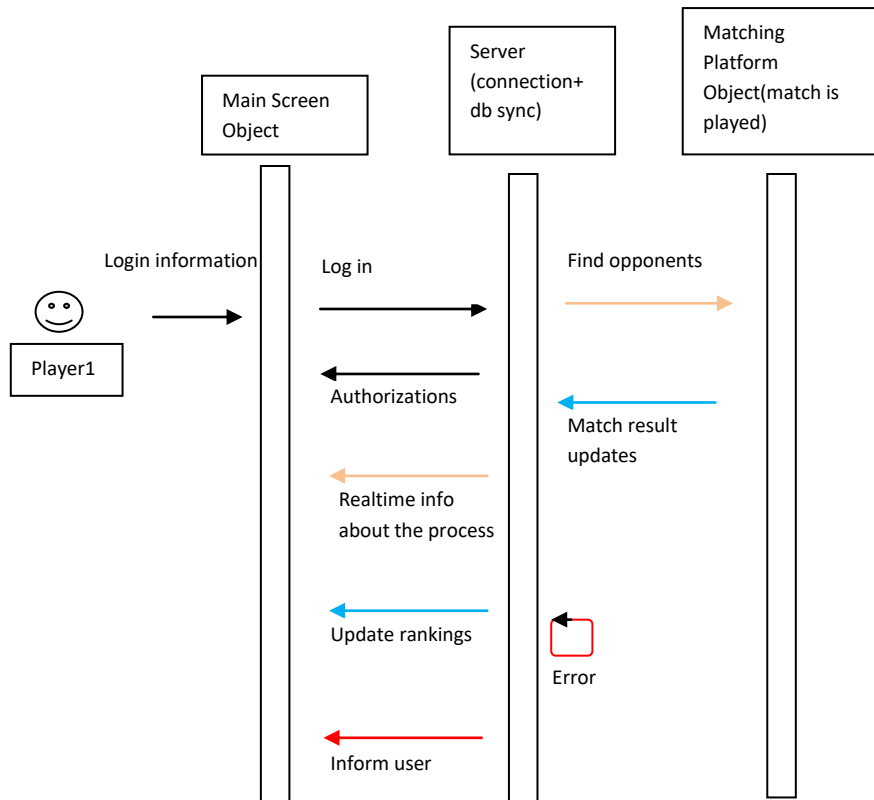
FUNCTIONAL DECOMPOSITION



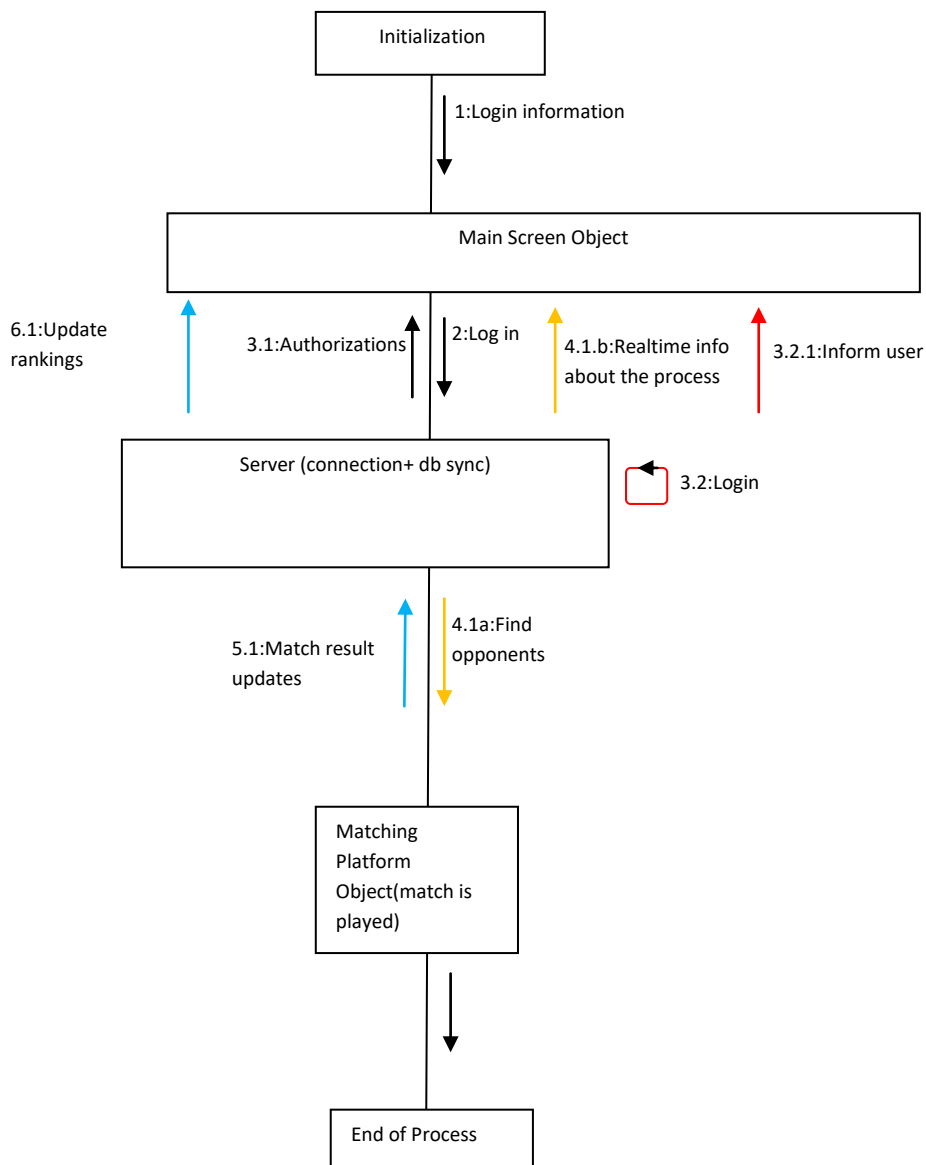
1. USE CASE DIAGRAM



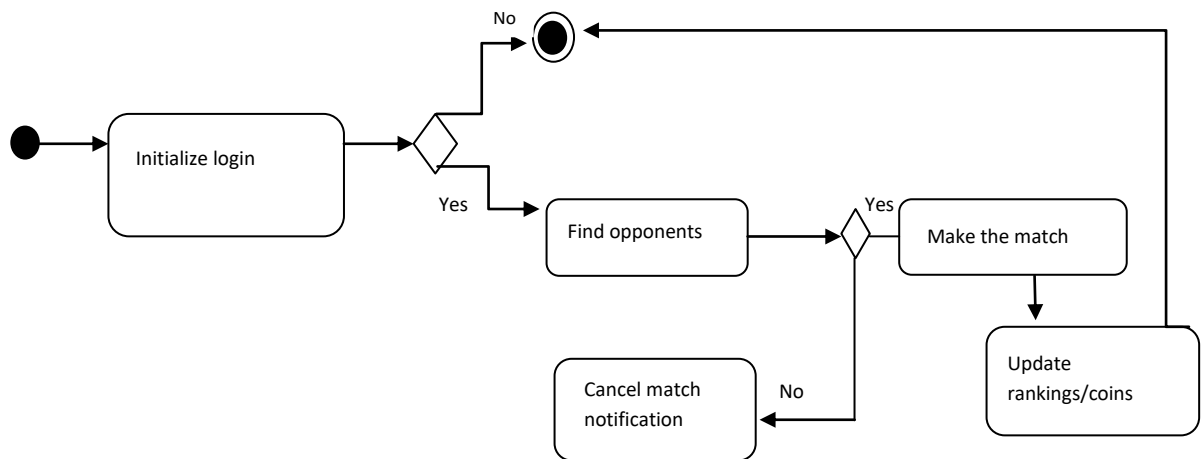
2. SEQUENCE DIAGRAM



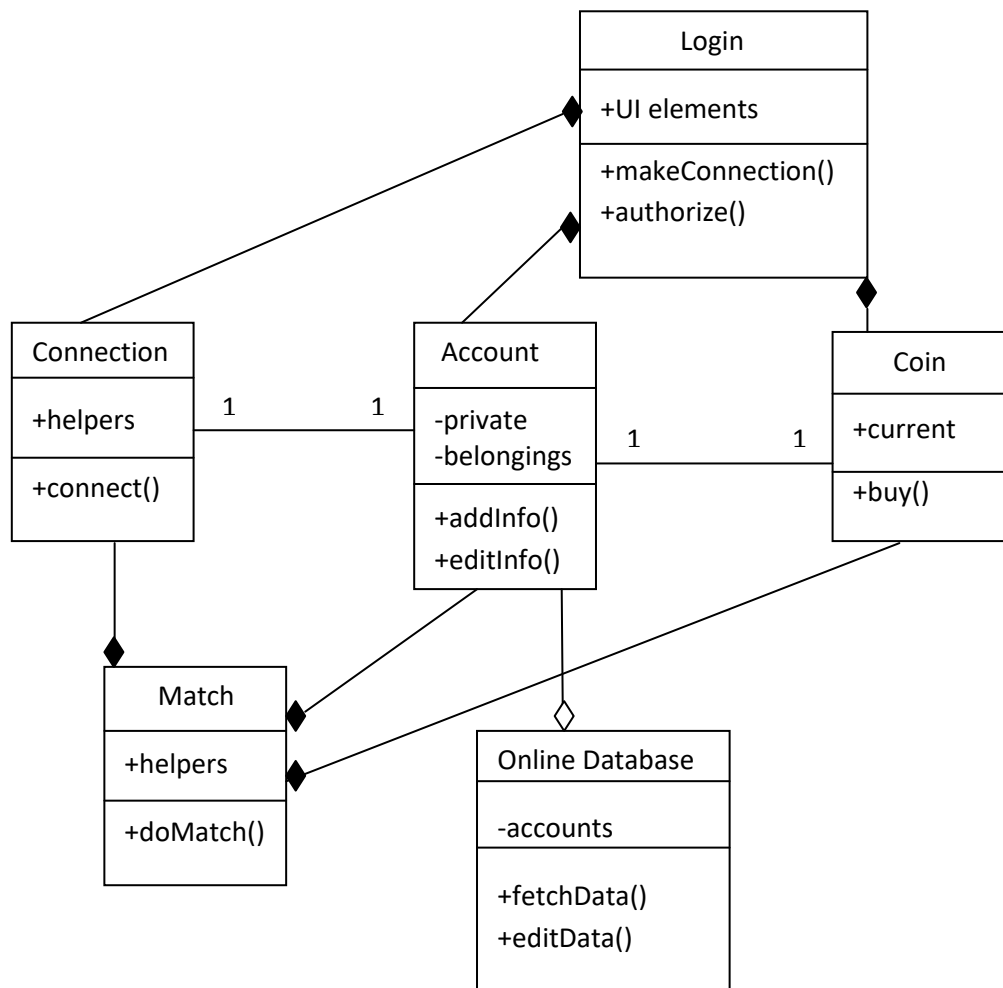
3. COLLABORATION DIAGRAM



4. ACTIVITY DIAGRAM



5. CLASS DIAGRAM (Iteration 1)

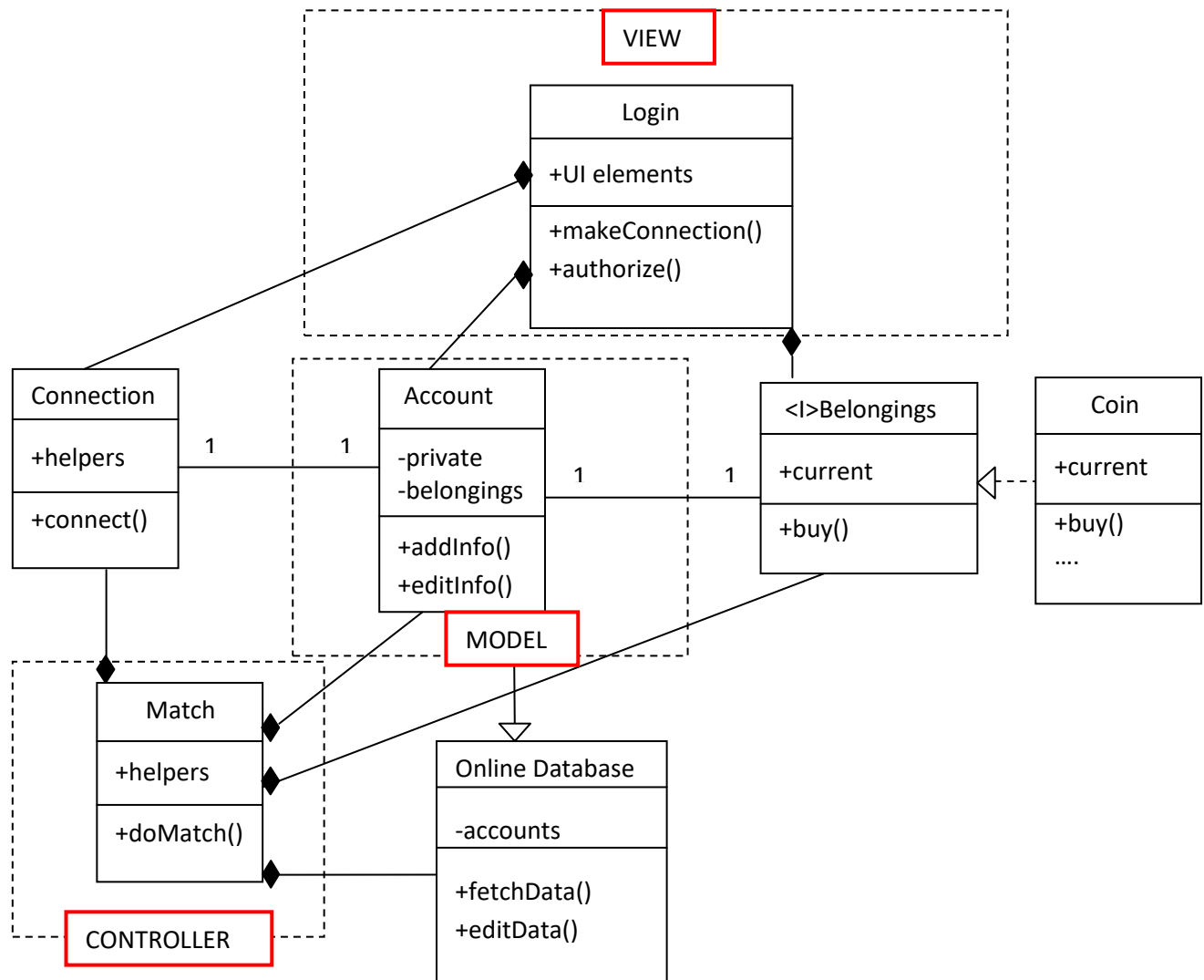


Why there is composition between Connection and Login? Because login activity(screen) can not function properly without existence of the Connection activity(or class). That's why it is an composition relationship.

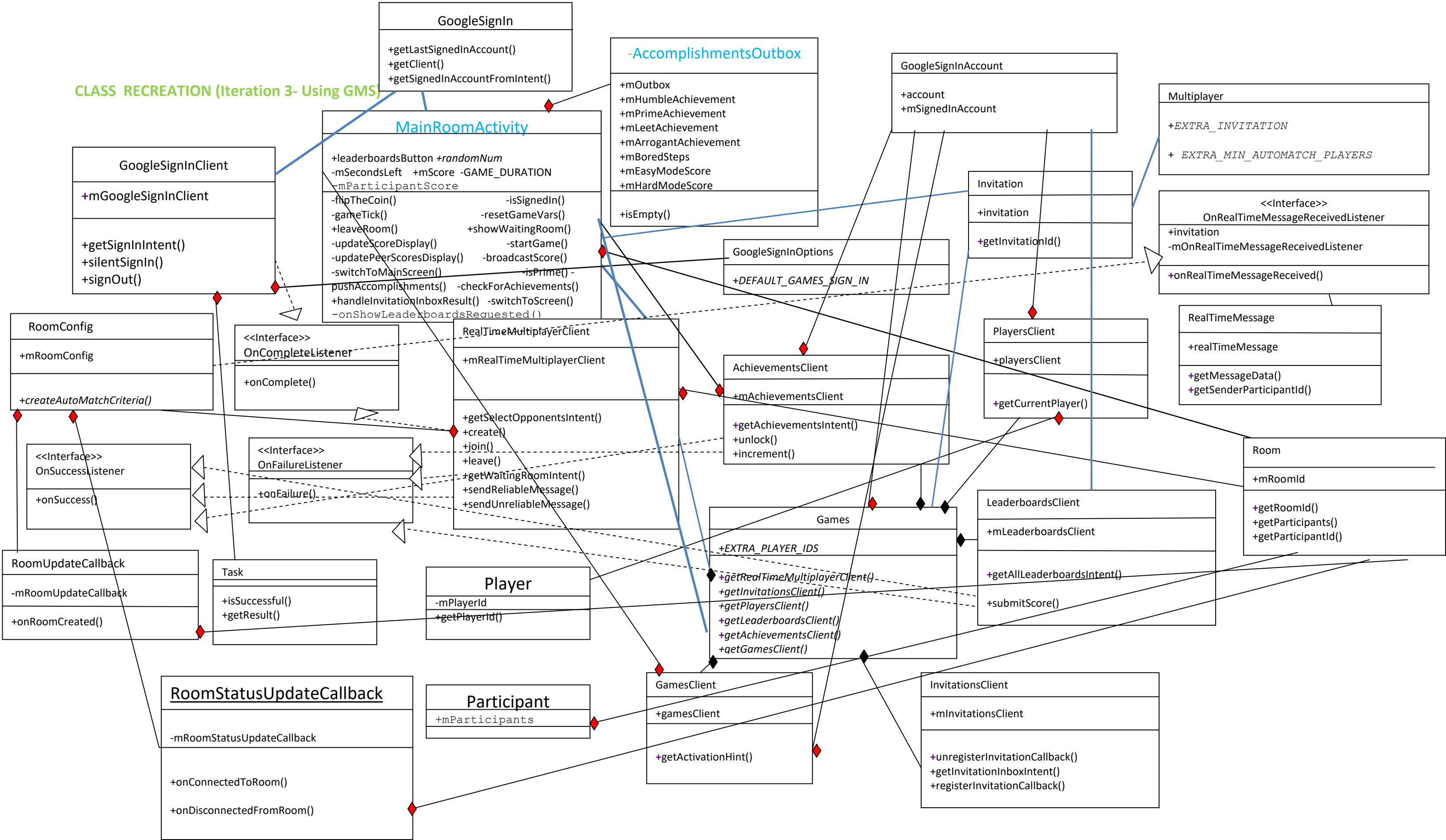
When it comes to Account-Coin relationship, it is an association one. Because both classes depend on each other and change in one can effect the other one. Unlike composition, there is a 2 ways information flow here.

For Account-Online Database relationship, we see that there is aggregation. While Account informations heavily depend on the Database activities, account can still exist without the functioning of the database(for example offline mode).

CLASS DIAGRAM (Iteration 2- after using SOLID Principles)



CLASS RECREATION (Iteration 3- Using GMS)



4) .:CARRYING OUT THE PLAN:.

a. Conforming to Plan

Not for the first or the second iteration, but when the application is recreated using a different model (using Google Play Games service) source code is compatible with the design model.

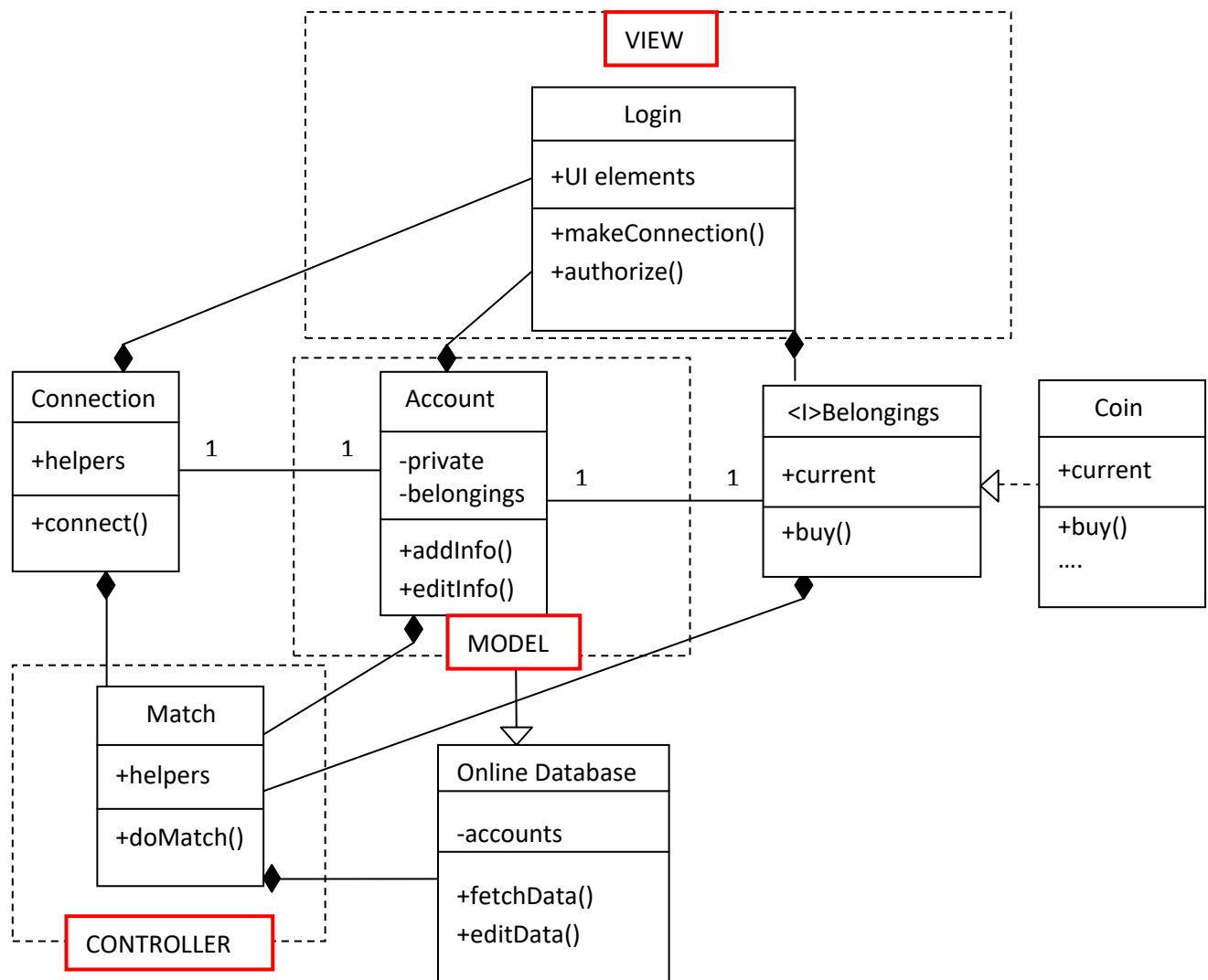
b. Component Checking

Application has been reviewed for correctness and has not been encountered any issue.

c. Applying SOLID Principles

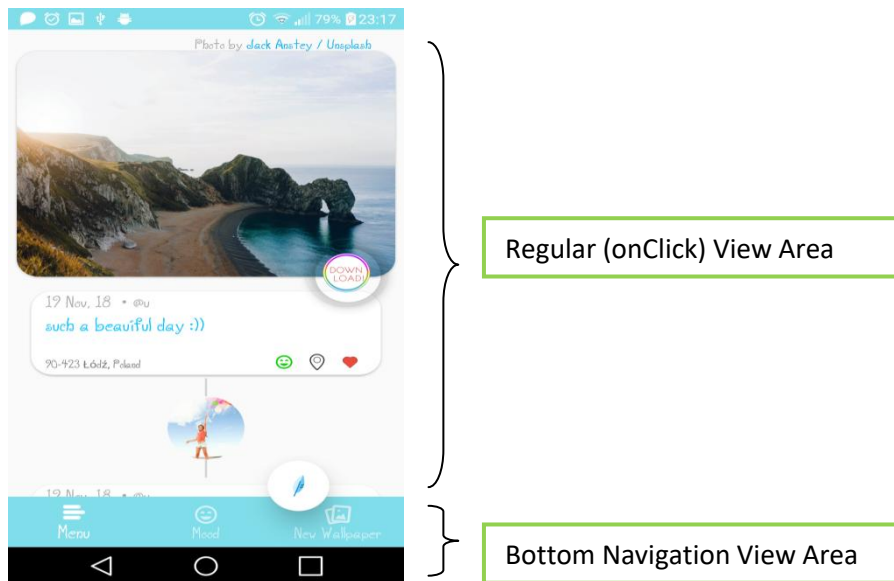
S: As is it seen, each class has one responsibility and only corresponding methods. MVC Pattern will be used to partitioning the responsibilities better and not cause coupling if changes are made.

O: Interface(or an abstract class) will be added for the “Coin” class to be able to break dependencies with the other classes when changes in the future “Belonging” types occur.



L: "Coin" class implements all the methods "Belongings" class has.

I: In the main screen(concerning UI components) we will have a bottom navigation view and regular view areas such as



For the bottom area we have a method “onNavigationItemSelectedListener” which derives from the corresponding interface “OnNavigationItemSelectedListener”.

On the regular area if we were to combine this method with the usual onClickListener interface’s onClick() method using an imaginary interface such as

```
myFictionalClickListener()
{
void setOnNavigationItemSelectedListener (View v);
void onClick(View v);
}
```

Everytime when we clicked on somewhere we should unnecessarily override bottom navigation view’s onNavigationItemSelectedListener() method although it wouldn’t be necessary. By applying proper design rules on the UI side ,we overcome this kind of problems.

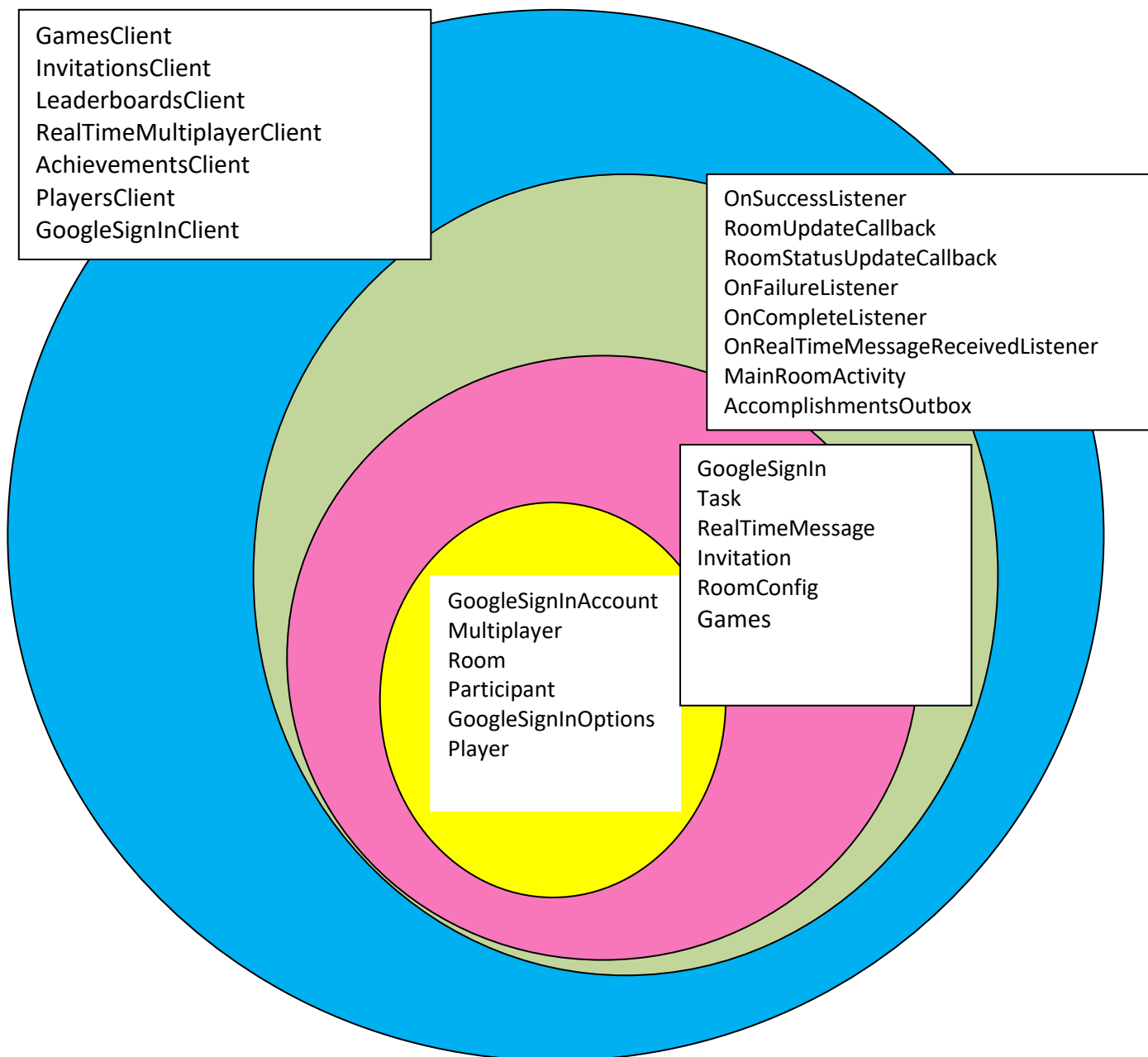
D: For registering to the online game platform and saving data we will use Firebase MBaaS platform. For authentication, instead of writing authentication logic for each of the possible sign-in methods (such as Google, facebook, email signins), I will use

`_FirebaseAuth.AuthStateListener()`
As an interface which I will implement the each signing way in one clean interface which allows further modification later.

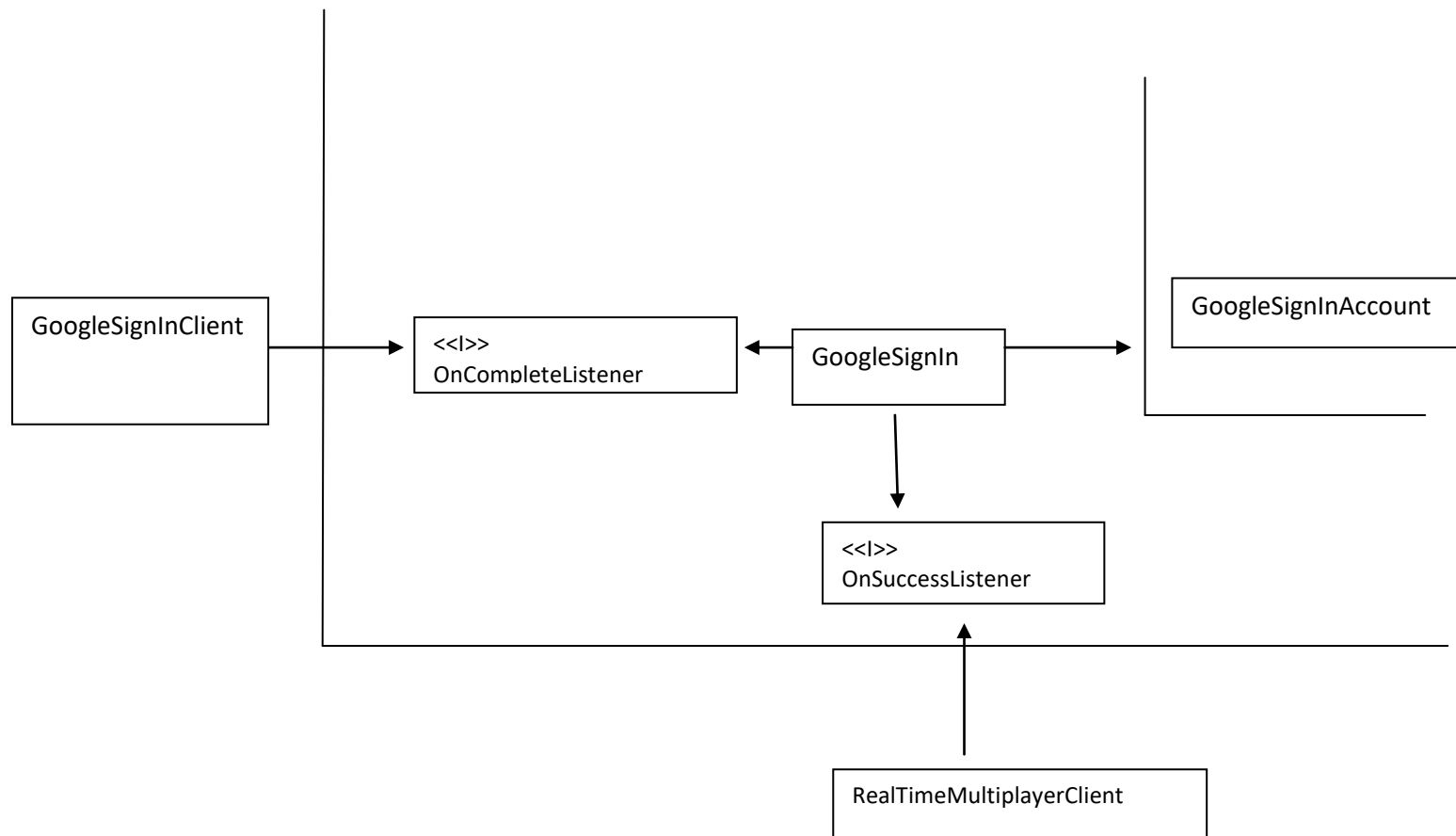
```
setAvailableProviders(Arrays.asList(
    new AuthUI.IdpConfig.GoogleBuilder().build(),

    new AuthUI.IdpConfig.EmailBuilder().build(),
    new AuthUI.IdpConfig.PhoneBuilder().build()
))
```

d. Components of the Project



Typical Scenario



5) .:EXAMINING THE RESULT:.

a. Testing

At the final phase, [Robotest](#) Android testing framework was also used for any inconsistencies or errors inbetween the components.

Want unlimited tests? [Upgrade](#)

📱 Flip Coin		Run a test		
Test matrix	Test type	Started	Total executions	Issues
✅ matrix-e25moyba1o9na	Robo	18 minutes ago	10	—
✅ matrix-35n3fu1m6e9ny	Robo	24 minutes ago	1	—

b. Confirmation

According to controls of the working application, all requirements and features are implemented successfully.

c. Reasons to Create Classes

GoogleSignInAccount: Hide implementation details, Package related operations

Multiplayer: Hide implementation details, Package related operations

Room: Package related operations

Participant: Package related operations

GoogleSignInOptions: Hide implementation details, Package related operations

Player: Package related operations

GoogleSignIn: Model real-world objects, Isolate complexity

Task: Model real-world objects, Isolate complexity

RealTimeMessage: Model real-world objects, Isolate complexity

Invitation: Model real-world objects, Isolate complexity

RoomConfig: Model real-world objects, Isolate complexity

Games : Model real-world objects, Isolate complexity

OnSuccessListener : Hide implementation details, Facilitate reusable code

RoomUpdateCallback: Hide implementation details, Facilitate reusable code

RoomStatusUpdateCallback: Hide implementation details, Facilitate reusable code

OnFailureListener: Hide implementation details, Facilitate reusable code

OnCompleteListener: Hide implementation details, Facilitate reusable code

OnRealTimeMessageReceivedListener: Hide implementation details, Facilitate reusable code

MainRoomActivity: Hide implementation details, Facilitate reusable code , Make central points of control

AccomplishmentsOutbox: Hide implementation details, Facilitate reusable code, Make central points of control

GamesClient: Package related operations, Model real-world objects

InvitationsClient: Package related operations, Model real-world objects

LeaderboardsClient: Package related operations, Model real-world objects

AchievementsClient: Package related operations, Model real-world objects

PlayersClient: Package related operations, Model real-world objects

GoogleSignInClient: Package related operations, Model real-world objects

RealTimeMultiplayerClient: Package related operations, Model real-world objects

d.e. Classes-CheckList:

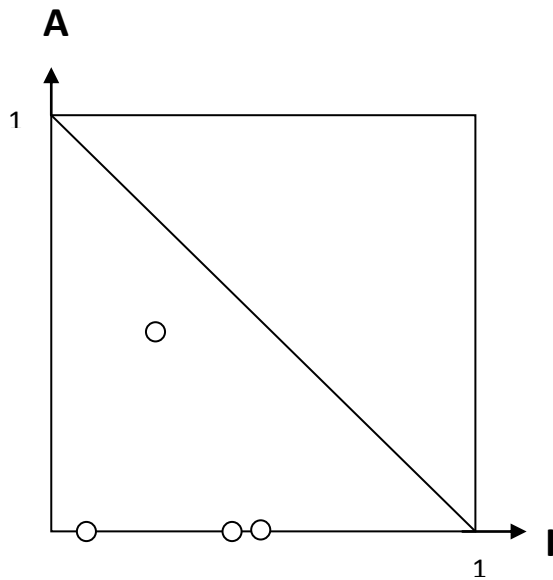
It s observed that all the inherited (gms) classes follow the best practices by means of the given criterias. The only issue is about an own written class MainRoomActivity:

This class hosts the main gaming screen as well as the main game logic (creating random coin values). It represents the activity so all the other connection classes need to take place inside. That's why it looks like a huge class but indeed it does its porpose only.

The only action that can be taken out of this class maybe the implementation of game logic to another activity but because of the easiness of the game logic and possible overhead to make the data transfer between two activities, this random number generation became the deviation of this class.

In addition to that it's worthwhile to mention that variables are not initialized in the constructor since this is the sole activity so it's constructor isn't needed to be initialized by something else.

f. Component Coupling

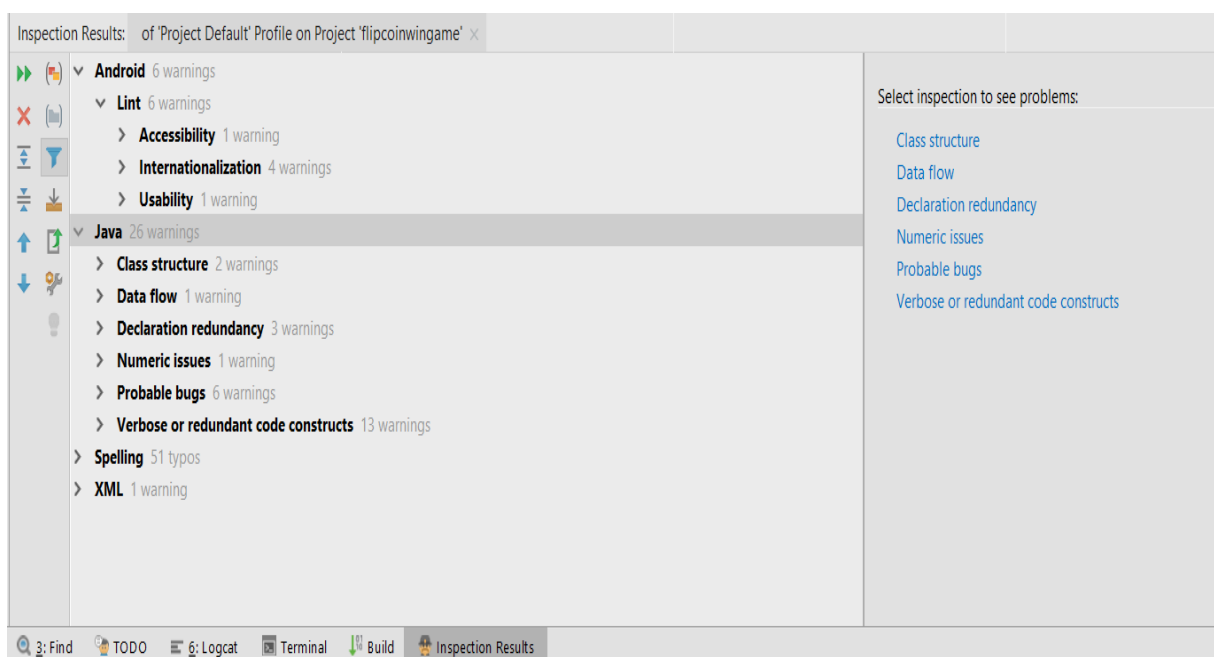


Entities (Core) Component (Yellow):	I= 1/14	A=0/6	D=0.92
Use Case Component (Pink):	I= 9/19	A=0/6	D=0.52
Adapters Component (Green):	I=5/23	A=4/8	D=0.28
FrameWorks Component (Blue):	I=10/27	A=0/7	D= 0.62

This looks like the only problematic area regarding to the application. Since almost all of the classes are inherited from GMS, it is not very probable there is an inconsistency in between the classes. Real reason why these relatively high numbers occurred possibly should be because of the higher level of abstractness in the inherited classes whose abstractness (or implementing interface level) can't be observed directly while importing them. It is obvious that for the sake of this application, analytics may not represent the true inner working of the coupling because of this specific mechanism used.

g. Static Analysis

FindBugs, PMD and Lint tools were the best alternatives. Lint was chosen for this application because of direct integrity to Android Studio. Generally verbose and spelling type suggestions have been observed.



6) .:PRESENTATION SCREENSHOTS:.

.:PROJECT TASK:.

Flipping Coin to Win

Android App Project



M.K. Arslan/376299

- ✿ Android application project about coin flipping
(to achieve the highest score)

- ✿ Can be utilized also as a multiplayer
game playing framework



.:UNDERSTANDING THE PROBLEM:.

.:Stakeholders:.

- 📱 End users
- 📱 Ads marketers
- 📱 Online community/tech writers/channels

.:SubTasks:.

- 📱 Matching two players from online pool
- 📱 Displaying the actual game on screen
- 📱 Updating the database

.:Unknowns/Features:.

- Establishing connection between a pool of players
- Using database to save game scores/profiles
- MBaaS
- User profiles
- Having a leaderboard

.:Requirements:.

Non-Functional Requirements/Operational

- Operation under Android environment
 - Wifi/3G connection for multiplayer mode
 - Database management
 - Package size < 10 mb max
-

.:Requirements:.

Non-Functional Requirements/Performance

- Automatch < 30-45 seconds
 - Real time ranking fetch < 6 seconds
 - Crash/single user ratio < 1 crash/3 months
-

.:Requirements:.

Non-Functional Requirements/Security

- Authorization, obfuscation mechanisms
 - Play Store policies (PG13)
 - Database level protection
-

Requirements:

Functional Requirements/User Capabilities

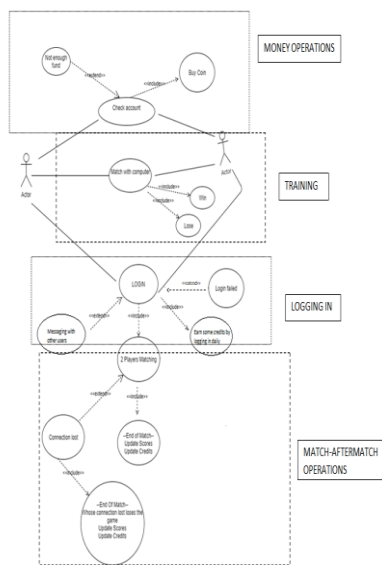
- Training match against the computer
- Real time matches with other users
- Sending/getting invitations to/from other users

Requirements:

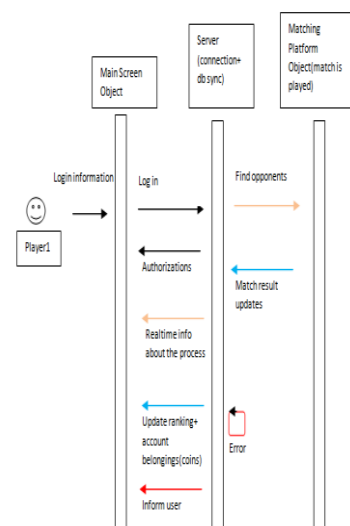
Functional Requirements/System capabilities

- Login function for online matching
- Finding opponents automatically
- Ranking system
- Updating scores in realtime

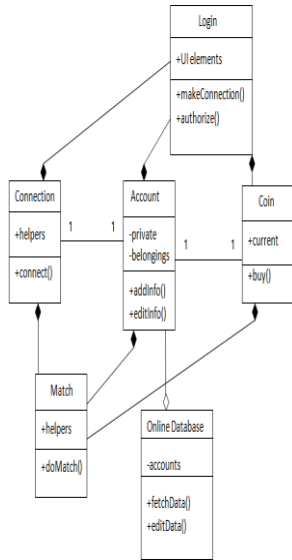
USE CASE DIAGRAM:



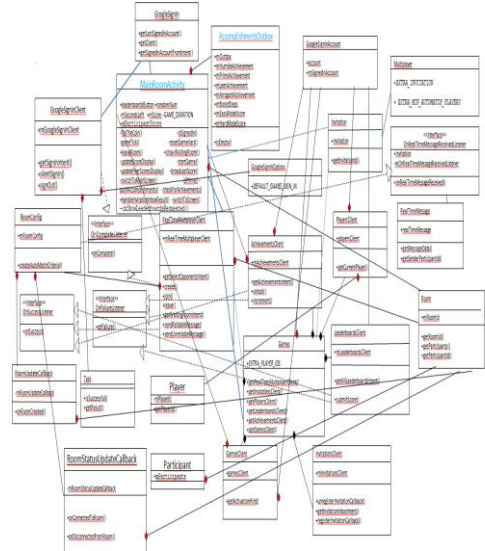
SEQUENCE DIAGRAM:



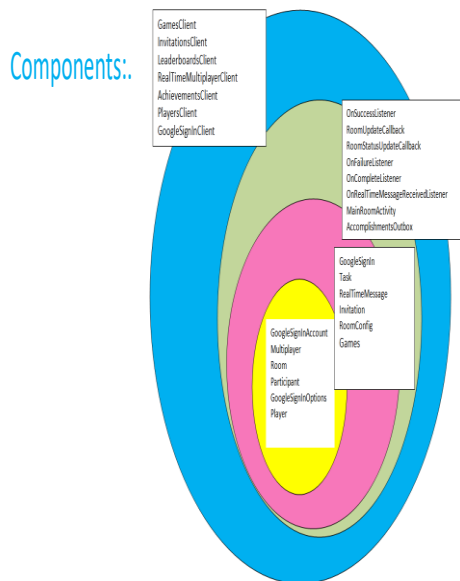
CLASS DIAGRAM (Iteration 1)



CLASS RECREATION (Iteration 3- GMS)






.:CARRYING OUT THE PLAN:.

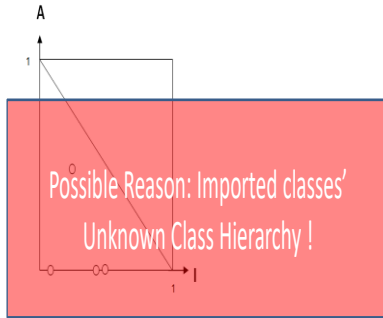


.:EXAMINING THE RESULT:.

Robotest Android testing frameworks

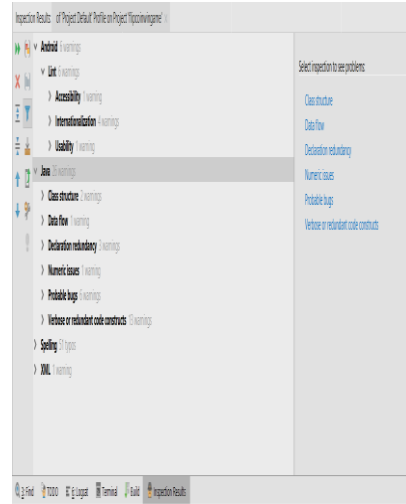
Test execution	Duration	Locale	Orientation
 Nexus IP Virtual, API Level 25	5 min 23 sec	English (United States)	Portrait
 Galaxy S6, API Level 23	48 sec	English (United States)	Portrait
 Nexus 9, Virtual, API Level 24	5 min 25 sec	English (United States)	Portrait
 Pixel 2, Virtual, API Level 28	1 min 41 sec	English (United States)	Portrait
 Nexus7 chrome.DVD 16:9 aspect ratio, Virtual, API Level 25	5 min 12 sec	English (United States)	Portrait
 Nexus 5, Virtual, API Level 22	2 min 0 sec	English (United States)	Portrait
 Pixel 2, API Level 28	28 sec	English (United States)	Portrait
 LG G6(G650G), API Level 24	52 sec	English (United States)	Portrait
 OnePlus One, API Level 22	34 sec	English (United States)	Portrait
 Nexus 6, Virtual, API Level 21	2 min 8 sec	English (United States)	Portrait

Component Coupling



Entities (Core) Component (Yellow):	I= 1/14	A=0/6	D=0.92
Use Case Component (Pink):	I= 9/19	A=0/6	D=0.52
Adapters Component (Green):	I=5/23	A=4/8	D=0.28
FrameWorks Component (Blue):	I=10/27	A=0/7	D= 0.62

Static Analysis



AS Lint

..:CONCLUSIONS:..

- All specifications & requirements are met
- App works w/out flow (automated tests)
- High component coupling numbers because of imported Google component classes with unknown hierarchy



7) .:CONCLUSIONS:.

In conclusion, the application is created and all specifications and requirements are met successfully. All the functions eachselves and the application in general is working perfectly fine - automated tests also confirm these. The only problem (as briefly mentioned in the Component Coupling section) is because of the high number of inherited classes' (which forced me to go this way because of the domain choice) abstraction family history is unknown, Main Sequence criteria is not reached due to this structure. Since the used framework is a Google framework, still it is safe to claim that it s very unlikely there is a problem in the dependency metrics in the parents of the inherited classes. All in all, framework is applied succesfully.