Raees Aamir

Matriculation number: s1617910

Abstract

A document containing design decisions about Coinz. A map based game where players have to collect cryptocurrency coins. Features of the app, choice of programming language and the timescale are all included.

DESIGN

Informatics Large Practical

Table of Contents

[Software Development Timetable 2](#_Toc525843402)

[Fundamental Features to Coinz 3](#_Toc525843403)

[Resolution to ambiguities 3](#_Toc525843404)

[Logo 3](#_Toc525843405)

[Welcome 3](#_Toc525843406)

[Menu 3](#_Toc525843407)

[Additional features of Coinz 4](#_Toc525843408)

[Programming Language Choice 5](#_Toc525843409)

[Decision made: Java 5](#_Toc525843410)

[Justification: *see below for discussion on why* 5](#_Toc525843411)

[References 7](#_Toc525843412)

# Software Development Timetable

|  |  |
| --- | --- |
| **Week** | **Objectives** |
| 2 | * Setup private repository for Coinz development project and include Professor Gilmore as a contributor. Complete by Tuesday. * Have the word document for the design created and added to the Git repository by Tuesday. * Decide on programming language choice; Java vs Kotlin by Thursday. * List all objectives in the software development timetable by Saturday. * Start watching the relevant Firebase tutorial videos produced by TVAC Studios on Friday. |
| 3 | * Complete list of fundamental features to the Coinz app by Tuesday. * Complete list of four additional features to add to the Coinz app by Wednesday. * Start watching the video tutorials on Mockito and JUnit testing from Udemy. [3] * Finish watching half of the relevant Firebase tutorial videos produced by TVAC Studios on Friday. [4] |
| 4 | * Start making paper designs of the MVC views by Monday. For example, the login view, the map view, the wallet view, the settings view, and the bank view. * Perform final spelling, grammar and content checks on design document and submit by Tuesday afternoon. * Finish the paper designs of the MVC views by Thursday. * Finish watching all the relevant Firebase tutorial videos produced by TVAC Studio by Friday. [4] |
| 5 | * Add a commons dependency like Google Guava or Apache Commons to the Gradle project by Monday. * Make a document for listing the algorithms and data structures used in developing the MVC models of entities by Monday. * Start programming the MVC models of entities involved in the gameplay by Monday. For example, the player model, the coin model, the map model, the wallet model, and the bank model. Whilst programming the models, add descriptions of algorithms and data structures used. * Finish implementation of MVC models of entities involved in the gameplay by Wednesday. |
| 6 | * Start translating all paper designs of MVC views to Android XML activity views by Monday. * Finish translating all paper designs of views to Android XML activity views by Friday. * Finish watching the videos from the Udemy Mockito and JUnit tutorials. [3] |
| 7 | * Make a document for listing the algorithms and data structures used in developing the Firebase backend server by Monday. * Start writing the backend Firebase server by Monday. Whilst programming the backend Firebase server, add descriptions of algorithms and data structures used to the document. * Start writing automation testing scripts for the MVC models and the MVC views by Thursday. * Make a document for listing the algorithms and data structures used in developing the MVC controllers by Wednesday. * Start implementing the MVC controllers by Wednesday. Whilst programming the controllers, add descriptions of algorithms and data structures used to the document. |
| 8 | * Finish writing automation testing scripts for the MVC models and the MVC views by Wednesday. * Finish programming the backend Firebase server by Wednesday. * Have the word document for the implementation report created and added to the repository by Friday. * Put all documents listing algorithm and data structure descriptions into the implementation report by Friday. |
| 9 | * Finish implementing the MVC controllers by Wednesday. * Add screenshots of the Coinz gameplay to the implementation report by Thursday. * Begin writing unit testing scripts for each MVC controller by Friday * Begin writing unit testing scripts for the backend Firebase server by Thursday. |
| 10 | * Finish writing unit testing scripts for each MVC controller by Friday. * Finish writing unit testing scripts for the backend Firebase server by Tuesday. * Start documenting the unit testing scripts for the backend Firebase server by Wednesday. |
| 11 | * Finish documenting the unit testing scripts for the backend Firebase server by Monday. * Start documenting each MVC controller by Tuesday. * Start documenting each MVC model by Wednesday. * Finish documenting the unit testing scripts for each MVC controller by Friday. |
| 12 | * Finish documenting each MVC controller by Monday. * Finish documenting each MVC model by Tuesday. * Start checking spelling, grammar and writing style of documentation and implementation report by Wednesday. |
| 13 | * Perform final checks of spelling, grammar and writing style on documentation and implementation report by Monday. * Perform final checks of the codebase and submit the project by Tuesday afternoon. |

# Fundamental Features to Coinz

## Resolution to ambiguities

* Coins are converted to GOLD at the time they’re paid into the bank. So a player could deposit 5 PENYs and 2 SHILLs and they would be changed into GOLD according the exchange rate. Once they are converted into GOLD, they cannot be converted back into PENYs and SHILs.
* If the coins aren’t deposited into the bank by the end of the day, then they expire. However, players can trade their coins to other players that don’t have enough coins.
* …

## Logo

* Since the application is based on collecting cryptocurrency coins around a map, the logo will be different coloured coins pinpointed on a map.

## Welcome

* On starting the application, a welcome splash screen will appear with the logo. Once the splash screen is complete, the application will redirect to the menu.

## Menu

* The menu will include the logo at the header of the view with an appropriate background to create an aesthetic look.
* In the body of the view (below the header and above the footer), there will be three buttons: Play, View Leaderboard and Exit.
* The play button will redirect to a login view if the user is not logged into the game, otherwise it will redirect to a map view where the player can play the game.
* If the player is logged in, at the right of the header there will be a door to signify the option to logout of the game.
* The view leaderboard button will redirect to a view showing the top x players and their total number of GOLD.

# Additional features of Coinz

# Programming Language Choice

## Decision made: Java

## Justification: *see below for discussion on why*

In this section, I’m going to discuss the advantages and disadvantages of choosing Kotlin over Java or by choosing Java over Kotlin. I will weigh up the advantages and disadvantages of each choice and I will reach a conclusion based on which language maximizes benefits and minimizes costs.

As of the release of Android Studio 3.0 in October last year (\*REF), Google added Kotlin as an officially supported programming language for the Android SDK. When developing a specification for a project, this might cause analysts to ponder on which choice of language would be the best for a new Android app. There are many things to consider such as the supported libraries, the Java to Kotlin translator and the syntax and semantics. A conservative developer might want to stay with Java until Kotlin matures; whilst a more liberal developer would be open to going with Kotlin in its youth.

Since it is possible to use Java libraries in Kotlin and vice versa, simply looking at which language has better libraries is not a valid comparison. A more important factor is thinking about how *effective* the Java to Kotlin translator is. It seems to be fine at translating toy Java code to toy Kotlin code, but there are situations where it can struggle translating enterprise level code to Kotlin. For example, the Stackoverflow question [1] is about a developer using a Java CallbackWrapper in Kotlin that works fine; however, when the Java code is translated to Kotlin the compiler throws cryptic errors. The developer who answered the question noticed that the problem was caused by the fact Kotlin “doesn’t allow using lambdas to implement Kotlin functional interfaces, so your lambda isn’t going to work (*it does allow implementing Java functional interfaces, because Java doesn’t have proper function types*)” Unfortunately the Android Studio translator assumed that the Kotlin lambdas would work despite that Kotlin has real function types unlike Java. If I were to improve the Android Studio translator, I would add a heuristic that searches the entire code bases for usages of Kotlin lambdas to implement Java functional interfaces because it won’t work in Kotlin.

Ones of Kotlin’s biggest advantage over Java is its syntax; lots of developers complain about Java verbosity and this is absolutely not a problem in Kotlin because the signal to noise ratio is much lower. For example, in Java it would take multiple lines to populate an array with content.

String[] names = **new** String[3];  
names[0] = **"Raees"**;  
names[1] = **"John"**;  
names[2] = **"Elliott"**;

In Kotlin the same thing can be done using one line:

**val** names = *arrayOf*(**"Raees"**, **"John"**, **"Elliott"**)

Whether this is a good thing depends on who you speak to. Personally, I don’t like writing lots of noisy code but there are developers who would dispute whether the extra code is noise. Some would say it’s signal because the String[] in front of the declaration tells you that you are declaring an array of strings. In Python this is a common concern because sometimes it’s difficult to determine the type of variable without calling the type method to explicitly check. Fortunately, in Kotlin there’s a way around this, we can explicitly tell the developer the type by changing the array declaration to:

**val** names: Array<String> = *arrayOf*(**"Raees"**, **"John"**, **"Elliott"**)

So far there’s a fair tie between Java and Kotlin; Kotlin wins in the syntax department and Java wins in the compatibility department. But we need to make a decision now because if we choose the wrong language, it could lead to further problems down the line. For example, even though Google has an official page on migrating Java projects to Kotlin, there have been disputes about whether it’s an effective way to move to Kotlin because of the problems it introduces. For example, in the migration article [2], Android Developer Paulina Sadowska talks about how following Google’s guidance will introduce problems if the Java project uses certain libraries. She talks about how using Java with Lombok to generate getters and setters causes errors in Kotlin because “when the Kotlin compiler runs it uses javac as well but with no annotation processing” Therefore, if we want to use Kotlin we must decide now because we will need to use different libraries than if we go with Java.

In conclusion, I have decided to program the Coinz app in Java. Java wins when we consider code migration and translation problems. Kotlin wins when we consider syntax. However, by using common dependencies like Google Guava we can reap the benefits of less noisy data structure instantiation by using its utility classes. Furthermore, I won’t be completing the Coinz app, I will just be making a prototype to pass onto a development team. There’s a significant time cost if 5 or 10 people have to spend a week learning the intricacies of Kotlin to understand good programming principles. We could just hire Kotlin developers but given the low supply and high demand, management will have to pay a higher salary.

# 

# References

[1] – Mgebrishvilli, J. (2017). Converted Java class file to Kotlin makes compilation error. Stackoverflow. [online], p.1. Available at: <https://stackoverflow.com/questions/43235423/converted-java-class-file-to-kotlin-makes-compilation-error> [Accessed 26 Sept. 2018]

[2] – Sadowska, P. (2018). How to f\*\*\* up Java to Kotlin migration in your existing Android app. *Usejournal.* [online], p. 1. Available at: <https://blog.usejournal.com/how-to-fuck-up-java-to-kotlin-migration-in-your-existing-android-app-325b57c9ddbb> [Accessed 26 Sept. 2018]

[3] – in28Minutes, O. (2018). Mockito tutorial with JUnit examples. *Udemy.* [online], p.1. Available at: https://www.udemy.com/mockito-tutorial-with-junit-examples/ [Accessed 29 Sept. 2018]

[4] – TVAC Studio. (2018). Android Studio – Firebase Backend Tutorial Full Course. *Youtube.* [online], p.1. Available at: <https://www.youtube.com/playlist?list=PLGCjwl1RrtcTXrWuRTa59RyRmQ4OedWrt> [Accessed 28 Sept. 2018]

Game summary:

* Players collect coins scattered around the central area
* App is considered a prototype that a team can further develop
* Location of coins are specified on map, they are collected by getting near to their location. (What if the player’s inventory is full? Drop/trade/bank)
* Each map has 50 coins
* A new map is released every day. (Do we generate the map or is the geo-json data given to us?)
* Four different coins: Penny (PENY), Dollar (DOLR), Shil (SHIL), Quid (QUID).
* Fluctuate relative to the value another currency just like real currencies do.
* That other currency is GOLD.
* What algorithm are we going to use to handle coin fluctuations ? Is it going to be a simulation or based on real data?
* Every coin on a map has a value > 0 but < 10 of its currency (relative to GOLD) E.g 5.72384765123 DOLR
* What happens to coins from an older map?

Justification and discussion on:

*Game design*

* Extra features
* Features

Define features:

* Answer ambiguities from underspecification [Map, coins, coin limit, daily map releases, types of coins, exchange rates, coin value, coins from older maps]
* Logo
* Welcome
* Menu
* Login/Logout

Define extra features:

* *Coin trading*
* *Leaderboard*
* Distance recording. The person who walks the shortest distance to obtain the coin for each pinpoint on the map receives a bonus.
* Difficulty level: normal and hard. Hard is timed. Time is based on projected walking distance time \* 0.8. Rewards TBD
* *Share on social media*
* *Backend game server. Firebase authentication, cloud functions programmed in Kotlin. Player updating and trading*