FACULTY OF SCIENCE & TECHNOLOGY

BSc (Hons) Software Engineering
May 2016

Gamifying Childhood

Incentivising Children to Achieve Through Video Games

by

Michael Porter

Faculty of Science & Technology
Department of Computing and Informatics
Final Year Project

Abstract

It can be difficult for parents to motivate children to perform given chores or tasks in a timely manner. This project aims to create a system for parents to help encourage children in the form of a gamified task management mobile app. The app will reward children for performing a task using a popular video game based reward structure by offering 'Experience Points', which will, when collected, allow them to level up their character. This report will detail how I have designed and developed the system surrounding these goals.

Research will be performed into the study of motivation and rewards, particularly with children, and into what will be the best tools I can implement to benefit this.

The artefact has been created as a central server based REST API that receives and handles web service requests to return JSON data. This API is interfaced via an Android application which will display the received data in an aesthetically pleasing and informative format.

I have approached this project using agile and test driven development methodologies to create a well structured and reliable software artefact. Requirements were gathered using brainstorming skills and were refined using the MoSCoW method to create a list of features that I believed would motivate children to perform tasks.

Dissertation Declaration

I agree that, should the University wish to retain it for reference purposes, a copy of my dissertation may be held by Bournemouth University normally for a period of 3 academic years. I understand that once the retention period has expired my dissertation will be destroyed.

Confidentiality

I confirm that this dissertation does not contain information of a commercial or confidential nature or include personal information other than that which would normally be in the public domain unless the relevant permissions have been obtained. In particular any information which identifies a particular individual's religious or political beliefs, information relating to their health, ethnicity, criminal history or sex life has been anonymised unless permission has been granted for its publication from the person to whom it relates.

Copyright

The copyright for this dissertation remains with me.

Requests for Information

I agree that this dissertation may be made available as the result of a request for information under the Freedom of Information Act.

Signed:
Name: Michael Porter
Date:
Programme: BSc Software Engineering

Original Work Declaration

This dissertation and the project that it is based on are my own work, except where stated, in accordance with University regulations.

Signed:

Acknowledgements

I would like to acknowledge my project supervisor, Dr. Damien Fay, for his invaluable advice for the design and development of the project artefact. I also extend thanks to Alysa Thomas for her support and excellent proofreading skills during this dissertation.

Contents

List of Figures

Chapter 1

Introduction

1.1 Problem Definition

Parents can often find it difficult to motivate their children to perform chores in a timely manner. Many parents have attempted to implement a simple reward system to incentivise their children, but often fail to keep up with the rewards or maintain the tracking needed to make the positive reinforcement truly effective. I believe this is a problem that all parents face, and one we have all faced as children ourselves. This project aims to give children the tools they need to succeed in life by giving them the motivation to manage their tasks, as well as make the lives of their parents easier.

1.2 Proposed Artefact

I want to create a proof-of-concept mobile application and backend server application that will allow parents to assign tasks and rewards to their children in the form of 'quests' in a role-playing game, effectively gamifying chores. The quests can take the form of "Tidy your room" or "Complete your homework" and offer rewards of experience points (XP) and gold. The application will also allow the child to create an in-game character that they can customise and level up using aforementioned gold and XP. The app is designed to provide children with incentive to achieve more using positive reinforcement and gamification.

1.3 Clients

The key clients for this app will be the parent(s) or guardians who will input quests into the game and mark a quest as complete once they have inspected the work. The child will also be able to view quests, notify the parents that a quest requirement is ready for inspection, and purchase real life rewards from a reward shop. Parents will also be able to monitor their children's activities and progress, whilst approving or rejecting various activities that the child may encounter whilst using the application.

I envisage this app to be most suitable for children between the ages of 10 and 17, though it has not been designed exclusively for that age range.

1.4 Aims and Objectives

The aim of this project is to evaluate the usefulness of game-based rewards for children performing chores. To achieve this, I have identified the following objectives for the software artefact:

- Create a well designed mobile application, capable of tracking progress and offering rewards for chores and adhering to the Android Developer specifications
- Create a secure and robust public API to allow applications to connect to the the artefact's service
- Utilise appropriate technologies to securely create the two deliverables to a high quality
- Utilise test-driven development to provide automated testing, ensuring test coverage of over 80% for the API

1.5 Risks

In this project, I will be working with a variety of technologies that I have not previously used or am not very experienced with, such as Flask, Android and REST. This presents a likely risk that the learning curves of these tools may be steeper than expected. If this occurs, it is likely that initial time-frame estimates for the development will be inaccurate, as features involving these technologies may incur delays. The scale and importance of the above delays could, in themselves, lead to the project not entirely fulfilling the above objectives. I believe this makes the above risk high severity. In order to mitigate this, I have researched a variety of different learning materials that I will be able to access throughout the project to help me better understand these tools. Furthermore, I have sourced potential help from my placement company, where colleagues have agreed to provide guidance in areas they are experienced in.

As this is a task management app, the artefact risks becoming more of a hindrance then a help. There is a possibility that the app could become too intrusive to parents giving their children a task, as what would normally be a simple request could be delayed by the additional process of entering and tracking the task. If this was to occur, it would result in the app being less desirable to use. However, I believe that if I put a sufficient focus on usability within the app and attempt to design my use cases to have as little resistance as possible, I can avoid this risk. I can also mitigate this risk by offering Parent

users their own version of the app, which would allow them to circumvent any security features designed to stop a Child from giving themselves rewards. Furthermore, the research performed in this paper will allow us to identify the usefulness of this app and determine areas where it may become intrusive.

Chapter 2

Literature Review

For this project I will need to research three key areas: the psychological aspects of motivation, particularly in children; gamification; video game and mobile application development. Research will need to be made specifically into how I can not only encourage children to use the app, but how I can best ensure that the app is both fun - to encourage continued play - and beneficial to the child's motivation skills. Other important research areas will be what software development methodology will be most beneficial to me as a lone developer, and what methods of testing are best suited towards ensuring a high quality deliverable.

2.1 Gamification

2.1.1 Defining Gamification

A definition of gamification is offered by Deterding et al. (2011, p.6) as "The use of design elements for games in non-game contexts". A game design element in this case is referred to as the characteristics of a game that appear in most games, are readily associated with games and are found to play a significant role in games. With this definition, it is necessary to identify the game design elements that will be included in the application.

In this paper, Deterding also highlights the fact that Gamification refers specifically to using game elements in a non-game application, rather than using full-fledged games. Therefore, by this strict definition, it is important to draw the distinction between a game application built to incorporate real-life rewards and a motivational application built to incorporate game-based elements.

Huotari and Hamari (2012, p.5) have also provided an alternative definition for gamification, stating that it is "service packaging where a core service is enhanced by a rules-based service system that provides feedback and interaction mechanisms to the user with an aim to facilitate and support the users' overall value creation."

2.1.2 Rewards and Punishments

In an experiment by Filsecker and Hickey (2014), children were separated into two groups: a public recognition (PR) group where the children's 'badges' were recorded on a prominently placed leaderboard in a room where others can see, and a non-public recognition (NPR) group. The children were then given an educational game called Taiga which would ask them to pose a hypothesis about decreasing numbers of fish in a pond, and then perform experiments to justify or disprove the hypothesis.

It was found that students achieved a statistically significant increase in understanding and performance of topics in the PR group when compared to the NPR group. Taiga had a variety of educational content embedded within the game that the users can access, and tracked which ones the children had checked during the tasks to determine whether the children were motivated to access more of these. However, it was unexpectedly also found that the PR group did not access any more of these educational materials than the NPR group, which - given the previous findings - could insinuate that the increased performance was derived from increased confidence in the subject matter.

Furthermore, the badges awarded by Taiga are synonymous with the common game design element of 'Achievements', which are awarded to the user after a certain set of requirements have been met, e.g. "Achievement Unlocked - Complete 10 Quests", and are a useful tool for encouraging users to complete many short tasks in order to earn a long-term reward Hamari and Eranti (2011).

A common issue with rewards in education is that 'extrinsic' rewards - i.e. rewards not relating to the activity they are awarded for - ultimately end up undermining the child's attention and motivation from the intrinsic learning activity once the rewards are removed (Deci et al. 2001, Tang and Hall 1995). It can be derived, therefore, that the rewards are less useful as long-term motivation for children, especially if the rewards cease to be given. However, Cameron (2001) highlights that this effect only occurs with children who had high initial interest in the task at hand. For example, rewarding a child for performing in his favourite sport only serves to undermine their original interest in the sport. An explanation for this effect might be that by offering the reward, the child's goal for playing the sport has shifted from 'I want to play Rugby' to 'I want to earn this reward offered for playing Rugby'. This means that once the reward has stopped, the incentive for playing has also stopped.

During a meta-analysis of studies into rewards in education, Deci et al. (2001) also found that unexpected rewards had the least negative impact and highest positive effect for children's long term attention to a task. This could mean that it would be beneficial to include an element of randomness to the rewards for completing a task, such as a random chance of earning an increased reward or an extra, but separate, reward. King et al. (2010) lists a variety of different reward structures in video games, such as in-game

currency, experience points, levelling up and 'rare-item' rewards. In role-playing games (RPGs), quests often have these key reward structures and, therefore, are established to work suitably well for this project.

2.2 Game Design

2.2.1 Game Design Elements

As previously mentioned, it is important to specify which game design elements I will choose to use in my application. In the book '100 Elements of Game Design' (Despain 2012), it mentions several useful elements to use.

Deterding et al. (2011) separated game design elements into five different levels as shown in appendix A. The key elements I will focus on in my research are 'Game interface design patterns', which are common components in games that are not specifically 'played with' themselves, but run alongside a game to offer more feedback or fun within the application. This includes elements such as badges/achievements, leader-boards, levels and experience points.

Achievements

Montola et al. (2009, p.94) performed an experiment to determine the usefulness of achievements within a non-game application by adding them to an image hosting and sharing website. Achievements are defined in this paper as "optional sub-goals in a secondary reward system" and are described as being separate to the core game system. These sub-goals do not affect the progress of the player and should be seen as more of a meta-game, rather than being integral to the game itself. Concerns were initially raised that adding achievements to non-game applications would encourage unproductive or unintended behaviours by users. An example of this would be an achievement on a forum for making 1000 posts. This may inadvertently motivate users to make a large amount of low-effort or spam posts, reducing the overall quality of the forum. However, Montola et al. (2009) noted that some users appreciated the changes and found the achievements to be motivating. Achievement features are one of the most commonly implemented game design patterns in gamification (Hamari and Eranti 2011).

Feedback Loops

A positive feedback loop involves the player becoming more powerful throughout the game, which, in turn, makes things easier to complete. This means they can complete more quests and become more powerful even quicker. Whilst this sounds like a good element to the game, it is important that developers avoid allowing this to destabilise the game, by making quests - and therefore the game - trivial. The project artefact will be

largely unaffected by this, because quests are completed in the real world and are not tied into the in-game character's power. However, if I seek to include an element of competitiveness into the game by allowing users to 'battle' each other, it is important to take steps to avoid players becoming so powerful that the battles are no longer enjoyable.

2.3 Smartphone Applications

2.3.1 Android vs. iOS

When planning the development of a mobile application, it must be considered which platform is best to target.

Market Share

Appendix C clearly shows Android's stronghold over the global market share, holding over 70% of units shipped since Q3 of 2012. Appendix D also shows an Android dominance in the UK market specifically. In these findings, however, the hold over the market is less strong - at 57% in 2014 and 52% in 2015 - and shows a downward trend in the market share of Android.

Fragmentation

A common issue with developing for mobile phones is that many people do not (or cannot) update their devices to the latest versions. This problem is referred to as 'Fragmentation' and is particularly problematic in Android, as shown in appendix B. Despite being released in October of 2015, the latest Android version of Marshmallow is still at less than 2.5% adoption rate. This is primarily due to software updates being held back by network carrier locked phones and phones falling out of support.

As Android phones are manufactured by a number of different companies, there is less standardisation between available handsets. OpenSignal (2015) reported that it detected nearly 24,100 distinct devices that downloaded its app in 2015, showing just how much of a problem fragmentation could pose to Android development. This also leads to concerns about the variety of screen sizes in these devices, as apps will have to be designed and maintained to support myriad phones. However, Ivanovic (2015) shows that, due to the similarities of resolutions between these phones, there are only four key aspect ratios that need to be managed within Android. This is due to the operating systems' implementation of scaling.

2.4 Development

2.4.1 Waterfall

The waterfall design methodology is a sequential design process focused on the project progressing to each stage of development one at a time. For example, the design stage of the software would not take place until all requirements have been gathered and finalised. Waterfall is considered to be a very structured methodology and is particularly useful when the requirements are well known and are fixed before the project begins and when the project and the technologies are well understood.

However, due to the rigidity of the Waterfall method, particularly around the planning phase, it is easy for mistakes made early on in the software development life cycle to become embedded within the project and be carried forward to later stages. For example, if a mistake is made during the requirements-gathering phase and is not noticed until a later phase of the project, it will be very costly to change. This approach means that risks to the project can be left unnoticed until it is too late in the project to change (Kruchten 2001).

2.4.2 Agile

The Agile design methodology is a well established and iterative design process that involves regularly moving between stages of the project in a non-sequential manner. Agile is very adaptable and allows for more flexibility to the software when design issues arise by not finalising the specification for the whole project too early. By iteratively planning the design of the project, it allows for quick responses to feed back and if one part of the project is changed, it should have minimal effect on the rest of the project stages.

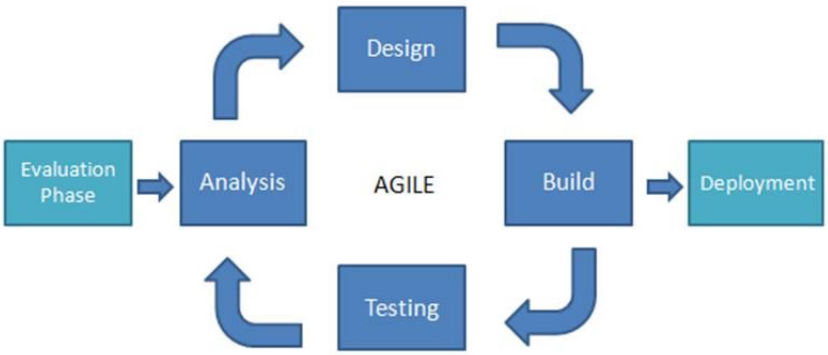

Figure 2.1: Iterative Agile Development Process

Agile dictates that testing should take place regularly throughout the project and by iteratively testing the software at various stages in the life cycle, it gives the developer more chances to catch bugs early. Under agile, defects discovered in software should

be fixed as soon as possible after discovery (Beck et al. 2001). In a study by Talby et al. (2006), it was found that by following this practice, defects took less time to fix than without. This may be due to the fact that the developed features are still fresh in the mind of the developer, who is therefore more readily able to see the problem area and fix it. The study also found that by keeping the bugs fixed and the code base stable throughout the project, it actually made the development phases of the project faster.

Furthermore, agile testing helps minimise the risks of bugs making it into the final artefact. If there was a single testing phase at the end of the project, it is possible that there would be more bugs than anticipated and that the developers would run out of time to properly fix them all. By using agile testing, this risk is minimised by allowing developers to better manage the time of testing, as the development of certain low priority features can be delayed or cancelled earlier in the project to allow more time to ensure a quality end product.

2.4.3 Test-Driven Development

As mentioned previously, one of the key principles of agile development is regular and iterative testing throughout the software development life cycle. This means that test-driven development (TDD) could be a useful tool to work alongside it. A key factor of TDD is planning out the testing phases of the project before the development of that phase begins. The workflow of TDD is as follows (Beck 2002):

1. Add a test that tests the new feature works
2. Run all tests and see the new one fail
3. Write the code that makes the test pass
4. Run all tests and see them all succeed
5. Refactor the code and tests to remove duplication
6. Repeat

A study by George and Williams (2003) found that developers following TDD produced higher quality code which passed 18% more functional tests.

A software group at IBM made the drastic change from an ad-hoc approach to testing to a TDD-based workflow. They found that, at the cost of a small drop in productivity, their defect rate was reduced by 50%, the code quality was generally increased and was more flexible to changes, and that morale was generally higher (Maximilien and Williams 2003). The paper also considered that the drop in productivity may well have been down to the change to an unfamiliar system rather than an inherent flaw in TDD, though - due to the increased time writing, preparing and running tests - it is not unusual that TDD might have this effect.

Chapter 3

Requirements and Analysis

3.1 Defining Requirements

3.1.1 Functional Requirements

Functional requirements are, in simple terms, things that the system should be able to do. Generally, functional requirements will specify behaviours, common goals, or functionality that one may require from the software being designed. Typical examples of these could include 'display customer' or 'add order'. The overall quality of the software can be evaluated by the quality and effectiveness of these individual features.

3.1.2 Non-functional Requirements

Non-functional requirements are defined as "Software requirements that describes not what the software will do, but how the software will do it" (Chung et al. 2012, p.6). This includes examples such as the software's performance requirements, usability, quality of services, reliability etc. These are generally more subjective aspects of the software requirements and are therefore harder to evaluate. It is important, however, that they still be considered during development iterations.

Software specifications are made up of both functional and non-functional requirements (Chung et al. 2012).

3.2 Gathering Requirements

3.2.1 Actors

When determining use cases, the first step is to define the actors in the software. In UML, an actor is defined as a "a type of role played by an entity that interacts with the subject (e.g., by exchanging signals and data), but which is external to the subject." (OMG 2007, p.586). Actors may represent human users of the system or other external systems

that will interact with the software being modelled. Different actors will have different objectives for using the application and it is important that the requirements of each actor are analysed to ensure the application is suitable. In this project there are two key actors, the Child and the Parent. The Child marks quests as complete and creates an character. In actual usage, this will usually be a dependent of the primary user. The Parent assigns, manages, and approves their child's quests. In actual usage, this will usually be a real-life parent or guardian of the secondary user.

3.2.2 Identifying Features

A useful first step in identifying the features of the software artefact is to use brainstorming to quickly list features that could be possible for the application, without regard to the suitability or feasibility of the ideas. Essentially, the goal of brainstorming is to achieve quantity over quality (Leffingwell and Widrig 2000, p.144). Once a wide breadth of ideas has been listed out on a brainstorm, it is recommended to begin an 'idea reduction' stage, where ideas are briefly evaluated to rule out obviously infeasible ideas to obtain a solid list of features that can be more deeply evaluated at a later stage. The process for brainstorming the application involved looking at the features of similar existing apps - such as task management apps or apps gamified for children - whilst keeping in mind the original objectives of this project. The results of this brainstorming process can be seen in figure 3.1.

3.2.3 MoSCoW

Developing a software project is a complicated procedure with many potential roadblocks to be faced. As identified in my risk analysis, there are several risks that can cause significant delays to the project, which in turn could be disastrous to the project due to its fixed deadline. Research by Standish Group (1994) has shown that only 16% of all software projects are delivered on time and within budget, which has harrowing implications for the schedule of the project.

In light of this, a useful next step is to prioritise the tasks ahead by determining which features are most vital to the project and will, therefore, require most time and urgency. MoSCoW is an easy method of dividing requirements into a clear hierarchy of four categories that establish a priority for the features or requirements of a software project (Hatton 2008, p.517). This MoSCoW list will serve as the final list of features that I hope to include within this project and will be referred to as my 'to-do list' throughout development. It is important to explicitly note, however, that this is not a target of features that will be in the application, but rather a hierarchical 'wish list' of features that may or may not be achieved, depending on how development progresses within the time frame. I have separated the functional requirements (FRxx) and non-functional requirements (NRxx)

Figure 3.1: Brainstorm of Potential Features

into the four MoSCoW categories in the following list:

Must Have

There are some features that will be vital to the success of the project and can cause delays to the release of the software if they are not finished on time. These should be the first features developed in the software and be thoroughly tested to ensure high quality.

No.	Feature	Description
FR01	Child Tasks App	A Child can login and see tasks to complete, and mark as completed accordingly
FR02	Parent Control App	A Parent can login and view details about the Child's account
FR03	RPG Rewards	Child users will be rewarded with experience points and will level up after obtaining certain amounts
FR04	Gold Rewards	Child users will be given a gold reward for completing tasks on time
FR05	Reward shop	Parents will be able to set real life rewards for the Child to purchase using earned gold
FR06	REST API back-end	Implement a REST API server that the Parent and Child apps can use to communicate with each other, instead of phone to phone communication
NR01	Reliability	App must not crash, and must handle errors gracefully

No.	Feature	Description
NR02	Security	Server must not make user information accessible without correct authentication

Should Have

These are features that are important to have, but are not integral to the app's main functionality, so will not render the app useless if they are incomplete at the end of the process.

No.	Feature	Description
FR07	Preset Quests	Allow Parent uers to choose from a list of preset tasks generated by the server, such as trending tasks
FR08	Friends List	Allow Child users to add each other as friends in the app. The adding of friends should be monitored and approved by Parent users
FR09	Notifications	Alert a user whenever their respective Parent/Child performs certain actions
NR03	Portability	The server should be platform agnostic and serve content regardless of client device
NR04	Usability	User must be able to easily navigate between various screens
NR05	Usability	The app should be responsive and reliable
NR06	Android Developer Guidelines	App must make a best effort to follow the Android Developer Material guidelines laid out by Google

Could Have

This describes features that would be preferable to have in the project and could be looked at, if there is time to, at the end of the project. However, it is important to be wary of scope creep and recognise that any feature added will not only add development time, but testing and documentation time too.

No.	Feature	Description
FR10	Friend Battles	Child users can select friends to 'battle' with and compete with
FR11	Graph Friend Selection	Graph selection UI for inviting friends to undertake group quests
FR12	Web application	Allow Parent/Child to use a web application to access their accounts

Won't Have

Suggested features that are not feasible for this release, due to time and resource constraints. These could potentially be re-raised for a future update to the software.

No.	Feature	Description
FR13	Sound Effects	Music and audio effects when certain events are completed
FR14	Character Graphics	Artistically designed graphical interface
FR15	Leaderboards	A constantly updated ranking of calculated 'scores' between befriended users

3.2.4 Use Case Diagrams

In order to properly specify the requirements for a user in more detail, it is useful to plan out the various use cases a user may take in a use case diagram. The purpose of a use case diagram is to show the actors the system, their goals, and how those goals work in relation to each other. Use case diagrams are also helpful to visualise an outside view of a system and gather requirements - although in this case I am using the diagram to refine the requirements that I have already mapped out.

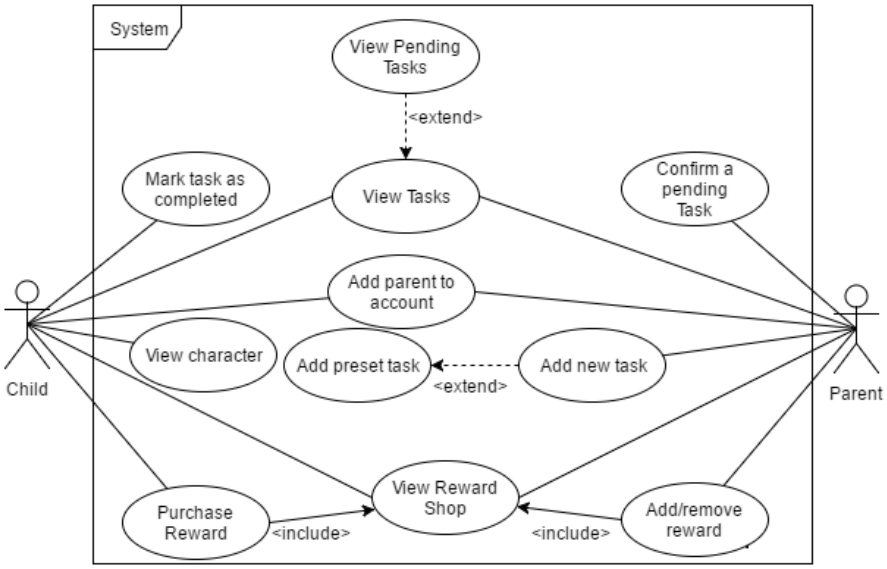

Figure 3.2: Use case diagram of the 'Must Have' functional requirements

In figure 3.2, you can see the two actors in the system and the relationships between the tasks that they may take. The diagram also lists tasks that are extended from another task such as 'view pending task', which, for all intents and purposes, is the same feature as 'view tasks'. The difference is that you only see a subset of the tasks. You can also more clearly see the layout of the reward shop, where the use case details how the Parent can view the shop to add or remove rewards, whereas the Child can view the shop to purchase rewards.

Chapter 4

Design

In this section, I will discuss both aesthetic and technical designs for the application as well as offer a design of the system as a whole.

4.1 System Design

In order to gain a better overview of the system, I mapped out how the individual pieces of the system work together and analyzed each part.

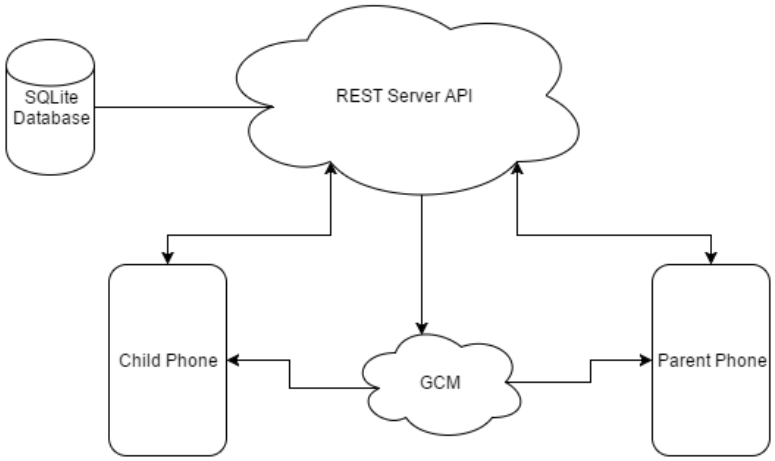

Figure 4.1: System Design Diagram

4.1.1 Server API

In the initial stages of design, there were incorrect assumptions made about the project, which led to early diagrams being inaccurate and ultimately unworkable. Initial designs attempted to implement all workflows into the applications and communicate directly between two phones, storing all necessary data/information on the client end. However, these designs presented many shortcomings. The main drawback was that communications between two mobile devices was unreliable, to the extent that it was not possible

to ensure that data would remain synchronised between them. For example, if a Child marked a quest as completed, their phone would send a notification to the Parent. However, if this notification was not received by the Parent, the quest would fall into a limbo-like state where the Parent is waiting for it to be completed and the Child is waiting for it to be confirmed.

In order to prevent this type of occurrence, the system calls for a centralised location. This will allow two users to reliably communicate with each other. By implementing the functionality into a server-based application, the system becomes significantly more portable. As all functionality of the system rests in a centralised location, any ported apps - such as iOS or a web app - would only have to tie into the existing Application Program Interface (API). This would make porting the application simple, as the only code that would need to be created for these apps would be to simply send and receive web service requests to the API and display the data that it receives.

Setting up a central server has also meant increased maintainability of the software. Updating software on mobile phones/devices becomes difficult as people do not always update their apps to the latest version. Furthermore, if two users working together had different versions of the app it could cause serious interoperability problems. The server API helps this, as all functionality updates to the system can be made in the server rather than user devices. This effectively distributes updated functionality to all users at once, instantly.

However, server-side implementation does have two main consequences that must be considered. Firstly, users would now require an internet connection to use the app so that it is able to communicate with the server. This may cause difficulty for users who have limited data plans on their phone contracts or limited connectivity. However, this issue can be mitigated by keeping the data sent to a minimum and making the app only send absolutely necessary requests. Furthermore, options exist in the Android operating system (OS) that restrict an application's data usage, meaning that requests will be halted until the user has connected to a usable wi-fi network.

Secondly, this would require all users to create an account with the server in order to use the app, as the system needs to store the data on the server end rather than on the users' phone. I would need to have high security requirements surrounding that data, making sure that it is sufficiently encrypted during transit between the user device and server, and that the data stored on the server itself is sufficiently protected as well.

To handle logins securely, I plan on implementing a token-based authentication system within the server. Upon their first request, the user will submit their username and password within a HTTP POST request to the server. The server then sends back an encrypted token that is derived out of various hashed attributes of their account, which the user client then stores. The app will then authenticate itself with the server for every future request using this token. This offers the main benefit that the app only has to store

the token and not the username and password, which would leave it vulnerable to other (potentially malicious) apps accessing critical user data. It will also mean that the app will not have to send the user's username and password with each request that it makes to the API, minimising the risk of the password being intercepted in transport. The password is only stored within the server, which has securely encrypted it with a salted hash.

4.1.2 Application

The application will greet the user with a login/register screen that will be used to authenticate with the server before the rest of the app can be accessed. The login process will be bypassed if the user currently has a valid authentication token.

The app will then communicate with the server to gather the relevant information needed for its current screen, being careful to only request what is necessary. Once the information has been received, the app will display that information in a well laid out manner and provide relevant options to add or update the data on the server. The app should follow the android developer guidelines (See section 4.3.2) to provide a familiar layout.

On the first login of the app, the Child will be prompted to enter a name for their character. They will then be instructed to pass the phone to the Parent who will create a PIN code. Performing certain functions of the app will require this PIN code to be entered, thus preventing the Child from circumventing the nature of the app by giving themselves unlimited rewards. Parents will be able to interact with the app solely through the Child's phone, but will also be able to set up an account on their own device which will provide the Parent functions remotely. As a safety measure, this Parent account must be set up on the Child's device first, and will require the aforementioned PIN code to be activated/installed. This should ensure the Parent account belongs to an actual parent of that user, and cannot be misused by a malicious, unknown user.

4.1.3 Database

Design

After planning out the requirements, the next step in the design of the project was to plan out the database structure. I created an entity relationship diagram (ERD) to map out the various tables needed and the relationships between them. ERDs are a useful visual representation of tables, columns and relationships within a database. An ERD for the server application can be found in figure 4.2. The requirements for the database are relatively simple: it is comprised of 3 tables and 26 data fields and will store all the necessary data about a user's account and their tasks.

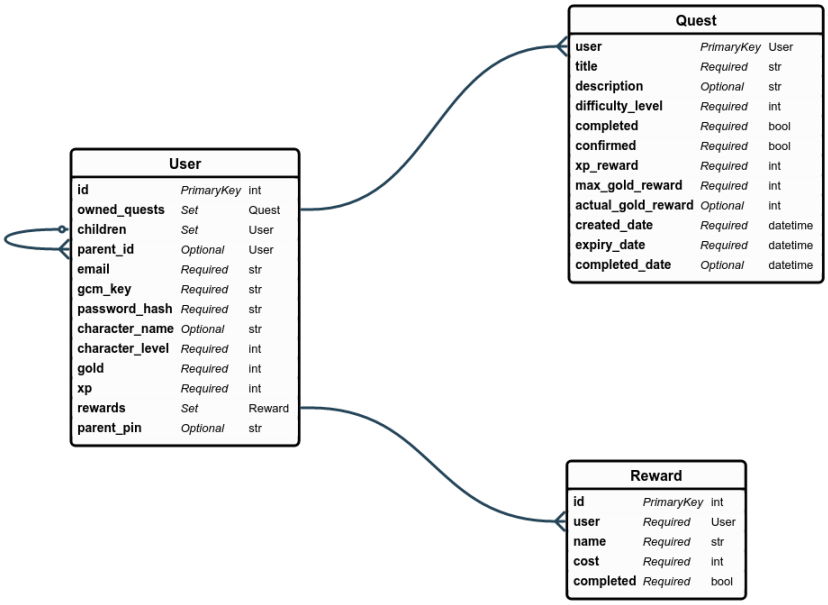

Figure 4.2: Server Entity Relationship Diagram

Database Access

In this design, the central database is only accessible via the server API, rather than using a direct connection to any of the users' individual devices. Instead, the server will offer publicly available web services that will provide functionality to input or update data. This has the benefit of allowing me to automatically validate and sanitise any data sent by the user before being added to the database and correctly reject requests that are unsuitable. These services will ensure the user making the request is authorised to take actions on that account, i.e. the user making the request must be either the owner of that account or a registered Parent of the user - before allowing them to perform any functionality. This segregation of the database helps to protect user data, as the database is not made accessible over the web at any point and is therefore less vulnerable to malicious entry.

4.2 Chosen Methodology

Balaji and Murugaiyan (2012) states that the choice of methodology depends on the project's requirements. As raised in my risk analysis, there is a high risk of the project requirements changing as various assumptions about the project are challenged. Therefore, influenced by the research in chapter 2, I have opted to develop the software artefact using an Agile methodology that would allow faster reaction to changes and issues that could otherwise put the project at risk.

4.2.1 Test-Driven Development

From the various development processes I have examined, I have chosen to use test-driven development (TDD), as it will be beneficial for the development of the server. Due to the nature of how devices will communicate with the server, it will be simple to generate test cases from the various functions that the server offers.

As I have very little experience with TDD, it is likely that there will be a drop in developmental progress, as mentioned in the research in chapter two. However, it must be considered that if the practice offers a higher quality of code, it is inherently likely to produce fewer errors and the productivity/time trade-off may be worth it. However, due to the complexities of generating unit tests for Android, I have chosen to develop the application using only an Agile methodology.

4.3 User Interface

One of the components of software quality is usability and usefulness of it's User Interface (UI) (ISO 25010 2011). As such, a focus must be put on making the screens of the application clearly usable. The screens should not only be easy to use but easy to learn and navigate between. The interface should present minimal obstruction in the use of the application. The server application does not require an interface as its intended use is solely machine-to-machine communication.

4.3.1 Screen Design

Stone et al. (2005) classifies a good user interface design as an 'easy, natural and engaging interaction between a user and a system'.

Wireframing is a useful tool to plan out the aesthetic design of screens and the UI elements used within them. A wireframe is also useful to gain a clear overview of the navigation between screens, as they can be used to plan out the button presses that will select the next screen. The wireframes shown in appendix E give an overview of the general aesthetic style that will be used.

An overview of the planned screens for the application can be seen in appendix F.

4.3.2 Android Developer Guidelines

Google (2014) have laid down clear guidelines on how to use the UI elements they have provided in the Android Source Development Kit. By using these guidelines, the application will be more portable, as its UI will be more scalable to the various sizes of devices that are available.Following these guidelines will also help make the application maintainable, as following standards is a useful way of ensuring that other developers will be able to better understand the design used in this artefact.

Chapter 5

Implementation and Evaluation

For clarity, I present the implementation and evaluation together and split the presentation by the natural split in the code which is the server and application.

5.1 Server

5.1.1 Representational State Transfer (REST)

To structure the server's web services, I have chosen to use a REST architecture style. REST involves dividing the web services into specific objects which can be accessed through sending a request to that object's URL. This is called an 'endpoint' or a 'URI'. These endpoints are then connected to via different HTTP methods, such as the following:

GET Retrieves an item or list of items.

POST Creates a new item.

PUT Updates an existing item or list of items.

DELETE Deletes an item.

For example, if someone wanted to retrieve all users of a REST service, they could send a GET request to the '/users/' endpoint. If, however, they wanted to retrieve a specific user, they would send a GET request to '/users/x/' where x is the user's ID. And finally, sending a POST request to '/users/x/quests/' would create a a new quest for user x.

Collectively, these request methods and endpoints will form the basis for all communication with the server. A full plan for the endpoints of the server artefact can be seen in appendix H and examples of requests in I.

JSON

In order to facilitate communication between the server and devices, JSON has been chosen as the primary messaging format for this system. JSON is a format of transmitting

data in a both human and machine readable way using key value pairs. When the server is required to return objects from the database, it will be able to serialise the python object into JSON text and return it in the request. The application will then be able to de-serialise the JSON back into a Java object, translating the JSON values back into object properties. Examples of possible requests can be seen in appendix I.

5.1.2 Flask

For the server-side implementation, I will need tools that allow me to easily create a web service API and provide controlled access to functionality over the web. For this task, I chose to use Flask as it is well established within the Python community and is relatively easy to set up and maintain. Flask is suitable for both quick prototyping of web-services and reliable deployment/hosting of the functionality over the internet.

By utilising Flask, I was able to easily translate a user's web request into a method call, with relevant parameters, using the following code:

```
@api.route('/users/<int:user_id>/rewards/', methods=['GET', 'POST'])
@auth.login_required
def user_rewards(user_id):
    # Get the user account relevant to the user_id parameter
    user = User.query.get(user_id)
    # Verify the logged in user is actually the user or an attached user.
    verify_user(user)

    if request.method == 'GET':
        # Return the user's rewards in JSON format.
        return jsonify(rewards=[r.serialize() for r in user.rewards])
```

This short code snippet shows the method used to return all rewards in the reward shop for a user. This method is activated upon sending a GET request to the endpoint such as 'server_host_name:5000/api/user/1/rewards/', and will return the rewards for the first user in JSON format, provided the attached authentication details match that user.

5.1.3 Database

Object-Relational Mapper

I chose to use a pre-built Object Relational Mapping (ORM) library to interface with the database, rather than crafting particular queries on an ad hoc basis. This was primarily for more ease-of-use purposes than any performance reasons. I decided upon using SQLAlchemy as the ORM for the server due to strong Flask support. Specifically, the Flask-SQLAlchemy library was utilised to provide Flask specific functions that could be used to interface with the database. Implementing an ORM also allows me to more easily protect the application against SQL injection, as Flask purposefully escapes special characters that would allow such an attack (Copeland 2008, p.15).

SQLite

I have opted to use SQLite to store the user data in the server. SQLite is a very lightweight RDBMS that is well supported by the Flask-SQLAlchemy library. The database tables will be automatically generated out of the classes created in Flask on the first run of an application, as well the relationships described in figure 4.2.

5.1.4 Evaluation

Of my objectives, there are two that are relevant to the server: 1. Create a secure and robust public API to allow applications to connect to the artefact's service. The server application successfully provides a public API that provides web services to connecting clients. The server provides the necessary information for clients securely, by requiring successful HTTP authentication to retrieve information about an user or any of it's quests/rewards. The server is resilient to crashes and automatically rejects bad requests before the invalid data can be stored in the database, and 2. Utilise test-driven development to provide automated testing, ensuring test coverage of over 80% for the API Using the practice of test-driven development for the server API was a general success. For some parts, the methodology was found to be rather strict and caused some delays to development time. However, once the practice had been used for a little while, it began to drastically improve development time, particularly in the case of debugging that arose during development. Furthermore, during testing phases, the tested code passed many of its functional tests on the first try, which is likely due to the fact that it is regularly tested during the development process. Overall, it was found that most features took slightly longer to develop, but took significantly less time during testing phases. These findings agree with the research in chapter 2, as Maximilien and Williams (2003) found similar results of reduced productivity, but it resulted in code with less defects.

The final result for automated test coverage came in at 91% code coverage for the server, exceeding the 80% target, and analysis of the code shows all major server functions were covered by automated unit tests.

5.2 Application

5.2.1 Android

Android has been chosen for this application due to the large success and market hold of the operating system. Primarily I will be aiming to support Android versions down to API level 16 (Jelly Bean 4.1) [1]. This decision has been made as appendix B shows that by supporting this far back, the application will be compatible with over 95% of current phones. Older versions than this will not be targeted, as development and testing time

is limited and it would mean that many features would have to be limited due to lacking functionality within these older versions.

5.2.2 Google Cloud Messaging

The app will be responsible for relaying information between the parent and child clients in a reliable and timely manner. In Android, this can be achieved using push notifications relayed through the Google Cloud Messaging (GCM) service. GCM can be utilised to send messages to an Android device, but is largely unreliable (Nava 2014) and therefore should only be used for notifications, rather than the transmission of data. The service requires that the server registers with the GCM to retrieve an API key and that the user device registers to retrieve a GCM registration ID (Google 2015). When a user registers with the artefact server, the registration ID is stored wtihin the database to be used to send notifications. When a particular action is performed on an account in the API, the server will check if that user has an attached account - i.e. either a Parent monitoring a Child or vice versa. Then, the server sends the message and the target GCM registration ID to the GCM system which passes on the message.

Within the application, a listener service has been implemented to be triggered upon the receipt of a GCM message that will create a push notification with the message sent by the server. Examples of the push notification and a list of possible messages can be seen in appendix G.

5.2.3 Evaluation

Incorrect assumptions were made in regards to the provided functionality of the GCM service. Initially, plans for the application involved using the GCM to communicate between two devices in order to share the information needed for the Parent app. However, it was discovered that GCM was *largely unreliable* (Nava 2014) and was prone to errors and failed messages. Therefore, it was decided that GCM was unsuitable for the use of communications between devices, as the risk of critical information being lost in transit was too high, and this functionality was moved to the server. However, GCM was still found to be suitable for the non-critical push notifications, as reliability is not strictly necessary in this case. Screenshots of the application can be seen in figure 5.1.

[1]Many features included in recent Android versions will not work in older versions. Android development allows you to specify a minimum Android version to prevent the use of features that will not work on older phones.

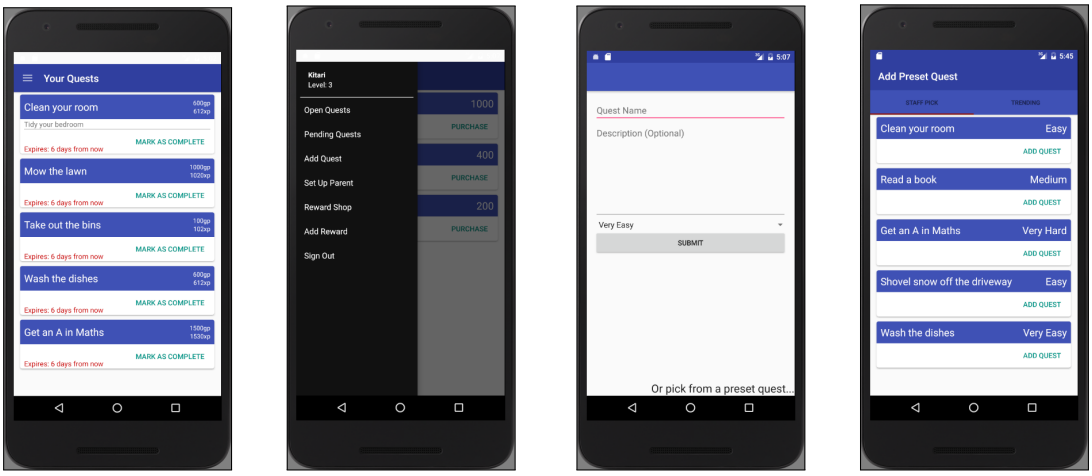

Figure 5.1: Screenshots of the Project Artefact

5.3 Game Design

5.3.1 Rewards

The rate at which these rewards are earned must be examined to ensure they still feel rewarding throughout. For example, if a quest returns 100 XP points and you require 300 XP to level up from level 1 to 2, this only requires you to complete three quests to level up, meaning these three quests feel rewarding as each quest is offering 1/3rd of a level. However, if quests still offer 100 XP and you require 6000 XP to level from 30 to 31, the reward from the same quest is now worth 1/60th of what it was before, despite being the same difficulty to the user. Therefore, some scaling of the rewards is required in the app to allow XP rewards to scale suitably with the XP required to level. This scaling is referred to as 'increasing cost' (Anderson 2016). Anderson also shows that it is important to determine reasonable caps for numerical relationships in video-games, to stop the increasing cost becoming out of proportion.

Anderson proposes that a useful pattern for these kinds of relationships is a classic triangular pattern, in which the first level requires 1 XP to level up, the second level requires 3 XP, the third requires 6 etc. To allow for more meaningful numbers to the user, I have multiplied the results of this formula by 100. The formula for this is $T_n = \frac{n(n+1)}{2} \times 100$, where T is the XP required to level up and n is the current level.

For XP rewards, the initial scaling formula involved using 60 XP as a base reward for a medium difficulty quest. Then using the player's current level, it would derive a multiplier that would be used to scale the XP reward, using the following formula: $\text{XP Gained} = 60 \times \frac{(n-1)\times k}{100} + 1$ where n is the current character level and k can be adjusted to determine the strength of the multiplier against the XP rewards. Under this formula, if k = 10, a character at level 31 would have a 3x multiplier on quests, meaning the same quest that

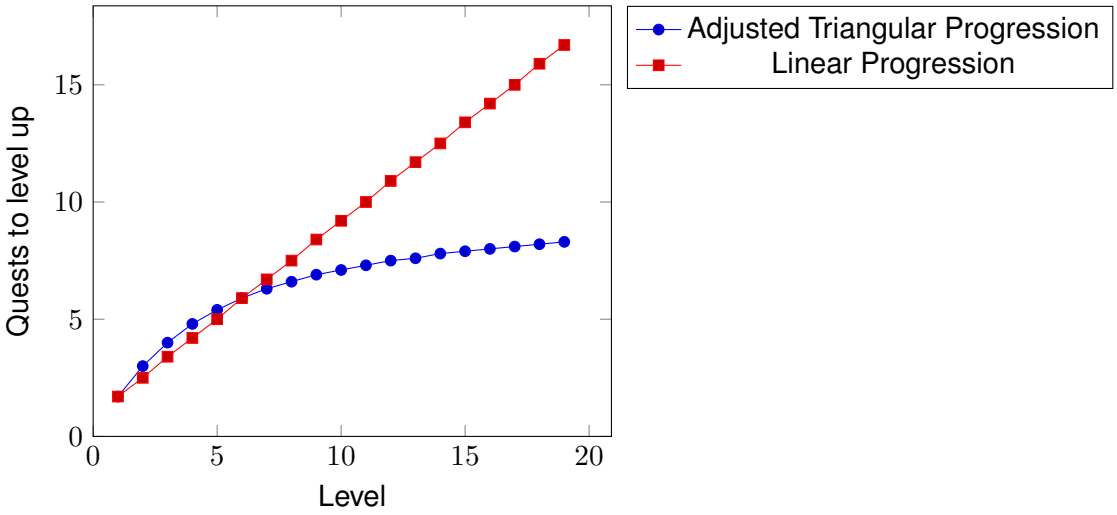

Figure 5.2: A comparison of two XP reward systems

gave them 60XP at level 1, now gives them 180XP.

However, when modelling this equation, the number of quests needed quickly becomes out of hand. At level 1, the player must complete 2 medium quests to level up - a reasonable starting point. However by level 25 the player must complete 22 quests to level up and by level 100 the player must complete an extraordinary 84 medium difficulty quests.

Therefore, I also modelled triangular numbers for calculating the XP gained per quest. I adjusted the initial triangular numbers formula to $\mathrm{XP\ Gained} = \frac{(n+2)(n+3)}{2} \times 10$ and found a suitable middle ground. This formula gives the new user the feeling of progressing quickly, but the rate of quests per level quickly slows down to a limit of approximately 10 quests per level by level 100. This will ensure that the player will still feel like they are progressing at a reasonable speed throughout.

The rate of quests required to level up using these two systems is mapped in figure 5.2. This figure was mapped out assuming all quests completed were of medium difficulty, whereas in real usage the speed will increase or decrease depending on the difficulty of quests.

5.3.2 Gold Rewards

A key goal of the project is achieving long term motivation for children, and to avoid the non-continuous effects mentioned by Deci et al. (2001) in chapter 2 research. In order to achieve this goal, I implemented two key elements to the rewards earned from a quest.

Firstly, in order to discourage Child users from delaying performing a task, I implemented a system that gradually diminishes the maximum achievable gold reward for a quest over time. In this system, a quest is added with an expiry date value, which defaults to one week from the quest creation date, unless otherwise specified. At around

halfway through the time available to complete the task, the reward from the quest will begin to reduce at a steady rate for the remaining time. Eventually, if the task remains incomplete until the expiry date, the gold reward drops to zero, the quest expires and the quest is removed from their tasks. This is an example of an 'open loop system', where previous performance has no bearing on the current reward.

Secondly, I extended this system to base the given reward for a quest off of the Child's time-based performance in their last three quests. This is referred to as a 'closed loop system', where the reward amount is calculated based off previous rewards. The end result is that if a Child allows a quest to begin expiring, it will effect the following three quests rather than just the current one. This system encourages the user to maintain a streak of fulfilling tasks in a timely manner and discourages them from allowing a quest to expire as it will not just be that reward that is effected, but their rewards for the next three quests. It is intended that this will influence children to begin thinking about their tasks in a long-term manner. If the user has completed fewer than three quests, the effect is ignored as there is not enough data. The reward is calculated using the following equation: $y_k = ax_k + by_{k-1} + cy_{k-2} + dy_{k-3}$ where k is the quest, y is the reward for that quest and a-d are adjustable coefficients that sum up to 1.

A comparison of the two types of rewards can be seen in figure 5.3.

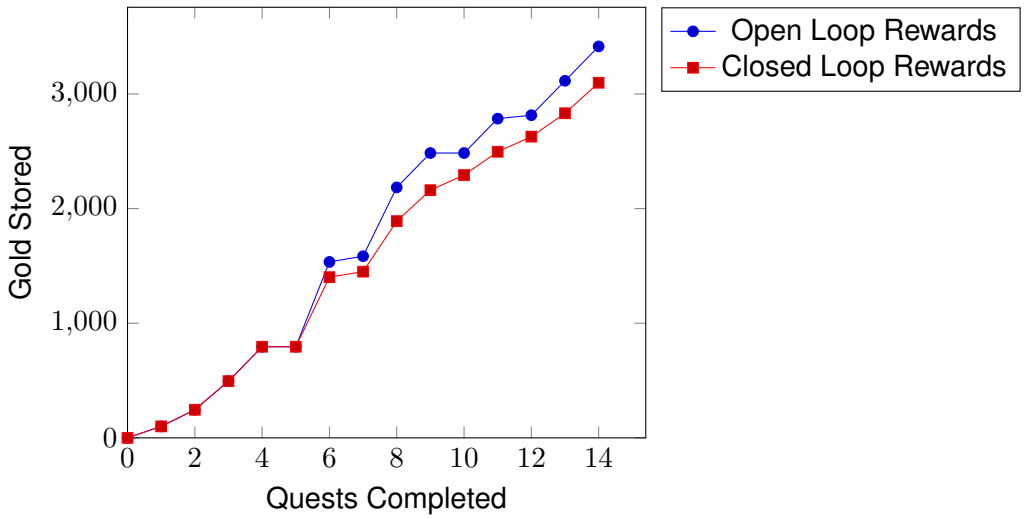

Figure 5.3: A comparison of the two reward types

5.3.3 Evaluation

In my evaluation of the artefact, I found that the triangular XP system provided a much more rewarding feel to the user. The application still provided a very fast gain when the user first starts using the app, but quickly levels off to a reasonable amount. The linear system quickly spiralled, which I felt was counter-intuitive to the intended motivational effects and would ultimately dishearten users who would have otherwise enjoyed the

app.

The gold and XP reward systems follow two separate paths. Where both the XP gained per quest and the XP required to level up increase each level, the gold rewards remain at a continuous gain and do not involve player levels. With XP, I preferred the system that rewards the player with more XP, whereas with gold rewards, the system that provided less was preferred. This is mainly due to the inclusion of a reward shop in the project, which exchanges some of the Child's gold storage for a real life reward set by the Parent. Therefore, in order for this to work successfully, the amount of gold that a child earns must remain stable.

5.4 Testing

Unfortunately, when developing a REST API, it is difficult to manually write and send the request to the API to test it. Programs and scripts exist to ease the process somewhat, but I found the easiest way to be automating the requests entirely. As a main principle of REST is to plan out the specific endpoints that can be messaged, it becomes rather simple to deduce automated tests by sending examples of valid and invalid requests to endpoints and then verifying the stored data. For example, the endpoint of '/api/users/$userId$/' allows two request types, GET and PUT, which easily generates four test cases.

- Valid GET
- Invalid GET
- Valid PUT
- Invalid PUT

However, this raises issues when considering that a request could be invalid for multiple reasons, and simply testing for invalid/valid may not reach adequate code coverage. A PUT request, for example, could be invalid due to an email address already existing in the system, or no new data being entered about the user. Both of these reasons are in separate sections of the code that each require their own tests. This issue is highlighted by Meyer (2008), who states that testing a program tells us 'little about its quality', arguing that the quality of a test case far outweighs sheer quantity of tests. Because of this, it must be analysed whether or not the test cases achieve sufficient code coverage, rather than just relying on the entry points into the software. Luckily, the python package 'coverage.py' allows for simple analysis of unit tests to determine the current code coverage for tests, which will provide a higher rate of confidence.

5.4.1 Test-Driven Development and Unit Testing

In the creation of a REST API, it is much easier to develop a test plan due to the clearly laid out workflow of the user. From the server's point of view, there are only a handful of requests that a user can make, and a small number of tests for those requests. Before writing each endpoint of the server, I developed a unit test that would test the various inputs to this endpoint. Usually, these unit tests consisted of python code to create and send a request to the testing server and then check the database to verify if the data was stored correctly.

I used an extension library of Flask called Flask-Testing, which streamlined the process of writing and running unit tests. The library allows you to create a new instance of your Flask application with a separate configuration file which can be changed to make the app connect to a different MySQL database file - in this case it connected to a freshly generated and empty test database. Then, as each separate test case is run, I used the library to delete the test database and regenerate it to ensure that previous unit tests would not interfere with new tests.

For example, the following code is a simple test that creates a Child user on the server, then attempts to get the user details from the server without authorising itself first. The test will pass if the server correctly rejects the request with a '401: Not Authorised' error or fail otherwise.

```
1 # Test no authentication details
2 child = create_child(self)
3 rv = self.client.get('/api/users/' + child['id'] + '/')
4 self.assert401(rv)
```

5.4.2 Integration Testing

Integration testing encompasses tests that ensure the various parts of the project work together correctly. For example, in this project, integration tests would test that the server and mobile app are able to correctly function together, by ensuring that the app can correctly send requests and that the server receives those requests as they were sent.

As the large majority of functionality took place in the server application, much of the integration testing that took place was to test that the mobile application was correctly requesting, sending and displaying data that it received from the server.

5.4.3 Black Box Testing

Black box testing describes the testing of software without using explicit knowledge of the code. Essentially, it is testing by using that the usage of the application is correct. The android application was mostly tested using black box testing by listing out various

actions that a user may perform on the app - whether valid or invalid actions - and testing that they are correctly accepted or rejected.

Due to the nature of the server code being primarily machine-to-machine communication, I consider black box testing to be inappropriate for testing the server.

5.4.4 White Box Testing

White box testing is the testing of a system using the knowledge of how the code works and encompasses tests such as boundary testing and equivalence partitioning testing. This was mostly performed on the server side code due to the fact that the majority of the android code was GUI based and provided no actual functionality.

5.4.5 Regression Testing

Regression testing is the practice of retesting the previously tested parts of the software to ensure that they are still performing correctly after a change elsewhere in the code, it can also be used to describe the process of testing previously detected and fixed bugs to determine that they have not reappeared.

This was made significantly easier by the implementation of strong automated unit tests, which allowed me to quickly retest the majority of the code by rerunning the test suite. I also followed a common development practice where a unit test is added for each defect found within the code to detect the presence of that bug specifically. This allowed me to easily spot any recurrences of legacy bugs that would have otherwise gone unnoticed.

5.5 Problems Encountered

Server Rewrite

As previously mentioned, issues arose regarding my assumptions about the GCM service. Initially, it was planned to have most of the functionality within the phone application and have monitoring phones communicate with it. Due to this, initial prototypes of both server and applications were inappropriate for the development of the system and had to be reworked. This incurred large delays in the progress of the artefact as alternative technologies were considered altogether. It was debated whether Flask was still suitable for the task, as I believed that all the extra code needed in the server might become too messy when written for Flask.

There were also initial issues with implementing automated testing within Flask, as I had not understood the extra requirements needed to run a separate, debugable instance of Flask and as such struggled to implement unit tests that successfully tested the web services. However, with the discovery of both the Flask-Testing library and the GCM

library it was found that the Flask code that had already been written would be able to reworked into a suitable API.

Data Loss

As predicted in the risk analysis in chapter 1, there was a small issue with data loss during the development of the application. During a regular reformat of my PC's hard drive, a small amount of code on the machine that had not been saved in a remote location was irretrievably deleted. However, by following the plans set out in the risk analysis, this issue was greatly mitigated by the fact that I had regularly committed and pushed code to the git repository. The lost code was not able to be recovered, but the loss was kept to an absolute minimum as the last commit had taken place only a few hours before the reformat.

Chapter 6

Conclusion

6.1 Features

Allowing Parent users to log into the app separately from a Child was an overall success and helped to limit the intrusiveness outlined in my risk analysis. The use of a server API back-end also allowed the two users' apps to update from each other very smoothly, and - to the user - seamlessly.

The addition of the reward shop is very useful to the system, as it provides a real-life reward element to the app. I believe this will be a key driving force in the adoption of the app, as it will provide the backbone for the motivational aspects of the RPG style rewards.

6.2 Personal Appraisal

I believe the system to be a very good proof of concept of the application, providing all of the important functionality detailed in the requirements chapter. The project has also allowed me to gain a great deal of knowledge into the development of an Internet connected mobile application as well as API and web service development, which will be invaluable going forward.

6.3 Evaluation

Upon evaluating the project, I feel that my initial decision to exclude outside participants for ethics reasons was a mistake. I believe that my evaluation of the software artefact suffered from a lack of structured external analysis and would have benefited from feedback such as beta testing or usability testing.

I also feel that choosing to make the system as a native Android app may not have been the best choice for this project. Whilst using a API back-end has improved the systems portability, a move to a responsive design web application would theoretically have provided an application that would be accessible from any device on any OS. However,

having a native app also comes with several of its own benefits such as efficiency and scalability, as using a website would have put much more strain on the server application.

6.4 Future Work

I fully intend to continue work on the application, though I believe it would require more work to be considered complete, as many features in the ‘Should Have’ and ‘Could Have’ lists did not make it into the final version of the artefact. There are a number of social media based features that would have been beneficial to the software, such as the ‘Friend Battles’ feature that would have introduced an element of competition into the motivational features. It is also my opinion that a complete version of the application would benefit greatly from a full aesthetic overhaul, including graphics, animations, and sound effects, to make it more appealing to younger children. Once the application is considered feature complete, the application could be made available for public beta testing.

Despite certain aspects that I may have chosen to do differently in this project, I feel that the the application was an overall success and achieved the aims and objectives set out at the beginning of the project.

Word count: 9983

Bibliography

Anderson, G., 2016. Level 2: Numeric relationships. Available from: https://gamebalanceconcepts.wordpress.com/2010/07/14/level-2-numeric-relationships/ [Accessed: 2016-4-4].

Appbrain, 2015. Top android sdk versions. Online. Available from: http://www.appbrain.com/stats/top-android-sdk-versions [Accessed: 2016-03-10].

Balaji, S. and Murugaiyan, M. S., 2012. Waterfall vs. v-model vs. agile: A comparative study on sdlc. *International Journal of Information Technology and Business Management*, 2 (1), 26–30.

Beck, 2002. *Test Driven Development: By Example*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J. and Thomas, D., 2001. Manifesto for agile software development. Available from: http://www.agilemanifesto.org/ [Accessed: 2016-04-01].

Cameron, J., 2001. Negative effects of reward on intrinsic motivation—a limited phenomenon: Comment on deci, koestner, and ryan (2001). *Review of Educational Research*, 71 (1), 29–42.

Chung, L., Nixon, B. A., Yu, E. and Mylopoulos, J., 2012. *Non-functional requirements in software engineering*, volume 5. Springer Science & Business Media.

Copeland, R., 2008. *Essential sqlalchemy*. " O'Reilly Media, Inc.".

Deci, E. L., Koestner, R. and Ryan, R. M., 2001. Extrinsic rewards and intrinsic motivation in education: Reconsidered once again. *Review of educational research*, 71 (1), 1–27.

Despain, W., 2012. *100 principles of game design*. New Riders.

Deterding, S., Dixon, D., Khaled, R. and Nacke, L., 2011. From game design elements to gamefulness: Defining "gamification". *In Proceedings of the 15th International*

Academic MindTrek Conference: Envisioning Future Media Environments, New York, NY, USA: ACM, MindTrek '11, 9–15. Available from: http://doi.acm.org/10.1145/2181037.2181040 [Accessed: 2016-03-25].

Fielding, R. and Reschke, J., 2014. Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. RFC 7231 (Proposed Standard). Available from: http://www.ietf.org/rfc/rfc7231.txt [Accessed: 2016-05-05].

Filsecker, M. and Hickey, D. T., 2014. A multilevel analysis of the effects of external rewards on elementary students' motivation, engagement and learning in an educational game. *Computers & Education*, 75, 136 – 148. Available from: http://www.sciencedirect.com/science/article/pii/S0360131514000426.

George, B. and Williams, L., 2003. An initial investigation of test driven development in industry. *In Proceedings of the 2003 ACM Symposium on Applied Computing*, New York, NY, USA: ACM, SAC '03, 1135–1139. Available from: http://doi.acm.org/10.1145/952532.952753 [Accessed: 2016-03-28].

Google, 2014. Material design. Online. Available from: http://www.google.com/design/spec/material-design/ [Accessed: 2016-03-11].

Google, 2015. Cloud messaging. Online. Available from: https://developers.google.com/cloud-messaging/ [Accessed: 2016-04-22].

Hamari, J. and Eranti, V., 2011. Framework for designing and evaluating game achievements. *Proc. DiGRA 2011: Think Design Play*, 115 (115), 122–134.

Hatton, S., 2008. Choosing the right prioritisation method. *In Software Engineering, 2008. ASWEC 2008. 19th Australian Conference on*, IEEE, 517–526. Available from: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=4483241 [Accessed: 2016-04-01].

Huotari, K. and Hamari, J., 2012. Defining gamification: A service marketing perspective. *In Proceeding of the 16th International Academic MindTrek Conference*, New York, NY, USA: ACM, MindTrek '12, 17–22. Available from: http://doi.acm.org/10.1145/2393132.2393137 [Accessed: 2016-03-15].

IDC, 2015. Smartphone os market share, 2015 q2. Online. Available from: http://www.idc.com/prodserv/smartphone-os-market-share.jsp [Accessed: 2016-04-05].

ISO 25010, 2011. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models [Accessed: 2016-04-21].

Ivanovic, R., 2015. The android screen fragmentation myth. Online. Available from: http://rustyshelf.org/2014/07/08/the-android-screen-fragmentation-myth/ [Accessed: 2016-03-10].

Kantar, 2015. Sales of the galaxy s5 in britain grow. Online. Available from: http://uk.kantar.com/tech/mobile/2015/february-great-britain-smartphone-sales-data/ [Accessed: 2016-03-10].

King, D., Delfabbro, P. and Griffiths, M., 2010. Video game structural characteristics: A new psychological taxonomy. *International Journal of Mental Health and Addiction*, 8 (1), 90–106.

Kruchten, P., 2001. From waterfall to iterative development—a challenging transition for project managers. *Rational Edge, Rational Software*.

Leffingwell, D. and Widrig, D., 2000. *Managing software requirements: a unified approach*. Addison-Wesley Professional.

Maximilien, E. M. and Williams, L., 2003. Assessing test-driven development at ibm. *In Software Engineering, 2003. Proceedings. 25th International Conference on*, 564–569 [Accessed: 2016-03-24].

Meyer, B., 2008. Seven principles of software testing. *Computer*, 41 (8), 99–101.

Montola, M., Nummenmaa, T., Lucero, A., Boberg, M. and Korhonen, H., 2009. Applying game achievement systems to enhance user experience in a photo sharing service. *In Proceedings of the 13th International MindTrek Conference: Everyday Life in the Ubiquitous Era*, New York, NY, USA: ACM, MindTrek '09, 94–97. Available from: http://doi.acm.org/10.1145/1621841.1621859 [Accessed: 2016-03-14].

Nava, E., 2014. Google cloud messaging is extremely unreliable for push notifications. Online. Available from: https://eladnava.com/google-cloud-messaging-extremely-unreliable/ [Accessed: 2016-04-22].

OMG, O., 2007. Unified modeling language (omg uml). *Superstructure*.

OpenSignal, 2015. Android fragmentation visualized. Online. Available from: http://opensignal.com/reports/2015/08/android-fragmentation/ [Accessed: 2016-03-10].

Standish Group, T., 1994. *The high cost of chaos*. Computerworld.

Stone, D., Jarrett, C., Woodroffe, M. and Minocha, S., 2005. *User interface design and evaluation*. Morgan Kaufmann.

Talby, D., Keren, A., Hazzan, O. and Dubinsky, Y., 2006. Agile software testing in a large-scale project. *IEEE Software*, 23 (4), 30–37.

Tang, S.-H. and Hall, V. C., 1995. The overjustification effect: A meta-analysis. *Applied Cognitive Psychology*, 9 (5), 365–404. Available from: `http://dx.doi.org/10.1002/acp.2350090502`.

Appendices

Appendix A Levels of Game Design Elements

Table 1. Levels of Game Design Elements

Level	Description	Example
Game interface design patterns	Common, successful interaction design components and design solutions for a known problem in a context, including prototypical implementations	Badge, leaderboard, level
Game design patterns and mechanics	Commonly reoccurring parts of the design of a game that concern gameplay	Time constraint, limited resources, turns
Game design principles and heuristics	Evaluative guidelines to approach a design problem or analyze a given design solution	Enduring play, clear goals, variety of game styles
Game models	Conceptual models of the components of games or game experience	MDA; challenge, fantasy, curiosity; game design atoms; CEGE
Game design methods	Game design-specific practices and processes	Playtesting, playcentric design, value conscious game design

Figure A.1: Levels of Game Design Elements

Appendix B Android Version Market Share

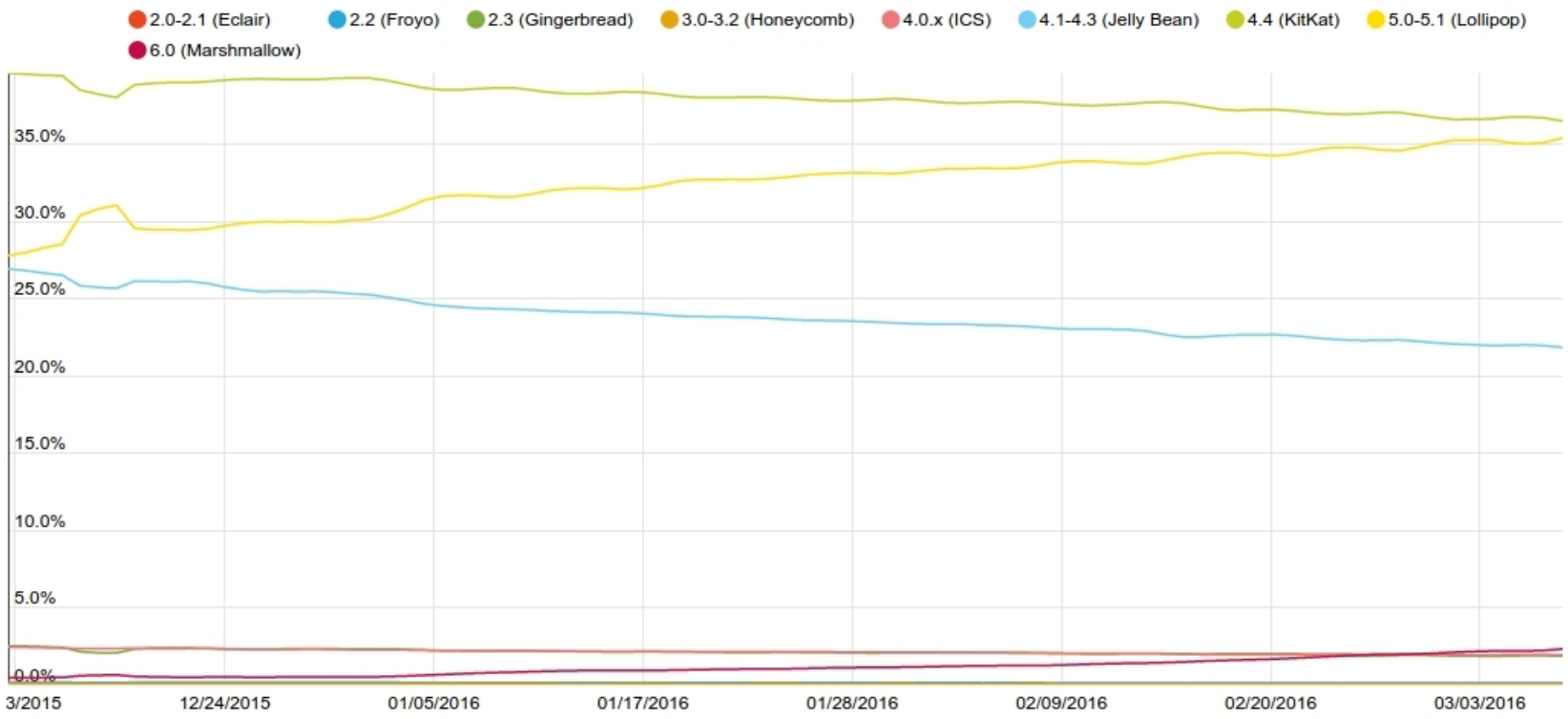

Figure B.1: Android SDK Market Share (Appbrain 2015)

Appendix C Worldwide Smartphone Market Share

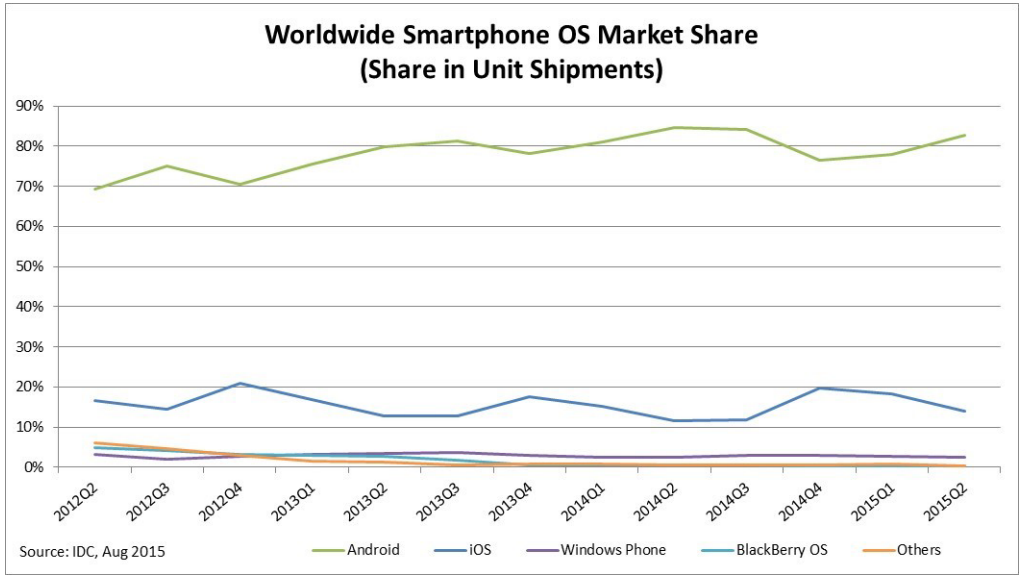

Figure C.1: Worldwide Smartphone OS Market Share (IDC 2015)

Appendix D UK Smartphone Market Share

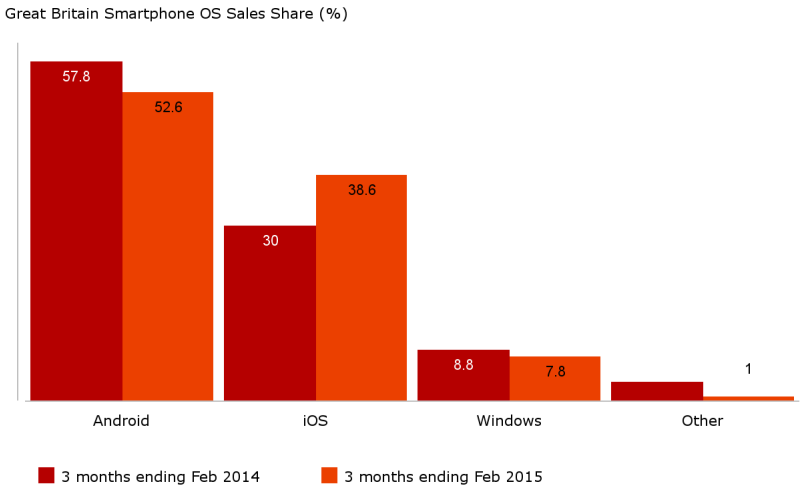

Figure D.1: Great Britain Smartphone OS Market Share (Kantar 2015)

Appendix E Wireframe Designs

Figure E.1: Your Quests Screen

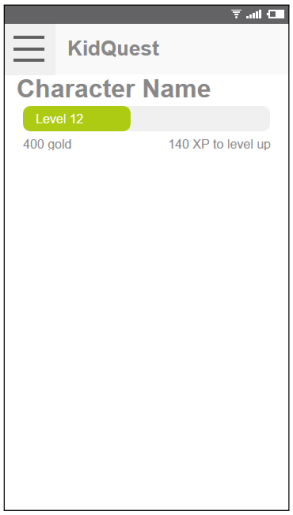

Figure E.2: Landing Page

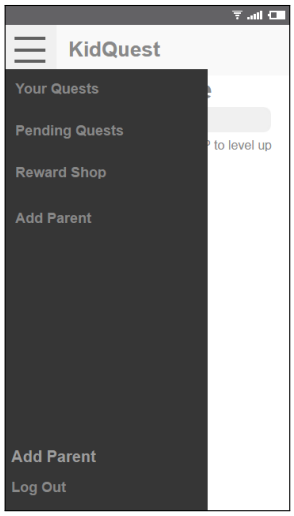

Figure E.3: Navigation Sidebar

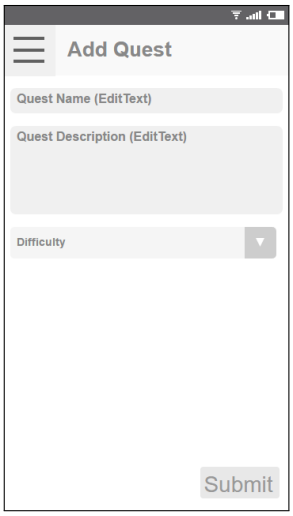

Figure E.4: Add Quest Form

Figure E.5: Wireframe Designs for the Project Artefact

Appendix F List of Planned Screens

Your Quests Screen

This will be a simple screen that displays the quests and potential rewards in an easy to read list. The screen also offers a button used to mark a quest as complete, which will send this information to the Parent for confirmation before rewards are offered.

Character Screen

This screen offers a quick overview of the user's character, gold, XP and the requirements needed to level up. This will be the first screen that a Child sees after logging in.

Navigation Menu

A navigation menu is necessary for the user to be able to change between screens with ease. The menu will be customised to the type of user - Parent or Child - and certain options will require a Parent PIN code to gain access.

Add Quest Form

This will be a simple screen for a Parent to input the necessary data about a quest and submit it to the server. The form only requires four inputs: A title for the quest; an optional longer description containing extra information; an estimated difficulty level of the task; and an expiry date. I have opted to use a difficulty field that will tell the server what the XP/gold rewards should be, rather than having Parents manually input the rewards. This is due to the fact that the Parent user is somewhat disconnected from the RPG elements of the application, and should see the app as more of a task management solution than a game. I also believe that Parent users will find manually entering values of XP/gold tiresome and would find selecting a difficulty level easier.

Add Preset Quest

This is an extension of the Add Quest form that will allow the user to automatically fill out the form by selecting a preset quest from the server.

Reward Screen

A screen that will show a list of possible real-life rewards that the Child can purchase using in game gold.

Appendix G Notifications

Event	Notifcation Text
Quest Complete	“You have completed a quest!”
Quest Confirmed	“A child you are monitoring has finished a task and requires approval”
Quest Added	“A new quest is available!”
Level Up	“You have levelled up!”
Reward Added	“New rewards are available in the store!”
Reward Purchased	“A child you are monitoring has purchased a reward from the store”

Appendix H REST Endpoint Plan

Endpoint	Request Type	Description	Authorization Required?
/users/	POST	Create a new user	N
/users/$userid$/	GET	Get details about a user	Y
/users/$userid$/	PUT	Update properties of a user	Y
/users/$userid$/quests/	GET	Get list of all user quests	Y
/users/$userid$/quests/	POST	Add new user quest	Y
/users/$userid$/quests/$questid$/	GET	Get details about a quest	Y
/users/$userid$/quests/$questid$/	PUT	Update a quest	Y
/users/$userid$/rewards/	GET	Get all user rewards	Y
/users/$userid$/rewards/	POST	Add a new user reward	Y
/users/$userid$/rewards/$rewardid$/	GET	Get a specific reward	Y
/users/$userid$/rewards/$rewardid$/	PUT	Update a specific reward	Y
/quests/getTrending/	GET	Get all trending quests	N
/quests/getStaffPick/	GET	Get preset staff pick quests	N

Appendix I Example Requests

I.1 Create new user

`POST http://kitari.ddns.net:5000/api/users/ No Authorization Required`

Required JSON:

```
{
  "email": "child@child.com",
  "password" "testpassword"
}
```

I.2 Create new quest request

`POST http://kitari.ddns.net:5000/api/users/1/quests/ Requires Authorization`

Required JSON:

```
{
  "title": "Clean your room",
  "description": "test",
  "expiryDate": "2016-04-19 13:04:47",
  "difficultyLevel": "VERY_EASY",
}
```

I.3 Get user details request

`GET http://kitari.ddns.net:5000/api/users/1/ Requires Authorization`

Example Response:

```
{
  "character_level": 1,
  "character_name": "TestCharacter",
  "children": null,
  "email": "child@child.com",
  "gold": 0,
  "id": 1,
  "parent": {
```

```
    "email": "parent@parent.com",
    "id": 2
  },
  "parent_pin": 1234,
  "quests": [
    {
      "completed": false,
      "completed_date": null,
      "confirmed": false,
      "created_date": "2016-04-20 21:58:53",
      "current_reward": 0,
      "description": "test",
      "difficultyLevel": "VERY_EASY",
      "expiryDate": "2016-04-19 13:04:47",
      "gold_reward": 100,
      "id": 3,
      "title": "Clean your room",
      "xp_reward": 100
    }
  ],
  "xp": 0,
  "xp_required": 100
}
```

Appendix J HTTP Response Codes

Upon receiving and processing a request, the server will return a response code to inform the user if the request was accepted or rejected. The codes are designed by Fielding and Reschke (2014) to give the sending machine context on what happened to the request. For example, a user's phone can send a quest to the server and check for the response code. If the code is 201, the phone knows it can show a success message and tell the user the quest was saved. A brief explanation of the response codes used in the server are listed below.

Response Code	Description
200 Success	Request has been accepted and is successful.
201 Created	Object has successfully been created in the server.
400 Bad Request	Request contains errors or is attempting to access an object that does not exist.
401 Not Authorized	Incorrect email or password, or not authorized to perform actions on that account.
404 Not Found	URL does not exist on server.
405 Method Not Allowed	Request method (GET/POST/PUT etc.) is not supported on this endpoint.
409 Conflict	Request could not be processed due to a conflict, e.g. Creating a user with email that already exists.
500 Internal Server Error	An error has occured within the server that means the request could not be processed.

Appendix K Screenshots

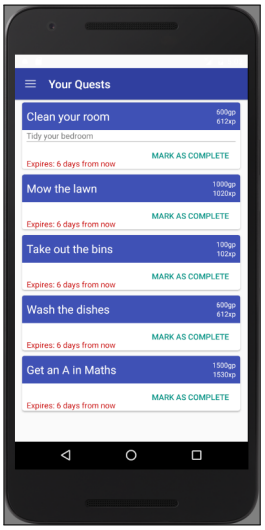

Figure K.1: Your Quests Screen

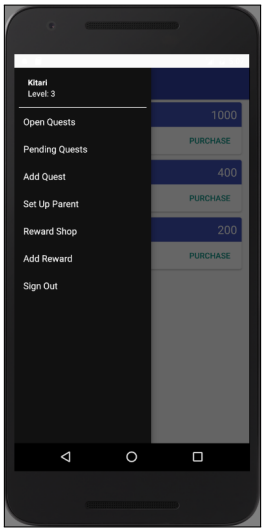

Figure K.2: Navigation Sidebar

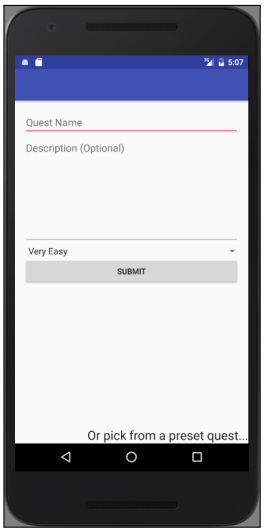

Figure K.3: Add Quest Form

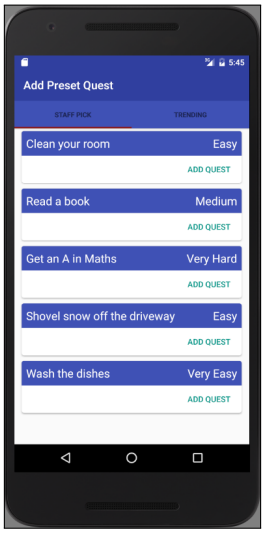

Figure K.4: Add Preset Quest Screen

Figure K.5: Screenshots of the Project Artefact

Appendix L Test Plan

L.1 Login Tests

No.	Description	Expected Outcome	Actual Outcome (If failed)	Test Data
1	Check the application launches	Application launches to login screen		Launch from homescreen
2	Check user can register	User receives “Registered” notification and user appears in server		Register with test@test.com and testpassword
3	Check user can login	The user can login successfully		Login with Test 2 details
4	Check user is rejected with incorrect password	“Incorrect email or password” is displayed	App hangs on sign in	Login with test@test.com and badpassword
5	Check user is rejected with incorrect email	“Incorrect email or password” is displayed	App hangs on sign in	Login with badtest@test.com and testpassword
6	Check user is auto-signed in	User should be automatically logged into the app		Login, then close and reopen app
7	Check user can sign out	App should revert to the login screen		Press sign out button in nav bar
8	Check user is not auto-signed in after manual sign out	App launches to login screen without auto-login		Press sign out button in nav bar, close and reopen app

L.2 Quest Tests

No.	Description	Expected Outcome	Actual Outcome (If failed)	Test Data
1	Check adding quest	Quest is added to server and displayed on Open Quests list		Add quest with title ‘Clean your room’, difficulty ‘Medium’
2	Check quests can be marked as completed	Quest moves from Open Quests to Pending Quests list		Click ‘Mark as Complete’ on quest in Open Quest list
3	Check quest can be confirmed	Quest disappears from both lists and XP/Gold rewards are given		Click ‘Confirm Quest‘ on quest in Pending Quest list. Enter correct PIN code
4	Check expiry date	Quest is removed from open quests list		Add quest then edit quest expiry date in server database
5	Check quest cannot be confirmed without PIN code	“PIN does not match” is shown, user does not change from current screen		Click ‘Confirm Quest’ on quest in Pending Quest list. Enter incorrect PIN code
6	Check quest cannot be added without PIN code	“PIN does not match” is shown, user does not change from current screen		Click ‘Add Quest’ on navigation bar. Enter incorrect PIN code
7	Check invalid quest - no title	Quest is not stored in server and “Must enter title” notification is shown		Add quest with no title
8	Check invalid quest - no expiry date	Quest is not stored in server and “Must enter expiry date” notification is shown	Invalid test, app defaults field to a week and cannot be removed	Add quest with no expiry date

L.3 Rewards Tests

No.	Description	Expected Outcome	Actual Outcome (If failed)	Test Data
1	Check adding rewards	PIN code check is shown, reward is added to server		Click ‘add reward’ in nav bar, add reward with title “Test Reward” and cost ‘100’
2	Check invalid reward - no name	Reward is not stored in server and “Must enter name” notification is shown		Add reward with a cost of 100 but no title
3	Check invalid reward - no cost	Reward is not stored in server and “Must enter cost” notification is shown	Reward is stored in server with cost of 0	Add reward with title of ‘Test reward’ but no cost
4	Check invalid reward - cost of 0	Reward is not stored in server and “Cost must be greater than 0” notifcation is shown	Reward is stored in server with cost of 0	Add reward with title “Test Reward” and cost ‘0’
5	Check invalid reward - incorrect pin	“PIN does not match” is shown, user does not change from current screen		Click ‘Add reward’ in nav bar, enter incorrect PIN code
6	Check purchasing reward	Reward is purchased, player gold is deducted by 100		Click ‘purchase’ reward
7	Check purchasing invalid reward - not enough gold	“Not enough gold” notification displayed, player gold is unchanged	Gold is deducted regardless, player in negative gold	Create new character with 0 gold, add new reward and attempt to purchase

L.4 General App Tests

No.	Description	Expected Outcome	Actual Outcome (If failed)	Test Data
1	Check navigation bar works	Each item takes user to the correct screen		Click each element in the navigation bar
2	Check first time set up	User is prompted to enter character name and parent PIN code		Clear app data in Android settings and register new account
3	Check parent restricted navigation items	Parent PIN code prompt appears		Click ‘Add Quest’, ‘Add Reward’ and ‘Set Up Parent’ in the navigation bar
4	Check gold display on character screen	‘Gold: 100’ is displayed on character screen or similar value		Complete and confirm a ‘Very Easy’ level quest
5	Check XP display on character screen	‘XP: 100’ is displayed on character screen or similar value		Complete and confirm a ‘Very Easy’ level quest
6	Check levelling up	Character level is 2 and XP is 300/600		Create new account, complete and confirm a ‘Medium’ level quest

L.5 Parent App Tests

No.	Description	Expected Outcome	Actual Outcome (If failed)	Test Data
1	Check setting up parent	Parent is logged in and can access the quests/rewards of the attached Child		Login as Child, click ‘Set up Parent’, enter parent pin and create Parent account
2	Check adding quest	Quest is added to server, Parent is not prompted for PIN, Child is notified		Click ‘Add Quest’ and input valid quest
3	Check adding reward	Reward is added to server, Parent is not prompted for PIN, Child is notified		Click ‘Add Reward’ and input valid reward
4	Check confirming quest	Quest is confirmed on server, Child is given reward, Quest disappears from Pending Quests, Child user is notified		Navigate to ‘Pending Quests’ and click ‘Confirm Quest’

Appendix M Project Proposal

BU Computing Programmes 2015-2016

Undergraduate Project Proposal Form

Degree Title: **Software Engineering**	**Student's Name: Michael Porter**
	Supervisor's Name: Damien Fay
	Proposed Project Title: Gamifying Childhood - Motivating Children to Achieve Through Video Games

1. What are the project's aims?

1.1 Problem definition - use one sentence to summarise the problem:
Parents often find it difficult to motivate children to perform chores in a timely manner.

1.2 Background - please provide brief background information, e.g., client:
I want to create a mobile application that will allow parents to assign tasks and rewards to their children in the form of 'quests' in a role-playing game. The quests can take the form of "Tidy your room" or "Complete your homework" and offer rewards of experience points and gold to level up the child's in-game avatar. Quests can have an expiry date and give an optional penalty if not completed in time or to a required standard.

The key interactions in this app will be the parent(s) inputting quests into the game and marking a quest as complete once they have inspected the work. The child will also be able to view quests, notify the parents that a quest requirement is ready for inspection and choose skills and equipment for their character.

1.3 Aims and objectives – what are the aims and objectives of your project?
Aim: To provide a fun, gamified way of motivating children to achieve more.
Objectives:

- Make a well designed Android application, adhering to the Android Developer specifications.
- Create a server that can handle many concurrent requests per minute to track the statistics of tasks being completed.

2. What is the artefact that you intend to produce?

The artifact will be an android app that provides a framework for parents to input chores for their children and assign them to be completed by a certain date. The app will also double up as a game that will allow children to earn rewards for completing the chores. In the interest of timesaving, the app will exist as a proof-of-concept only, and will not be a fully featured game. I will also need to create a server backend in either Java or Python to handle the statistics gathering involved in the application.

BU Computing Programmes 2015-2016

3. How are you going to evaluate your work?

I plan on using several friends and acquaintances of mine who have children to assess the quality of the features before I begin work. I will also use friends who have a strong interest in video games to judge the quality of the gaming side of the app through traditional beta and usability tests.

I have opted not to use children to beta test the app in the interest of the University's ethics code.

4. Why is this project honours worthy?

To create this project, I will need to use all of the skills I have learned from my degree collectively.

I also believe this project is honours worthy as I intend to create software that I believe has real world applications in aiding parents to raise children.

5. How does this project relate to your degree title outcomes?

This project will have a strong emphasis on all stages of the software development process. I will need to have this project planned out in it's entirety before starting the project, due to the complex nature of the app.

I will need to use a wide variety of technologies to fulfill the task, including database design, Android/Java and Python development and security.

I will need to ensure that my project is sufficiently tested and meets the criteria for good quality software, which is a strong facet of the Software Engineering degree title.

6. What are the risks in this project and how are you going to manage them?

The key risk in this project is re-learning how to develop an Android application. Whilst I have recently worked with Java, it has been a long time since I have developed an Android app and this will certainly be the most complex app I will have created.

Another main risk is the 'trending tasks' feature, which will require data analysis to generate that I am not familiar with. However, to combat this, Damien has offered advice and help with this.

Appendix N Proposed Project Plan

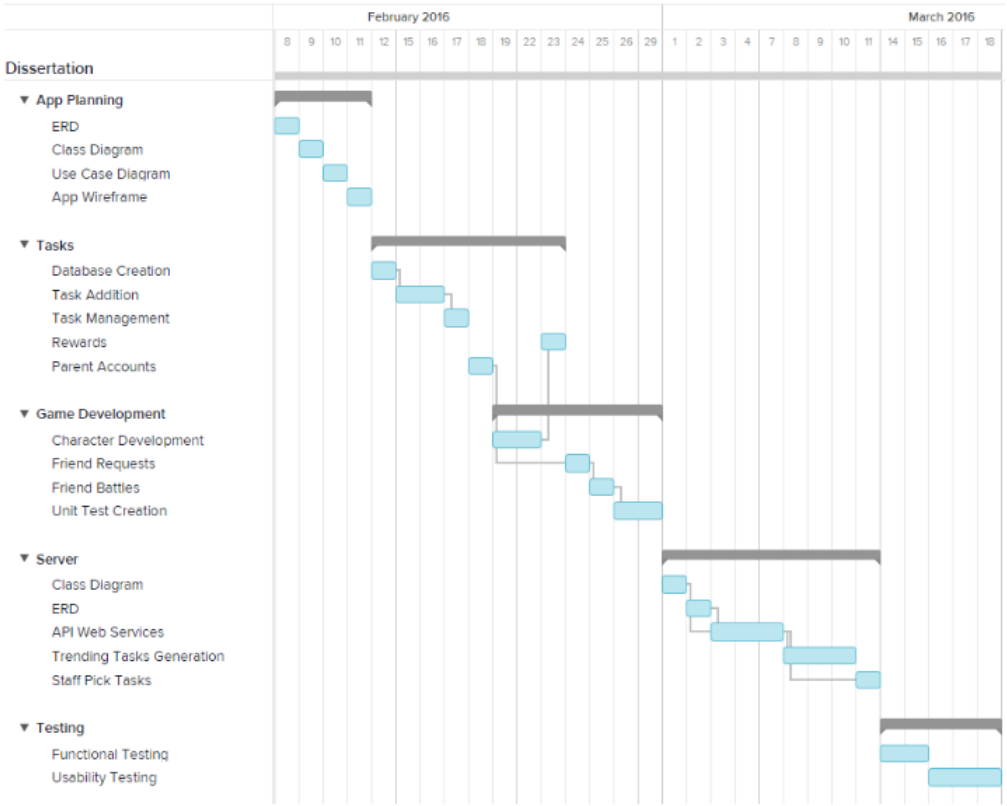

Figure N.1: A proposed schedule for the artefact development

Appendix O Ethics Form

Appendix P Attached CD Contents

P.1 Contents

- Project Report
- Artefact
  - KidQuest (Mobile Application)
    - README file
    - Compiled APK file
    - Java Source Code
  - KidQuestServer (Server Application)
    - README file
    - Python Source Code