Electronics and Computer Science
Faculty of Physical Sciences and Engineering
University of Southampton

Thomas Smith, tcs1g20
April 25, 2023

Using Blockchain for Video Game Distribution

Project Supervisor: Leonardo Aniello
Second Examiner: Heather Packer

A project report submitted for the award of
BSc Computer Science

Abstract

Centralised video game marketplaces are in a large cost to game developers, are at a greater risk of censorship, and result in brittle ownership of games that is fully dependent on the platform.

Using blockchain technology, IPFS and a custom peer-to-peer file sharing network this project creates a proof-of-concent decentralised video game marketplace. This application allows for the uploading, updating and purchasing of games through direct interactions between developers and users.

This shows the potential for a decentralised video game marketplaces and how they offer an alternative to existing platforms that benefit both developers and their users.

Statement of Originality

- I have read and understood the ECS Academic Integrity information and the University's Academic Integrity Guidance for Students.
- I am aware that failure to act in accordance with the Regulations Governing Academic Integrity may lead to the imposition of penalties which, for the most serious cases, may include termination of programme.
- I consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to verify whether my work contains plagiarised material, and for quality assurance purposes.

You must change the statements in the boxes if you do not agree with them.

We expect you to acknowledge all sources of information (e.g. ideas, algorithms, data) using citations. You must also put quotation marks around any sections of text that you have copied without paraphrasing. If any figures or tables have been taken or modified from another source, you must explain this in the caption and cite the original source.

I have acknowledged all sources, and identified any content taken from elsewhere.

If you have used any code (e.g. open-source code), reference designs, or similar resources that have been produced by anyone else, you must list them in the box below. In the report, you must explain what was used and how it relates to the work you have done.

I have not used any resources produced by anyone else.

You can consult with module teaching staff/demonstrators, but you should not show anyone else your work (this includes uploading your work to publicly-accessible repositories e.g. Github, unless expressly permitted by the module leader), or help them to do theirs. For individual assignments, we expect you to work on your own. For group assignments, we expect that you work only with your allocated group. You must get permission in writing from the module teaching staff before you seek outside assistance, e.g. a proofreading service, and declare it here.

I did all the work myself, or with my allocated group, and have not helped anyone else.

We expect that you have not fabricated, modified or distorted any data, evidence, references, experimental results, or other material used or presented in the report. You must clearly describe your experiments and how the results were obtained, and include all data, source code and/or designs (either in the report, or submitted as a separate file) so that your results could be reproduced.

The material in the report is genuine, and I have included all my data/-code/designs.

We expect that you have not previously submitted any part of this work for another assessment. You must get permission in writing from the module teaching staff before re-using any of your previously submitted work for this assessment.

I have not submitted any part of this work for another assessment.

If your work involved research/studies (including surveys) on human participants, their cells or data, or on animals, you must have been granted ethical approval before the work was carried out, and any experiments must have followed these requirements. You must give details of this in the report, and list the ethical approval reference number(s) in the box below.

My work did not involve human participants, their cells or data, or animals.

Acknowledgements

I would like to thank my supervisor, Leonard Aniello, for his continued support and advice throughout this project. He has been a big part of making this project so great. I would also like to thank Heather Packer on her input for my final submission.

Contents

Chapter 1

Introduction

Millions of worldwide users enjoy video games, which are large pieces of software that require complex platforms to distribute them, which results in them being generally provided by multinational corporations. However this approach often results in these platforms:

- taking a large cut of revenue from developers[1],
- being prone to censorship from entities like governments[2],
- relying on a single platform to stay active, distribute games and maintain a user's ownership[3].

1.1 Objectives

Modern web ideas revolve around taking power away from large corporations and having platforms that are built, run and maintained by those who use them. This project aims to produce a proof-of-concept, distributed video game marketplace that will allow developers to continuously release and update their games on a public network where they can interact directly with their users.

1.2 Scope

This project will strictly focus on creating a distributed platform in which users can upload, purchase and share games and will consist of the following components:

1. An Ethereum smart contract that will allow us to maintain a library of games that can be queried and added to by any user. This will then be deployed to an Ethereum test-net.
2. A local application to be run by users to interact with the smart contract and allow them to join a peer-to-peer network where they can download and upload games.

This project will not present methods for preventing or stopping the distribution of illegal content or include tangentially related features such as achievements, or message boards.

[1] Steam take a 30% cut [1, 2]
[2] See the Chinese version of Steam [3]
[3] For example, the Nintendo eShop closing down [4]

Chapter 2

Literature Review

This section will be used to examine the literature surrounding the distributed storage of data. Initially I will look at various distributed file-sharing protocols and then how blockchain has been used to enhance them.

2.1 P2P File Sharing

One finding from the previous section was that a lot of blockchain-based cloud storage systems rely on an external form of distributed file-sharing. Therefore it is important for me to understand the existing options, their architecture, as well as their pros and cons to determine the best implementation for this project.

Table 2.1 looks at various implementations of P2P file-sharing networks.

System	Description of Solution
IPFS [5]	IPFS is a set of protocols for transferring and organising data over a content-addressable, peer-to-peer network. IPFS is open source and has many different implementations, such as Estuary [6] or Kubo [7].
Swarm [8]	Swarm is a distributed storage solution linked with Ethereum that has many similarities with IPFS [9]. It uses an incentive mechanism, Swap (Swarm Accounting Protocol), that keeps track of data sent and received by each node in the network and then the payment owed for their contribution.
BitTorrent [9]	See Section 3.1.
AFS [10, 11]	The Andrew File System was a prototype distributed system by IBM and Carnegie-Mellon University in the 1980s that allowed users to access their files from any computer in the network.
Napster [12]	Napster uses a central cluster of servers that maintain an index of every file on the network and which users have a copy of it. Clients will query the cluster for this information and will choose peers based upon their bandwidth. This protocol is dependant on a central cluster existing and will not function properly without it.

Gnutella [12]	In Gnutella, nodes form an overlay network and will discover other peers through *ping-pong* messages, where any node that receives a *ping* message will forward it to their neighbours and send a *pong* message to the originator. A user will flood a download request to their peers until they find a suitable peer to download off of. A network flood in a large network will result in a significant amount of congestion and and a large amount of *pong* messages going to a single node results in a DDoS attack.

Table 2.1: *Various global distributed file systems.*

Some of the common themes found include:

- **Trust** Nodes are typically anonymous so users have to trust that the data they download isn't malicious.
- **Illegal Content** There are no measures in place to stop or prevent the distribution of illegal content across these networks.
- **Payment** None of these networks natively support payment and thus don't support the access control systems required for payment systems.

It is clear that using blockchain in conjunction with these systems could mitigate a lot of their issues and thus a combination of these two ideas will be used for this project.

2.2 Blockchain-Based Cloud Storage

This project will need to maintain an immutable and public record across a P2P network such that it searchable and can be trusted. Table 2.2 shows how blockchain technology has been used to provide a solution distributed cloud storage systems.

Paper	Description of Solution
Blockchain Based Data Integrity Verification in P2P Cloud Storage [13]	The Ethereum blockchain is used to add trust to a data verification system for a P2P network. It analyses how varying structures of Merkle Trees affect the performance of verification of data stored in the network.
Deduplication with Blockchain for Secure Cloud Storage [14]	This paper implements a deduplication scheme by uploading storage information to Ethereum and uses smart contract based protocols to provide secure deduplication over encrypted data.
Block-secure: Blockchain based scheme for secure P2P cloud storage [15]	Users divide their own data into encrypted blocks and upload them randomly into a blockchain, P2P network. It uses a custom genetic algorithm to solve the file block replica placement problem and ensure data availability.
Blockchain-Based Medical Records Secure Storage and Medical Service Framework [16]	Describes a blockchain-based platform that would allow for the secure and immutable storage of user medical records, such that they are independant from any individual medical institution and users have greater control over who can access their personal data.

A Blockchain-Based Framework for Data Sharing With Fine-Grained Access Control in Decentralized Storage Systems [17]	An attribute-based encryption scheme that allows the distribution of access keys to users based upon their allocated groups. Data is uploaded to IPFS and an Ethereum smart contract is used to implement a keyword search of the data stored.
Blockchain based Proxy Re-Encryption Scheme for Secure IoT Data Sharing [18]	A distributed cloud system for encrypted IoT data that uses a proxy re-encryption scheme that allows the data to only be visible to the owner and any persons present in the smart contract.

Table 2.2: *Examples of blockchain cloud storage systems [19]*

Some common themes found include:

- **Access Control** If these platforms offer access control it is typically up for the owner of the data to distribute access rights.
- **Data Integrity** The blockchain is primarily used to add trust across a distributed network in the form of immutability or providing data integrity services.
- **Data Location** Data stored on the blockchain is commonly metadata that refers to some data stored elsewhere, typically on a form of distributed storage network.

2.3 Overview

Chapter 3

Background Research

This chapter will consider two popular technologies from Chapter 2 to provide key ideas on how I could use ideas from them to build this project.

3.1 BitTorrent

BitTorrent [20, 9] is a protocol for sharing data across a distributed network[1] and is one of the most popular P2P file-sharing protocols active today, being responsible for 3.35% of global bandwidth [24]. As such, it is important to understand what makes BitTorrent so successful and how I could include key ideas in my project. In BitTorrent, users barter for blocks of data from a network of peers in a tit-for-tat fashion, such that users with a high upload rate will also typically have a high download rate.

Key Ideas

Trackers A tracker server keeps track of which peers have what data, which of those peers are available at, and will provide network statistics to *recommend* which peers to connect to first.

Block Priority Users will download blocks from other peers using the following priority:

1. **Strict Priority** Data is split into pieces and sub-pieces where pieces are ideally downloaded together.
2. **Rarest First** Download pieces that have the fewest copies to boost availability.
3. **As Soon As Possible** A user with no pieces will try to get a random piece quickly so they can contribute to the network.

Optimistic Unchoking A peer allocates a portion of their bandwidth for communicating with unknown peers. This allows new users to join the network and be able to contribute without being ignored and gives a way for existing peers to seek better peers.

Availability

One of the most significant issues facing BitTorrent is the availability of torrents, where *'38% of torrents become unavailable in the first month'* [20] and that *'the majority of*

[1] Popular implementations include BitTorrent Web [21], qBittorrent [22], and µTorrent [23]

users disconnect from the network within a few hours after the download has finished' [9]. This paper [25] looks at how the use of multiple trackers for the same content and DHTs can be used to boost availability. However, good availability requires users to *choose* to contribute and often the built-in incentives aren't enough to encourage users to contribute for a long period of time.

3.2 Ethereum

A large amount of the platforms discussed in Section 2.2 use Ethereum smart contracts so this section will look at what it is and how it can be used to write distributed applications.

Ethereum [26, 27] is a distributed transaction-based blockchain that comes with a built-in Turing-complete programming language that allows any user to design their own transactions. Each block will include a list of transactions, where each transaction includes bytecode that can be run by each node in the network to update their copy of the global state.

Smart Contracts

A smart contract is an executable piece of code, usually written in Solidity [28], that will automatically execute on every node in the Ethereum network when certain conditions are met. Smart contracts are enforced by the network, remove the need for intermediaries and reduce the potential of contractual disputes, due to their transparency and immutability.

Gas is the computational effort of running a smart contract and must be paid, in Ether, before a transaction can be processed and added to the blockchain. This helps prevent DoS attacks and provides economic incentives for users to behave in a way that benefits the whole network.

Miners receive Ether for mining transactions based upon their gas price, which results in gas price varying according to supply and demand. For example, in a period of congestion users will offer a higher gas price to have their transaction be processed more quickly.

Test Networks

An Ethereum test network is an instance of Ethereum in which users can deploy their smart contracts and test them in a live environment. Ether for these networks can be gained for free from a faucet provided by a node from the network. Some notable examples include Sepolia [29], Goerli [30], and Ropsten [31].

Chapter 4

Design

This section will identify the key requirements for this project, describe a design that can be used to meet them, and finally a breakdown of an application based upon that design.

4.0.1 Stakeholders

Stakeholders allow us to understand the different types of user by looking at what they will use the platform for, as well as what they want and don't want from it.

Game Developers PRIMARY
Users who upload (and update) games to the platform that will be purchased by other users who want access to it. As the content providers for this platofrm, they are essential in attracting users.

Players PRIMARY
This group will purchase games, then upload and download game data to other other users in the network. They will be responsible for the long-term availability of games and making the platform appealing to developers.

Other Platforms SECONDARY
Other digital marketplaces that serve as the main competitors to this platform. Users will expect a similar experience as found on these other platforms so this project will share several key ideas from them.

4.0.2 Requirements

To determine the requirements of this project, I used a mind-map to show how the primary stakeholders, key pieces of data, and predicted technologies would relate to one another and what constraints would be on those relations. This can be seen in Appendix C.

Tables 4.1 and 4.2 show the functional and non-functional requirements of this project organized using MoSCoW prioritisation.

Functional

ID	Description
	Must
F-M1	Developers must be able to release games by uploading metadata to the Ethereum blockchain.
F-M2	Developers must be able to release updates to their existing games.
F-M3	An owner of an existing game must be an owner of all future updates to that game.
F-M4	This application must include a smart contract that is deployable to the Ethereum blockchain[1].
F-M5	Users must be able to purchase games off of develoeprs.
F-M6	Users must be able to prove they have purchased a game.
F-M7	Users must be able to create and maintain many concurrent connections to other users.
F-M8	A user must be able to communicate with other users by exchanging structured messages.
F-M9	A user must be able to upload and download data to and from other users.
F-M10	A user must be able to verify the integrity of all data that they download.
F-M11	A user must be able to download games in their entirety.
F-M12	A developer must be able to upload a hash tree[2] of a game such that all users can access it.
	Should
F-S1	Users should only upload game data to users who own that game.
F-S2	Users should interact with the application using a GUI.
F-S3	Users should be able to prove the amount of data that they have uploaded to other users.
F-S4	Users should have a way to discover new peers from their existing ones.
	Could
F-C1	Developers could be able to release downloadable content (DLC) for their games.
F-C2	Allow developers to upload promotional materials such as cover art and an overview to be shown to the user.

Table 4.1: *These requirements define the functions of the application in terms of a behavioural specification*

Non-Functional

ID	Description
	Must

[1]This project will only test deploying to an Ethereum test network
[2]See Section 4.1.3.

NF-M1	This application must be decentralised and cannot be controlled by any singular party.
NF-M2	Any user must be able to join and contribute to he network.
NF-M3	Developers who upload games to the network must be publicly identifiable.
NF-M4	The data required to download a game must be immutable.
NF-M5	Only the original uploader must be able to make any changes or release any updates to a game.
	Should
NF-S1	This application must be scalable, such that many users can upload and download the same game at the same time.
NF-S2	This application's GUI should be intuitive to use for new users.
	Could
NF-C1	The GUI could include detailed support and or instructions for new users.

Table 4.2: *Requirements that specify the criteria used to judge the operation of this application*

4.1 Design Considerations

This section will give a high-level design showing how each of the functional requirements can be met by considering key functions for the applications.

4.1.1 Data Types

Table 4.3 discusses the different types of data we are going to need to store and where they should be stored based upon their properties.

Data	Size	Location	Explanation
Game Metadata (**F-M1**)	100 – 200B	Blockchain	The minimal set of information required for the unique identification of each game. See Section 4.1.2. This data is appropriate to store on the blockchain as it is public, small in size, and essential to the correct functioning of the application as all users will need to be able to discover all games.
Game Hash Tree (**F-M12**)	˜15KB	IPFS	The hash tree that will allow users to identify and verify blocks of data they need to download for a game. The user will download this immediately after purchasing the game. This data is public but its size makes it costly to store on the blockchain at a large scale. IPFS will be used for fast, reliable access at a large scale and we can store the generated CID in the blockchain instead.

Game Assets (**F-C2**)	Variable[3]	IPFS	Any promotional material provided for the game that can be viewed on the game's store page. This should include cover art and a description file but isn't required to purchase or download the game. The user will download this when they first view the game in the store. For similar reasons as the hash tree, this data will be also be stored on IPFS and have its CID stored on the blockchain instead.
Game Data	*avg. 44GB* [4]	Peers	the data required to run the game that is fetched based upon the contents of the game's hash tree. This data is very large and has restricted access so wouldn't be appropriate to store on either the blockchain or IPFS. Therefore, this project will use a custom P2P network for sharing data, which is described in Section 4.1.3.

Table 4.3: *The different types of data required for each game.*

4.1.2 Blockchain

This section will describe how blockchain technology will enable the storage of game metadata and the purchasing of content using a distributed immutable record that can be trusted by any user.

To satisfy (**NF-M1**) and (**NF-M2**), we will need to use a public blockchain. This will benefit my project by:

- being accessible to more users, which will boost both availability and scalability (**NF-S1**),
- reducing the risk of censorship (**NF-M1**), and
- providing greater data integrity (**NF-M4**)

Ethereum is a public blockchain that allows developers to publish their own distributed applications to it. It comes with an extensive development toolchain so is an obvious choice for this project (**F-M4**).

Uploading Games

To satisfy (**F-M1**) and (**F-M2**), the data stored on the blockchain will be used for the identification of games. Table 4.4 shows the fields that will stored as part of the smart contract for each game and to manage the whole collection of games. Fields in *italics* are generated for the user and non-italic fields are entered manually.

Name	**Description**
	Metadata for each game
title	The name of the game.

[3]Some games may include many promotional materials, whilst some could include none. Therefore, it is hard to estimate the expected size.

[4]Calculated based off of the top 30 games from SteamDB [32].

version	The version number of the game.
release date	The timestamp for when the game was uploaded.
developer	The name of the developer uploading the game (**NF-M3**).
uploader	The Ethereum address of the developer (**NF-M3**).
root hash	The root hash of the game that uniquely identifies the game and is based upon its contents.
previous version	The root hash of the most previous version of the game if it exists.
next version	The root hash of next update to this game.
price	The price of the game in Wei.
hash tree CID	Required for downloading the hash tree from IPFS.
assets CID	Required for downloading the assets folder from IPFS.
Managing the Collection of Games	
library	A mapping for all games uploaded to the network, where a game's root hash is the key used to find its metadata.
game hashes	Solidity doesn't allow us to enumerate maps so we will also store a list of hashes for all games uploaded.
purchased	A mapping which allows us to easily check if a user has purchased a game (**F-M6**).

Table 4.4: *the data to be stored on Ethereum using a smart contract*

Purchasing Content

Users will purchase games from developers over Ethereum by transferring Ether (**F-M5**). The user's address will then be added to a public record of all users who have purchased the game (**F-M6**).

4.1.3 Distributed File Sharing

This section will look at how various types of game data can be shared across a distributed set of peers using a content-addressable hash tree to describe the data.

Hash Tree

The hash tree of a given directory is used to represent its structure as well as the contents of its files. Each file is represented by an ordered list of SHA-256 hashes that match a fixed-size block of data. This allows users to easily identify and verify game data (**F-M10**). Users should be able to specify which folders/files to exclude when generating a hash tree.

Hash trees will be stored on IPFS with the CID being kept with the other game metadata on Ethereum.

Downloading Content

Games are content addressable using their root hash field, which will allow users to request data from that game from other users. When a peer seeking data $P_{downloader}$ forms a connection with another peer P_{seeder} they will:

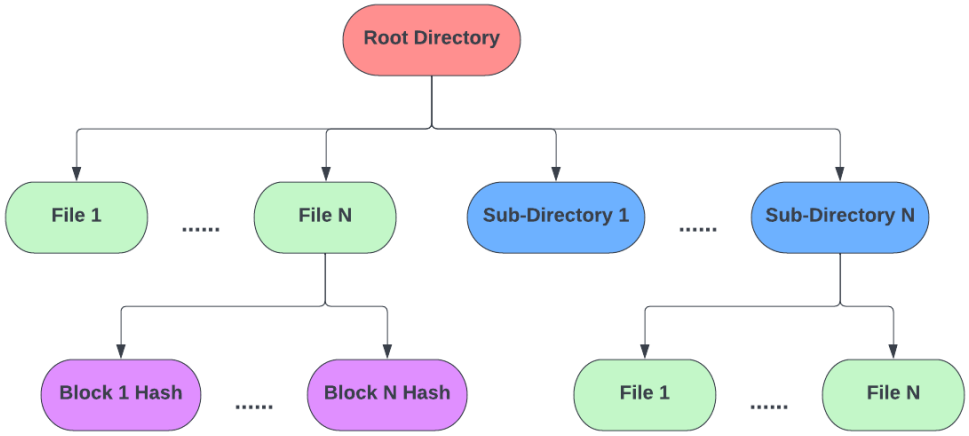


Figure 4.1: *The structure of a hash tree*

1. Perform a handshake to determine each other's Ethereum address and public key.
2. $P_{downloader}$ will send requests for individual blocks to P_{seeder} (**F-M9**).
3. Upon receiving the first request, P_{seeder} will verify that $P_{downloader}$ owns the game by checking the *purchased* mapping on the smart contract (**F-M6**) (**F-S1**).
4. Upon receiving a block, $P_{downloader}$ will verify the contents using the block's hash (**F-M10**) before writing it to disk in the appropriate location.
5. Repeat Steps 3–4 until the entire game has been downloaded (**F-M11**).
6. P_{seeder} may request a signed receipt that details the blocks they uploaded (**F-S3**) to $P_{downloader}$.

Users will be able to connect and send requeststo many peers at once (**F-M7**). Requests will be sent in a round-robin fashion to evenly distribute the requests and prevent overloading a single peer (**NF-S1**). Requests that cannot be completed will be retried when connecting to a new peer or when a peer has a change in library.

Updating Content

To satisfy (**F-M2**), developers will upload the game and also provide the root hash of the most previous version of the game, where only the original uploader will be allowed to publish an update to an existing game (**NF-M5**). Any users who own a previous version of a game will be allowed access to all future versions (**F-M3**).

Each version is considered its own game and will require users to download the updated version separately. Whilst this isn't reflective of how updates are typically managed, this will be acceptable for the scope of this project.

Downloadable Content

Downloadable Content (DLC) (**F-C1**) represent optional additions for games that users will buy separately. DLCs will act similarly to how updates are treated. Each DLC will need:

1. **Dependency** The root hash of the oldest version of the game this DLC supports.
2. **Previous Version** (Optional) The root hash of the previous version of the DLC.

Users must own the original game to buy any of its DLC.

Proving Contribution

As a user downloads blocks of data, they will keep track of which users have sent them which blocks. A peer may then request their contributions in the form of a signed message that can be sent to the developer (**F-S3**) in return for some kind of reward. The contents of the reward isn't specified for this project but could include in-game items, digital assets or Ether. This solution assumes that developers have knowledge of which Ethereum address maps to which of their game's users.

4.2 Architecture

This section will detail the architecture and implementation details for an application based upon the design considerations given in the previous section.

This application is structured using the Model-View-Controller MVC pattern to create a separation of concern between its main layers. Figure 4.2 shows a high level overview of the architecture and below I discuss the purpose for each.

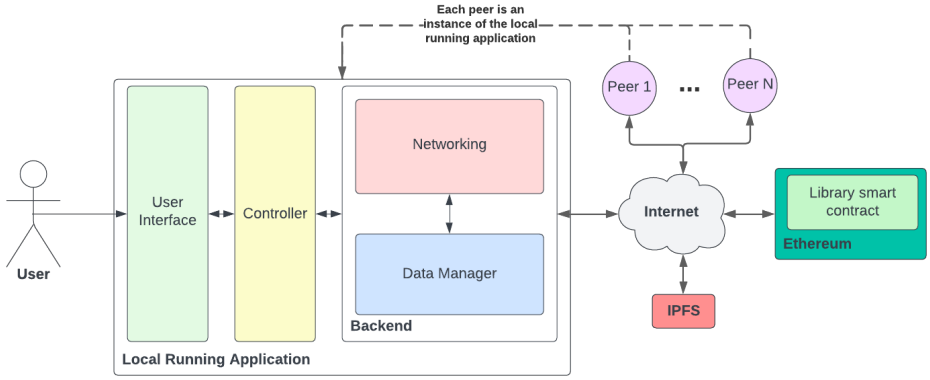


Figure 4.2: *The components of the application*

4.2.1 Persistence

The Persistence layer shows how the data for the application is divided across several mediums; namely the **Ethereum Smart Contract**, **IPFS**, and a **P2P Network**. Each component stores a different type of data, which is outlined in Section 4.1.1.

There are several things to note about using Ethereum as a platform for selling games:

- Ethereum is a less stable currency than most traditional currencies so games may fluctuate largely in price. This design describes no solution to this issue but an example solution might attempt to map a real-world currency to Ether.
- All write functions to the smart contract will incur a gas fee so uploading or updating data will not be free.
- Many people are sceptical of blockchain technology so may be hesitant to adopt the application.

This layer will also provide interface functions that will allow our backend code to interface with our smart contract. Interactions with IPFS will be very simple uploads or downloads of compressed data.

A peer's behaviour is described by the Backend in Section 4.2.2, which will be part of the application run locally by the user.

4.2.2 Backend

The Backend containts all functionality required for a peer to use and contribute to the platform and can be broken down into two major components:

- **Networking** The creation and maintenance of network connections with other peers over the internet with the purpose of locating and sharing data. See Section 4.2.2.

- **Data Manager** The management of local data and the processing of data received and to be uploaded by the Networking component. See Section 4.2.2.

Both sections will interface with Ethereum as appropriate to complete their functionality.

Networking

This component will connect users to the distributed P2P file-sharing network, where it will create and maintain a set of TCP connections with other users (**F-M7**) in the network and will communicate by sending structured messages to each other (**F-M8**). Section 4.2.2 describes these commands in detail.

Peer Identification Peers are identified by their Ethereum addresses, which will allow us to view which games they've purchased and are allowed access to. Upon forming a connection, each peer will request the other return a signature for a generated message, from which we can derive their address and public key.

Commands Structured messages (**F-M8**) will typically come as part of a request/response pair involving the sharing of information between peers. Command responses are not awaited to remove unnecessary blocking of the connection channel as a user may be responding to many different requests at once by the same peer. Table 4.5 shows the list of commands used bu the application.

Message Format	**Description**
LIBRARY GAMES;[$hash_1$];[$hash_2$];...;	Request that a peer sends their library of game. The user sends a list of their games as a series of unique root hashes. These root hashes will map to games on the blockchain.
BLOCK;[$gameHash$];[$blockHash$];	The user will request a block of data off of a peer by sending the root hash of the game and the hash of the block being requested. The response will be a SEND_BLOCK message (**F-M9**) and if it isn't received after a given amount of time then it is resent.
SEND_BLOCK;[$gameHash$]; [$blockHash$];[$compressedData$];	The user sends a block of data in response to a BLOCK message (**F-M9**). The data is compressed using the *compress/flate* package to reduce message size (**NF-S1**).
VALIDATE_REQ;[$message$]	The user is requesting for a message to be signed using the receiver's Ethereum private key. This is used to verify the receiver's identity and thus their owned collection of games (**F-S1**).
VALIDATE_RES;[$signedmessage$]	The user responds to a VALIDATE_REQ message with a signed version of the received message. From this signature, the receiver can determine the address and public key of the user (**F-S1**).
REQ_RECEIPT;[$gameHash$]	A user will request a RECEIPT message from a peer detailing the data that has been sent by the user for a specific game (**F-S3**).

RECEIPT;[*gameHash*];[*signature*] ;[*message*]	A user will respond to a REQ_RECEIPT message with a signed message detailing all of the blocks that the requester has sent to the user from a given game. This will allow for users to prove their contributions to the game developer who could then reward them (**F-S3**).
REQ_PEERS	A user requests the list of peers which the receiver peer is connected to. This will be sent immediately after a peer's identity is validated and will help increase the connectivity in the network (**F-S4**).
PEERS;[$p_1 hostname$] : [$p_1 port$];...	A user will send a list of their active peers. This will be limited to those peers which they have connected to and thus know the hostname and port of their server (**F-S4**).
SERVER;[*hostname*] : [*port*]	When we form a connection with a peer we send them the address of our server that peers use to connect to us. This allows that peer to share our address through the PEER command so we are more easily discoverable.
ERROR;[*message*]	An error message that can be used to prompt a peer to resend a message.

Table 4.5: *The set of structured messages sent between peers*

Data Manager

This component is responsible for interacting with local storage and managing the user's collection of owned and installed games.

It will interact directly with Ethereum to discover, purchase (**F-M5**), and upload (**F-M1**) (**F-M2**) games and use IPFS to upload and distribute game assets (**F-C2**) and hash trees (**F-M12**).

Download Order Each download will have a downloader thread that selects the order at which blocks are to be fetched. Using the hash tree, it will queue whole files at a time and to verify the whole file once all blocks have been fetched.

4.2.3 Frontend & Controller

Frontend

This application will have a GUI (**F-S2**) (**NF-S2**) where users can interact with the platform. Having a GUI is essential to making the platform as easy to use as possible so that it is accessible to new users. At minimum it will need to include the following pages:

- **Library** The user's collection of owned games, where they can view details for each game as well as manage their download status. Users should be able to view both old and new versions of a game and check for any new updates.
- **Store** Where user's can find and purchase new games that have been uploaded by other users. Games should be searchable by their root hash.
- **Upload** Where users can fill in details about their new or updated game and have it be uploaded to the network.

- **Downloads** Where user's can track all of their ongoing downloads and see their progress.
- **Peers** Where users can manage their list of connected peers. Here a user can form new connections, break existing ones and request contribution data from their peers.
- **Help** A help page to describe the application and all of its functionality (**NF-C1**)

To satisfy (**NF-M3**), a developer must always be displayed with both their chosen name and their Ethereum address. A developer should publically provide their Ethereum address to ensure their users can identify it.

Controller

The Controller will be represented as a set of interface functions that allow the backend and frontend code to communicate. This can be done to trigger actions such as starting a game download or to fetch data like the list of a user's owned games.

4.3 Sequence Diagram

Figure 4.3 shows how all of the components from this section are combined to download a game off of a single peer in the network.

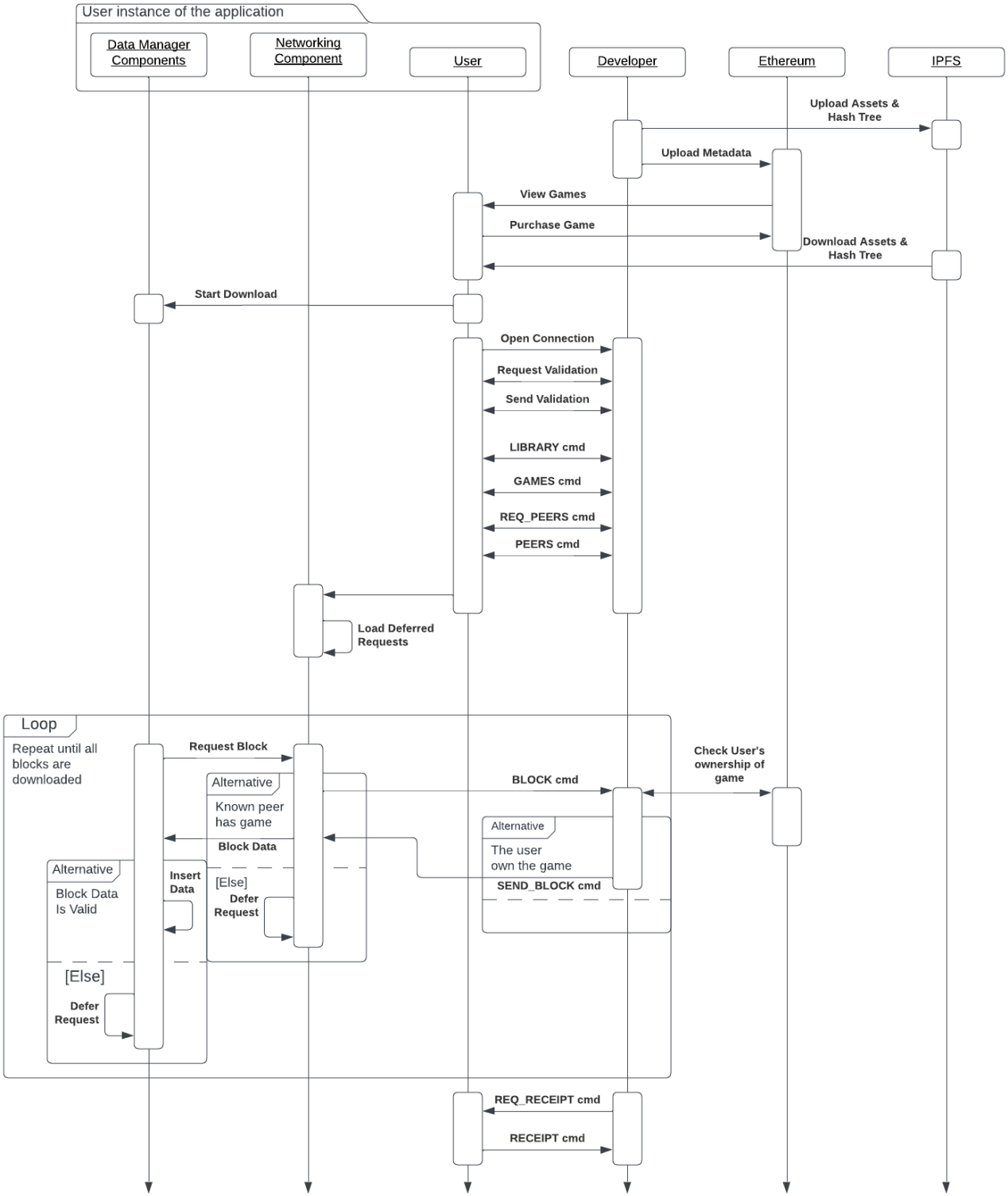


Figure 4.3: *A sequence diagram showing the main interactions needed to download a game from a single peer*

4.4 Benefits

This application presents the following benefits when compared with centralised game marketplaces:

- **Privacy** A user's personal information and usage isn't collected. Traditional platforms require users to enter personal information and will actively collect data about a user's actions through the platform.
- **Ownership** A user's ownership of a game isn't tied to a single platform and use of Ethereum means that a user's ownership is upheld by all computers in the network.
- **Censorship** Similar to the previous point, no one party has control over the platform so it is much harder for third parties, such as governments, to restrict the content uploaded to it.
- **Profits** Developers and gamers communicate directly and this means developer's won't have to pay a hefty fee for a middle-man. This will result in potentially larger profits for the developers.
- **Concurrent Downloads** Many downloads can be completed in parallel whereas a centralised platform would do them sequentially.

4.5 Limitations

This application presents the following limitations when compared with a centralised game marketplace:

- **No Social Features** Social features, such as friends or achievements, were not included within the scope of this project.
- **Availability** Section 3.1 highlights the issue of availability within P2P file-sharing systems and it is likely this platform will face similar issues. The use of a contribution system was implemented to help identify those users who have been contributing but there is no automatic rewards system[5].
- **Inefficient Updates** As updates are treated as individual games, they will require users to download the entire game again. This is highly inefficient and results in lots of duplicate data being downloaded.
- **Illegal Content** Currently there is no policy, automated or otherwise, to stop the distribution of illegal content. This is an incredibly complex problem to tackle and any obvious solution would risk violating the anti-censorship goal of this project.

[5] An example would be how the micro-payment system works in Swarm [8]

Chapter 5

Implementation

The implementation was completed according to the design specified in Chapter 4 using the collection of tools specified below. See Section 7.2 for a breakdown of how sprints were used to structure the implementation.

5.1 Backend

Table 5.1 shows the tools to create the backend of my application as it was described in Section 4.2.2.

Tool	Description & Reasoning
Go [33]	Go was chosen because of its simple syntax, high performance, strong standard library and third party packages for interacting with Ethereum.
go-ipfs-api [34]	A Go package used for interacting with the Kubo implementation of IPFS [7] that gave an easy interface for downloading and uploading data to Kubo.
go-ethereum [35]	A collection of tools used for interacting with Ethereum including an Ethereum CLI client (Geth), and a tool for converting Ethereum contracts into Go packages (abigen). This was essential for interfacing with the smart contract.
Zap [36]	A logging library that performs much better and provides easier customisation when compared to Go's standard library implementation.
Viper [37]	A configuration file management library that helps read, write, and access configuration options written to file. This makes it simple to access global configuration settings and made it easy to configure test profiles.

Table 5.1: *The tools used to develop the backend*

5.2 Smart Contract

Table 5.2 shows the tools used to create and deploy the Library smart contract as it was described in Section 4.1.2.

Tool	Description & Reasoning

Solidity [28]	The first language used to write smart contracts for the Ethereum blockchain. Most tutorials are focused around Solidity so it made the most sense to learn it for this project.
Sepolia [29]	An Ethereum test-net that used to deploy my smart contract to. One of the main benefits was that it provides a fast transaction time for quick feedback. It was chosen of Goerli [30] as the faucet gave out 5 times as much Ether.
Alchemy [38]	Alchemy provides useful tools for interacting with Ethereum and specifically Sepolia, such as an ETH faucet and an RPC URL.
MetaMask [39]	A browser-based wallet that can easily be connected to other tools such as Alchemy or Remix. This tool can generate new accounts and show you the transaction history of each using an intuitive UI.
Remix [40]	A browser-based IDE for writing smart contracts that allows for easy deployment. I did have an implementation that would deploy using go-ethereum but due to a bug in package it was unable to work for the Sepolia test-net.
Ganache CLI [41]	Ganache CLI was used to create a local Ethereum test-net that I could develop my application with. With one command I could quickly boot a lightweight blockchain with a predetermined set of keys. When compared to Geth, from go-ethereum, Ganache CLI was much more beginner friendly.

Table 5.2: *The tools used for deployment of my smart contract*

The contract was successfully deployed [42] to the Sepolia test-net and can be interacted with by any user.

5.3 Frontend

The frontend code was developed from Section 4.2.3 to provide a user a GUI to interact with. Table 5.3 details the list of tools used.

Tool	Description & Reasoning
Wails [43]	Allows you to add a webkit frontend to a Go application, so that you can use a modern web framework. This allowed me to easily create a reactive UI using tools I was previously familiar with. Wails allows you to implement a controller using functions written in that can be called from the frontend and can emit events that trigger actions in the frontend.
Vue.js v3 [44]	A reactive, component based web-framework that allows me to create reusable components that react to changes in state and can trigger events at different points in a components lifecycle. The Vue Router [45] package was used to add multiple pages to the application and markdown-it [46] was used to render markdown files. Vue was chosen due to me already having lots of experience using it.

Pinia [47]	A state management tool for Vue.js that boosts the reusability of components and reduces the overall complexity of the frontend by allowing state to be accessed globally through stores. Pinia integrates directly with Vue's composition API and is very beginner friendly and well documented, especially when considered to similar tools from other frameworks.
SASS [48]	An extension of CSS that is used to style DOM elements. This was essential in making the UI look nice and be accessible without having styling that was hard to maintain and search through.

Table 5.3: *The tools used to develop the application's GUI*

5.4 Other Tools

Table 5.4 shows the other tools used for the design, development and write-up of this application.

Tool	Description & Reasoning
Git [49] GitHub [50]	A version control system used in conjunction with GitHub. Creating periodic commits meant I always had a recent backup available and could easily backtrack to help find issues. Use of a GitHub Actions helped remind me that not all of my tests passed at all times :(.
LaTeX [51]	Used for the write-up of this document. LaTeX was useful in creating a large document and has many packages that help with referencing and design.
VSCode [52]	My code editor of choice for this project as it allowed me to seamlessly work on both my frontend and backend code at once.
Lucidchart [53]	Lucidchart was used to create all of the diagrams for this project. Lucidchart offers a better user experience when compared to alternatives like Draw.io but locks several features behind a paywall.
Zotero [54]	A reference manager used to help me organise all of the documents cited in this paper. This includes a browser extension to easily add new references.

Table 5.4: *General purpose tools used for this project*

Chapter 6

Testing

My approach to testing will consist of the following principles:

1. **Test Driven Development** Tests should be written alongside the code to reduce the risk of bugs and improve robustness.
2. **Fail Fast (Smoke testing)** Automated tests should be ran in a pipeline where the fastest tests are always ran first to reduce the time spent running tests.
3. **Documentation** Test cases should be well documented and grouped contextually such that they are easy to maintain and extend.

Tools

Table 6.1 shows the different tools used to write automated tests for my application.

Tool	Description & Reasoning
Go Testing [55]	The testing package included with Go's standard library was sufficient to produce most of the test cases required for this project.
testify [56]	This package is included as it provides several useful testing features that aren't present in the standard library testing package. This includes assert functions to boost code readability, mocking tools for better unit testing, setup/teardown functionality, and more.

Table 6.1: *The tools used for testing my project*

6.1 Unit Testing

Unit tests were written alongside the code they were testing to ensure my code was robust and responded appropriately to all inputs. I aimed for a 70% test coverage to ensure a significant amount of the application was tested. Table 6.2 shows a breakdown of test coverage by package.

Package	Coverage	Tests Written
model/manager/hashtree	75.2%	69
model/manager/games	61.3%	107

model/manager/ignore	90.6%	13
model/net/tcp	71.9%	27
model/net/peer	55.8%	111
model/persistence/ethereum	81.7%	20
model/util	61.2%	24
Total	71.1%	371

Table 6.2: *Code coverage by package. Missing entries do not have code in them for example model/manager is only a wrapper for its child packages.*

Automated tests were also written for the smart contract functions and tested locally using Ganache CLI [41]. Tests were not written for generic functions like getters and setts; this includes the Controller functions that were typically wrappers around Model functions. Equally, automated UI tests were omitted due to time constraints and that UI components were partially tested by the user walkthroughs.

6.2 Integration Testing

The purpose of integration tests were to evaluate how the application fared when interacting with various types of peer. Each *profile* will mimic a type of behaviour that could be expected in a real-world deployment and the different types are detailed in Table 6.3.

Name	Purpose
Listen Only	A peer who will listen and respond to all requests perfectly but will never request anything. This is useful for testing when we want a lightweight client to just download data off of. This will also be able to upload a game to the locally running smart contract.
Sender	This peer will respond to requests but will also periodically send requests. This can be used to show how my application reacts to more realistic peers.
Unreliable	This peer will pseudo-randomly not respond to messages or send incorrect data in response. This is used to show how my application recovers from faults sent by other users.
Selfish	This peer will send requests but will never respond to any. This is used to show how my application handles expired requests.

Table 6.3: *The different profiles used to simulate real-world peers.*

The main outcome from these different profiles shows the need for a reputation system where a user can distinguish between peers that reliably respond to requests and peers that don't. This would help mitigate some of the overhead of timed-out requests or receiving incorrect data.

6.3 Benchmarking

To assert that my application is scalable (**NF-S1**), I will be using a benchmark to show the download speed of a game varies depending on the number of peers a user is connected to.

Each test case will:

- connect to N **Listen Only** peers.
- dowload the entirety of the generated test data. This data will be of size 40GB to match the average game size mentioned in Section 4.3.
- be run three times to ensure consistency.

Table 6.4 shows the results of the benchmark.

	Runtime (s)			
Peers	**1**	**2**	**3**	**avg.**
1	1,513	1,511	1,511	1,512
2	777	775	777	776
4	397	397	396	397
8	227	223	224	225

Table 6.4: *Download times for the same piece of data by varying the number of connected peers.*

These results show that the download speed we can massively be increasedbby connecting to more peers. This shows that my application can correctly handle many concurrent peers (**F-M7**) and will distribute requests among many different connections to increase scalability (**NF-S1**).

6.4 Acceptance Testing

A user walkthrough is a series of steps to take that, if completed, prove the completeness of a set of requirements. All user walkthroughs use a contract deployed to the Sepolia test-net [42], satisfying (**F-M4**), (**NF-M2**) and (**NF-M1**), and all transactions completed can be seen publicly.

Table 6.5 describes all user walkthroughs and Appendix A shows the evidence for their completeness.

Id	Requirements	Description	Success
1	(**F-M1**) (**F-M5**) (**F-M12**) (**F-S2**) (**F-C2**) (**NF-M3**)	1. P_1 uploads a game G_1. 2. P_2 finds G_1 on the store. 3. P_2 purchases G_1. 4. P_2 shows G_1 added to their library.	YES

2	(**F-M6**) (**F-M8**) (**F-M9**) (**F-M10**) (**F-M11**) (**F-S1**) (**F-S2**) (**F-S3**) (**NF-M2**)	1. P_2 connects to P_1. 2. P_1 and P_2 exchange Ethereum addresses. 3. P_2 starts a download for G_1. 4. P_2 sends requests for blocks to P_1. 5. P_1 queries the smart contract to verify that P_2 owns G_1. 6. P_1 will respond to P_2 with the requested data. 7. P_2 will verify each block of data received using its hash. 8. P_2 will have full downloaded G_1. 9. P_1 will request and receive a contributions receipt for G_1 from P_2.	YES
3	(**F-M2**) (**F-M3**) (**F-M6**) (**NF-M5**)	1. P_1 is the original uploader of G_1 and P_2 has already purchased G_1. 2. P_1 uploads an update to G_1, G_2. 3. P_2 will hit the 'check for updates' button and see G_2 in their library.	YES
4	(**F-S4**) (**F-M7**) (**NF-M2**)	1. P_1 is connected to P_2. 2. P_3 forms a connection with P_1. 3. P_3 requests a list of P_1's peers and P_1 responds with the details for P_2. 4. P_3 forms connections with P_2.	YES

Table 6.5: *The set of user walkthroughs used to prove the completeness of this project's requirements.*

Chapter 7

Project Management

7.1 Risks

This section will describe any anticipated risks for this project that were considered at the start of the project and then discuss whether any occurred and how effective my mitigation strategies were.

Risk Assessment

Table 7.1 shows the risks that were considered at the start of this project.

Risk	Loss	Prob	Risk	Mitigation
Difficulty with blockchain development	2	3	6	I will seek advice from my supervisor about how to tackle certain problems and decide on any changes my project might need. I could also use online documentation or forums for support.
Personal illness	3	2	6	Depending on the amount of lost time, I will have to choose to ignore some lower priority requirements. Use of effective sprint planning will help ensure I can produce at least a minimal viable product.
Laptop damaged or lost	3	1	3	Thorough use of version control and periodic backups to a separate drive will ensure I always have a relatively recent copy of my work. I have other devices available to me at home and through the university to continue development.
The application is not finished	2.5	4	10	Effective use of agile development and requirement prioritisation will ensure that even if I do not complete the project I will have the most significant parts of it developed. It is important to consider a cut off point for development, where I will have to purely focus on the write-up and final testing.

Lack of large-scale testing infrastructure	2	5	10	Local benchmarks can be used to determine theoretical upper limits on my application or could be tested using a variety of hardware owned personally. Other tests may show it working on a smaller scale over the internet but it would be difficult and expensive to obtain the hardware to test it at a large scale.

Table 7.1: *The risk assessment of this project*

Risk Evaluation

This section will look at whether any risks outlined in the previous section occurred and how effective my mitigation was.

The application is not finished Section 7.5 shows the scoped requirements that were not met and Section 4.5 shows some of the limitations within my design when compared to similar platforms. However, use of MoSCoW prioritisation and sprint planning meant I was still able to produce a largely functional application that met the majority of the requirements and having a cut off point for implementation ensured I had sufficient time to complete this report.

One reason for not finishing was the size of the project, which took a considerable amount of time to implement and test. Table 7.2 shows a breakdown of the source code.

Language	Files	Comments	Code
Go	44	683	4621
Go Tests	29	791	3205
Vue.js Components	18	147	2855
JavaScript	19	74	209
Solidity	1	30	51

Table 7.2: *The lines of code written for this project calculated using CLOC [57] excluding any auto-generated code.*

Lack of large-scale testing infrastructure Testing distributed applications is challenging due to many factors, such as having to source homogenous devices, having access to networks that would allow me host a peer, and more. However, several useful results were produced from the benchmarks (Section 6.3) and acceptance tests (Section 6.4) that can be used to improve the application. If this project were to move forward then testing it on a large-scale would be essential.

7.2 Sprint Plans

By dividing my implementation into sprints, I was able to incrementally build upon my project by completing requirements according to their priority and expected difficulty.

Sprint 1

I anticipated that the P2P game distribution network would be the most complex and time consuming set of requirements in this project so I decided to focus on it for this first sprint. Table 7.3 shows the requirements included for Sprint 1 and whether they were completed or not.

Req.	Complete	Evidence/Reasoning
(**F-M7**)	YES	Unit tests for the *model/net/tcp* package and the peer count benchmark tests.
(**F-M8**)	YES	Unit tests for the *model/net/peer/message_handlers* file test the handling of structured messages and the structured responses sent back.
(**F-M9**)	YES	The benchmark test show the downloading of data to a large scale.
(**F-M10**)	YES	Unit tests to show incorrect messages being rejected.
(**F-M11**)	YES	User walkthrough 2 shows the download of a game in its entirety.
(**F-M12**)	STARTED	The algorithm to generate hash trees and the ability to use them to download data was implemented. Unit tests for the *model/manager/hashtree* and *model/manager/games* packages show this. Uploading this to distributed storage was planned for Sprint 2.
(**NF-M2**)	YES	User walkthrough 2 shows that any user can establish a connection with any other user.
(**NF-S1**)	STARTED	Users can perform many concurrent connections and channels are used at component boundaries to allow for multiple producers/consumers of data. This requirement was a consideration throughout all sprints.

Table 7.3: *Requirements included for Sprint 1*

Sprint 2

Sprint 2 was about increasing the scope of the application by focusing on two main aspects:

1. The integration with Ethereum using a Smart Contract, and
2. Allowing users to interface with the application via a GUI.

This sprint had a much slower start compared to the first one as I was largely unfamiliar with smart contract development and the related packages needed to interface with them. On top of this, I considered several UI framework's before settling on my final choice which increased the length of this sprint.

Table 7.4 shows the requirements pitched for Sprint 2 and whether or not they were completed.

Req.	Complete	Evidence/Reasoning
(**F-M1**)	YES	Unit tests for the Library smart contract and user walkthrough 1 show the ability to upload game metadata to Ethereum.

(**F-M2**)	YES	Unit tests for the Library smart contract and user walkthrough 4 show the ability to upload an update to an existing game to Ethereum.
(**F-M3**)	YES	Unit tests for the Library smart contract show users of an existing game being given ownership of an updated version.
(**F-M4**)	YES	The smart contract was successfully deployed the Sepolia test-net [42]. All user walkthroughs will form connections to this smart contract.
(**F-M5**)	YES	Unit tests for the Library smart contract and user walkthrough 1 show the successful purchase of a game.
(**F-M6**)	YES	Unit tests for the Library smart contract show a user being added to a mapping containing all users who have purchased the game.
(**F-M12**)	YES	Hash trees are now uploaded to IPFS and the CID is stored on Ethereum.
(**F-S2**)	STARTED	Basic pages were added according to Section 4.2.3. These pages had little styling or reactivity but could perform the required basic functions. See Appendix B for screenshots of the final versions.
(**NF-M1**)	YES	The use of the Ethereum blockchain means that no single user can control what is uploaded to the network.
(**NF-M3**)	YES	Developers can be uniquely identified using their Ethereum address. This should be made publically verifiable by the developers.
(**NF-M4**)	YES	Data stored on Ethereum is inherently immutable.
(**NF-M5**)	YES	Unit tests for the Library smart contract show the restriction that only the original uploader can release an update.

Table 7.4: *Requirements included for Sprint 2*

Sprint 3

This sprint was about extending the minimum viable application reached by the end of Sprint 2 with some necessary additions. Table 7.5 shows the list of requirements for this sprint and whether or not they were completed.

Req.	**Complete**	**Evidence/Reasoning**
(**F-S1**)	YES	Users will validate each other's Ethereum address after forming a connection and unit tests for the model/net/peer/message_handlers file show this being performed.
(**F-S2**)	YES	The UI was overall improved to improve the user experience.
(**F-S3**)	NO	Users will track the blocks sent to them by each of their peers but this application has no mechanism for redeeming these. Due to time constraints, I was unable to implement a sufficient solution. Moreover, I felt that a micropayment system, like present in Swam [8], would be a much better implementation.

(**F-S4**)	YES	Users will exchange the REQ_PEERS/PEER commands to discover neighbouring peers. However a better implementation might have the developer of the game be able to provide a list of peers who have the game. This would allow a user to easily find peers who are interested in the same content.
(**F-C1**)	NO	Due to time constraints I was unable to implement this at all.
(**F-C2**)	YES	Game assets are uploaded to IPFS and the CID is stored with the game metadata on Ethereum.
(**NF-S1**)	YES	Benchmark tests show the scalability of my application by varying certain parameters and that the target file size can be downloaded within an acceptable best-case.
(**NF-S2**)	YES	Changes to the UI made it more interactive and easier to navigate. Designs were inspired by pages from existing platforms to make the UI feel familiar. See Appendix B for screenshots of the final versions.
(**NF-C1**)	YES	A help page was included answering some questions that new users may have about the application.

Table 7.5: *Requirements included for Sprint 3*

7.3 Gantt Chart

A Gantt chart was used to give a high-level overview of my project to give myself a realistic timetable of when different aspects had to be completed by. Figure 7.1 shows the complete version.

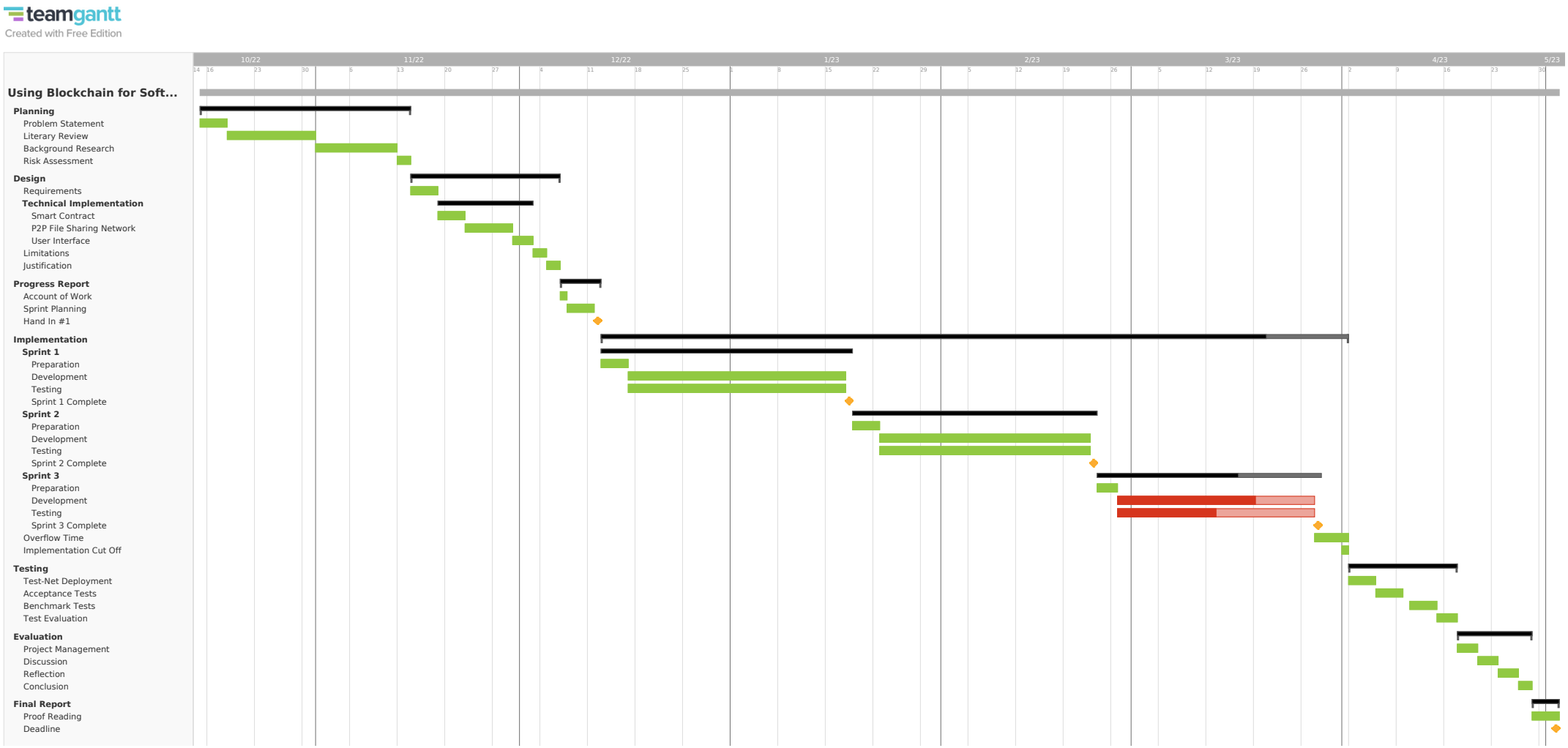


Figure 7.1: *The Gantt chart showing a breakdown of my project by task and milestones, where the red areas incidicate incomplete work. Created with TeamGantt [58].*

Chapter 8

Evaluation

8.1 Discussion

Due to the scale and complexity of the platforms this project was aiming to replace, this project was never going to be more than a proof-of-concept as to what a distributed games marketplace could look like. However, this application does have several benefits over its centralised competitors, which are outlined in Section 4.4

This project presents interesting ideas surrounding how games, and other proprietary software, can be distributed between its users without the need for an intermediary. Furthermore, making this project open source would encourage community development through updates or extensions, and improve transparency.

8.2 Reflection

Using Agile Development

The use of an agile methodology alongside test-driven development meant I could incrementally expand the scope of my application and ensure that existing code was tested sufficiently. Separating my implementation into three sprints benefitted me by:

- having a smaller set of requirements to focus on at once helped me to feel less overwhelmed,
- working on the most important aspects first to ensure I was able to produce a minimum viable product, and
- taking time in-between sprints to take a break from the project and prepare for the next sprint.

On top of this, using a Gantt chart (Figure 7.1) allowed me to have a clear overview of my project timeline so I could realistically gage just how much time I would need for each section.

Chapter 9

Conclusion

This project set out to demonstrate how video game distribution could be migrated to a distributed platform with the aim of reducing the risk of censorship, improving ownership and increasing profits for developers.

By researching related topics and reviewing the literature around key areas of this project, I was able to combine modern ideas and technologies to develop a functional proof-of-concept application. The heavy use of automated testing allowed me to continuously write robust and correct code and the successful deployment to an Ethereum development meant I was able to demonstrate the correctness of my implementation.

As most of the requirements set out in Section 4.0.2 were met, I can say that this project was a success.

9.1 Future Work

New Features

Some notable features that would serve as excellent extensions to this application:

- Fleshing out the store page will make it easier for users to discover new content. Games could be given extra metadata to help make them more searchable.
- Allowing users to post reviews or posts to message boards for individual games will allow for greater community integration.
- Add customisable user profiles and allow for users to add each other as friends. Friends would auto-connect as peers to help maintain a network.

Optimisations

There are several optimisations we could add to improve the overall performance and scalability of the application:

- **Peer Reputation** By ranking peers based upon their reliability we can improve the success rate and latency of requests we send. An optimistic unchoking algorithm like seen in BitTorrent could also be used.
- **Block Selection** Currently blocks are not ranked in any way but by considering ideas from Section 3.1 we could improve the efficiency, availability and throughput of our network.

Chapter 10

References

Peer-to-Peer File Sharing

[5] Juan Benet. *IPFS - Content Addressed, Versioned, P2P File System*. July 14, 2014. DOI: 10.48550/arXiv.1407.3561. arXiv: 1407.3561[cs]. URL: http://arxiv.org/abs/1407.3561 (visited on 11/02/2022).

[6] *Estuary*. URL: https://estuary.tech (visited on 04/09/2023).

[7] *ipfs/kubo*. original-date: 2014-06-26T08:14:34Z. Apr. 9, 2023. URL: https://github.com/ipfs/kubo (visited on 04/09/2023).

[8] J.H. Hartman, I. Murdock, and T. Spalink. "The Swarm scalable storage system". In: *Proceedings. 19th IEEE International Conference on Distributed Computing Systems (Cat. No.99CB37003)*. Proceedings. 19th IEEE International Conference on Distributed Computing Systems (Cat. No.99CB37003). ISSN: 1063-6927. June 1999, pp. 74–81. DOI: 10.1109/ICDCS.1999.776508.

[9] Johan Pouwelse et al. "The Bittorrent P2P File-Sharing System: Measurements and Analysis". In: *Peer-to-Peer Systems IV*. Ed. by Miguel Castro and Robbert van Renesse. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2005, pp. 205–216. ISBN: 978-3-540-31906-1. DOI: 10.1007/11558989_19.

[10] James H. Morris et al. "Andrew: a distributed personal computing environment". In: *Communications of the ACM* 29.3 (Mar. 1, 1986), pp. 184–201. ISSN: 0001-0782. DOI: 10.1145/5666.5671. URL: https://doi.org/10.1145/5666.5671 (visited on 11/25/2022).

[11] John H. Howard et al. "Scale and performance in a distributed file system". In: *ACM Transactions on Computer Systems* 6.1 (Feb. 1, 1988), pp. 51–81. ISSN: 0734-2071. DOI: 10.1145/35037.35059. URL: https://doi.org/10.1145/35037.35059 (visited on 11/25/2022).

[12] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble. "Measurement study of peer-to-peer file sharing systems". In: *Multimedia Computing and Networking 2002*. Multimedia Computing and Networking 2002. Vol. 4673. SPIE, Dec. 10, 2001, pp. 156–170. DOI: 10.1117/12.449977. URL: https://www.spiedigitallibrary.org/conference-proceedings-of-spie/4673/0000/Measurement-study-of-peer-to-peer-file-sharing-systems/10.1117/12.449977.full (visited on 11/23/2022).

[20] S. Kaune et al. "Unraveling BitTorrent's File Unavailability: Measurements and Analysis". In: *2010 IEEE Tenth International Conference on Peer-to-Peer Computing (P2P)*. 2010 IEEE Tenth International Conference on Peer-to-Peer Computing (P2P). ISSN: 2161-3567. Aug. 2010, pp. 1–9. DOI: 10.1109/P2P.2010.5569991.

[21] BitTorrent Inc. *BitTorrent Web — The Best Online Torrent Downloader*. BitTorrent. URL: https://www.bittorrent.com/products/win/bittorrent-web-free/ (visited on 04/08/2023).

[22] *qBittorrent Official Website*. URL: https://www.qbittorrent.org/ (visited on 04/08/2023).

[23] BitTorrent Inc. *µTorrent (uTorrent) — A Very Tiny BitTorrent Client*. uTorrent. URL: https://www.utorrent.com/ (visited on 04/08/2023).

[24] *Application Usage & Threat Report*. URL: https://www.paloaltonetworks.com/blog/app-usage-risk-report-visualization/# (visited on 04/08/2023).

[25] G. Neglia et al. "Availability in BitTorrent Systems". In: *IEEE INFOCOM 2007 - 26th IEEE International Conference on Computer Communications*. IEEE INFOCOM 2007 - 26th IEEE International Conference on Computer Communications. ISSN: 0743-166X. May 2007, pp. 2216–2224. DOI: 10.1109/INFCOM.2007.256.

Blockchain

[13] Dongdong Yue et al. "Blockchain Based Data Integrity Verification in P2P Cloud Storage". In: *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*. 2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS). Dec. 2018, pp. 561–568. DOI: 10.1109/PADSW.2018.8644863.

[14] Jingyi Li et al. "Deduplication with Blockchain for Secure Cloud Storage". In: *Big Data*. Ed. by Zongben Xu et al. Communications in Computer and Information Science. event-place: Singapore. Springer, 2018, pp. 558–570. ISBN: 9789811329227. DOI: 10.1007/978-981-13-2922-7_36.

[15] Jiaxing Li, Jigang Wu, and Long Chen. "Block-secure: Blockchain based scheme for secure P2P cloud storage". In: *Information Sciences* 465 (Oct. 1, 2018), pp. 219–231. ISSN: 0020-0255. DOI: 10.1016/j.ins.2018.06.071. URL: https://www.sciencedirect.com/science/article/pii/S0020025518305012 (visited on 11/23/2022).

[16] Yi Chen et al. "Blockchain-Based Medical Records Secure Storage and Medical Service Framework". In: *Journal of Medical Systems* 43.1 (Nov. 22, 2018), p. 5. ISSN: 1573-689X. DOI: 10.1007/s10916-018-1121-4. URL: https://doi.org/10.1007/s10916-018-1121-4 (visited on 11/24/2022).

[17] Shangping Wang, Yinglong Zhang, and Yaling Zhang. "A Blockchain-Based Framework for Data Sharing With Fine-Grained Access Control in Decentralized Storage Systems". In: *IEEE Access* 6 (2018). Conference Name: IEEE Access, pp. 38437–38450. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2018.2851611.

[18] Ahsan Manzoor et al. “Blockchain based Proxy Re-Encryption Scheme for Secure IoT Data Sharing”. In: *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. 2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC). May 2019, pp. 99–103. DOI: 10.1109/BLOC.2019.8751336.

[19] Pratima Sharma, Rajni Jindal, and Malaya Dutta Borah. “Blockchain Technology for Cloud Storage: A Systematic Literature Review”. In: *ACM Computing Surveys* 53.4 (July 31, 2021), pp. 1–32. ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3403954. URL: https://dl.acm.org/doi/10.1145/3403954 (visited on 11/22/2022).

[26] Dejan Vujičić, Dijana Jagodić, and Siniša Randić. “Blockchain technology, bitcoin, and Ethereum: A brief overview”. In: *2018 17th International Symposium INFOTEH-JAHORINA (INFOTEH)*. 2018 17th International Symposium INFOTEH-JAHORINA (INFOTEH). Mar. 2018, pp. 1–6. DOI: 10.1109/INFOTEH.2018.8345547.

[27] Chris Dannen. *Introducing Ethereum and Solidity: Foundations of Cryptocurrency and Blockchain Programming for Beginners*. Berkeley, CA: Apress, 2017. ISBN: 978-1-4842-2534-9 978-1-4842-2535-6. DOI: 10.1007/978-1-4842-2535-6. URL: https://link.springer.com/10.1007/978-1-4842-2535-6 (visited on 11/23/2022).

[30] *Goerli Testnet*. Goerli Testnet. URL: https://goerli.net (visited on 04/03/2023).

[31] *Ropsten Testnet*. original-date: 2017-03-22T16:17:48Z. Mar. 23, 2023. URL: https://github.com/ethereum/ropsten (visited on 04/03/2023).

Tools Used

[28] *Solidity — Solidity 0.8.18 documentation*. URL: https://docs.soliditylang.org/en/v0.8.18/ (visited on 03/22/2023).

[29] *Sepolia Resources*. Sepolia Resources. URL: https://sepolia.dev/ (visited on 03/30/2023).

[33] *The Go Programming Language*. URL: https://go.dev/ (visited on 03/22/2023).

[34] *go-ipfs-api*. original-date: 2015-05-13T05:09:55Z. Mar. 29, 2023. URL: https://github.com/ipfs/go-ipfs-api (visited on 03/30/2023).

[35] *go-ethereum*. go-ethereum. URL: https://geth.ethereum.org/ (visited on 03/22/2023).

[36] *Zap*. original-date: 2016-02-18T19:52:56Z. Mar. 30, 2023. URL: https://github.com/uber-go/zap (visited on 03/30/2023).

[37] *viper package - github.com/dvln/viper - Go Packages*. URL: https://pkg.go.dev/github.com/dvln/viper (visited on 03/30/2023).

[38] *Alchemy - the web3 development platform*. URL: https://www.alchemy.com/ (visited on 03/30/2023).

[39] *The crypto wallet for Defi, Web3 Dapps and NFTs — MetaMask*. URL: https://metamask.io/ (visited on 03/30/2023).

[40] *Remix - Ethereum IDE*. URL: https://remix.ethereum.org/ (visited on 03/30/2023).

[41] *trufflesuite/ganache*. original-date: 2017-03-27T17:04:47Z. Mar. 31, 2023. URL: https://github.com/trufflesuite/ganache (visited on 04/01/2023).

[42] etherscan.io. *Library — Address 0x9675d36f7ab5f5a8163928f6982375e4ebc2c746 — Etherscan*. Ethereum (ETH) Blockchain Explorer. URL: http://sepolia.etherscan.io/address/0x9675d36f7ab5f5a8163928f6982375e4ebc2c746 (visited on 04/19/2023).

[43] *The Wails Project — Wails*. URL: https://wails.io/ (visited on 03/22/2023).

[44] *Vue.js - The Progressive JavaScript Framework — Vue.js*. URL: https://vuejs.org/ (visited on 03/22/2023).

[45] *Vue Router — The official Router for Vue.js*. URL: https://router.vuejs.org (visited on 03/30/2023).

[46] *markdown-it*. original-date: 2014-12-19T22:54:53Z. Mar. 30, 2023. URL: https://github.com/markdown-it/markdown-it (visited on 03/30/2023).

[47] *Pinia — The intuitive store for Vue.js*. URL: https://pinia.vuejs.org (visited on 03/22/2023).

[48] *Sass: Syntactically Awesome Style Sheets*. URL: https://sass-lang.com/ (visited on 03/22/2023).

[50] *Github*. GitHub. URL: https://github.com (visited on 03/30/2023).

[51] *LaTeX - A document preparation system*. URL: https://www.latex-project.org/ (visited on 03/30/2023).

[52] *Visual Studio Code - Code Editing. Redefined*. URL: https://code.visualstudio.com/ (visited on 03/30/2023).

[53] *Lucidchart*. Lucidchart. URL: https://www.lucidchart.com (visited on 03/30/2023).

[54] *Zotero — Your personal research assistant*. URL: https://www.zotero.org/ (visited on 04/12/2023).

[55] *testing package - testing - Go Packages*. URL: https://pkg.go.dev/testing (visited on 04/12/2023).

[56] *Testify - Thou Shalt Write Tests*. original-date: 2012-10-16T16:43:17Z. Mar. 30, 2023. URL: https://github.com/stretchr/testify (visited on 03/30/2023).

[57] *AlDanial/cloc: cloc counts blank lines, comment lines, and physical lines of source code in many programming languages*. URL: https://github.com/AlDanial/cloc (visited on 03/30/2023).

Project Management

[49] *Git*. URL: https://git-scm.com/ (visited on 03/30/2023).

[58] *Free Online Gantt Chart Maker That's Easy to Use — TeamGantt*. URL: https://www.teamgantt.com/h2 (visited on 04/12/2023).

Other

[1] Tom Marks. *Report: Steam's 30% Cut Is Actually the Industry Standard*. IGN. Oct. 7, 2019. URL: https://www.ign.com/articles/2019/10/07/report-steams-30-cut-is-actually-the-industry-standard (visited on 12/10/2022).

[2] Andy Brown. *Valve defends taking 30 per cent cut of Steam sales in response to lawsuit*. NME. July 30, 2021. URL: https://www.nme.com/news/gaming-news/valve-defends-taking-30-per-cent-cut-of-steam-sales-in-response-to-lawsuit-3007255 (visited on 12/10/2022).

[3] *Steam China officially launched*. SteamDB. URL: https://steamdb.info/blog/steam-china-launched/ (visited on 04/06/2023).

[4] "Nintendo receives backlash from fans over ending eShop purchases". In: *BBC News* (Feb. 16, 2022). URL: https://www.bbc.com/news/technology-60405561 (visited on 04/06/2023).

[32] *Steam Charts · Most Played Games on Steam*. SteamDB. URL: https://steamdb.info/charts/ (visited on 03/22/2023).

Appendix A

User Walkthroughs

This section will go through each of the acceptance tests outlined in Section 6.4 and provide the evidence that they pass. This evidence will be given in the form of:

- Smart contract transaction logs from the Sepolia test-net. These are public at https://sepolia.etherscan.io/address/0xca2522592219954c270451f0994fafcaba8ff104. The smart contract code was uploaded so all functions called and data uploaded are visible.
- Snippets of logs written that correspond to certain actions. For example, receiving a message over the network.
- Screenshots to show changes being reflected in the user interface.

For this the game will be a simple folder of text files that we can easily compare to show correctness.

- G_1 = [title="User WT", version="1.0", developer="tcs1g20", uploader=0xfBC8D99Bb9ab8F781fF04F9b45Fe5c97AACE1916, previousVersion=None, price=0, rootHash='']
- G_2 = [title="User WT", version="2.0", developer="tcs1g20", previousVer ion=G_1, price=0, uploader=0xfBC8D99Bb9ab8F781fF04F9b45Fe5c97AACE1916, rootHash='']

User Walkthrough 1

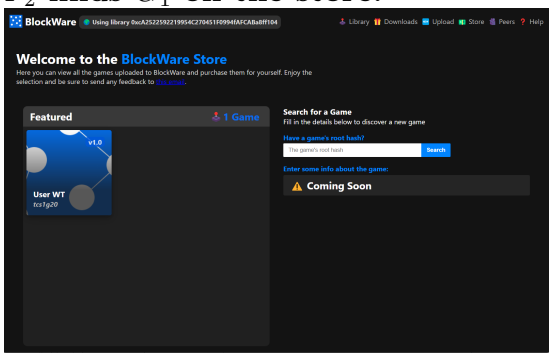
User Walkthrough 1 shows a user P_1 uploading a new game G_1 and another user P_2 purchasing it.

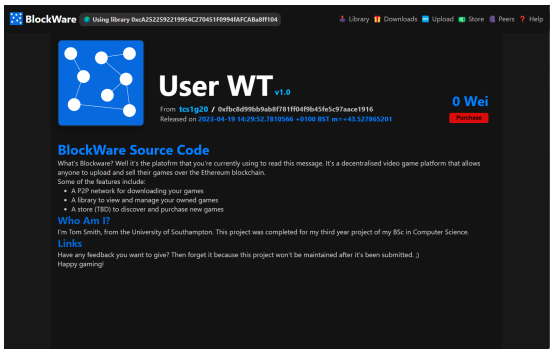
1. P_1 **uploads a game** G_1**.**
   Transaction hash = 0x0cb69f095a7f6e9625f6af91a72bea732c6f3ce58a75ff0ce3ba8a379248ab10

Function: uploadGame((string,string,string,string,bytes32,bytes32,bytes32,uint256,address,string,string))

#	Name	Type	Data
0	_game.title	string	User WT
0	_game.version	string	1.0
0	_game.releaseDate	string	2023-04-19 14:29:52.7810566 +0100 BST m=+43.527865201
0	_game.developer	string	tcs1g20
0	_game.rootHash	bytes32	0x7843e810bd21ca00cfe09b75e7990a672c7016b5f47d2760528876606 70f952b
0	_game.previousVersion	bytes32	0x0000000000000000000000000000000000000000000000000000000000000000
0	_game.nextVersion	bytes32	0x0000000000000000000000000000000000000000000000000000000000000000
0	_game.price	uint256	0
0	_game.uploader	address	0xfBC8D99Bb9ab8F781fF04F9b45Fe5c97AACE1916
0	_game.hashTreeIPFSAddress	string	QmYwv31FpJT8RPuEbGSvYGBbHaJLPNJBRtCi9TM1rUCaJt
0	_game.assetsIPFSAddress	string	QmbC5ezWsH6mS8R8e6t3SjvVT29aWvKoabZRwgE4TS7UvL

2. P_2 **finds** G_1 **on the store.**



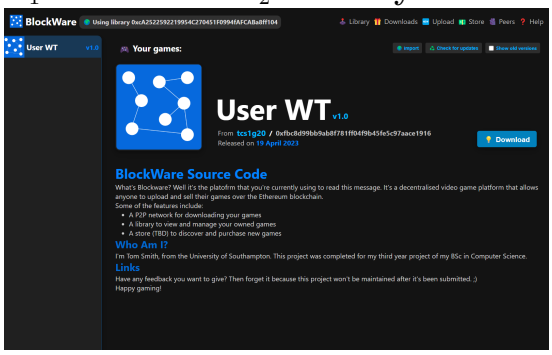


3. P_2 **purchases** G_1**.**
Transaction hash = 0x11b4932b5e1a72defbea862305fd3fd1dfa6bfc1c04ea2191b552c0f4c0a1981

```
Function: purchaseGame(bytes32 _game)

MethodID: 0x3e093f79
[0]:  7843e810bd21ca00cfe09b75e7990a672c7016b5f47d2760528876606 70f952b
```

4. G_1 **is added to** P_2**'s library.**



User Walkthrough 2

User walkthrough 2 shows a user P_1 downloading a game G_1 off of another user G_2. Both P_1 and P_2 already own G_1.

1. P_2 **connects to** P_1**.**

Logs showing P_2 forming a TCP connection with P_1:

```
2023-04-19T11:54:40.976+0100  [INFO]  tcp/client.go:42  Attempting to open
    ↪ connection to localhost:6051
...
2023-04-19T11:54:41.285+0100  [INFO]  controller/peers.go:52  Connected to peer
    ↪ localhost:6051
```

2. **P_1 and P_2 exchange Ethereum addresses.**
   Logs showing P_2 sending P_1 a VALIDATE_REQ message and receiving a VALIDATE_RES to verify its address:

```
2023-04-19T11:54:41.285+0100  [INFO]  ethereum/identity.go:54 Generating address
    ↪ validation
2023-04-19T11:54:41.285+0100  [DEBUG] tcp/client.go:93  Sending VALIDATE_REQ
    ↪ ;323032332
    ↪ d30342d31392031313a35343a34312e32383531333435202b303130302042535 4206d3d2
    ↪ b32372e355133303537393031
...
2023-04-19T11:54:41.286+0100  [DEBUG] tcp/client.go:79  message received
    ↪ VALIDATE_RES;258002
    ↪ cbc118da4900081729c53a6640b8d08ec6e2b2d408fac38a7e7f44cfa705c4ac3231bcc
    ↪ aae333cc1c0846fa7e32513bccad5dfac4cfe3d7fd32a60155300
2023-04-19T11:54:41.286+0100  [INFO]  ethereum/identity.go:80 Checking signature
2023-04-19T11:54:41.286+0100  [INFO]  ethereum/identity.go:97 Signature valid: true
```

3. **P_2 starts a download for G_1**
   P_2 Hits the download button in the libraray entry page.
4. **P_2 sends requests for blocks to P_2**
   Logs showing P_2 choose a block and send a request for it P_1:

```
2023-04-19T11:57:33.035+0100  [DEBUG] games/download.go:339 Requesting file
    ↪ downloads\User WT-1.0\test.txt for game
    ↪ e47a04d248a4f1e6863bcc5303a8e51b3c485129ad0055a82054648037cb4d6c
2023-04-19T11:57:33.035+0100  [DEBUG] games/download.go:343 Requesting shard 8
    ↪ ad06d76f6ac8729ab24e81b5e8d273c33bd6914f40c6709f2e6c02534428923 for file
    ↪ downloads\User WT-1.0\test.txt in game
    ↪ e47a04d248a4f1e6863bcc5303a8e51b3c485129ad0055a82054648037cb4d6c
...
2023-04-19T11:57:33.035+0100  [DEBUG] peer/requests.go:17 Processing request for
    ↪ block 8ad06d76f6ac8729ab24e81b5e8d273c33bd6914f40c6709f2e6c02534428923
...
2023-04-19T11:57:33.035+0100  [DEBUG] tcp/client.go:93  Sending BLOCK;
    ↪ e47a04d248a4f1e6863bcc5303a8e51b3c485129ad0055a82054648037cb4d6c;8
    ↪ ad06d76f6ac8729ab24e81b5e8d273c33bd6914f40c6709f2e6c02534428923
```

5. **P_1 queries the smart contract to verify that P_2 owns G_1**

```
2023-04-19T11:57:33.035+0100  [DEBUG] peer/peer_data.go:76  Verifying ownership of
    ↪ game e47a04d248a4f1e6863bcc5303a8e51b3c485129ad0055a82054648037cb4d6c for
    ↪ user 127.0.0.1:3260
2023-04-19T11:57:33.410+0100  [DEBUG] peer/peer_data.go:84  User 127.0.0.1:3260
    ↪ owns game e47a04d248a4f1e6863bcc5303a8e51b3c485129ad0055a82054648037cb4d6c
```

6. **P_1 will respond to P_2 with the requested data.**
   Logs showing P_2 receive a SEND_BLOCK message with the specified data:

```
  2023-04-19T11:57:33.411+0100  [DEBUG] tcp/client.go:79  message received
    ↪ SEND_BLOCK;e47a04d248a4f1e6863bcc5303a8e51b3c485129ad0055a82054648037cb4d6c
    ↪ ;8ad06d76f6ac8729ab24e81b5e8d273c33bd6914f40c6709f2e6c02534428923;< DATA
    ↪ OMITTED>
```

7. **P_2 will verify each block of data received using its hash.**
   The data matches the expected contents, no error is thrown and the shard is inserted:

```
2023-04-19T11:57:33.412+0100  [DEBUG] games/download.go:206 Attempting to insert
    ↪ shard 8ad06d76f6ac8729ab24e81b5e8d273c33bd6914f40c6709f2e6c02534428923 into
    ↪ 67aaf70a4b60379a1503b740547756bc524d4e5e7be53cbdbf05752394669dae
2023-04-19T11:57:33.412+0100  [DEBUG] hashtree/dummy.go:123 Writing shard to
    ↪ downloads\User WT-1.0\test.txt:0
```

```
3 2023-04-19T11:57:33.417+0100  [DEBUG] games/download.go:256 successfully inserted
      ↪ shard 8ad06d76f6ac8729ab24e81b5e8d273c33bd6914f40c6709f2e6c02534428923 into
      ↪ 67aaf70a4b60379a1503b740547756bc524d4e5e7be53cbdbf05752394669dae
```

8. P_2 **will have a full downloaded copy of** G_1
   The output of a diff command comparing the original directory with the downloaded one. An empty output here indicates they are identical.

```
1 thoma@TOM-LAPTOP MINGW64 ~/coursework/part-iii-project/test-net (main)
2 $ diff -rq ./game-1 ./peer-2/downloads/User\ WT-1.0/
```

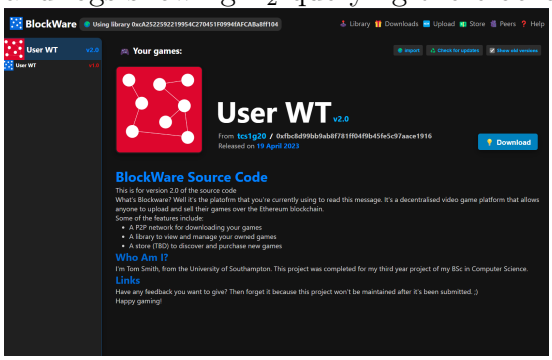
9. P_1 **will request and be sent a contributions receipt for** G_1 **from** P_2**.**
   Logs showing P_1 send a REQ_RECEIPT message to P_2 and receive a signed RECEIPT back:

```
1 2023-04-19T12:03:02.214+0100  [DEBUG] tcp/server.go:147 Sending REQ_RECEIPT;
      ↪ e47a04d248a4f1e6863bcc5303a8e51b3c485129ad0055a82054648037cb4d6c
2 2023-04-19T12:03:02.224+0100  [DEBUG] tcp/server.go:135 message received RECEIPT;
      ↪ e47a04d248a4f1e6863bcc5303a8e51b3c485129ad0055a82054648037cb4d6c;7790162
      ↪ ad42178687fd8353414423a8a2f8cfb54015b1b91be38d125667bc0a1604532bd9aeb3c552
      ↪ a849b8eeec36c3ad69a97b9e340f621f4b0e83abc0fa2c400;< BLOCKS OMITTED>
3 2023-04-19T12:03:02.224+0100  [INFO]  peer/message_handlers.go:422  Received
      ↪ receipt for 6 blocks
```

User Walkthrough 3

User Walkthrough 3 shows P_1 releasing an update to G_1, G_2, and P_2 downloading it. If User Walkthrough 2 is valid then we will be able to successfully download G_2 as we did with G_1.

1. P_1 **is the original uploader of** G_1 **and** P_2 **has already purchased** G_1**.**
   Given from User Walkthrough 1
2. P_1 **uploads an update to** G_1**,** G_2**.**
   Transaction hash = 0x0ea6739a1b93add5135e02db3e6030103582ed1cf4240c00ed2e05cfad943a80
3. P_2 **will hit the 'check for updates' button and see** G_2 **in their library.**
   The screenshot showing the updated library screen with the new version of the game and logs showing P_2 querying the blockchain for the new update.




```
1 2023-04-19T14:39:14.424+0100  [DEBUG] library/library.go:291  Checking for game
      ↪ updates
2 2023-04-19T14:39:14.915+0100  [INFO]  library/library.go:303  Found 1 updates
3 2023-04-19T14:39:14.915+0100  [DEBUG] library/library.go:307  1 game updates added
      ↪ successfully
```

User Walkthrough 4

User Walkthrough 4 shows a user discovering more peers through a request sent to a newly met peer through the use of the SERVER, REQ_PEERS and PEERS commands.

1. P_1 **is connected to** P_2**.**
   Logs from P_1 show us forming a connection with P_2 and sending its server address:

```
2023-04-20T15:51:58.918+0100  [INFO]  tcp/server.go:70  Server listening on
    ↪ localhost:6051
< other application setup >
2023-04-20T15:52:46.812+0100  [INFO]  tcp/client.go:41  Attempting to open
    ↪ connection to localhost:6052
2023-04-20T15:52:47.116+0100  [DEBUG] tcp/client.go:89  Sending SERVER;localhost
    ↪ :6051
< eth address validation  >
2023-04-20T15:52:47.116+0100  [INFO]  controller/peers.go:53  Connected to peer
    ↪ localhost:6052
```

2. P_3 **forms a connection with** P_1 Logs from P_1 show P_3 initiating a connection and sending their server address:

```
2023-04-20T15:52:54.514+0100  [INFO]  tcp/server.go:94  Client joined:
    ↪ 127.0.0.1:19335
...
2023-04-20T15:52:54.515+0100  [DEBUG] tcp/server.go:131 message received from
    ↪ 127.0.0.1:19335: SERVER;localhost:6053
```

3. P_3 **requests a list of** P_1**'s peers and** P_1 **will send the details for** P_2**.** Logs from P_1 show P_3 requesting a list of peers and P_1 replying with P_2's information.

```
2023-04-20T15:52:54.515+0100  [DEBUG] tcp/server.go:131 message received from
    ↪ 127.0.0.1:19335: REQ_PEERS
2023-04-20T15:52:54.515+0100  [DEBUG] tcp/server.go:143 Sending PEERS;localhost
    ↪ :6052
```

4. P_3 **forms a connection with** P_2**.** Logs from P_3 show the receing P_2's information from P_1 and forming a connection with P_2.

```
2023-04-20T15:52:54.515+0100  [DEBUG] tcp/client.go:73  message received from
    ↪ localhost:6051 PEERS;localhost:6052
2023-04-20T15:52:54.515+0100  [INFO]  tcp/client.go:41  Attempting to open
    ↪ connection to localhost:6052
2023-04-20T15:52:54.820+0100  [DEBUG] tcp/client.go:89  Sending SERVER;localhost
    ↪ :6053
```

Appendix B

Screenshots

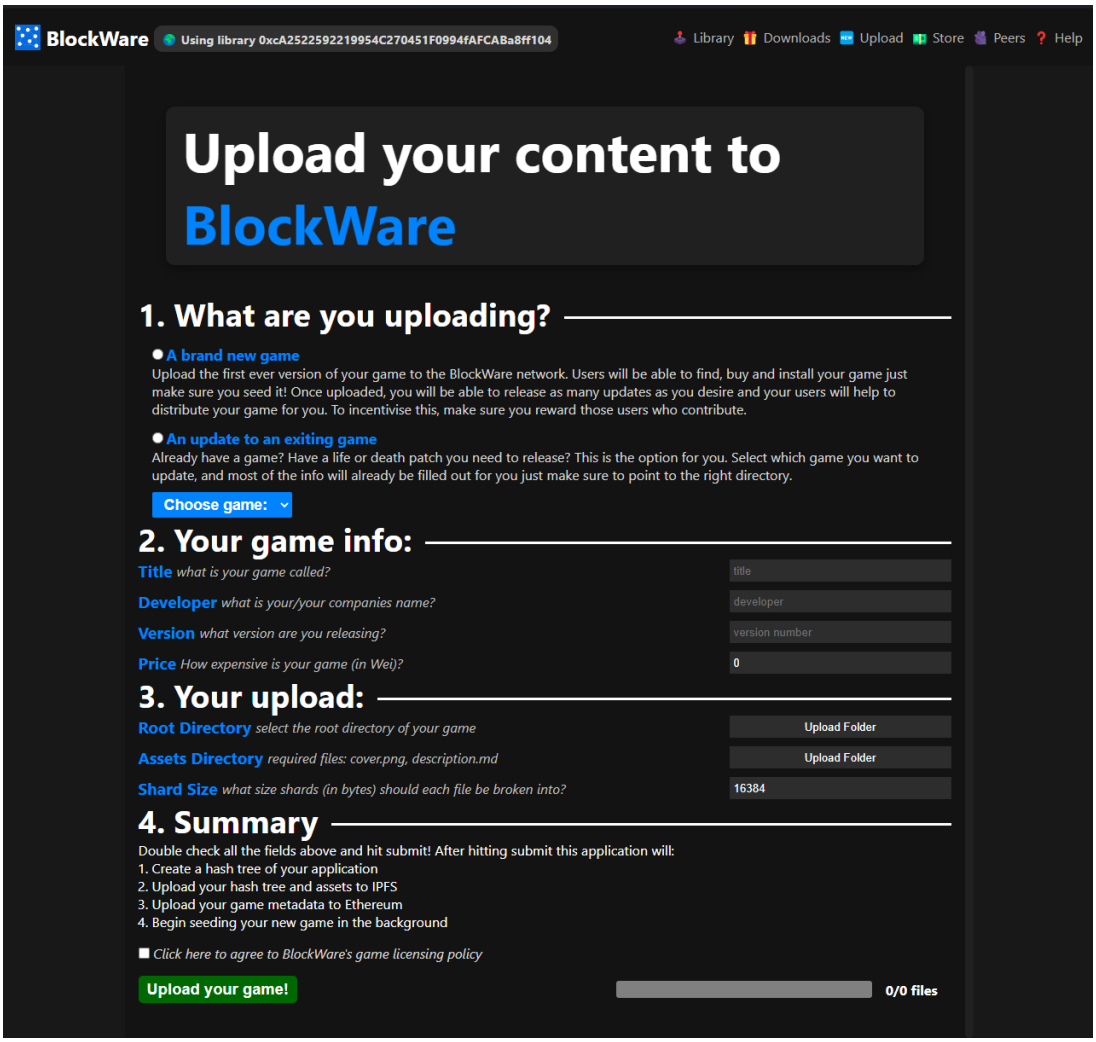


Figure B.1: *The page where users input the details about their game and can upload it to the Ethereum network. If a user wants to update a game they can choose from their uploaded games.*

Figure B.2: *The login page where a user will enter their Ethereum private key and connect to the BlockWare contract instance deployed to Sepolia. Users can access the help page from here.*

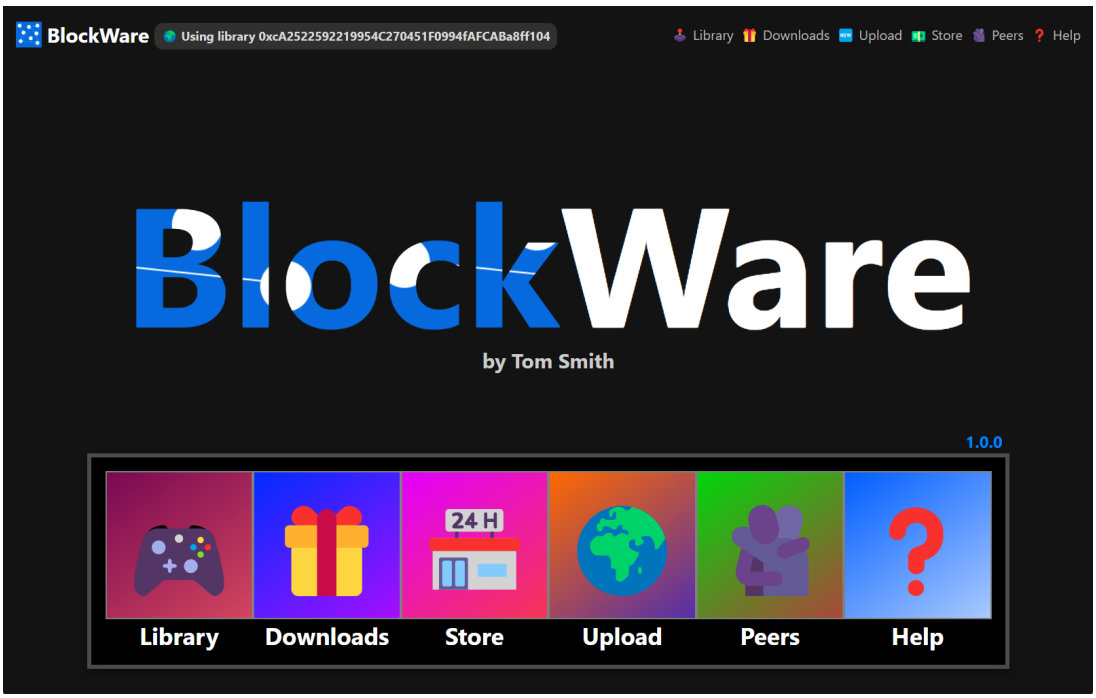


Figure B.3: *The home page where users can navigate between the main pages.*

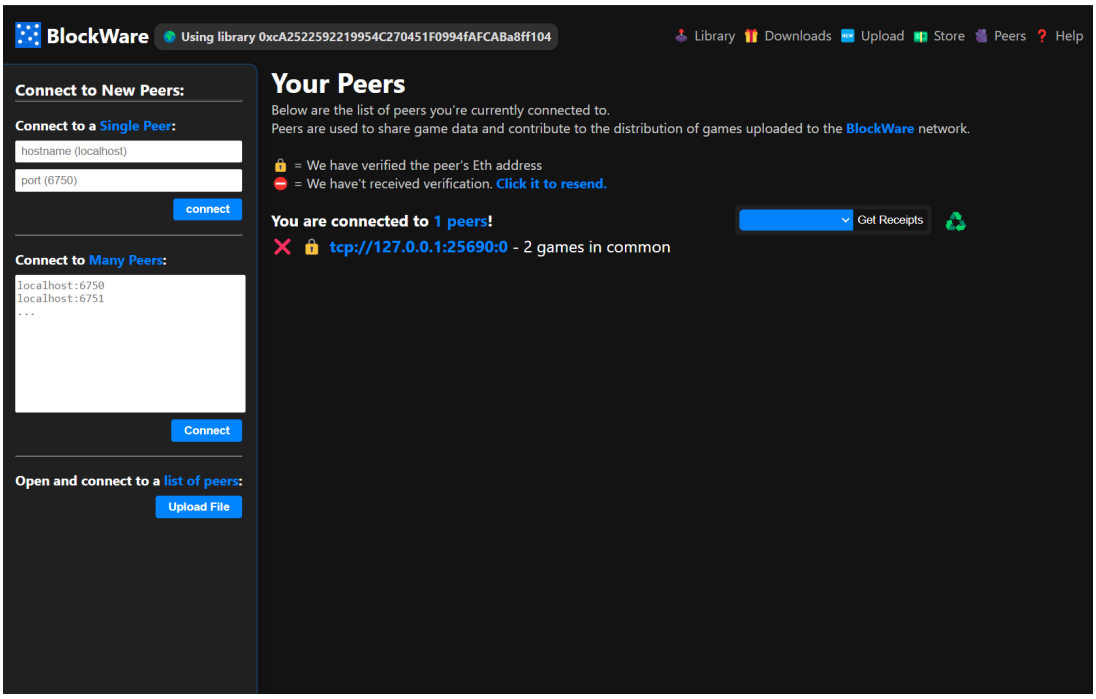


Figure B.4: *The page where users can manage their connections to peers with whom they will download game data off of. Users can request receipts from peers that show the amount of data they've sent.*

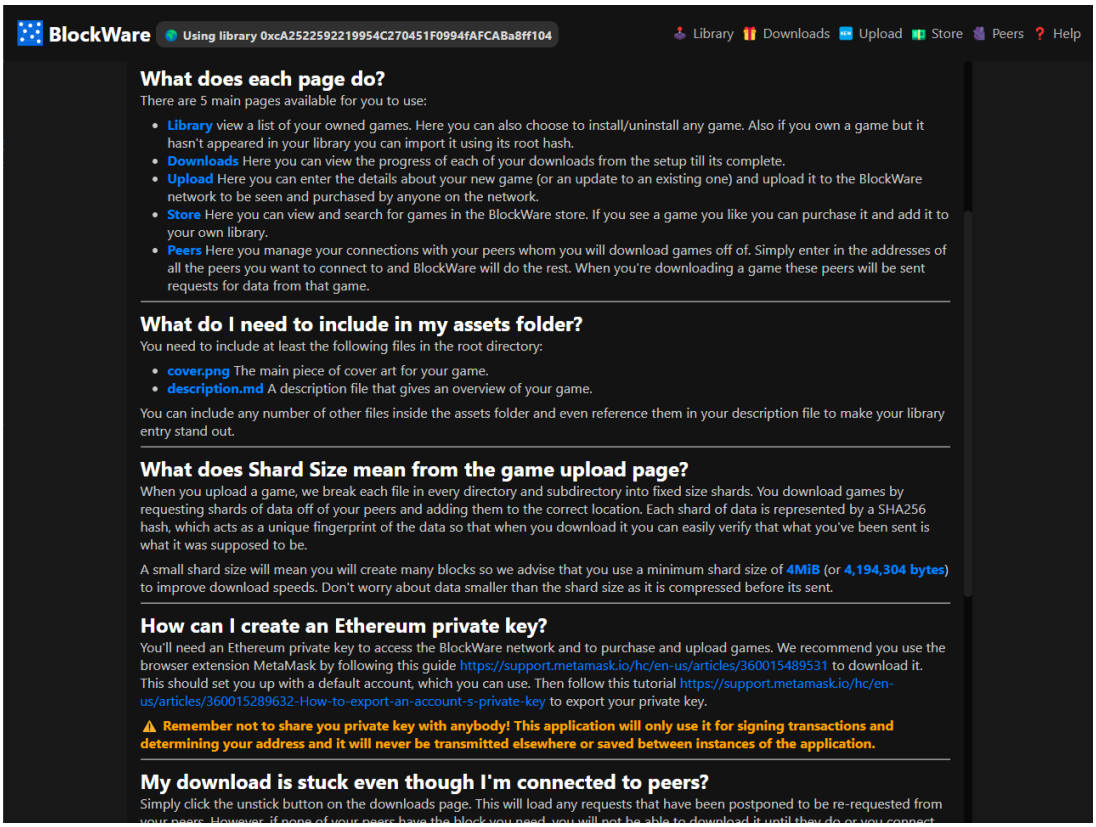


Figure B.5: *This page has a list of common questions, about the application, expected by users with answers.*

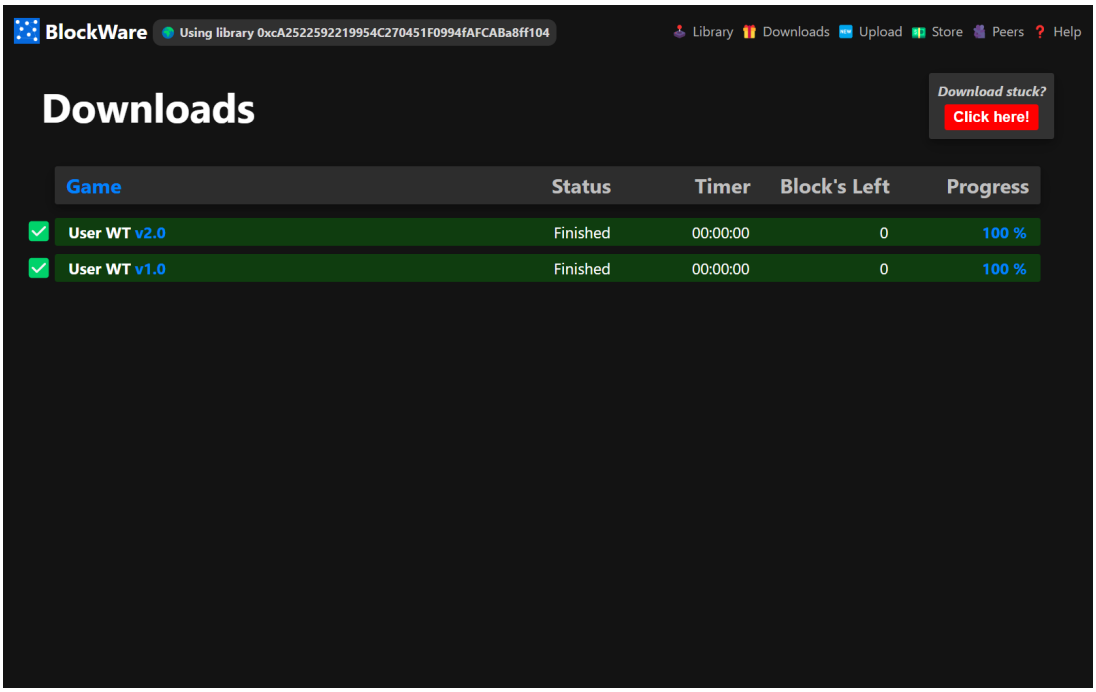


Figure B.6: *This page is where users can track their downloads. It will update in real-time as data is downloaded.*

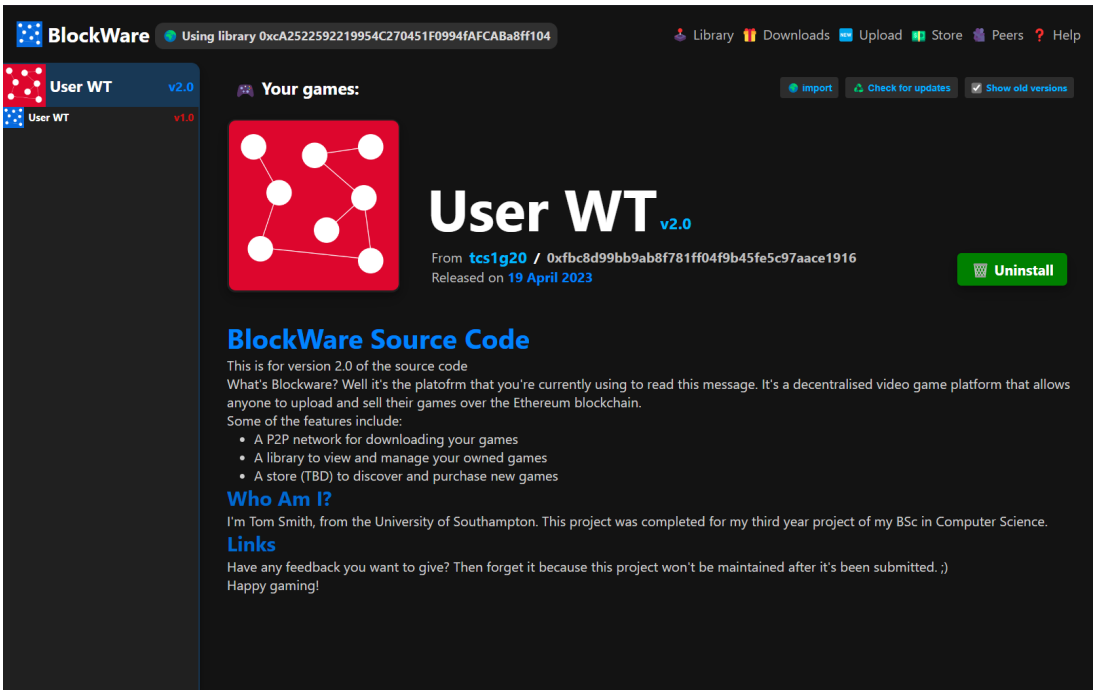


Figure B.7: *This page shows the user's library of games and allows them to view the assets for them. Smaller games represent older versions of the game and these can be hidden using a the 'show old versions' toggle.*

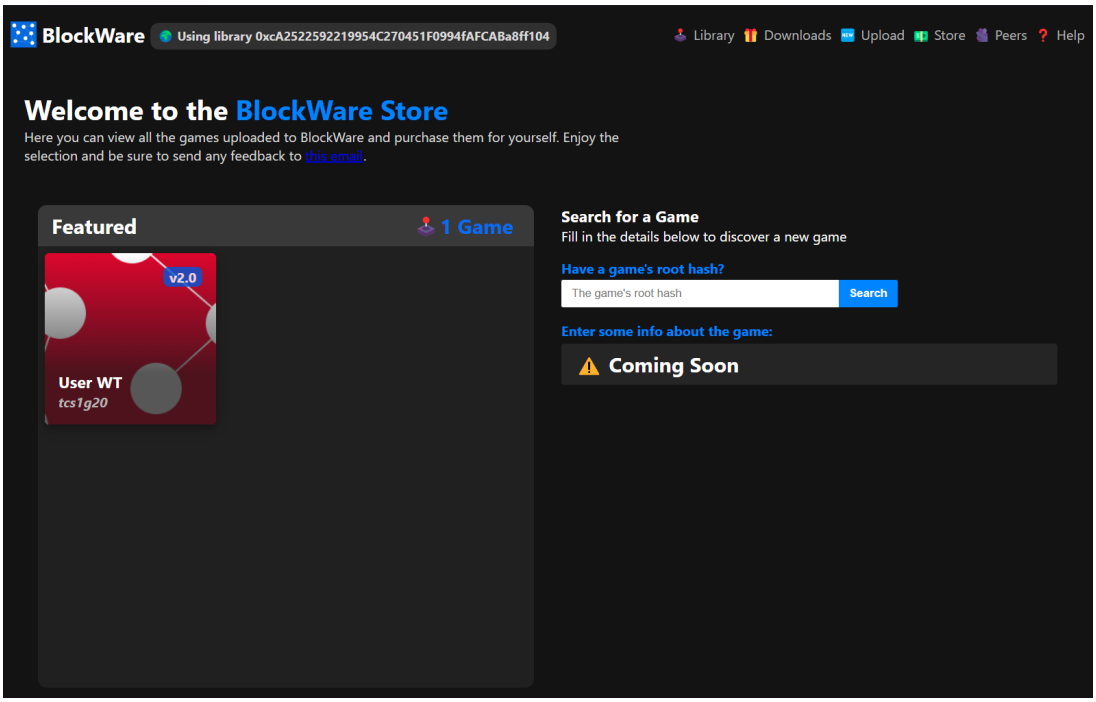


Figure B.8: *This is the store page where users will be able to find games. Users can find the store page of a game using its root hash. A lot of the typical store page functionalities were left out of scope due to their complexity.*

Appendix C

Other Diagrams

Requirement Analysis

Figure C.1 shows a mind-map which shows the relationships between primary stakeholders, key pieces of data and key technologies that were expected to be of use for this project. This allowed me to come up with a suitable set of requirements.

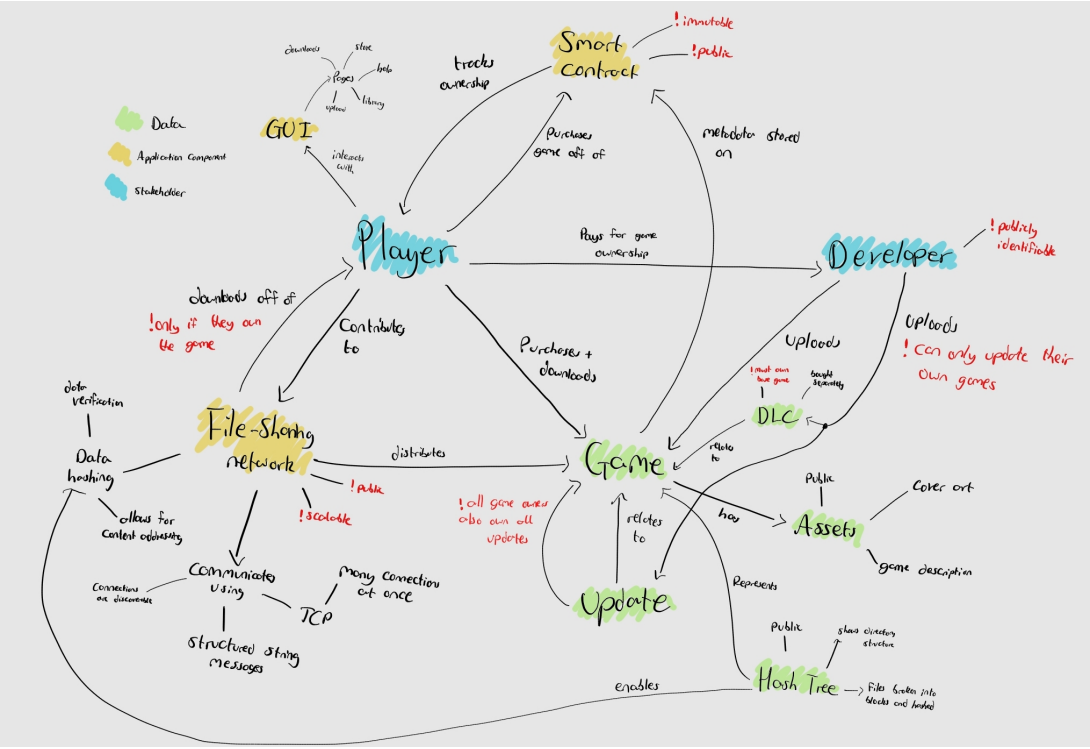


Figure C.1: *How key elements of the project are related to each other. This diagram was used to help determine the requirements for the application.*

Appendix D

Progress Report

Chapter 1

Problem Statement

1.1 The Problem

Video games are often large and highly popular pieces of software that are typically distributed for developers by a third party platform like Steam or Epic Games. Whilst these platforms provide benefits such as availability, and some social features they have some major downsides that include:

(a) taking a large cut of all revenue,
*Steam take a 30% cut [**?**, **?**]*
(b) being vulnerable to censorship from governments,
*The Chinese version of Steam is heavily censored [**?**]*
(c) the user's access to their games is linked to the platform.
If the platform shuts down, the user loses all their games

1.2 Goals

The goal of this project is to implement a large-scale distribution platform that will allow game developers to release and continuously update their games on a public network by directly interacting with their users. This is in the aim to provide greater profits to developer's, freedom from censorship, and better digital ownership for the user.

1.3 Scope

This project will look at how the Ethereum blockchain and smart contracts can be used to create a large-scale distribution platform for video games. This will be deployed to a 'testnet', where Ether has no value and applications can be tested in a live environment. Whilst the focus on this application is on video games, this should be a valid solution for software of all types.

The application will consist of a set of smart contracts, written in Solidity, with tests and a basic GUI to be written using TypeScript and React.js. Tools like Ganache and Truffle will also be useful for blockchain development.

Chapter 2

Background Research

2.1 BitTorrent

It is an unrealistic to expect that every game uploaded to the network will be downloaded by every user so only a subset of users will have the game installed and available to share. In this section and Section 3.2, I will look at how various peer-to-peer file-sharing networks allow users to discover and download content that is fragmented across the network.

BitTorrent [**?**, **?**] is the most popular P2P file-sharing platform, in which users will barter for chunks of files by downloading and uploading them in a tit-for-tat fashion, such that peers with a high upload rate will typically also have a high download rate. It is estimated that tens of millions of users use BitTorrent every day [**?**].

Download Protocol

For a user to download data from BitTorrent they would:

1. Find the corresponding .torrent file that contains metadata about the torrent.
2. The user will find peers, using a tracker identified in the .torrent, that are also interested in that content and will establish connections with them.
3. The user will download blocks[1], from peers, based upon the following priority:
   (a) **Strict Priority** Data is split into pieces and sub-pieces with the aim that once a given sub-piece is requested then all of the other sub-pieces in the same piece are requested.
   (b) **Rarest First** Aims to download the piece that the fewest peers have to increase supply.
   (c) **Random First Piece** When a peer has no pieces, it will try to get one as soon as possible to be able to contribute.
4. The node will continuously upload blocks it has while active.

Availability

One of the most significant issues facing BitTorrent is the availability of torrents, where *'38% of torrents become unavailable in the first month'* [**?**] and that *'the majority of users disconnect from the network within a few hours after the download has finished'* [**?**]. This paper [**?**] looks at how the use of multiple trackers for the same content and DHTs can be used to boost availability.

[1] nodes may reject downloads without the user providing data themselves in a tit-for-tat fashion

2.2 Ethereum

Ethereum is a Turing-complete, distributed, transaction-based blockchain that allows the deployment of decentralized applications through the use of smart contracts. Ether is the currency used on Ethereum and can be traded between accounts and is used to execute smart contract code on the network.

Smart Contracts

A smart contract is an executable piece of code, written in Solidity, that will automatically execute on every node in the Ethereum network when certain conditions are met. Smart contracts are enforced by the blockchain network and remove the need for intermediaries and reduce the potential of contractual disputes.

Gas is used to measure the computational effort of running a smart contract and must be paid, in Ether, before being processed and added to the blockchain. This helps prevent DoS attacks and provides economic incentives for users to behave in a way that benefits the whole network.

Example Use Cases

Some examples of applications that can be deployed to the Ethereum network are:

- Financial applications, such as decentralised exchanges and payment systems,
- supply chain management and tracking,
- voting and governance systems,
- unique digital asset systems, and
- data storage and sharing platforms.

Chapter 3

Literature Review

3.1 Blockchain-Based Cloud Storage

Blockchain technology can be leveraged for distributed cloud storage to provide both public and private storage. In table 3.1, I detail some examples of how blockchain has been used to create cloud storage platforms:

One gap found when researching these solutions was that few offered file versioning that would allow a user to view previous versions of uploaded data. File versioning is a particularly important to this project as users will likely all have varying versions of the same software.

Paper	**Description of Solution**
Blockchain Based Data Integrity Verification in P2P Cloud Storage [**?**]	This paper uses Merkle trees to help verify the integrity of data within a P2P blockchain cloud storage network. It also looks at how different structures of Merkle trees effect the performance of the system.
Deduplication with Blockchain for Secure Cloud Storage [**?**]	This paper describes a deduplication scheme that uses the blockchain to record storage information and distribute files to multiple servers. This is implemented as a set of smart contracts.
Block-secure: Blockchain based scheme for secure P2P cloud storage [**?**]	A distributed cloud system in which users divide their own data into encrypted chunks and upload those chunks randomly into the blockchain, P2P network.
Blockchain-Based Medical Records Secure Storage and Medical Service Framework [**?**]	Describes a secure and immutable storage scheme to manage personal medical records as well as a service framework to allow for the sharing of these records.
A Blockchain-Based Framework for Data Sharing With Fine-Grained Access Control in Decentralized Storage Systems [**?**]	This solution uses IPFS, Ethereum and ABE technology to provide distributed cloud storage with an access rights management system using secret keys distributed by the data owner.

Blockchain based Proxy Re-Encryption Scheme for Secure IoT Data Sharing [?]	An IoT distributed cloud system for encrypted IoT data that uses a proxy re-encryption scheme that allows the data to only be visible to the owner and any persons present in the smart contract.

Table 3.1: *Examples of blockchain cloud storage systems [?]*

3.2 P2P File Sharing

It is unreasonable to expect every node to have a copy of each game uploaded to the blockchain so data will be fragmented across the network. This project will use ideas from various P2P file-sharing networks to help connect nodes interested in the same content Table 3.2 shows some example p2p file-sharing networks.

The main issues involving these networks are:

1. **Trust** Nodes are typically anonymous and you can never fully trust that what you're downloading isn't malicious, and
2. **Payment** These platform don't allow users to pay for content and are generally large sources of piracy.

System	Description of Solution
IPFS [?]	IPFS is a content-addressable, block storage system which forms a Merkle DAG, a data structure that allows the construction of versioned file systems, blockchains and a Permanent Web.
BitTorrent [?]	BitTorrent is a p2p file-sharing system that has user bartering for chunks of data in a tit-for-tat fashion, which provides incentive for users to contribute to the network. More on BitTorrent can be found in Section 2.1
AFS [?, ?]	The Andrew File System was a prototype distributed system by IBM and Carnegie-Mellon University in the 1980s that allowed users to access their files from any computer in the network.
Napster [?]	Napster uses a cluster of centralized servers to maintain an index of every file currently available and which peers have access to it. A node will maintain a connection to this central server and will query it to find files; the server responds with a list of peers and their bandwidth and the node will form a connection with one or many of them and download the data.
Gnutella [?]	Gnutella nodes form an overlay network by sending *ping-pong* messages. When a node sends a *ping* message to their peers, each of them replies with a *pong* message and the *ping* is forwarded to their peers. To download a file, a node will flood a message to its neighbors, who will check if they have and return a message saying so; regardless, the node will continue to flood their request till they find a suitable node to download off of.

Table 3.2: *Various global distributed file systems.*

Chapter 4

Design

4.1 Stakeholders & Requirements

Stakeholders

Game Developers ***primary***

This group will use the application to release their games and its updates to their users, who they will reward for helping to distribute it.

Players ***primary***

This group will use this application to downloaded and update their games off of. They may also contribute to the distribution of the games to other players for an incentive provided by the developers.

Other Platforms ***secondary***

This group consists of platforms like Steam or Epic Games, which serve as the main competitor to this application. It is likely that as more developers choose this application, this group will see a loss in revenue.

Requirements

Tables 4.1 and 4.2 show the functional and non-functional requirements of this project organized using MoSCoW prioritisation.

Functional Requirements

ID	Description
	Must
F_M1	Store game metadata on a blockchain
F_M2	A node must data as constant-sized shards from its peers
F_M3	A node must be able to discover peers who have their desired game installed
F_M4	Games must be updatable through the blockchain
F_M5	A node must be able to upload games

F_M6	A node must be able to download games in their entirety from nodes in the network.
F_M7	A node must be able to verify the integrity of each block it downloads
F_M8	The application should run on the Ethereum network
F_M9	Users must be able to purchase games from developers over the network
F_M10	Users must be able to prove they have purchased a game
	Should
F_S1	Seeders should have a way to prove how much data they have seeded
F_S2	Seeders will only upload content to users who have a valid proof of purchase
F_S3	Allow for the distribution of DLC for games
	Could
F_C1	Allow users to request specific game versions
F_C2	Provide a simple GUI for interacting with the blockchain

Table 4.1: *These requirements define the functions of the application in terms of a behavioural specification*

Non-Functional Requirements

ID	Description
	Must
NF_M1	The application is decentralized and cannot be controlled by any one party
NF_M2	Any user must be able to join and contribute to the network
NF_M3	Game uploaders should be publicly identifiable
NF_M4	Metadata required to download the game should be immutable
	Should
NF_S1	This application must be scalable, such that many users can upload and download the same game at the same time.
NF_S2	Only the original uploader can upload an update to their game
NF_S3	Only the original uploader can upload a DLC for their game
	Could
NF_C1	The application could have an intuitive GUI

Table 4.2: *Requirements that specify the criteria used to judge the operation of this application*

4.2 Design Considerations

Architecture

Figure 4.1 shows the architecture of this application:

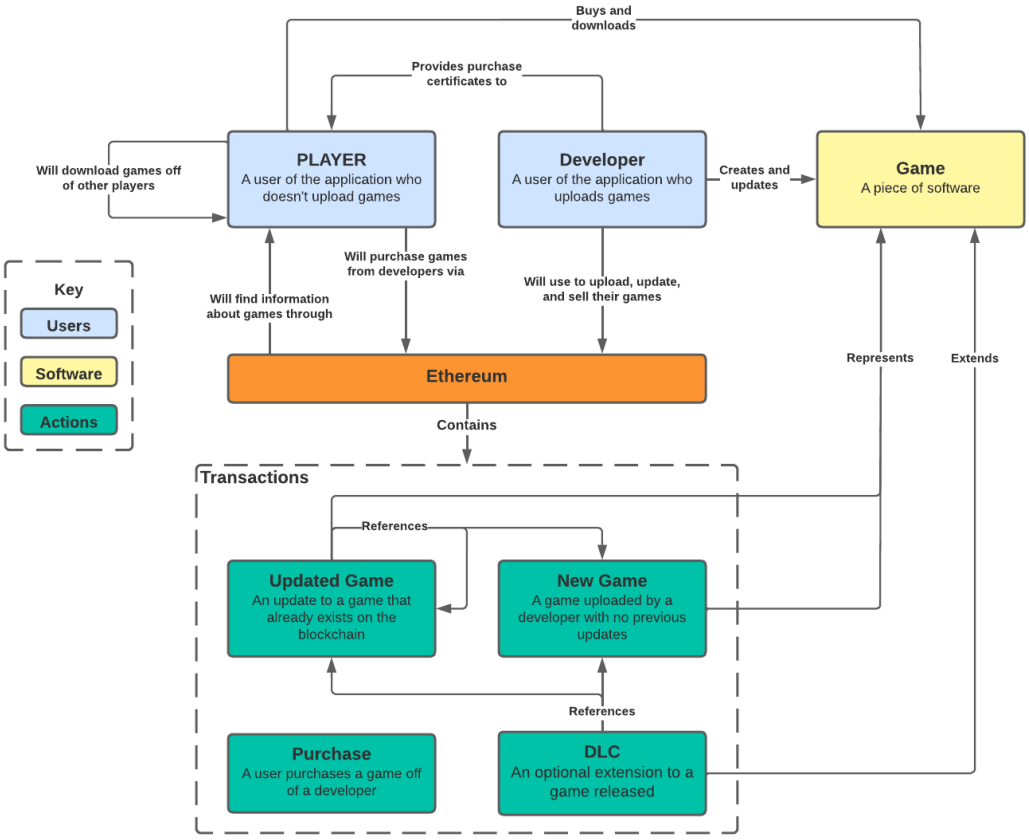


Figure 4.1: *Architecture of the application*

Type of Blockchain

To satisfy **NF_M1** and **NF_M2**, we will need to use a public blockchain, which will benefit our project by:

- being accessible to a larger user-base, which should boost availability and scalability (**NF_S1**),
- reducing the risk of censorship (**NF_M1**), and
- providing greater data integrity (**NF_M4**)

Ethereum is a public blockchain that allows developers to publish their own distributed applications to it. It comes with an extensive development toolchain so is an obvious choice for this project (**F_M8**).

Sequence Diagram

Figure 4.2, shows the main interactions between actors in the application.

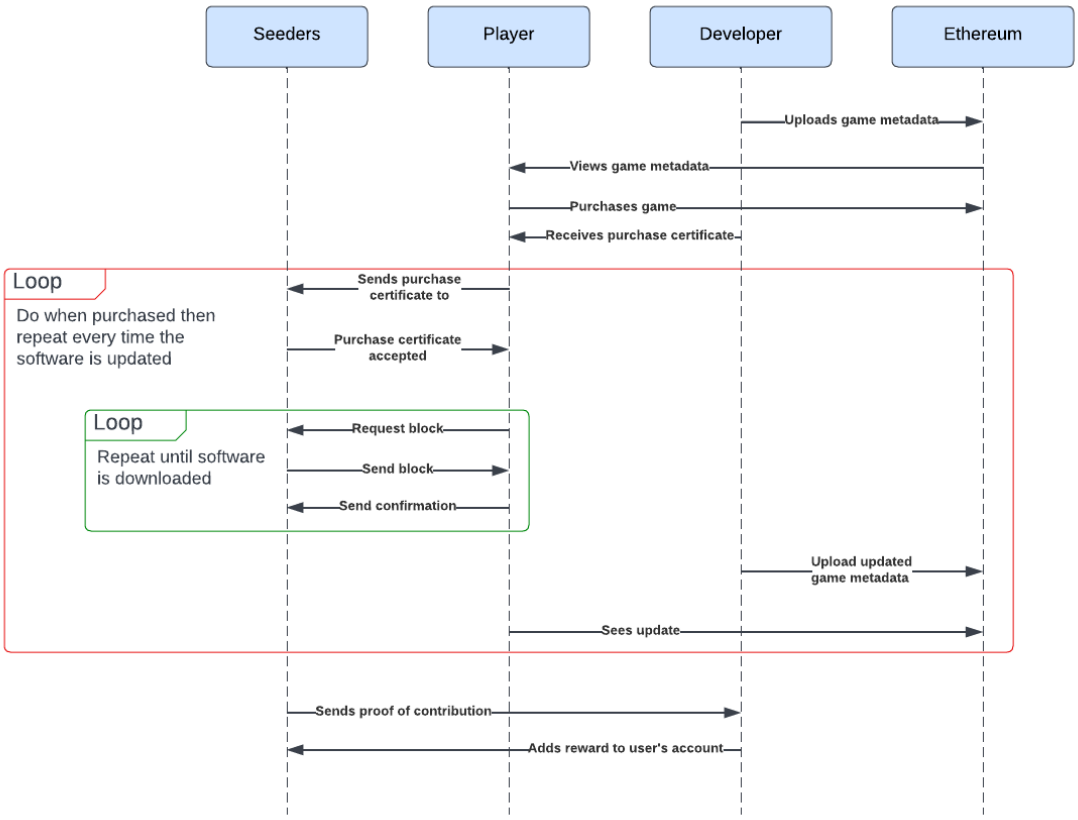


Figure 4.2: *A sequence diagram showing some of the main interactions within this application*

Uploading Content

For developers to upload their game (**F_M5**), they must provide a digital certificate to prove their identity (**NF_M3**) as well as the required metadata (**F_M1**) for identifying, downloading and verifying their game. The developer is then expected to allow users to purchase the game off them and seed the game to at least an initial group of users.

Purchasing Content

Users will purchase content from sellers using Ether (**F_M9**) and will be provided with a proof of purchase (**F_M10**) that is encrypted with the developer's private key. By using public key infrastructure, it gives proof of purchases authenticity that allows them to be validated by any node in the network.

The proof of purchase will include: the root hash of the title purchased, the seller's address, the buyer's address, a timestamp, and a unique seeder token to be used for proving distribution.

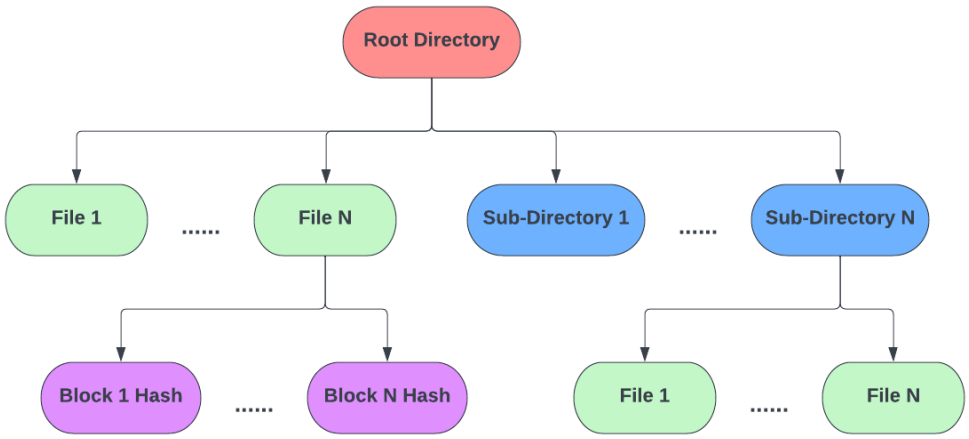


Figure 4.3: *Hash data about shards will be stored in a tree that mimics the folder structure of the data*

Downloading Content

Games will be content addressable, using their root hash stored on the blockchain. This will be used to help connect nodes who are interested in the same content (**F_M3**). Once a user connects to a node it will:

1. Send their proof of purchase (**F_S2**).
2. request individual shards from the node using the shard's hash (**F_M2**),
3. use the metadata from the blockchain to verify the shard's contents (**F_M7**),
4. send a confirmation message that proves the successful transfer of a block (**F_S1**),
5. and repeat this until the entirety of the game is installed (**F_M6**).

As the blockchain will only be used to store metadata about games uploaded to the network, not every node will have each game installed. Instead, this project will take ideas from networks like BitTorrent, where nodes will seek out peers who have the game installed using a tracker hosted by the uploader.

Updating Content

To satisfy **F_M4**, developers will perform the same steps as in **Uploading Content** but will also include the hash of a previous block that contains the older version of the game. This will include the restriction that only the original uploader can upload an update for their game (**NF_S2**).

It is likely that many shards will persist between versions so a node will only ever download the new or changed data. To satisfy **F_C1**, a node may optionally keep older shards that have been removed or changed.

Downloadable Content

Developers will typically offer paid or free DLC (**F_S3**) for their games that users will purchase separately. Each DLC will need:

1. **Dependency** A reference to the oldest version of the game they apply to, and
2. **Previous Version** A pointer to the previous version of the DLC.

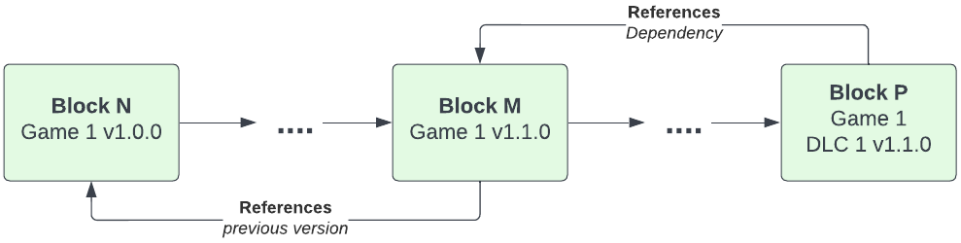


Figure 4.4: *How blocks relate to each other. An update will reference the previous version whilst a DLC will reference, which piece of software and version it is dependent on.*

Proving Contribution

A purchase certificate will include a confidential *seeder token*. When a user successful downloads a shard of data they will send a confirmation message, including their seeder token, that is encrypted with the game developer's public key. A user will prove their contribution (**F_S1**) by sending a collection of these messages to the developer, who can decrypt and validate the tokens.

4.3 Limitations

This project will not attempt to mimic any of the social features (friends, achievements, message boards, etc.) provided by platforms like Steam.

Section 2.1 highlights the issue of availability in p2p sharing networks and this platform will likely suffer from similar issues. These can be mitigated by having an active community or good incentives to help distribute the game.

Chapter 5

Project Management

5.1 Risk Assessment

Risk	Loss	Prob	Risk	Mitigation
Difficulty with blockchain development	2	3	6	I will seek advice from my supervisor about how to tackle certain problems and if necessary, what aspects of my project I should change.
Personal illness	3	2	6	Depending on the amount of lost time, I may ignore some of the SHOULD or COULD requirements.
Laptop damaged or lost	3	1	5	All work is stored using version control and periodic backups will be made and stored locally and in cloud storage. I have other devices that could be used to continue development.
The application is not finished	1	3	3	Using agile development will ensure that I will at least have a minimal working application. If I feel that I am running out of time, I will focus on testing and finalising the report.

Table 5.1: *The risk assessment of this project*

5.2 Work to Date

My work has primarily been on research, looking at how blockchain has been used to build cloud storage systems as well as how various peer-to-peer function and perform. I have proposed a design for the application to be built on the EVM.

5.3 Plan of Future Work

Below is a high level plan for the work I have remaining.

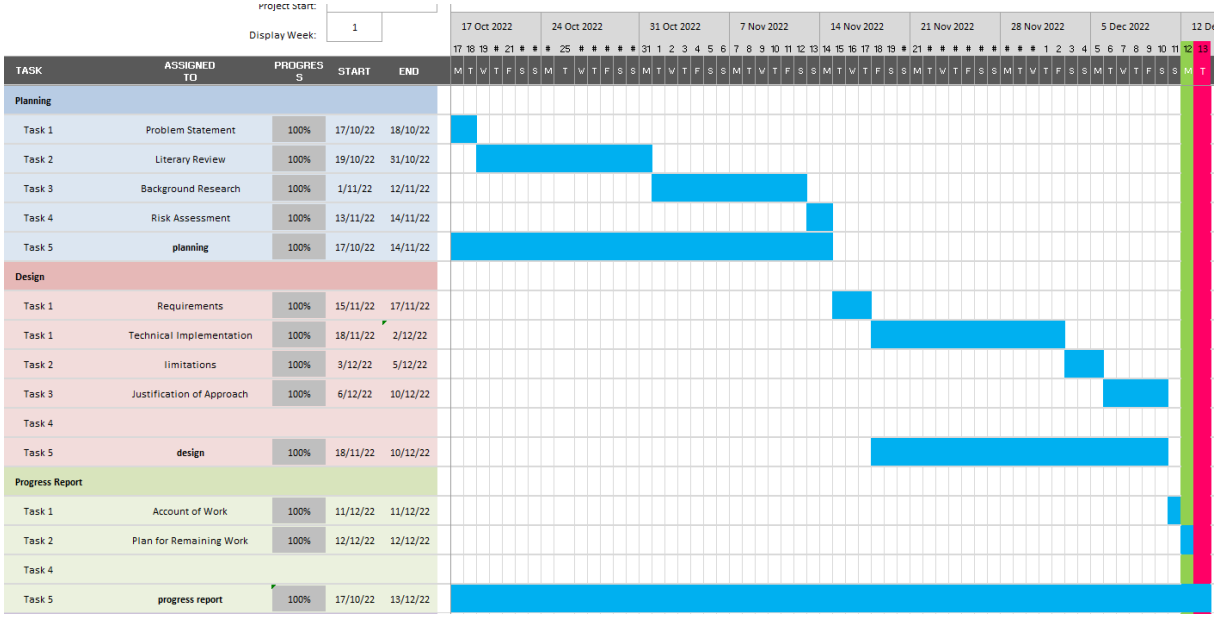


Figure 5.1: *A Gantt chart for my work up until the progress report.*

Implementation & Testing

In this phase I will use the agile development methodology to build my application. My sprints will all be structured into three phases:

1. **Preparation** Deciding on the set of requirements to complete and making any initial design decisions and diagrams,
2. **Implementation** Using test-driven development, I will work on requirements based on their prioritization, and
3. **Review** I will discuss the completed work in that sprint including design choices, what was completed, and any issues.

By using a preparation and review phase with each sprint, I can make detailed notes on my implementation as I go so when I come to write the full report I have a strong starting point.

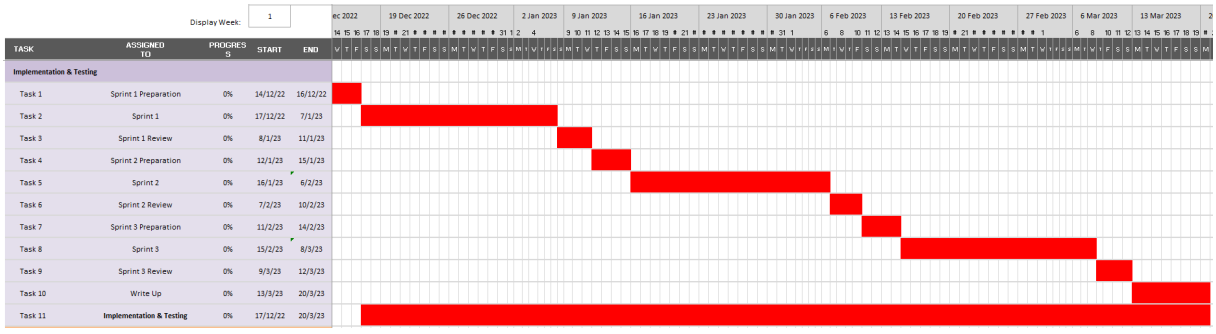


Figure 5.2: *The plan for my three sprints*

Testing Strategy and Results

This phase will consists of several sections:

1. **Testing Strategy** The approach I used for writing tests throughout the implementation phase and how these were used to evaluate the completion of the requirements set out in Section 4.1.
2. **Test Results** A report on the results of my automated and manual testing.

Evaluation

This phase will have me evaluating how successful my project was, as a whole, by focusing on several key areas:

1. **Project Organisation** How successfully did I structure my time in this project?
2. **Outcome of the Application** How successful was my application in regards to a solution to the problem set out in Section 1.1 and in terms of the requirements set out in Section 4.1?
3. **Limitations and Future Improvements** What were the limitations of my project and what would I change about it if I had more time or were to start again?

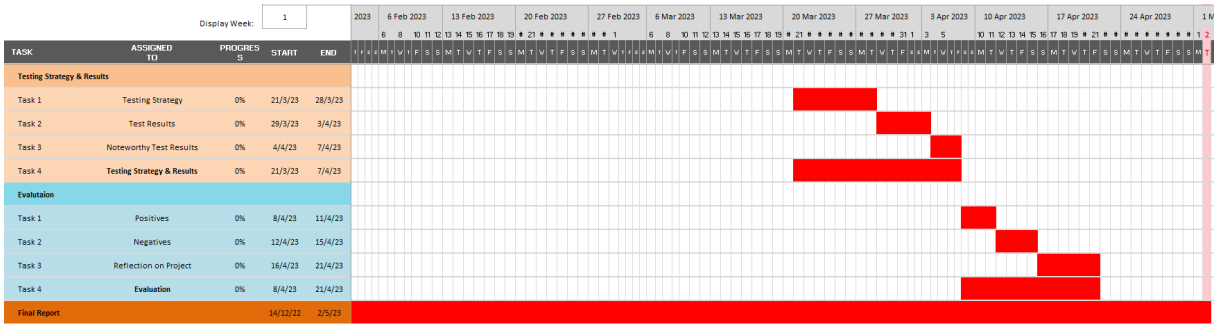


Figure 5.3: *The plan for my testing and evaluation phases*

Final Report

I will be writing the first draft of my report throughout the project, aiming to finalise sections at the end of each phase of this project. Any leftover time at the end will be focused on finishing the report.