Architecture Document:

**1: Purpose**

This document outlines the overall architecturally significant requirements for this project. Based upon the main NFR’s and the system assumptions/dependencies this document will justify the decisions made and the constraints that come with them. It will also show the overall system view in many forms including class diagrams, activity diagrams and deployment diagram. Basic architectural frameworks are also mentioned to aid in visualizing how this project will be implemented.

**2: Architectural goals and philosophies**

The application we wish to develop, as found in the vision document, is a text based trivia game. The difference of our game to others on the market already is it will be targeted towards specific topics. These would be what you call ‘nerdy’ topics and will try to encapsulate that specific demographic of people. We wish for this game to be multiplayer and contain current leaderboards against every other player in the game.

Since our goal is to create a multiplayer game we propose using a server running a MySQL database.The database will be sufficient for our proposed plan and also allow cross platform play.

We want to reach as many people as possible in our target demographic so opening this up to cross platform across iOS and Android will reach more people. Also by making the game cross platform this will bring more players online and allow for faster match making and bring more change to global leaderboards.

More specifically we will have two players play online in a choice quiz game against each other. As well as having a global leaderboard and a multitude of available questions. The game will determine a winner and then update their individual and worldwide leader board. The game also allows players to be a part of multiple games simultaneously, allowing players to start new games or continue existing ones.

Additionally, users will be able to play in a single player version of this game.

Based on the overall goals set out in the vision document the following are the high priority NFR’s. Usability, Reliability, Performance and Maintainability.

**Usability** is the most important NFR as without a usable system all other NFR’s can’t even be considered. The game is intended to be run on mobile devices and have a UI catered to those users. The game must be intuitive and self-explanatory in all degrees of operation.

**Reliability** is important to be considered due to the game running off a server. The entire operation of the game requires a connection to the server to be reliable. Without a reliable connection, users will not be able to play any of the functionality of the application. (maybe add in redundancy if connection drops from user, will submit next time online)

**Performance** is still important in regards to this being a mobile application. The app must be tailored to mobile use and be quick to respond to inputs from the user. It must also have quick response times from the server as delays longer than 1-2 seconds is enough for mobile users to stop using the application. Mobile users expect everything to happen instantly.

**Maintainability** is important as the game must be able to be continued to be upgraded and bugs fixed without hindering the overall functionality of the game. Without being maintainable users could experience large downtimes. By focusing on making the game maintainable the server can be kept up for as much time as possible.

**3: Assumptions and Dependencies**

Assumptions:

Dependencies:

* Each user must have an internet connection in order to play a multi-player game. Although users do not need to be online the entire time they are using the application, a connection to the server must be made at some point to download new game data and to upload results from current games.
* We are depending quite heavily on the availability of the server. If the server is unavailable at the end of a turn, then multiplayer will not function as intended. A backup system is in place to upload game data once an internet connection is made. In saying this our server provider boasts a close to 100% up time.
* Team member’s availability is a dependency which can limit our progress in this project. Each team member has stated their week to week time availability in the team charter. Based on this we have been able to set reasonable goals and stretch goals that should be achievable in the given time.
* Our project has a vast amount of different technologies in play including, Unity, C#, PHP, MySQL, Facebook and Google interaction. We are depending on our team members to deliver on their promises to learn how to use and create services using these technologies. To limit risk, we will deliver Technical Competency Applications to prove we can use required technologies.

**4: Architecturally Significant Requirements**

Listed below are the architecturally Significant Functional Requirements for Let’s Quiz.

1. The game must be implemented with C# script.

2. The game must be cross platform

3. The game must receive questions and answers from a server

4. The game must update server with result of the game

5. The game must be able to generate leaderboards based on data in the server

6. The game must allow users to log in as a guest, Let’s Quiz, Facebook or Google account

7. The game must allow users to submit questions to the server

Necessary Requirements for Architecture Realization:

* A log in screen allowing users to register, sign in with Facebook or Google Play will be implemented. SDK’s will be used to implement the Facebook and Google Play services which will provide the application with necessary user data to log them in.
* Once a multiplayer game has been created a connection to the FTP server must be established. If no connection possible the device will try again later, notifying the user.
* Each time the game is loaded the app will download the latest question pool and store locally on the phone.
* An option for the user to submit questions to the server to be added to relevant question pools. Subsequently users will have options to vote on given in game questions, if a question reaches a certain number of down votes it is removed from the pool.
* All logic inside the application must be complete. The game must be able to determine correct answers, calculate scores, determine game winners and display the correct answers at the end of game.

**5: Decisions, constraints & justifications**

Decisions:

* We will use Unity as our development environment.
* We will allow users to log in using either a Let’s Quiz account or their existing Facebook or Google accounts. Or log in a as a guest.
* The game will be cross platform and be available on both Android and iOS.
* A server will host a MySQL database for storing all game data.

Constraints:

* An internet connection is required for many functions of the game including, log in, register, downloading questions, updating score to database, viewing leaderboards and submitting questions.
* All of our code will be done using C# and PHP.
* The database will be a MySQL database.
* The user interface will have different levels of authentication handled by separate SDK’s.

We are using Unity as our development environment as it is a program that all team members have used before and it is a suitable program for meeting the other decisions made. Unity allows us to create a cross platform application for Android and iOS. It is also an environment suitable for game development.

By allowing users to log in using Facebook and Google Play users can authenticate themselves quickly and using services they are familiar. Users may feel safer doing this rather than signing up directly with our server. A guest option is also available.

None of the architecture being used is specific for either Android or iOS so we are able to develop the same application for both devices. The result of which widens our audience when it comes to delivering and gaining a user base.

We do not require users to join a lobby in order to allow multiplayer functionality. Rather the user plays locally for most of the gameplay except when it comes to updating a current round at its beginning and end. Therefore, we can use a MySQL server that just gets updated with current game information as required. This saves significant resources and skill requirements to allow multiplayer functionality.

Since we are allowing multiplayer we bring in the constraint of requiring an internet connection from the user. However as stated above this internet connection does not need to be consistent but just once at the beginning and end of a round. We will also require functionality to store data that needs to be sent to the server if for some reason an internet connection is not available when required.

Inside of Unity we will use the language of C# to code our application. We need to specify a language to be used and we have chosen this off of the skills of the team members. PHP will be used to communicate commands from the user to the server for the same reason.

Similar to above we have chosen to use a MySQL server due to the fact that team members all have previous experience in implementing and it is an obvious choice for this kind of application.

We require authentication of all users entering into multiplayer. In order to handle this, we will implement both Facebook and Google Play SDK’s to assist in gathering user data. These are the two most popular services that allow authentication and are services that most users still use.

It is known that all team members have many other commitments over the course of this project. All of these constraints have been laid out in the Team Charter. Team members will be held to their promise of time commitment, if not then action will be taken.

**6: Architectural Mechanisms**

Play Game:

Two options from this button push. The user can either start a new game or select from an ongoing one. Both options require a connection to the internet.

Submit Question:

Users will be able to submit their own questions to become part of the normal game question rotation.

Multiplayer:

Refers to the multi-layer functionality allowing players to play against each other. Currently being implemented through a MySQL database.

User Login:

Login using either an email address or the Facebook/Google Play SDK.

GUI:

We will use Unity to design the GUI for display on mobile devices. Will follow standard design principles and not be too dissimilar to other applications of the same nature.

**7: Layers or Architectural framework**

The majority of the application will follow a mobile architecture pattern. Meaning it will be designed specifically for mobile. The GUI will be relevant and use touch input to move through the application. The application will also be optimized to run on a mobile device.

The application will use a server-oriented architecture model. This model describes application components connecting together through a communication protocol over a wireless network.

**8: Architectural Views**

## Architecturally significant use cases for realization.

### i) Use Case: Register

When the user

Wants to register a Let’s Quiz account, they must input user details and click register

So that the application creates an account for them

### ii) Use Case: Login

When the user

Wants to login, they must choose login option

So that that the application allows login and displays pregame screen

### iii) Use Case: Login Registered users-Sub-function End Goal: Login

When the user

Wants to play by logging in, they then press login

So that that the application opens to the login screen

### iv) Use Case: Login with Facebook-Sub-function End Goal: Login

When the user

Wants to play by logging in with Facebook, they then press Facebook login

So that that the application connects to the Facebook Authentication server and allows login

### v) Use Case: Login with Google Play Services-Sub-function End Goal: Login

When the user

Wants to play by logging in with Google Play Services, they then press Google Play Services login

So that that the application connects to the Google Play Services Authentication server and allows login

### vi) Use Case: Play as Guest-Sub-function End Goal: Login

When the user

Wants to play without logging in or first registering they press play as guest

So that the application opens to the pre-game screen

### vii) Use Case: End Goal: User Submit Question

When the user

Wants to submit a question they will press the submit question button

So that the application opens to the submit question scene

### viii) Use Case: End Goal: Start a Game

When the user

Wants to start a new game they will press the start new game button

So that the application will either start a new game or join an existing game.

### ix) Use Case: Continue an existing game-Sub-function End Goal: Start a Game

When the user

Wants to take their turn in a previously started game they press the games description

So that the application opens the correct game state

### x) Use Case: End Goal: Answer question

When the user

Wants to answer a question they select the correct answer

So that the game can check the answer for correctness

### xi) Use case: End Goal: Submit Score

When the system

Wants to submit score, the system connects to Let’s Quiz Server

So the application can send score data to data base for updating

### xii) Use Case: End Goal: Check Leader board

When the user

Wants to check the leader board scores they press the leader board button

So that the application connects to the Google Play Services server to display the leader board

Listed above are the 12 use cases required for the architecture to be realized. This covers all the main requirements of login, answering questions, leader boards and submitting questions and will use all of the proposed architecture of Unity, c#, PHP and MySQL.

Full use Case diagram showing all aspects of the architecture:



## Activity Diagrams

### Register



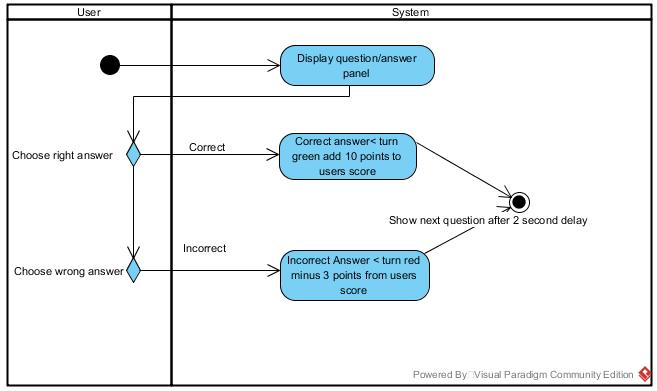
### Login



### Start a game



### Answer Question



### Play round



## Domain Model



## Logical View

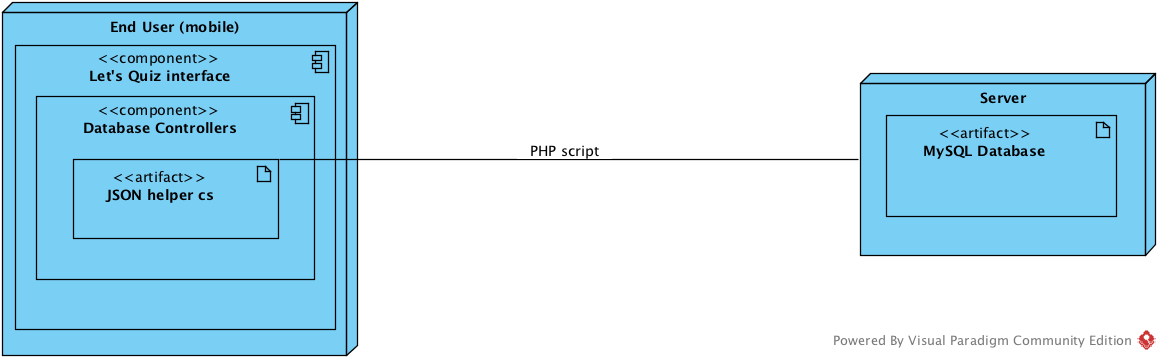
**Important Classes:**

The player class holds the name, email and login id for every user. A player can login, through the authentication services provided (Facebook, Google Play or Let’s Quiz account) or they can skip and play as a guest. Once authenticated a player can play a quiz game. The quiz game here is the multiplayer quiz game. The functionality for a single player game will be slightly different as users will not require an internet connection.

The quiz game becomes part of the games list held on the MySQL database. During gameplay the Quiz game will request a question from the question pool. The current game status after every question will be sent to the MySQL database. The question pool also has a list of answers to all questions which will be made available at the end of the game to both players. The quiz game will create a score class to keep track of player score and high score data. Once a game is finished it is removed from the current games list on the database.

Users will require an internet connection for all updates to the database. This happens at the beginning of a game and at the beginning and end of any question. It is required at the beginning of a question in order to retrieve the result of the other player’s answer. It is required at the end in order to update the database with the player’s response to the question.

## Deployment Diagram



**9: Implementation of CCRD:**

As stated in the Initial Requirement Model, the CCRD use case involves the high level items of Register, Login, Answer Question and Submit Score. These use cases cover all of the necessary architecture for the entire system to be realised.

**Register:** Requires input fields for necessary information as well as buttons for submitting the data. The application must have a connection to the database in order to submit the information and register the new user. If not connection is available this will not be possible and time out. Registering will submit the given information into the relevant fields in the player table on the MySQL database. A unique identifier will also be created at this point for the new user. Once this is complete the user is given feedback that is was successful and is logged in.

**Log in:** Input fields are given to the user to authenticate themselves with a username/email and password. This will require a connection to the database. Once a connection is made the given information is compared to the fields in the player table and if a match is found the player is logged in a feedback given. If not the player is notified that their supplied information is incorrect.

Once a player has logged in before they will automatically be logged in each time they open the app as their username and password is stored locally inside the app for quicker log in.

**Answer Question:**

All gameplay takes place inside the Game Controller. In order for a player to answer questions, firstly questions must be available for the player to answer.

When the player performs log in a connection to the database is established and all questions currently on the database are downloaded in a single JSON string. This string is stored inside the PlayerController, once this has been completed at least once offline gameplay is available.

When the player launches the game the QuestionController is called and serializes the JSON into useable QuestionData objects containing a question, 4 answers and whether they are correct or incorrect.

Before questions are shown they are randomised inside the questionPool array to ensure the player does not receive the same set of questions each time they PlayGame. The order of answers is also randomised.

Inside the Game scene the player is presented with a UI containing the questionText and the four answers. When a user selects an answer the Game Controller determines if the selected answer is correct and changes the colour of the button, green or red, to reflect the response.

At this point the game score is also updated for correct and incorrect responses.

**Submit Score:**

Submit Score is required to update the players total accumulated score over all games as well as to compare between their opponent. When a round or game is finished the score is updated to the players accumulated score, a check is also made to see if this is a new high score for the player.

The game controller then establishes a connection to the database and updates the game table with the scores. Once the player and opponent’s data is received a winner is determined and the mobile clients display the result.