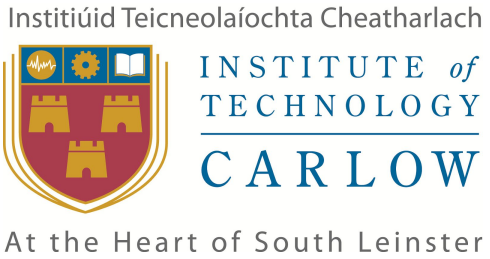



# Computer Games Development CW208
# Technical Design Document
# Year IV

DION BUCKLEY
C00220868
May 03rd 2020


# Faculty of Science

# Department of Computing and Networking

# Open-Book and Remote Assessment Cover Page

**Student Name: Dion Buckley**

**Student Number: C00220868**

**Lecturer Name: Philip Bourke**

**Module: Project II (Games)**

**Stage/Year: Final**

**Date: 02-05-2020**

**Declaration**

This examination/assessment will be submitted using GitHub/Google Drive as the online submission tool. By submitting my examination/assessment to GitHub/Google Drive, I am declaring that this examination/assessment is my own work. I understand that I may be required to orally defend any of my answers, to the lecturer, at a given time after the examination/assessment has been completed, as outlined in the student regulations.

## Contents

**Architecture** **3**

UML **3**

Class Diagram: (most basic diagram showing key classes and relationships) 4

**Features** **4**

Feature 1: Video Player - http://jira.itcarlow.ie:8080/browse/ARGO-105 4

Feature 2: Badges - http://jira.itcarlow.ie:8080/browse/ARGO-106 5

Feature 3: Quiz - http://jira.itcarlow.ie:8080/browse/ARGO-106 5

Feature 4: Stack Navigator - http://jira.itcarlow.ie:8080/browse/ARGO-106 6

CRC Cards **6**

Class Name : App 6

Class Name : SplashScreen 6

Class Name : HomeScreen 7

Class Name : VideoScreen 8

Class Name : ExpoVideoPlayer 8

Class Name : CoinScreen 9

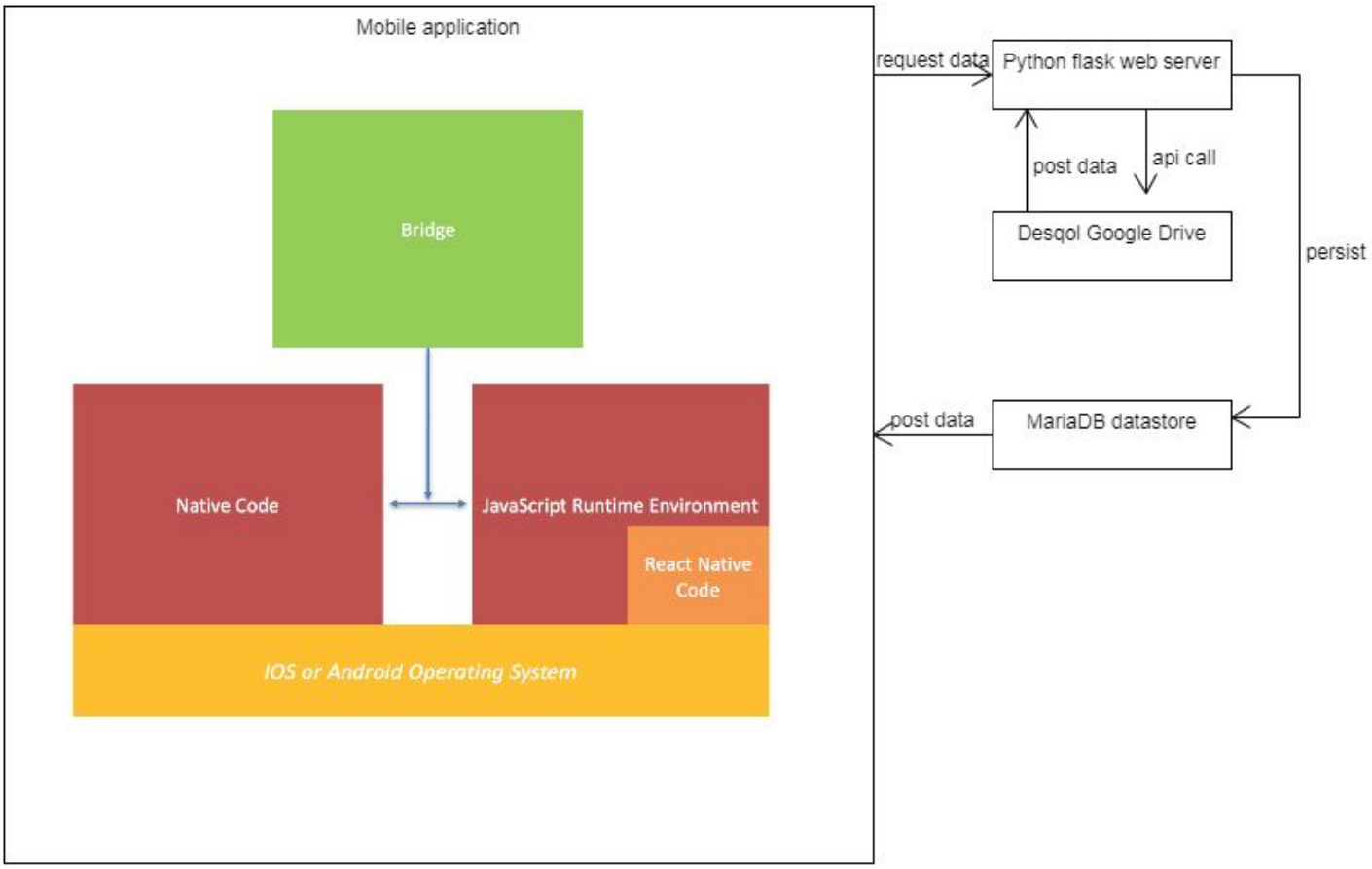
Class Name : Coin 9

Class Name : QuizScreen 9

Class Name : Quiz 10

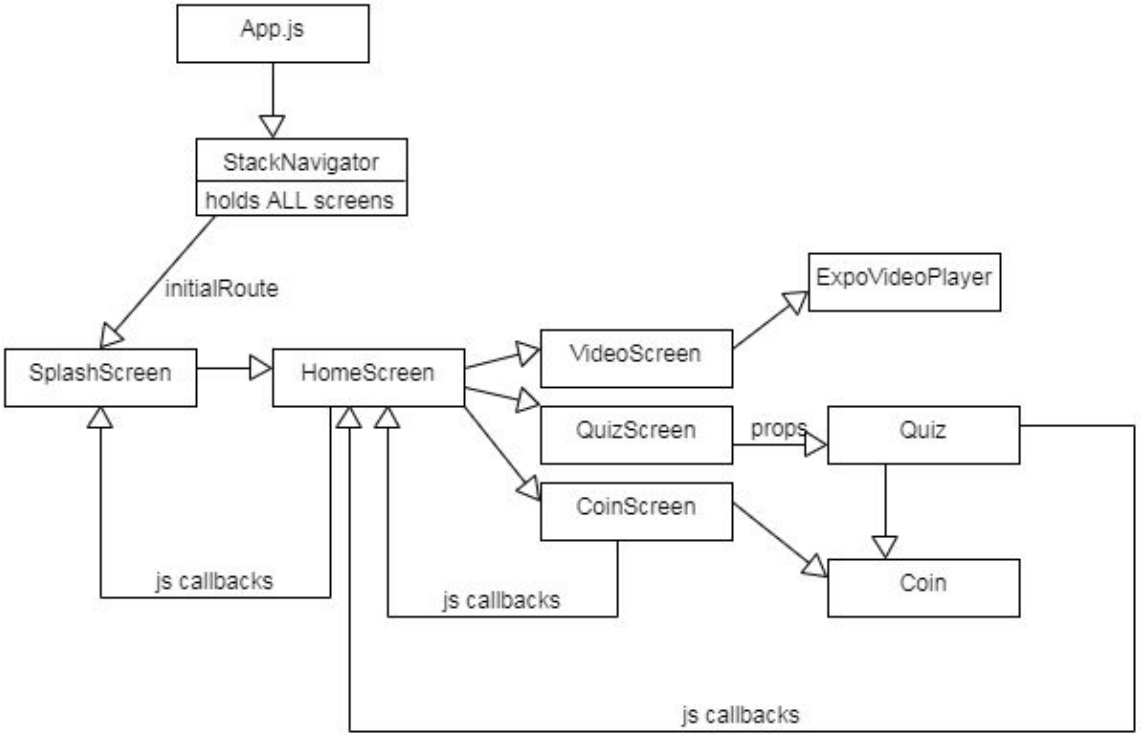
**Architecture**

Specific Subsystems and Engine Architecture



**UML**

***Class Diagram: (most basic diagram showing key classes and relationships)***




**Features**

***Feature 1: Video Player - [http://jira.itcarlow.ie:8080/browse/ARGO-105](http://jira.itcarlow.ie:8080/browse/ARGO-105)***

Tasks:

1. Find appropriate video library
2. Test if we can load the videos from the web or are they small enough to store locally
3. The loading, playing, pausing, stopping and unloading of the files requires extensive knowledge and use of a**synchronous javascript** functions and keywords and object states
4. Ensure unload of all asynchronous processes upon unmounting this video component to **avoid memory leak**
5. Wire everything up to use **native controls**
6. Create a video **loader** which loads all videos and displays appropriately
7. Create thumbnails using a **custom image button** and display as a React **ScrollView** to allow fluid scrolling through video options

***Feature 2: Badges - [http://jira.itcarlow.ie:8080/browse/ARGO-106](http://jira.itcarlow.ie:8080/browse/ARGO-106)***

Tasks:

1. Create **animated** coin sprite sheet
2. Utilize this animated coin component where the user achieves a right answer on the quiz
3. Show it **spinning** or shining with the **score** as the **number** of coins earned
4. Show at the end of quiz total coins earned and suggest to check out area to spend them
5. Build screen for **viewing** and **spending** collected coins
6. **Persist** the coins frequently and upon app inactive state, restoring them upon re-open or boot or app
7. Add **boosters** to be bought and implement that with quiz and demo real world purpose
8. For **real world purpose** use the **webview** component I build previously to display the site where the product for which the approved discount is pending

***Feature 3: Quiz - [http://jira.itcarlow.ie:8080/browse/ARGO-106](http://jira.itcarlow.ie:8080/browse/ARGO-106)***

Tasks:

1. Attempt to load quiz from React.js to Native in **WebView**
2. Realise this should be ported **locally**
3. Sift through the **complex structure** of last years quiz, **re-use** any **shared** code **scrapping** all **unnecessary** code and **rewrite huge chunks** of the quiz for native
4. From here build on **my own** gamified elements; **sound** effects, **coins**, hot **streaks** with fuel **booster**, **better particle system** AND whole new look and feel for entire quiz, including new **animated radio buttons**
5. Clearly mark which parts of this component were shared and which parts were my own through detailed comments
6. Add **extra back handler** separate to the navigator (mentioned below) so that if the user tries to exit the quiz we get an **alert warning** us that we will lose coins
7. Create custom loading bar so that while we are **waiting for the data from the backend** we can display a **dynamic loading** circle rather than nothing or just some static text (also used while waiting for WebView to load as mentioned above)

***Feature 4: Stack Navigator - [http://jira.itcarlow.ie:8080/browse/ARGO-106](http://jira.itcarlow.ie:8080/browse/ARGO-106)***

Tasks:

1. Create a stack navigator **as an App Container** to hold all of the app screens
2. Set this up with **initial route** and **header format options**
3. On each screen point to the next desired screen using **navigation options**
4. This is setup as a stack so that when the user presses the **Back** button on their mobile device this will **pop the stack** and the **last routed screen** displays again
5. Learn how using this approach has specific way of **passing and accessing props different** than **regular react** components
6. Similarly needed to **pass data back up** through requires learning about **javascript callbacks**
7. Create **custom fonts** through **async loading** on initial route page so that font can be used through **entire app** (since each routed page while not rendered will not be unmounted until popped either)

**CRC Cards**

| ***Class Name : App*** | |
|---|---|
| **Subclasses :** | |
| **Superclasses :** React.Component | |
| **Responsibilities** | **Collaborators** |
| All react application need to have a base App.js so standard is to have an App class in here | StackNaviagator.jsx (this file has no class, simply creates and exports the app stack - with all screens the initial route screen and stylings) |
| Manages creation of Stack Navigator (as an App Container) | |

| ***Class Name : SplashScreen*** | |
|---|---|
| **Subclasses :** | |
| **Superclasses :** React.Component | |
| **Responsibilities** | **Collaborators** |
| Load font and splash sound | ImageButton.jsx (component) |

| | |
|---|---|
| Retrieve and parse (without parse we have a bunch of appended 0s from the string version) coins from persisted async storage | expo-font package |
| Ensure calling back of streakKeeper and coins from other screens to ensure that state is set with correct values upon returning to this screen and going back again | AppState, AsyncStorage + standard react-native imports (StatusBar, StyleSheet, View, Text, Image) |
| Play splash sound upon exit towards home screen | expo-av for Audio |

| ***Class Name : HomeScreen*** | |
|---|---|
| **Subclasses :** | |
| **Superclasses : React.Component** | |
| **Responsibilities** | **Collaborators** |
| load coins and streak keeper as props from splash | Button from react-native-elements (much more style options than regular react-native Button) |
| Ensure calling back of streakKeeper and coins from other screens to ensure that state is set with correct values upon returning to this screen and going back again | standard react-native imports (StatusBar, StyleSheet, View, Text, Image) |
| For the callbacks need a separation of responsibility between callback to set the state of coins (for coin screen callback) AND state set with addition (for quiz callback) | |
| This is main landing screen so have the warm mascot along with welcoming and then send the app to either of the three main areas through buttons | |

| *Class Name : VideoScreen* | |
|---|---|
| **Subclasses :** | |
| **Superclasses : React.Component** | |
| **Responsibilities** | **Collaborators** |
| display list of video options separated into two sections, the animated intro type learning and the cross contamination examples | ImageButton.jsx (component) |
| Go to correct video screen then to load that video into the expo player later | standard react-native imports (StatusBar, StyleSheet, View, Text, Image) Plus ScrollView |
| | ExpoVideoPlayer.jsx (component) |

| *Class Name : ExpoVideoPlayer* | |
|---|---|
| **Subclasses :** | |
| **Superclasses : React.Component** | |
| **Responsibilities** | **Collaborators** |
| load all video files here and depending on which screen gets called by the VideoScreen we pass this file to expo-av Video component as a reference | ImageButton.jsx (component) |
| Async loading playing and unloading, Dimensions allows us to get the size of screen and fit video to that size screen | standard react-native imports (StatusBar, StyleSheet, View) Plus Dimensions |
| Extra back handler here to stop the async playing (I wrote this before I realised I could have used ComponentWillUnmount for same purpose actually) | react-navigation-backhandler |

| *Class Name : CoinScreen* | |
|---|---|
| **Subclasses :** | |
| **Superclasses : React.Component** | |
| **Responsibilities** | **Collaborators** |
| set state to navigation based props and send back data to parent callbacks | expo-av for Audio |
| load sound as mentioned before | standard react-native imports (Image, StyleSheet, View, Text) |
| managed spending of coins | Coin.jsx (component), MyWebView (component) |
| Load GIP stick page through webview to demo potential real world impact of coins | Button from react-native-elements (much more style options than regular react-native Button) |

| *Class Name : Coin* | |
|---|---|
| **Subclasses :** | |
| **Superclasses : React.Component** | |
| **Responsibilities** | **Collaborators** |
| set config for spritesheet; fps, animation type, looping, reset | rn-sprite-sheet |
| handle playback reference | standard react-native imports |
| render animation using columns  rows width and order of display | |

| *Class Name : QuizScreen* | |
|---|---|
| **Subclasses :** | |
| **Superclasses : React.Component** | |
| **Responsibilities** | **Collaborators** |

| | |
|---|---|
| Show loading indicator until quiz is ready after starting the quiz from quiz splash | Quiz.jsx (component) |
| handle audio as before (but much more this time so **proper unloading** is important) | standard react-native imports plus ActivityIndicator |
| callbacks for parent | expo-av for Audio |
| call the api from the server, splitting up the content( including questions, answers, id and correct answer) appropriately to serve later | ImageButton.jsx (component) |

| ***Class Name : Quiz*** | |
|---|---|
| **Subclasses :** | |
| **Superclasses : React.Component** | |
| **Responsibilities** | **Collaborators** |
| a lot of state to set in beginning including loading both hot streak images to be swapped in and out later | Coin.jsx (component) |
| handle audio as before (but much more this time so **proper unloading** is important) | standard react-native imports plus Alert and BackHandler |
| time the quiz so we can score and move through quiz | expo-av for Audio |
| Alert user when trying to go back out of quiz that they will lose progress if continue | ImageButton.jsx (component) |
| handle user answers and react accordingly | { Emitter } from 'react-native-particles' |
| shuffle incoming question on setup | RadioForm from 'react-native-simple-radio-button' |
| Display results at end of quiz | |
| check how many lines of text on screen the questions are to help stabilize the answer options vertically on page | |