Raees Aamir

Matriculation number: s1617910

Abstract

A document containing implementation details about Coinz. Coinz is a map based game where players have to collect cryptocurrency coins.

IMPLEMENTATION

Informatics Large Practical

Table of Contents

[Algorithms/Data Structures and Architectural Details 3](#_Toc532160032)

[About 4](#_Toc532160033)

[Share 4](#_Toc532160034)

[Menu 4](#_Toc532160035)

[Login 4](#_Toc532160036)

[Registration 5](#_Toc532160037)

[Game 5](#_Toc532160038)

[Downloading & Parsing 5](#_Toc532160039)

[Persistent Storage 6](#_Toc532160040)

[Detection of Coins 6](#_Toc532160041)

[Wallet and Bank 7](#_Toc532160042)

[Unrealized parts of my design 8](#_Toc532160043)

[Additional features that were not described in my design 9](#_Toc532160044)

[Screenshots 10](#_Toc532160045)

[Acknowledgements 17](#_Toc532160046)

[References 18](#_Toc532160047)

# Algorithms/Data Structures and Architectural Details

Coinz makes use of the MVC paradigm. MVC is an acronym for model-view-controller; it is an architectural pattern commonly used for developing user interfaces that divides the application into three connected parts. [2]



The views are XML layout files, the controllers are the activity class files named with a ‘controller’ suffix and the models are plain java classes that represent features of the game.

The code makes use of fragments in all the subsections apart from login and registration. The reason why I decided to use fragments is because it allows for easy navigation between sections of the app. A bottom navigation bar is used to hold the sections that you can transition to.

Any subsections of the app that require an internet connection will show the user an error if there is no internet connection available. They get presented with a dialog telling them that no connection is available and that they should come back later.

## About

No particular interesting algorithms and data structures have been used in the implementation of the About section. The about text is a constant that is loaded into a text view in the about view of the AboutFragment.

## Share

The share system uses the Facebook Sharing API to let players share their achievements with their friends on social media.

There are no interesting data structures in this section. However, the installation of the Facebook Sharing API requires the use of a security algorithm so I will go into some detail about the installation procedure. The first involved adding the dependencies to the module Gradle build file:

implementation **'com.facebook.android:facebook-share:[4,5)'**

The second step is adding a Facebook App ID to the Android Manifest. [1]

The third step requires a unique key hash. I did this by generating a SHA64 key for the app’s signature. Java has a very useful inbuilt security framework which lets me do this without having to manually implement the SHA64 key hashing algorithm.

**final** MessageDigest md = MessageDigest.*getInstance*(**"SHA"**);  
md.update(signature.toByteArray());  
**final** String hashKey = **new** String(Base64.*encode*(md.digest(), 0));  
Log.*i*(**"AppLog"**, **"key:"** + hashKey + **"="**);

The final step involved creating an empty activity called FacebookActivity. This is needed for the Facebook share button to segue to Facebook.

## Menu

Menu items are defined in an Enum called MenuItem. MenuItem takes a class reference to the controller that manages the navigation for the particular subsection of the app. Menu items are loaded into a list view that is referenced in MenuFragment. An ImmutableSet is used to make all the items from the Enum iterable. In particular, an ImmutableSet is used because we don’t want to let the developer add/remove/edit/delete MenuItem instances outside of the MenuItem Enum.

## Login

Both the login and registration controllers are subclasses of an abstract authentication controller. The authentication controller handles the UI code that is common to both the login and registration, e.g. the progress spinner, the error message alerts and the on click listeners.

Before the Firebase login is called, appropriate checks are done to make sure the email address and the password are valid. The criteria for determining a valid email address is whether the email contains an “@” and the criteria for determining a valid password is whether the password has atleast 6 characters. If the checks pass then the Firebase method signInWithEmailAndPassword is called.

(Note that firebaseAuth in the below code is an instance of FirebaseAuth)

**firebaseAuth**.signInWithEmailAndPassword(email, password).addOnCompleteListener((@NonNull Task<AuthResult> task) -> {  
**...**  
});

## Registration

The registration form has four fields: display name, email, password and confirm passwords. Before the appropriate firebaseAuth method is called there are checks to make sure the password and confirm password fields are equal. If they are then the Firebase registration is carried out with the code below:

**firebaseAuth**.createUserWithEmailAndPassword(email, password).addOnCompleteListener((@NonNull Task<AuthResult> task) -> {

...

}

## Game

The game section of the app is the most complex so I will divide the explanation into three sections.

### Downloading & Parsing

#### Downloading

The dependency OkHttp is used to download files from the internet. I created an abstract class called DownloadFileTask which is a subclass of DownloadFileTask<String, Void, T>. T is a generic type which is defined in the class definition as anything that is an object. In Java that means everything apart from primitive types. DownloadFileTask has two methods:

**protected abstract** T readStream(String inputStream);

**protected** T doInBackground(String... params);

The method doInBackground downloads the file from the internet. The URL to the file is found at index 0 of params.

Request.Builder builder = **new** Request.Builder();  
builder.url(params[0]);  
Request request = builder.build();  
  
**try** {  
 Response response = **client**.newCall(request).execute();  
 **return** readStream(Objects.*requireNonNull*(response.body()).string());  
} **catch** (Exception e) {  
 e.printStackTrace();  
}

readStream is returned callback when the download is completed.

#### Parsing

A dependency called Gson is used to parse the JSON string into a Java object. A anonymous class of DownloadFileTask is created and it returns a one liner which handles the JSON decoding:

**return new** Gson().fromJson(json, FeatureCollection.**class**);

### Persistent Storage

After parsing the feature collection JSON data, it is stored in the app’s shared preferences and an instance of a Java class called FeatureCollection is used to manipulate the data. When a player collects a coin from the map, the coin is removed from today’s instance of the player’s feature collection.

Feature[] features = **featureCollection**.getFeatures();  
features[indexOfFeature] = **null**;

The FeatureCollection Java class is serialized back into JSON data using the Gson dependency.

String jSONDocument = gson.toJson(**featureCollection**);

String key = **mUser**.getUid() + **"/"** + dateFormatted;  
preferences.edit().putString(key, jSONDocument).commit();

The persistent storage of the player’s coins in the wallet is discussed further down in the Wallet section.

### Detection of Coins

Every time a player moves a check is carried out to see if the player is within a 25m radius of a coin or a group of coins.

**if** (featureLatLng.distanceTo(playerLatLng) <= 25) {  
 featureMap.put(i, feature);  
}

where featureLatLng is the latitude and longitude of the marker and playerLatLng is the latitude and longitude of the player’s current location.

If the player is within a 25m radius of a coin or group of coins, then the coins are added to the player’s wallet. If the player has more than 25 coins in their wallet, then the coins are added to a spare change wallet that can only be used for trading.

## Wallet and Bank

# Unrealized parts of my design

# Additional features that were not described in my design

# Screenshots

|  |  |
| --- | --- |
| Subsection | Screenshot |
| Splash screen |  |
| Login |  |
| More Login |  |
| Registration |  |
| About |  |
| Share |  |
| Play |  |
| Wallet |  |
| Menu |  |
| Messaging  (the screenshot on the right is a chat with a person called test) |  |
| More Messaging |  |
|  |  |

# Acknowledgements

# References

[1] <https://developers.facebook.com/docs/android/getting-started#app_id>

[2] https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller