CS102	**Fall 2016/2017**	Project Group	9
Instructor:	**Öznur Taştan Okan**		
Assistant:	**Halil İbrahim Kayım**		

~ Happy Real Estate ~

Happy_Koalas

Burak Erkılıç 21501035
Deniz Evrensel 21401841
Özlem Yıldız 21502657
Selim Fırat Yılmaz 21502736
Mert Yurdakul 21401628

Criteria	TA/Grader	Instructor
Presentation		
Overall		

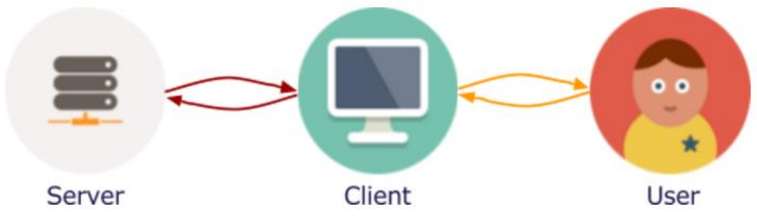
Project Requirements Specification Report
(Version 1.0)
25 November 2016

1. Introduction

In the 21st century, people has not enough time to allocate time for looking for real estates. In this respect, Happy Real Estate will take a part. The aims of the program are that easing the user to find real estates and helping seller to demonstrate his/her real estates. This program will demonstrate an online map which helps to perceive the location of the real estate and give detailed information and photos about the real estate. Additionally, with the colors in the map, to distinguish the price and type of the real estate will be easier. This is a desktop application and is on purpose of buyers and sellers/renters. The purpose of this report is giving information about the outline design of the program.

2. System Overview

2.1 Client-Server Architecture



In the 21st century, people has not enough time to allocate time for looking for real estates. In this respect, Happy Real Estate will take a part. The aims of the program are that easing the user to find real estates and helping seller to demonstrate his/her real estates. This program will demonstrate a an online map which helps to perceive the location of the real estate and give detailed informations and photos about the real estate. Additionally, with the colors in the map, to distinguish the price and type of the real estate will be easier. This is a desktop application and is on purpose of buyers and sellers/renters. The purpose of this report is giving information about the outline design of the program.

2.2 Database

Map will help user to perceive appropriately the location of the house. User will be able to browse around the area which houses are in and explore the advantages of the surrounding of the house. When a user decides to buy a house, the contact information of the seller will be available to this user.

2.3 Environments

2.3.1 Development Environment

During the development stage, the server mostly will run on the same computer with the client. Obviously, it will be needed to test server-client communication for some situations such as database or network connection assurance. Both the server and the client projects will require a JVM with Java 8 support.

The collaboration of group members will be through Github. Each group member will update the project from Github repository before making changes and then new changes will committed to Github repository. Also, Github repository Issues page will be used to track requirements for the development. For example, we opened the "HouseFeatures" issue because the HouseFeatures class needed to be filled with features of house like room number, and size of the house.

2.3.2 Production Environment

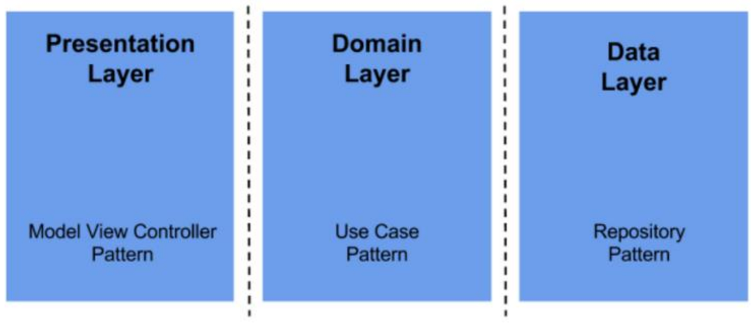
When the initial development is completed, the server project will be deployed into a server with JVM that supports Java 8. The server provider most probably will be heroku.com. Its simplified configurations will boost our process and rescue us from the operating system's problems. Since the the Happy Real Estate is designed as a desktop program and will be written in Java, thanks to Java's cross-platform feature, it will be able to run on many computers with many different operating systems. Thus, nearly all computer owners will be able to use the Happy Real Estate. Finally, the resulting application will be in .jar format.

3. Core Design Details

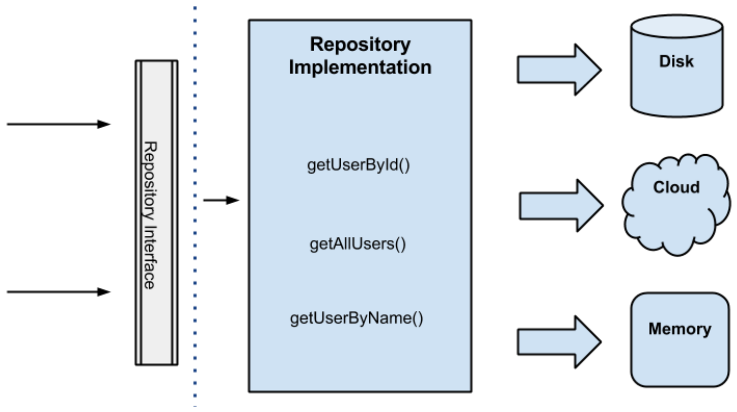
Happy Real Estate consists of two modules. These are named the client and the server. These modules will be compiled and built as separate programs and there will not be any dependency relation between them. Since the Happy Real Estate consists of two modules, the data integrity is very important. To provide data integrity, the patterns of the client and the server must be consistent. Thus, we tried to make them consistent. The client and the server design details will be explained below.

3.1 Client

Our client is a complex software, thus we decided to separate it into three layers. These are presentation layer, domain layer, and data layer.



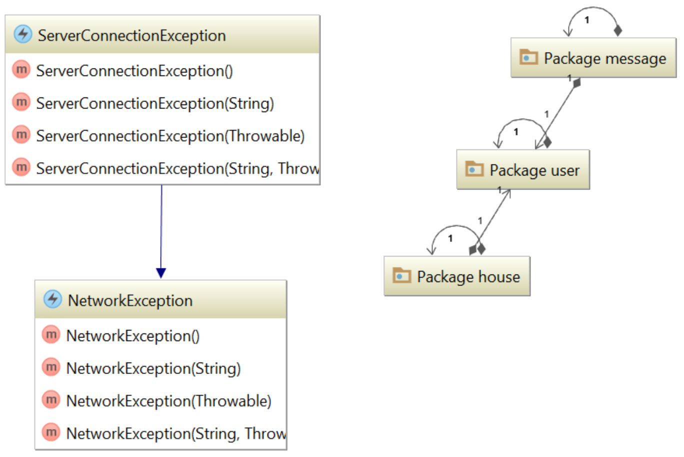
3.1.1 Data Layer



The data layer will use the Repository pattern with a idea that, through a factory, gets data from different data sources such as remote(server) or local(disk/memory) depending on definite conditions. For instance, while traversing around the map, if the area traversed and accordingly cached before and no new entries, the cache from local data source will be used or else, data will be requested from the remote data source. The idea behind repository pattern is that domain and presentation layers do not know anything about the source of the data.

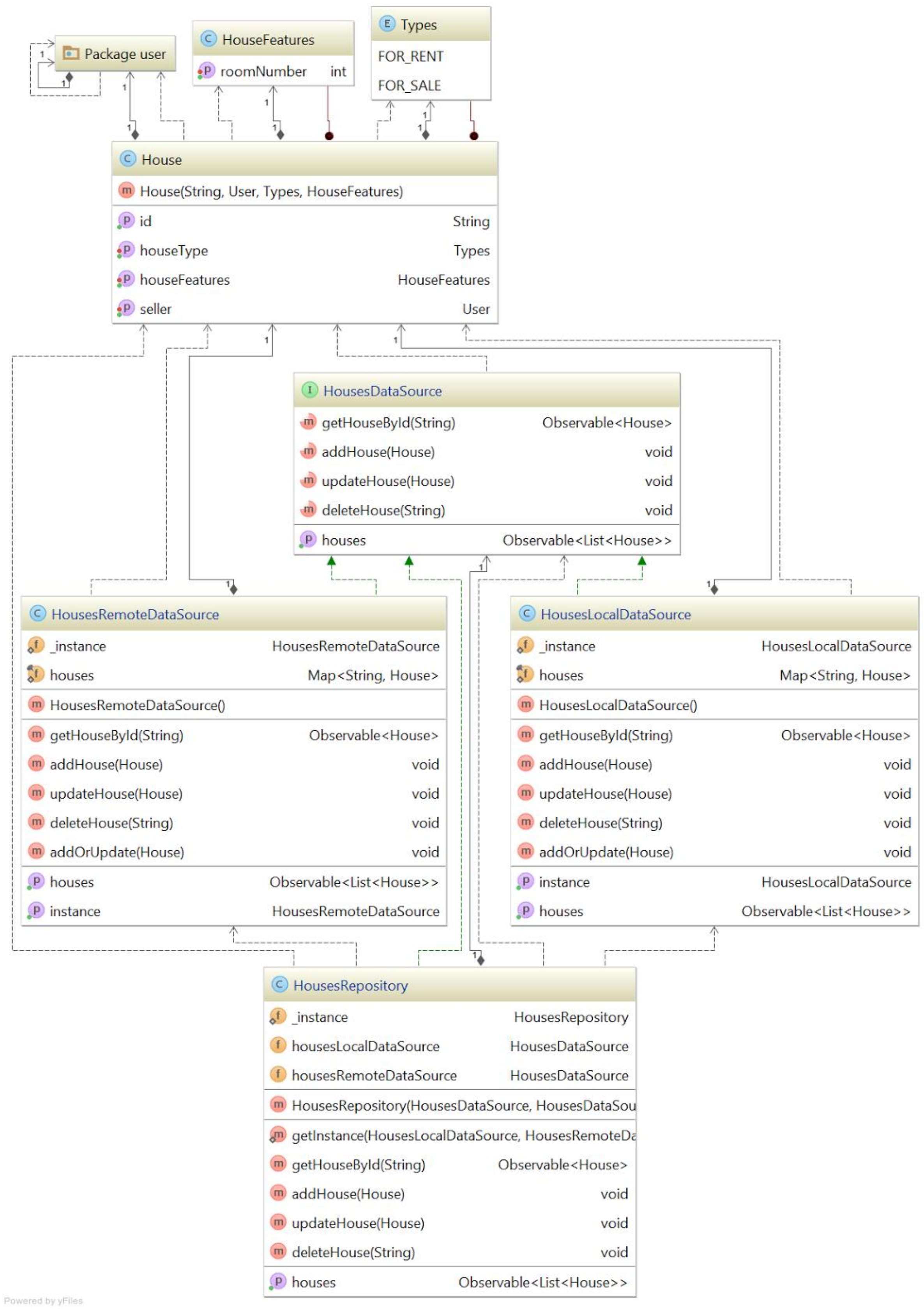
For unique objects, the Entry pattern is useful because they will have identity on the database. Entries will be House, User, and Message. For each entry, there will be a repository and data sources. User has no local data source because a user and its operations must always be in synchronization with the database. Thus, a local data source is not needed but to provide consistency in between data layer, still the same pattern will be applied. The domain and presentation classes will use entries as modals.

General diagram for data layer, namely the ***com.koala.app.client.data*** package:


ServerConnectionException
ServerConnectionException()
ServerConnectionException(String)
ServerConnectionException(Throwable)
ServerConnectionException(String, Throw
NetworkException
NetworkException()
NetworkException(String)
NetworkException(Throwable)
NetworkException(String, Throw
Package message
Package user
Package house
1
1
1
1
1
1

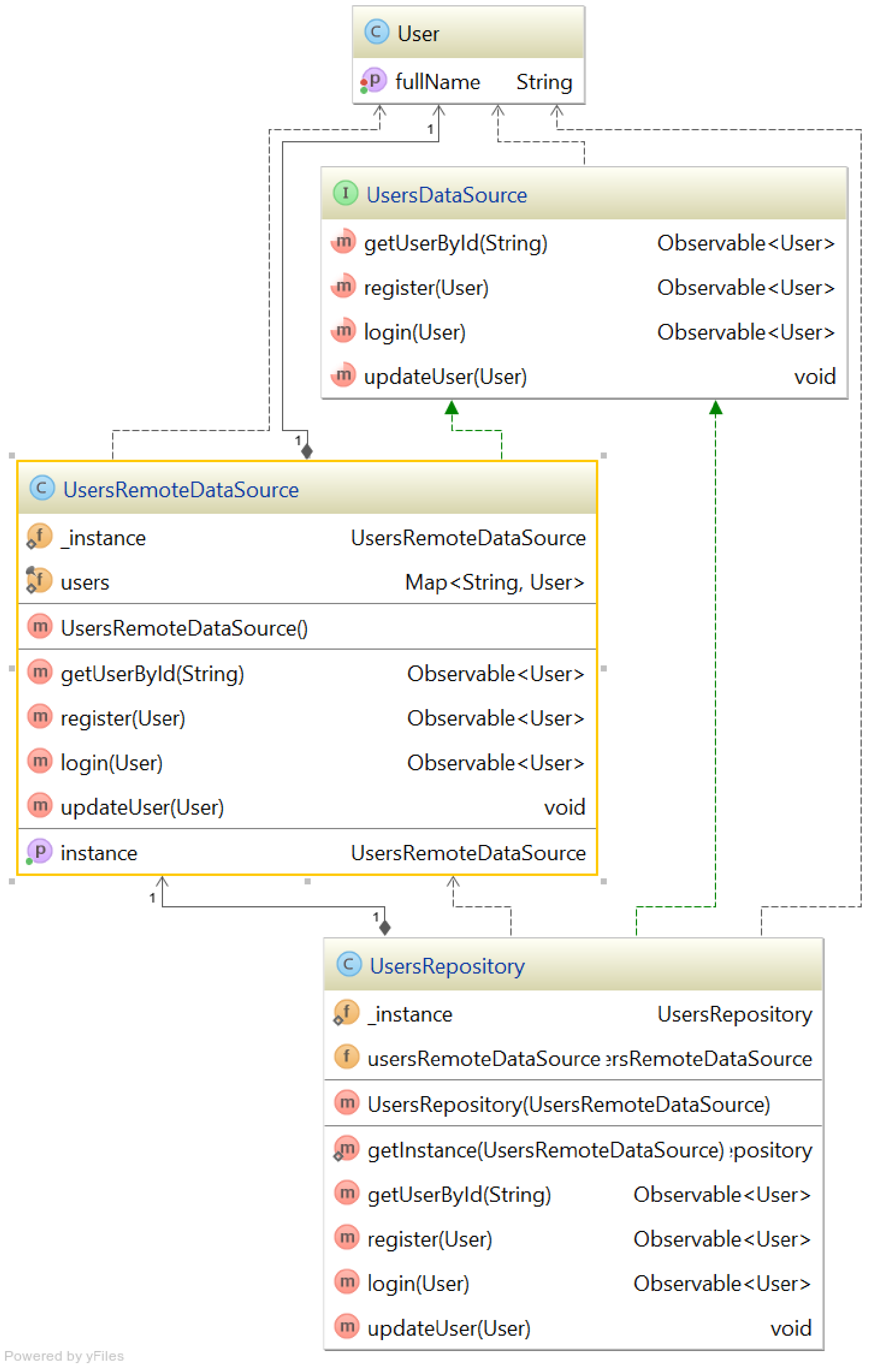
Data layer contains three packages which are *message*, *user*, and *house* exceptions for network failures.

3.1.1.1 House


Package user
HouseFeatures
roomNumber int
Types
FOR_RENT
FOR_SALE
House
House(String, User, Types, HouseFeatures)
id String
houseType Types
houseFeatures HouseFeatures
seller User
HousesDataSource
getHouseById(String) Observable<House>
addHouse(House) void
updateHouse(House) void
deleteHouse(String) void
houses Observable<List<House>>
HousesRemoteDataSource
_instance HousesRemoteDataSource
houses Map<String, House>
HousesRemoteDataSource()
getHouseById(String) Observable<House>
addHouse(House) void
updateHouse(House) void
deleteHouse(String) void
addOrUpdate(House) void
houses Observable<List<House>>
instance HousesRemoteDataSource
HousesLocalDataSource
_instance HousesLocalDataSource
houses Map<String, House>
HousesLocalDataSource()
getHouseById(String) Observable<House>
addHouse(House) void
updateHouse(House) void
deleteHouse(String) void
addOrUpdate(House) void
instance HousesLocalDataSource
houses Observable<List<House>>
HousesRepository
_instance HousesRepository
housesLocalDataSource HousesDataSource
housesRemoteDataSource HousesDataSource
HousesRepository(HousesDataSource, HousesDataSou
getInstance(HousesLocalDataSource, HousesRemoteDa
getHouseById(String) Observable<House>
addHouse(House) void
updateHouse(House) void
deleteHouse(String) void
houses Observable<List<House>>
Powered by yFiles

House package contains an entry class House and repository for Houses. There are two data sources for repository which are remote and local.

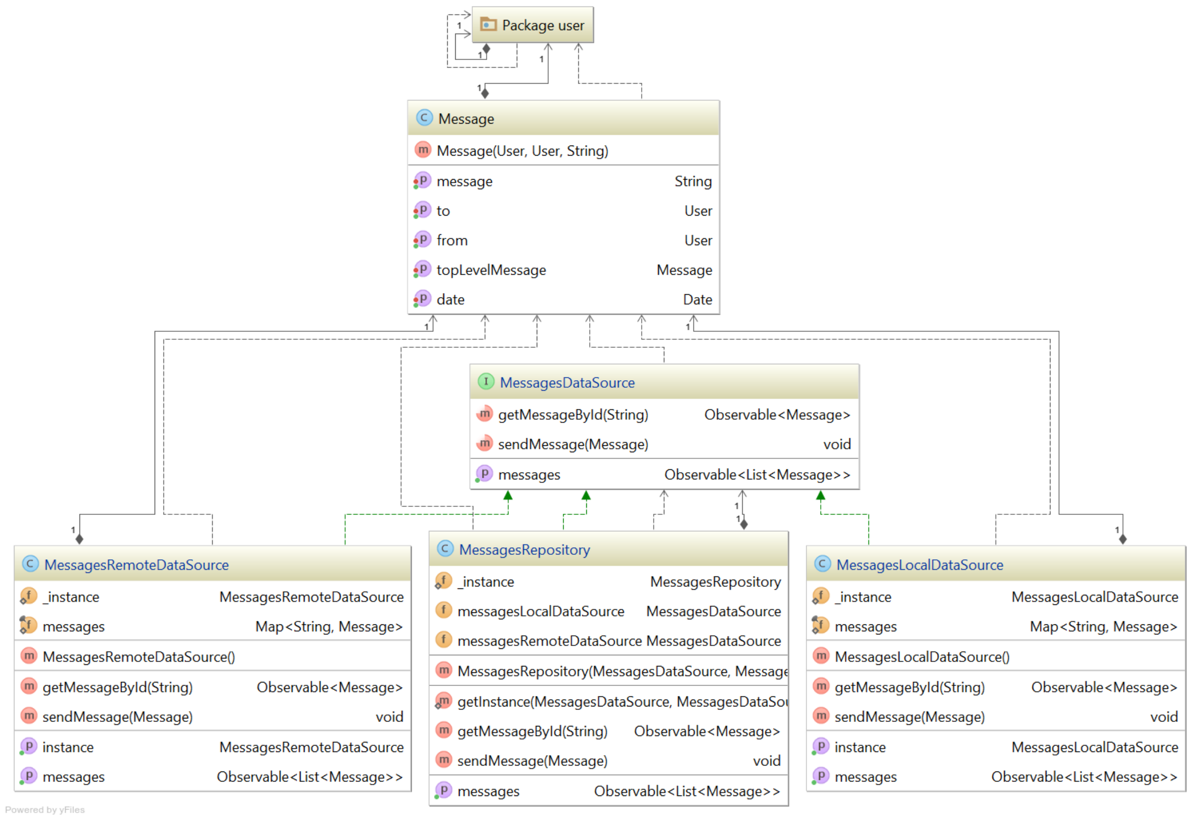
3.1.1.2 User



User
fullName String
1
UsersDataSource
getUserById(String) Observable<User>
register(User) Observable<User>
login(User) Observable<User>
updateUser(User) void
1
UsersRemoteDataSource
_instance UsersRemoteDataSource
users Map<String, User>
UsersRemoteDataSource()
getUserById(String) Observable<User>
register(User) Observable<User>
login(User) Observable<User>
updateUser(User) void
instance UsersRemoteDataSource
1
1
UsersRepository
_instance UsersRepository
usersRemoteDataSource ersRemoteDataSource
UsersRepository(UsersRemoteDataSource)
getInstance(UsersRemoteDataSource) pository
getUserById(String) Observable<User>
register(User) Observable<User>
login(User) Observable<User>
updateUser(User) void
Powered by yFiles

House package contains an entry class User and repository for Users. There is only one data source for Repository because user must always be up-to-date.

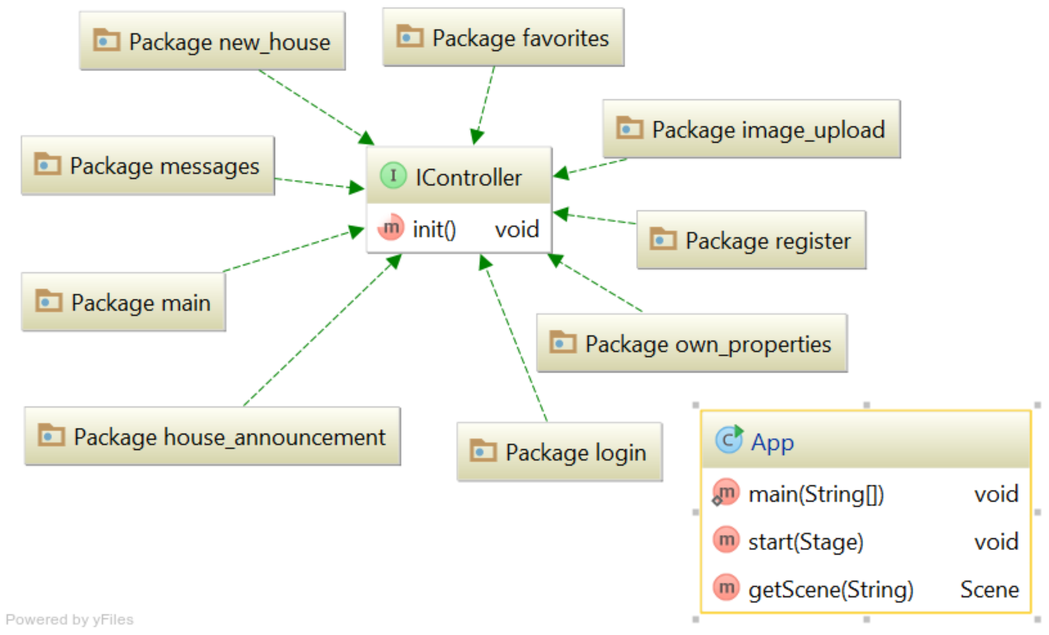
3.1.1.3 Message


Package user
Message
Message(User, User, String)
message String
to User
from User
topLevelMessage Message
date Date
MessagesDataSource
getMessageById(String) Observable<Message>
sendMessage(Message) void
messages Observable<List<Message>>
MessagesRemoteDataSource
_instance MessagesRemoteDataSource
messages Map<String, Message>
MessagesRemoteDataSource()
getMessageById(String) Observable<Message>
sendMessage(Message) void
instance MessagesRemoteDataSource
messages Observable<List<Message>>
MessagesRepository
_instance MessagesRepository
messagesLocalDataSource MessagesDataSource
messagesRemoteDataSource MessagesDataSource
MessagesRepository(MessagesDataSource, Message
getInstance(MessagesDataSource, MessagesDataSo
getMessageById(String) Observable<Message>
sendMessage(Message) void
messages Observable<List<Message>>
MessagesLocalDataSource
_instance MessagesLocalDataSource
messages Map<String, Message>
MessagesLocalDataSource()
getMessageById(String) Observable<Message>
sendMessage(Message) void
instance MessagesLocalDataSource
messages Observable<List<Message>>
Powered by yFiles

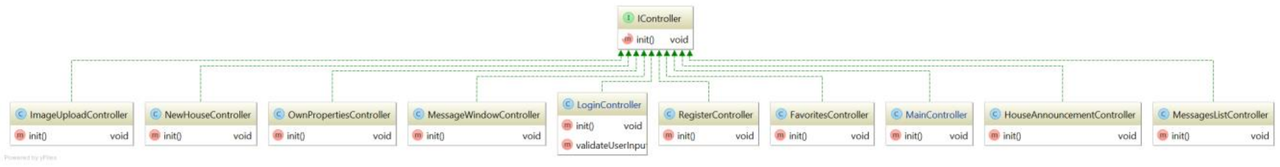
Message package contains an entry class Message and repository for Messages. There are two data sources for repository which are remote and local.

3.1.2 Presentation Layer

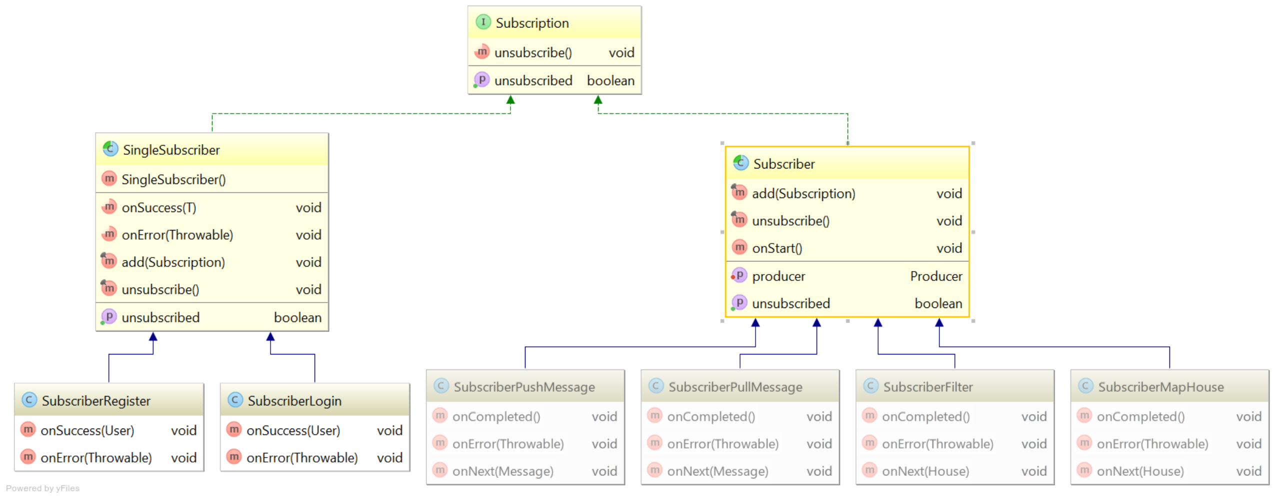
General diagram for data layer, namely the ***com.koala.app.client.presentation*** package:



The Presentation Layer is where model-view-controller pattern lives. For each window and dialog there will be a package as seen above. As we use JavaFX for GUI, the views will be in .fxml format. Accordingly, they will be separated from controllers & modals with no effort.



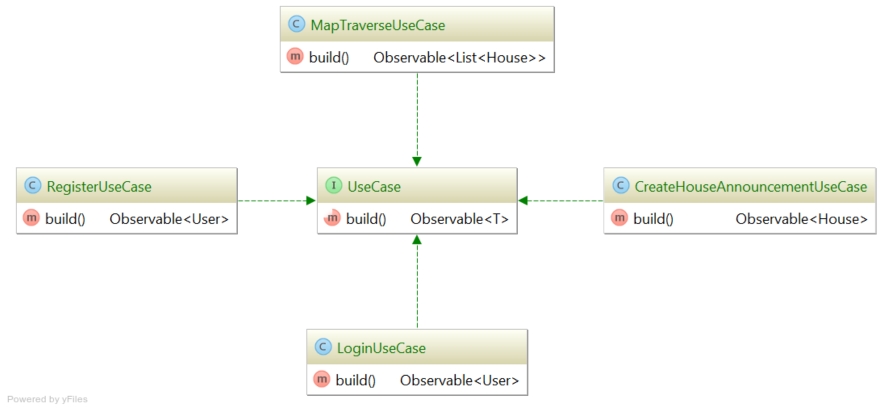
Views are binded to the controllers automatically by JavaFX. And will manage the events on views.



Subscribers are the listeners & requesters that works with use cases on domain layer.

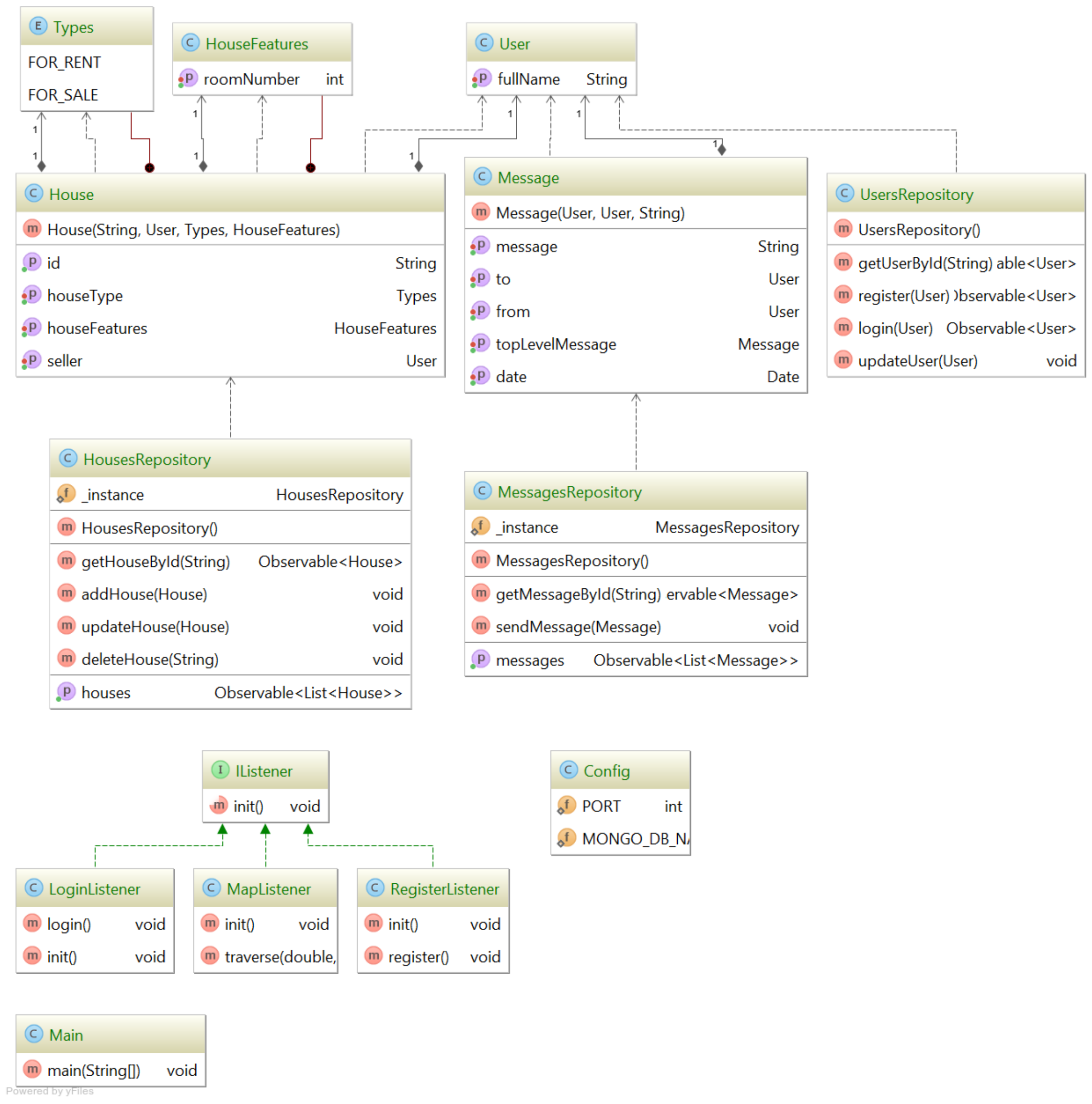
3.1.3 Domain Layer

Domain layer will be on ***com.koala.app.client.domain*** package.



Domain layer will use the Use Case pattern to help the communication between data layer and presentation layer. Use Case pattern will encapsulate the related algorithms.

3.2 Server



The server contains models, listeners for client requests, config. The server will communicate with client via sockets using a library specified on external libraries section below.

3.3 External Libraries

It is known that reinventing the wheel is not a good practice. Thus, we will use the libraries below for the explained reasons.

3.3.1 JavaFX

As of JavaFX 2.2 and Java SE 7 update 6, JavaFX is part of standard Java libraries. Anyway, it worth to mention again that we will use JavaFX for our GUI in the client. We chose JavaFX because it has better graphics and easier to use. JavaFX also contributes to the MVC pattern because views are always in a separate files .fxml, in particular the XML files.

3.3.2 GluonHQ Maps

GluonHQ Maps v1.0.1 is a wrapper library for google maps container in the client. Instead of using web browser and dealing with javascript, we will use this library for map actions. It will contribute to our project a lot because we have lots of map actions.

3.3.3 RxJava

As the Happy Real Estate consists of client and server, it will have lots of asynchronous actions. To handle such situations, we made our design utilizing RxJava v1.2.2. RxJava is a library for reactive programming which is a developed version of Observable/Iterable/EventBus patterns. This library will be used in both client and server. It eases nearly all the cases like client-server communication, communication between client's classes, and the communication between server's classes.

3.3.4 RxJavaFX

RxJavaFX v0.3.0 is a library for JavaFX events based on the design principles of reactive programming. Usage of this library will increase consistency on core design and help to do more with less code. This library will only be used in the client.

3.3.5 ControlsFX

ControlsFX v8.40.12 provides many pre-built controls that we will need to use such as tableview. This library will reduce our efforts on creating such views. This library will only be used in the client.

3.3.6 Google GSON

As the Happy Real Estate consists of client and server, object serialization and deserialization will be required because Java objects cannot be transmitted on network before serialized. JSON fits for such situations. GSON v2.8.0 will help us to deal with serialization/deserialization events, namely mapping.

3.3.7 MongoDB RxJava Driver

Since we have to deal with data, we chose MongoDB as our database. To ease the communication with this database, we needed to use a driver. MongoDB RxJava Driver v1.2.0 follows the reactive programming design principles. Thus, it will also increase the consistency of our code. This library will only be used in the server.

4. Other

4.1 Security

The case that all users' data is stored in the database and the access to these data is restricted with application's needs would enhance the security of their data. The code for login & register will be written considering the brute-force attacks. Users' passwords will be encrypted via MD5 and sha1 in order to ensure no one is able to access plain passwords. Also, the server provider we are considering has its own security so we do not have to deal with operating system related security issues, it is already secure enough. Lastly, we realized the necessity for the encryption of intermittent data between client and server. Accordingly, we decided to encrypt these outgoing data before transmitted. And the key for decryption of these data will be initialized at application starting by the server and the server sends this key to the client and the client will use this key for future request to encrypt data and the server uses this key to decrypt data. Likely, the server encrypts request and the client decrypts via this key.

4.2 Documentation

The documentation of Happy Real Estate source code will be generated automatically by IntelliJ. Also, the documentation will be published to Github repository wiki of *Happy Real Estate* repository.

4.3 Testing

The GUI will be tested manually. If the time is enough for the end of initial development of *Happy Real Estate* or if we decide to continue development in future, we may write unit and integration tests especially for the parts like GUI, high-pressure cases for client-server communication, and map traversing.

5. Conclusion

5.1 Task Assignments

- Özlem will work on the classes about house properties and types and she will be actively working on the repositories. In addition to data part, she will be responsible of Business objects about house in the domain.
- Mert will be responsible about the feature of the messaging and he will work the repositories, too. Additionally, he will be working on the business objects of the messaging in the domain
- Deniz will work on the user features of the application and he will have a part in the repositories. Moreover, he will be responsible of the business objects of user feature.
- Selim will work on the presentation part, specifically on the classes of favorites, filter, image upload, house announcement, own properties. In addition to these, he will be responsible of the client communication in the server.
- Burak will work on the presentation part. In particular, main, messages, new_house, and register packages. Additionally, he will be responsible of database in the server.

5.2 Discussions

This report gives brief demonstration of the program, however, this demonstration may change with time. In particular, as a group, we are not sure whether server should exist in the program. If server will be in the program, how can we store the database and server is not clear enough for us. Although we have unclear parts in the program, this report will be the main guidelines for this application.

5.3 Summary

Briefly, this report has a purpose to demonstrate the detailed design of the Happy Real Estate. In general, the features of this application are that online map which makes the perception of the location easier, messaging which ensures quick communication, house properties, user login.