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

Thomas Smith, tcs1g20
April 3, 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

Video game developers will often have to rely on third party platforms for the distribution of their games; this comes at a large monetary cost to the developer and leaves users at a greater risk of censorship and with weak digital ownership that is reliant on the platform staying active. This project uses the Ethereum blockchain to facilitate the large-scale distribution and continuous updating of video games that allows developers to directly interact with their users, who will now have true digital ownership.

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, Leonardo Aniello, for his support throughout this project.

Contents

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 [1, 2]

(b) being vulnerable to censorship from governments,
*The Chinese version of Steam is heavily censored [**steamdb_steam_2021**]*

(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 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.
2. A local application to be run by users to interact with the smart contract and allow users to join a peer-to-peer network where they can download and upload games to.

Both sections will need to pass a series of acceptance tests to ensure that they meet the requirements I set out in Section 4.1.2. Further tests will consider how the application fares in a live environment as well as a local development one.

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 [3, 4] was chosen as part of my background research as it is one of the most popular P2P file-sharing platforms. In 2013 it was estimated that tens of millions of users used BitTorrent every day [5]. In BitTorrent, users barter for chunks of data 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.

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'* [3] and that *'the majority of users disconnect from the network within a few hours after the download has finished'* [4]. This

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

paper [6] looks at how the use of multiple trackers for the same content and DHTs can be used to boost availability.

2.2 Ethereum

Ethereum [7, 8] 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 in an Ethereum blockchain wil include a list of transactions that show the most recent state that can be calculated by any node by applying the smart contract code given for each transaction.

Smart Contracts

A smart contract is an executable piece of code, written in Solidity [9], 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.

Test Networks

An Ethereum test network is an instance of the Ethereum network 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 [10], Goerli [11], and Ropsten[2].

[2]The Ropsten test-net is declared as end of life as of December 2022 [12]

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 [13]	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 [14]	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 [15]	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 [16]	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 [17]	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 [18]	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 [19]

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 [20]	IPFS is a set of protocols for transferring and organising data over a content-addressable, peer-to-peer network. Data uploaded to an IPFS network is addressed using it content identfier CID, which is a cryptographic hash based upon its content. IPFS is open source and has many different implementations, such as Estuary or Kubo.
BitTorrent [4]	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.
Swarm [21]	Swarm is a distributed storage solution linked with Ethereum that has many similarities with IPFS [4]. 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.
AFS [22, 23]	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 [24]	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 [24]	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 Analysis

4.1.1 Stakeholders

Game Developers PRIMARY
This group will upload games, and any subsequent updates, to the applications. Users will purchase access to their games and contribute to the distribution of it in return for some reward that could be provided by this group.

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.

4.1.2 Requirements

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
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 publically 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.

[1]This project will only test deploying to an Ethereum test network

[2]See Section 4.2.

	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	This application could include measures to prevent/stop the distribution of illegal content.
NF-C2	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*

4.2 Design Considerations

Data

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	Ethereum	This data is the minimal set of information required for the unique identification of each game. See Section 4.2. This data is appropriate to store on Ethereum 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	This will be the compressed hash tree that will allow users to identify and verify the 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 will make it costly to store on Ethereum at a large scale and given that only a subset of users will actually ever want access to it, it would be wrong to store it on Ethereum. Instead IPFS can be used for reliable and fast access at a very large scale and we can embed the generated CID, from the upload to IPFS, in our smart contract instead.

Game Assets (**F-C2**)	Variable[3]	IPFS	This will represent any promotional material provided for the game that can be viewed on the game's store page. This will typically include cover art and a markdown file for the description and 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 Ethereum instead.
Game Data	*avg. 44GB* [4]	Peers	This will the data required to run the game and will be 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 Ethereum or IPFS[5]. Therefore, this project will use a custom P2P network for sharing data, which is described in Section 4.2.

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

Swarm [21] was considered as a decentralised storage and distribution platform over IPFS but was decided against as it would couple this project more tightly with Ethereum and the fact that IPFS has a much greater adoption.

Blockchain

Type of Blockchain

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.

[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 [25].
[5]IPFS and similar platforms provide no access control for the data stored there and any encryption based technique would be unviable.

Name	Description
	Metadata for each game
title	The name of the game.
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.
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, on the smart contract, of all users who have purchased the game (**F-M6**). Upon purchasing a game, a user will broadcast their new library to all of their peers.

Distributed File Sharing

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**).

Uploading Content

For a developer to upload their game (**F-M1**), they must provide the following:

- the metadata outlined in Section 4.2,
- a hash tree created from the root directory of the game, and
- an assets folder containing a piece of cover art *(cover.png)* and a description file *(description.md)*.

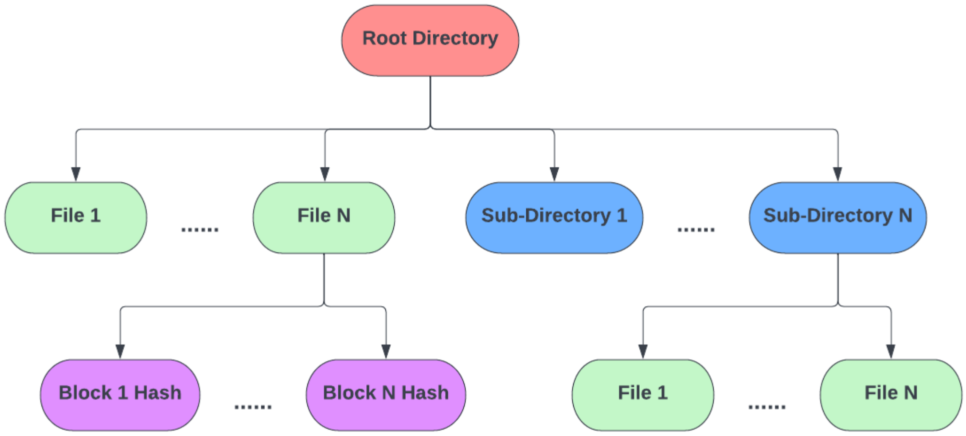

Figure 4.1: *The structure of a hash tree*

The developer should be able to enter the required fields into an upload page of the GUI and have the data generated and uploaded for them (**F-S2**).

Downloading Content

Like mentioned in Section 4.2, it is impractical to store the game's data on the blockchain or IPFS. Instead we will consider ideas from decentralised file-sharing networks, like discussed in Sections 3.2 & 2.1.

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:

1. Perform a handshake to determine each other's Ethereum address and public key.
2. P_{seeder} will verify that $P_{downloader}$ owns the game by checking the *purchased* mapping on the smart contract (**F-M6**) (**F-S1**).
3. $P_{downloader}$ will send requests for individual blocks to P_{seeder} (**F-M9**).
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 to many peers at once (**F-M7**) and will send download requests to the subset of their peers who also own the game. 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 perform the same steps outlined in Section 4.2 but must also provide the root hash of the most previous version of the game. Any users who have purchased the previous version will be added to the list of users who have purchased the new version (**F-M3**). Additionally, this will include the restriction that only the original uploader can upload an update for their game (**NF-M5**).

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.

4.3 Architecture

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.

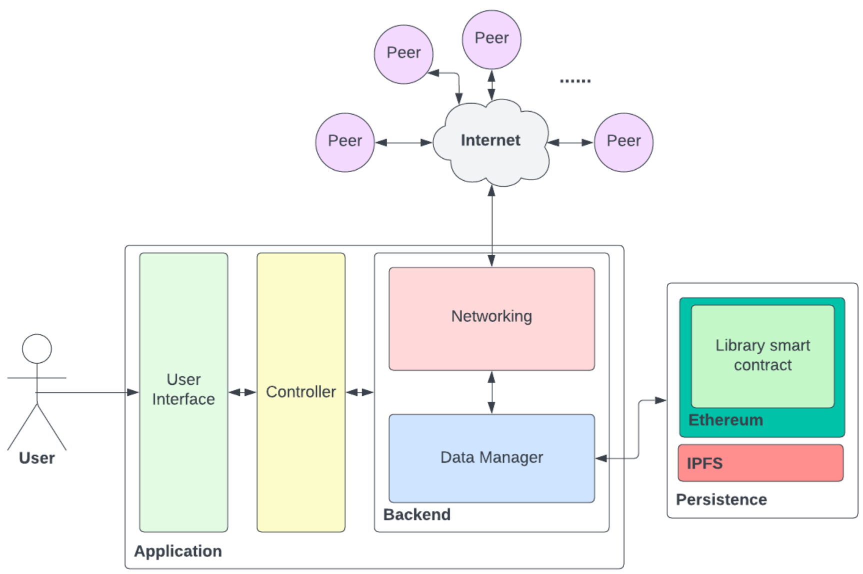

Figure 4.2: The layers of the application

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 4.2.

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 like GBP or USD 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.
- Users will have to source Ether from elsewhere before being able to purchase games, which may be intimidating to users who are not already familiar with the Ethereum ecosystem. Moreover, 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.

Backend

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

Networking

Users running this application will be a part of a distributed network of peers by creating and maintaining 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 4.3 describes these commands in detail.

The use of TCP will add a computational overhead to maintain proportional to the number of peers, which is not ideal for inactive channels. A UDP approach would be more scalable but would add greater complexity to the project, but should be considered when extending this application.

Address Verification When two peers connect they will perform a handshake to exchange their Ethereum addresses and public keys by sending signed messages to each other. This will allow a peer to identify what games another peer is allowed access to.

Message Handling One of the responsibilities of this component is to respond to requests sent by the Data Manager by sending and tracking messages to other peers to fetch requested data. Each message should be tracked by the peer for a given time period and resent if an appropriate response has not been received. Any duplicate requests sent by the data manager will be ignored if a pending request is active.

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;[$gameHash$];[$blockHash$]; SEND_BLOCK;[$gameHash$]; [$blockHash$];[$compressedData$];	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**). 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;[$message$] VALIDATE_RES;[$signedmessage$]	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**). 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;[$gameHash$] RECEIPT;[$gameHash$];[$signature$] ;[$message$]	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**). 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 PEERS;[$p_1hostname$] : [p_1port];...	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**). 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**).
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

The Data Manager has several responsibilities:

1. Track the user's owned games and know which are installed locally and which aren't.
2. Interact with the Library contract deployed to the Ethereum blockchain to discover, upload and purchase games (**F-M1**) (**F-M2**) (**F-M5**).
3. Fetch shards of game data from the disk to send to other peers.
4. Interact with IPFS to upload/fetch a game hash tree and assets (**F-M12**) (**F-C2**).
5. Sending requests to the networking layer to find and retrieve blocks of data from peers in the network.
6. Generate hash trees for games.

Other Details

Ignore File A standard implementation of a .ignore file should be included to indicate to the hash tree algorithm which files/folders to ignore. This is useful to ignore temporary or non-static files, which contents will vary by user and thus won't need to be distributed. To do this, the uploader adds a *.blockwareignore* file to the game's root directory.

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.
- **Store** Where user's can find and purchase new games that have been uploaded by other users.
- **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.
- **Peers** Where users can manage their list of connected peers.
- **Help** A help page to describe the application and all of its functionality (**NF-C2**)

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.

4.4 Downloading a Game

Figure 4.3 shows the standard sequence of events used for a user to download a game from this application. The developer will upload a game that is purchased by a user; this user will then proceed to download the game off of the developer using the commands described in Section 4.3.

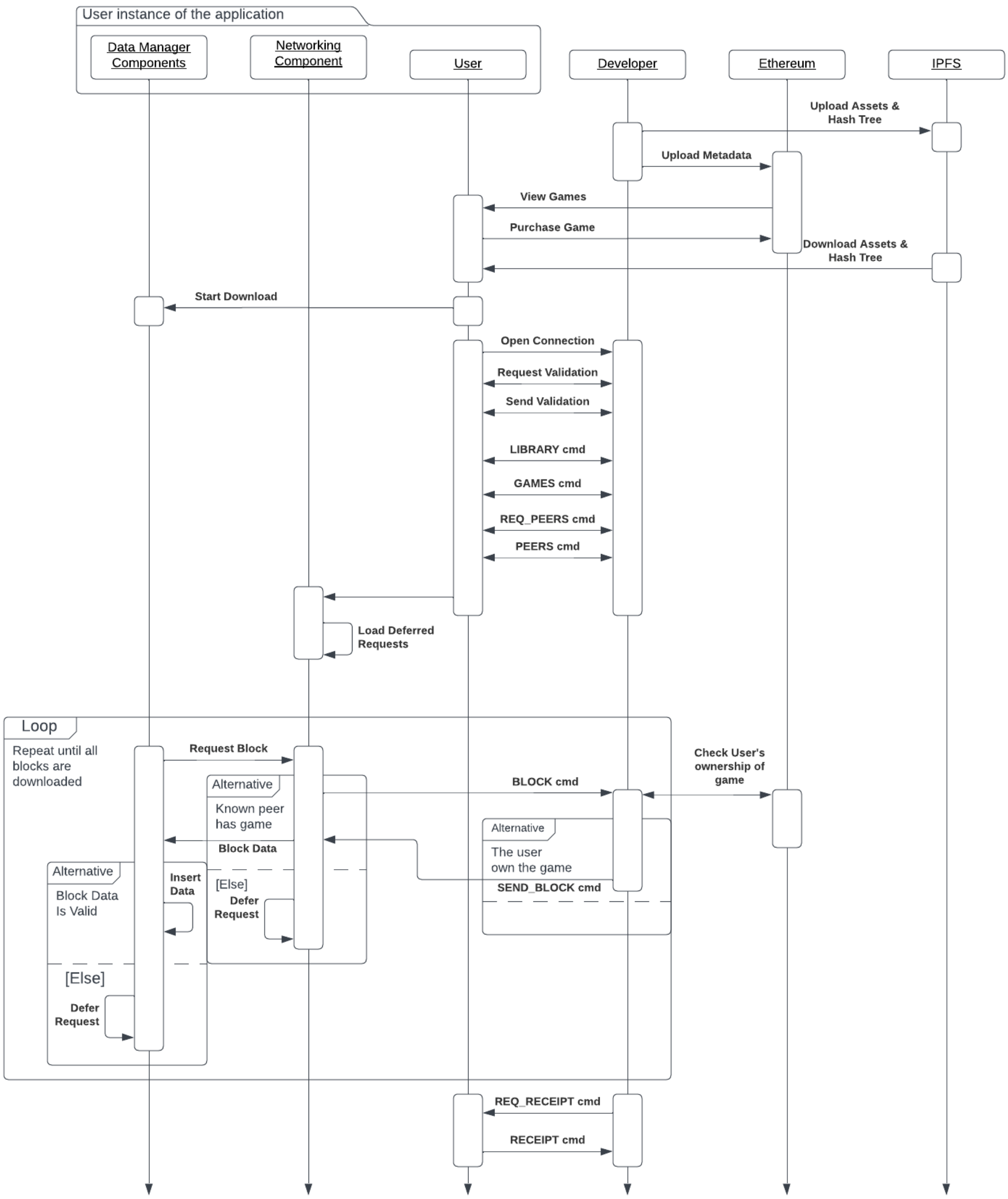

Figure 4.3: *A sequence diagram showing the main interactions needed to download a game from a single peer*

4.5 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.

4.6 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 2.1 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[6].
- **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.
- **Hash Trees** Modelling data as a hash tree means that each file will need at least one block so a game may have a large amount of blocks for not a lot of data and as each block has to be requested separately this will add a lot network overhead.

[6] An example would be how the micro-payment system works in Swarm [21]

Chapter 5

Implementation

5.1 Backend

The backend code for this application was written based upon the design from Section 4.3 and was done using the following tools:

Tool	Description & Reasoning
Go [26]	Go was chosen because of its simple syntax, high performance, strong standard library and third party packages for interacting with Ethereum.
go-ipfs-api [27]	A Go package used for interacting with the Kubo implementation of IPFS that gave an easy interface for downloading and uploading data to Kubo.
go-ethereum [28]	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.
Zap [29]	A logging library that is much faster than the standard library implementation and has better customisation.
Viper [30]	A configuration file management library that helps read, write, and access configuration options written to file.

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

5.2 Smart Contract

To develop and deploy a smart contract that met the criteria specified in Section 4.2, I used the following tools:

Tool	Description & Reasoning
Solidity [9]	The language used to write smart contracts for the Ethereum blockchain.
Sepolia [10]	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.

Alchemy [31]	Alchemy provides useful tools for interacting with Ethereum and specifically Sepolia, such as an ETH faucet and an RPC URL.
MetaMask [32]	A browser-based wallet that can easily be connected to other tools such as Alchemy or Remix.
Remix [33]	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 [34]	Ganache CLI was used to create a local Ethereum test-net that I could develop my application with.

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

The contract was successfully deployed [35] to the Sepolia test-net and can be interacted with by any user.

5.3 Frontend

The frontend code was developed from Section 4.3 to provide a user a GUI to interact with. Table 5.3 details the list of tools used.

Tool	**Description & Reasoning**
Wails [36]	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 [37]	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 [38] package was used to add multiple pages to the application and markdown-it [39] was used to render markdown files.
Pinia [40]	A state management tool for Vue.js that boosts the reusabiltiy of components and reduces the overall complexity of the frontend.
SASS [41]	An extension of CSS that is used to style DOM elements. This was essential in making the UI look nice and be accessible.

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 development of this application.

Tool	Description & Reasoning
Git [42] GitHub [43]	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 [44]	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 [45]	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 [46]	Lucidchart was used to create all of the diagrams for this project.

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

Chapter 6

Testing

6.1 Overview

My approach to testing will consist of the following principles:

1. **Test Driven Development [47]** 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 add to. See Appendix B.1 for an example of the documentation and structure expected for each function test.

Tools

Table 6.1 shows the different tools I used to test my application and a justification as to why they were included.

Tool	Description & Reasoning
Go Testing	The testing package included with Go's standard library was sufficient to produce most of the test cases required for this project.
testify [48]	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*

6.2 Unit Testing

Unit tests are to ensure the correct functionality of individual blocks of code within the application. This is to ensure that each function responds appropriately to both well-formed and illegal arguments and considers

6.3 Integration Testing

Profiles

Profiles are minimal versions of the application that can be run on external devices for the purposes of testing how my application fares in a simulated environment. The following profiles are included:

- **Listen Only** A client which will fetch a repository from Git and upload it as a game to the network. Once uploaded it will listen for incoming messages and reply accordingly. This is supposed to simulate an ideal peer.
- **Send Only** This client will never respond to messages but will send them periodically to the peer. This represents a selfish client.
- **Spam** This client will spam the peer with expensive messages, such as requesting a block, and should trigger the client to disconnect the spammer.
- **Unreliable** This client will represent an unreliable peer who will take a long time to respond to messasges and may reply with the wrong contents at random.

Deployed Implementation

Distributing Instances

To prove that the application could work in a live environment, we will need to deploy a set of instances of the application to different devices that communicate over the internet. An example deployment might consist of instances deployed to:

- a local machine to initiate tests,
- a VPS running on a cloud service provider,
-

Ethereum Test-Net

The Library Smart Contract, from Section **??**, will need to be deployed to an Ethereum test-net to allow instances of the application, that are distributed over the

6.4 Acceptance Testing

To ensure this application meets the requirements as described in Section 4.1.2, each requirement will be part of at least one user walkthrough. In Table 6.2 we give a high level description of each user walkthrough and any comments or results found from it. Appendix B.1 will give a more in-detail example of user walkthrough 2.

Id	Requirements	Description	Success
1	(**F-M1**) (**F-M5**) (**F-M12**) (**F-S2**) (**F-C2**)	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**)	1. P_1 and P_2 own game G_1. 2. P_1 connects to P_2. 3. P_1 and P_2 exchange Ethereum addresses. 4. P_1 starts a download for G_1. 5. P_1 sends requests for blocks to P_2. 6. P_2 queries the smart contract to verify that P_1 owns G_1. 7. P_2 will respond to P_1 with the requested data. 8. P_1 will verify each block of data received using its hash. 9. P_1 will have full downloaded G_1	YES
3	(**F-M7**) (**F-M8**) (**NF-S1**)	Refer to the benchmark tests described in Section 6.5.	YES
4	(**F-M2**) (**F-M3**) (**F-M6**) (**NF-M5**)	1. P_1 is the original uploader of G_1. 2. P_1 uploads an update to G_1, G_2. 3. P_2 owns G_1 and thus G_2. 4. P_2 will successfully download G_2.	NO

Table 6.2: *The set of user walkthroughs used to prove the completeness of this project's requirements.*

6.5 Benchmarking

This project uses benchmarking to determine the overall performance and scalability of the application, whilst also being useful in discovering any bottlenecks or bugs. Specifically, this section will test how varying the following conditions will effect the download speed of a game:

1. how many peers we are connected to,
2. how the data for the game is distributed in terms of file size and the number of files, and
3. the block size that each file is broken up into.

To ensure consistency in these results, we include the following constraints:

- All benchmarks should be ran on the same hardware using the same OS,
- All tests should be ran three times,
- Test data should be pseudo-random, and
- All projects should aim for the target size of 40GB to match the average game size given in Section 4.2.

Number of Peers

This benchmark allows us to observe how the application scales when dealing with many peers at the same time and how if affects the overall performance of the application.

For each run, we will create a project with 500 files, each of size 80MB and a shard size of 4MiB. We will then run N peers locally to simulate a perfect network connection.

	Runtime (s)			
Peers	**1**	**2**	**3**	**avg.**
1	56	65	63	61.3
2	65	64	60	63
4	66	65	65	65.7
8	66	62	60	62.7

Table 6.3: *How varying the peer count affects download speed*

These results show us that increasing the number of peers does not increase or decrease the download speed. However, some important notes about these results are:

- Having more peers distributes the workload across the network. This reduces the load on individual peers, who can then share data with other users.
- In reality, peers may have variable or weak network connections so having more peers may be a much larger contributing factor to download speed.
- There may be a bottleneck elsewhere in the application. For example, we may see better performance using UDP instead of TCP.

Game Size

This benchmark will be useful in discovering an optimal strategy for determining the directory structure of games uploaded to the network to allow developers to optimise their uploads to give the greatest download speed.

For each run, we will create a project with F files of size SMB, such that $F \times S = 40GB$, and a shard size of 4MiB. We will then run 1 peer locally to simulate a perfect network connection.

File		**Runtime (s)**			
Count	**Size (MB)**	**1**	**2**	**3**	**avg.**
200	200	57	55	58	57
100	400	56	52	58	55
50	800	110	109	110	110
25	1,600	110	108	114	111
5	8,000	311	318	311	313
1	40,000	1,548			

Table 6.4: *How varying file count and size affects download speed*

From these results, we can see that distributing data across a larger pool of files results in a greater download speed. This occurred for several reasons:

1. A file can only be written to by one process at one time so having less files will reduce the potential for parallelisation.
2. For each block downloaded: we open a writer to a file, write a single block, and close the writer. This adds a lot of overhead for files that have many blocks.

There are several optimisations we could make to improve this result:

1. Using batch requests would increase the amount of data we would have to write to a file at once.
2. Using a memory buffer before writing to disk to increase the chance that we are writing many blocks to the same file at once.

Block Size

This benchmark will be useful in determining an optimal block size to use that maximises download speed.

For each run we will create a new project with 500 files, each of size 80MB, and a shard size of BMiB. We will then run 1 peer locally to simulate a perfect network connection.

		Runtime (s)			
Block Size (MiB)	**Total Blocks**	**1**	**2**	**3**	**avg.**
1					
2					
4					
8					
16					

Table 6.5: *How varying the shard size of the hash tree affects download speed*

Chapter 7

Project Management

7.1 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 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*

7.2 Sprint Plans

The use of the Agile Methodology [49] with sprints was essential in managing my time and ensuring that I was working on the most important aspects of my project first. The use of MoSCoW prioritisation and by then dividing my requirements into logical groups I was able to effectively target key aspects of my application in bulk. The use of test-driven development [47] meant that at the end of each sprint, each piece of code I wrote was tested and I could move on.

For each sprint we will detail the planned requirements, whether they were completed or not, as well as any general comments about that sprint.

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 7.2 shows the requirements included for Sprint 1 and whether they were completed or not.

This sprint was largely problem-free as I didn't have much to learn to be able to complete this and could rely heavily on my design to structure my implementation.

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	All benchmark tests 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 shows the download of a game in its entirety.
(**F-M12**)	STARTED	The algorithm to generate a hash tree and the using of it to download data was implemented but no way to upload it anywhere.
(**NF-M2**)	YES	User walkthrough ... shows that any user can establish a connection with any other user.
(**NF-S1**)	STARTED	Users will form many connections concurrently and optimisations were made using the producer/consumer pattern to complete actions like inserting data, or reqeusting data.

Table 7.2: *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 7.3 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 ... show the ability to upload game metadata to Ethereum.
(**F-M2**)	YES	Unit tests for the Library smart contract and user walkthrough ... 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 [35]. All user walkthroughs will form connections to this smart contract.
(**F-M5**)	YES	Unit tests for the Library smart contract and user walkthrough ... 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 4.3. These pages had little styling or reactivity but could perform the required basic functions. See Appendix A 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 smart contract show the restriction that only the original uploader can release an update.

Table 7.3: *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 7.4 shows the list of requirements for this sprint and whether or not they were completed.

I definitely found this sprint to be the most difficult for a number of reasons:

1. As I was beginning to test my application with other simulated applications, I kept coming across bugs that were hard to locate and replicate, which made fixing them tedious.
2. I didn't have as much free time to spend working on this project due to the other modules I was taking at the time.
3. Some of the requirements pitched introduced relatively complex mechanics that I was not entirely satisfied with but didn't have the time to redesign them.

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 micro-payment system, like present in Swam [21], 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 A for screenshots of the final versions.
(**NF-C1**)	NO	Completing this requirement would be incredibly complex and was decided against being completed. Preventing the distribution of illegal content is an important consideration moving forward to help keep the platform safe.
(**NF-C2**)	YES	A help page was included answering some questions that new users may have about the application.

Table 7.4: *Requirements included for Sprint 3*

7.3 Gantt Chart

A Gantt chart was used to give myself a high-level overview of how my time should be spent so that I would have appropriate time to complete the necessary aspects of this project. This was useful in prioritising different aspects and knowing when to plan deadlines for myself (for example, having a hard cut-off point for my implementation).

Figure 7.1 shows the Gantt chart for this project up util the progress report submission, which includes the planning and design phases.

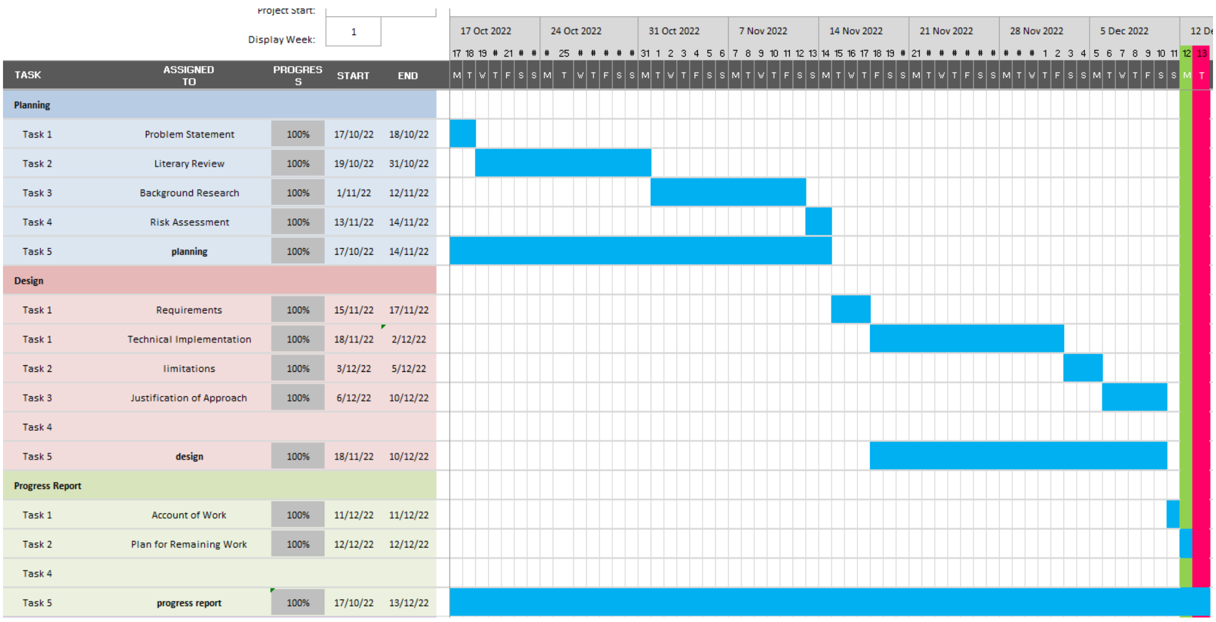

Figure 7.1: *Gantt Chart leading up to the progress report*

Figure 7.2 shows the Gantt chart for the implementation phase of this project. The implementation phase was split into three sprints, as detailed in Section 7.2, each with their own planning phase.

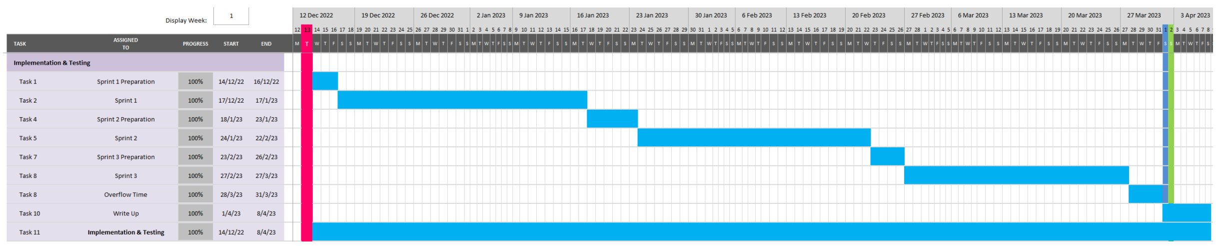

Figure 7.2: *Gantt Chart showing the three sprints*

Figure 7.3 shows the Gantt chart for the final testing and evaluation phases of my application. As I used test-driven development, the testing phase only needed to consist of acceptance and benchmark testing among some other fixes.

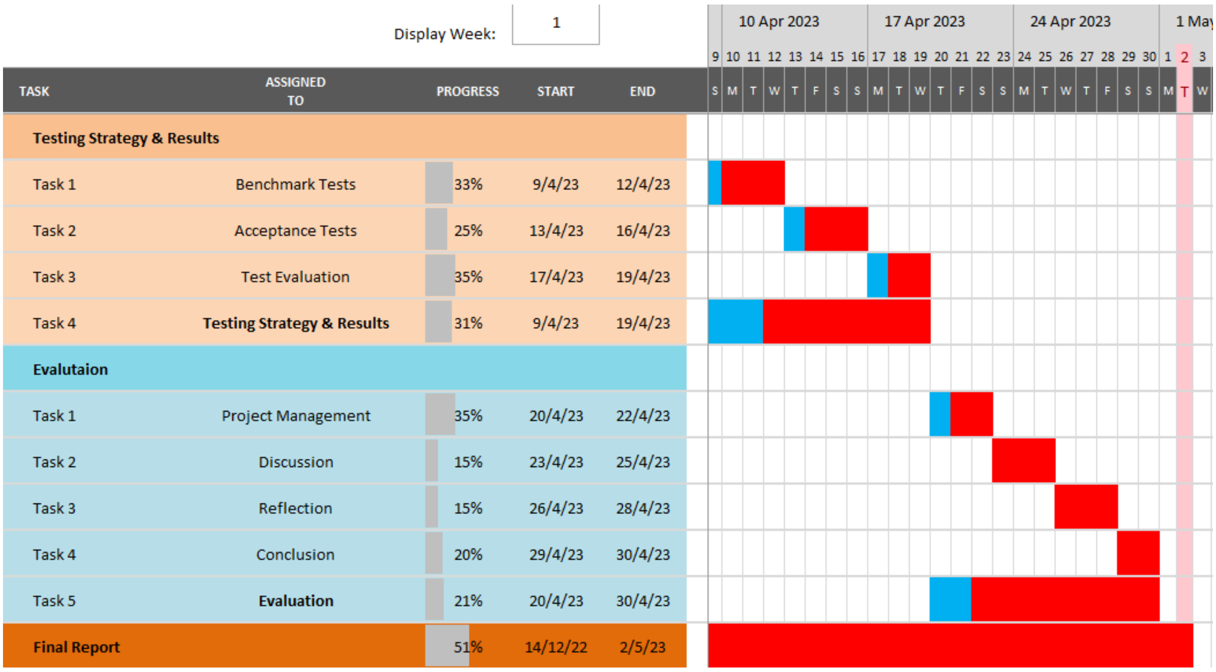

Figure 7.3: *Gantt Chart showing the final testing and evaluation*

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 4.5

This project presents interesting ideas surrounding how games, and other proprietary software, can be distributed without the need for a middle-man. This follows modern web ideas of taking away reliance on large data-centres and having a system built and maintained by the people who use it. Moreover, this type of project is usually open source, which can help attract community contributions to improve the security and user experience of the application.

8.2 Reflection

Risk Assessment

Below I'll detail some the risks mentioned in Section 7.1 that came up during development and how my mitigation strategy helped me cope with the issues as they came up.

Difficulty with blockchain development As expected there were several difficulties with blockchain development:

- A decent amount of the documentation is minimal or outdated. As blockchain development is a smaller field there are little in the ways of forum postings.
- Some of the libraries used had bugs in them that hindered development. For example, I could deploy the smart contract to the Sepolia testnet from Remix easily but not through Geth.

Despite the problems mentioned above, I was able to complete the blockchain aspect of the project successfully. Effective use of sprint planning meant that I had considered this to be a potential issue and had allocated time to being able to deal with it.

The application is not finished Section 7.2 shows that not all of the requirements were finished and Section 4.6 discusses the limitations with my current implementation. It was expected that this project would not be entirely finished but a lot of the major requirements were met.

The application ended up being incredibly large and took a lot more time to implement and test than expected. Table 8.1 shows a breakdown of the source code. Agile development could only help me so much so by having a hard cut-off point for the implementation I was still left with plenty of time to complete the report.

Language	**Files**	**Comments**	**Code**
Go	45	829	5006
Go Tests	26	665	2836
Vue.js Components	17	141	2414
JavaScript	19	88	377
Solidity	1	33	46

Table 8.1: *The lines of code written for this project calculated using CLOC [50]*

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 6.5) 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.

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.

What Went Well

What Could Have Gone Better

Complexity Overall, I felt that this project felt too large and I had to commit an extremely large amount of time to complete the application and its tests. This resulted in a massive codebase.

Large Scale Testing Like anticipated in Section 7.1, this application was hard to test at a large enough scale to emulate real world usage.

Chapter 9

Conclusion

9.1 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 many modern ideas and techniques to develop a functional proof-of-concept application. The heavy use of automated testing allowed me to continuously write robust and correct code.

As most of the requirements set out in Section 4.1.2, I can say that this project was successful in providing a proof-of-concept application t

9.2 Future Work

Content Discovery

Creating a fully functioning store page, where users can search for and discover new games, would be a vital next step for this application. Some of the techniques used to achieve this could be:

- **Indexing** Periodically generate an index of all games uploaded that allows users to easily search the store without having to make large amounts of requests to the blockchain. Use of ranking algorithms could allow for users to be shown the most relevant and useful results.
- **More Metadata** Adding more metadata to games would allow for them to be more easily searchable and indexable. For example, each game might be given a set of tags that can be searched for.

This would help flesh out our store page to make the user experience even better as user's will be able to search for games through this application.

Optimisations

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

- **New Commands** By extending the commands from Section 4.3 we could reduce the amount of network overhead required to communicate between peers.
  This could include batch block requests, or only forming connections with peers who we share some games with.
- **Block Selection** Currently blocks are not ranked in any way but by considering ideas from Section 2.1 we could improve the efficiency, availability and throughput of our network.

Chapter 10

References

Peer-to-Peer File Sharing

[3] 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.

[4] 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.

[5] Liang Wang and Jussi Kangasharju. "Measuring large-scale distributed systems: case of BitTorrent Mainline DHT". In: *IEEE P2P 2013 Proceedings*. IEEE P2P 2013 Proceedings. ISSN: 2161-3567. Sept. 2013, pp. 1–10. DOI: 10.1109/P2P.2013.6688697.

[6] 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.

[20] 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).

[21] 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.

[22] 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).

[23] 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).

[24] 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).

Blockchain

[7] Dejan Vujičić, Dijana Jagodić, and Siniša Ranđić. “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.

[8] 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).

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

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

[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).

Tools Used

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

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

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

[27] *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).

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

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

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

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

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

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

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

[35] etherscan.io. *Deployed Smart Contract*. Ethereum (ETH) Blockchain Explorer. URL: http://sepolia.etherscan.io/address/0x2899dab55a4a20d698062bbf4d4ce9f1073ce052 (visited on 03/30/2023).

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

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

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

[39] *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).

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

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

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

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

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

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

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

[48] *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).

[50] *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

[47] Kent Beck. *Test-driven Development: By Example*. Google-Books-ID: CUlsAQAAQBAJ. Addison-Wesley Professional, 2003. 241 pp. ISBN: 978-0-321-14653-3.

[49] S. Ilieva, P. Ivanov, and E. Stefanova. “Analyses of an agile methodology implementation”. In: *Proceedings. 30th Euromicro Conference, 2004*. Proceedings. 30th Euromicro Conference, 2004. ISSN: 1089-6503. Sept. 2004, pp. 326–333. DOI: 10.1109/EURMIC.2004.1333387.

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).

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

Appendix A

Screenshots

Figure A.1: *The login page where a user will enter their Ethereum private key and connect to a BlockWare contract instance.*

Figure A.2: *The home page where users can navigate between the main individual pages.*

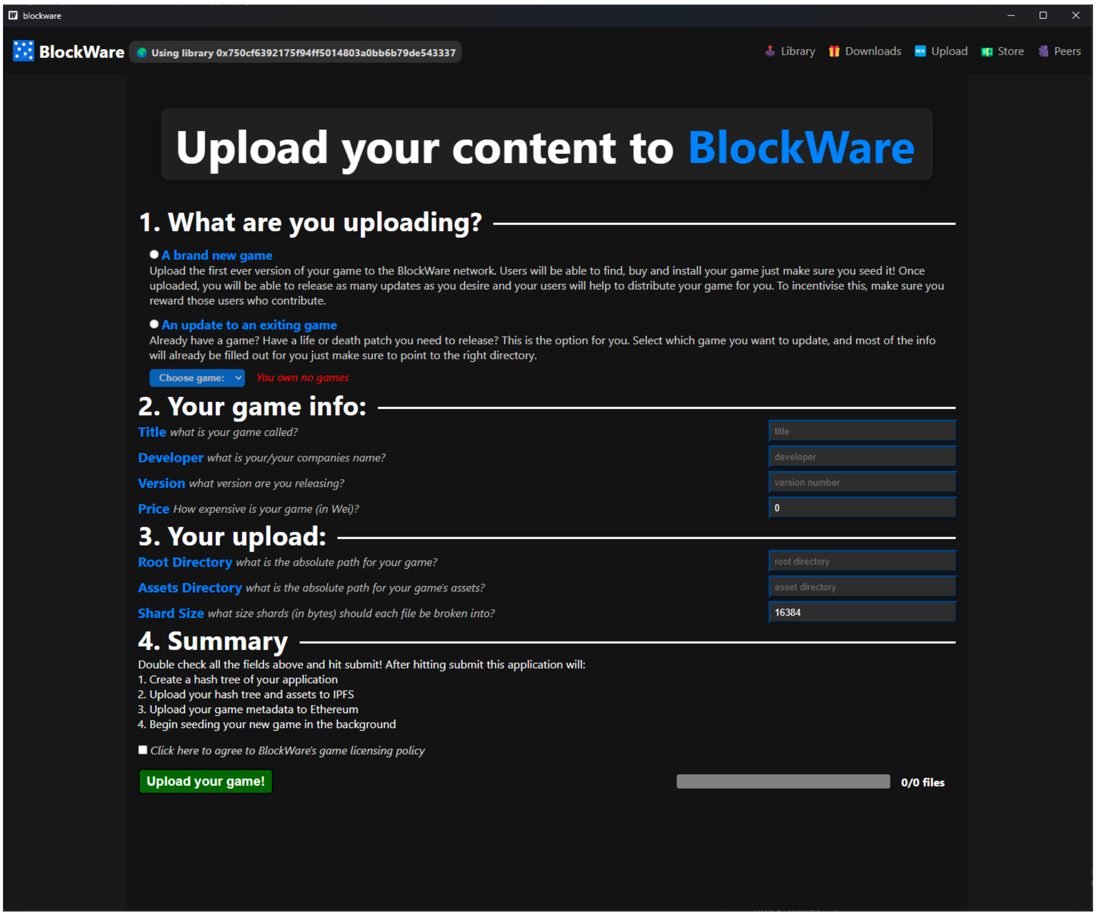

Figure A.3: *The page where users input the details about their game and can upload it to the Ethereum network.*

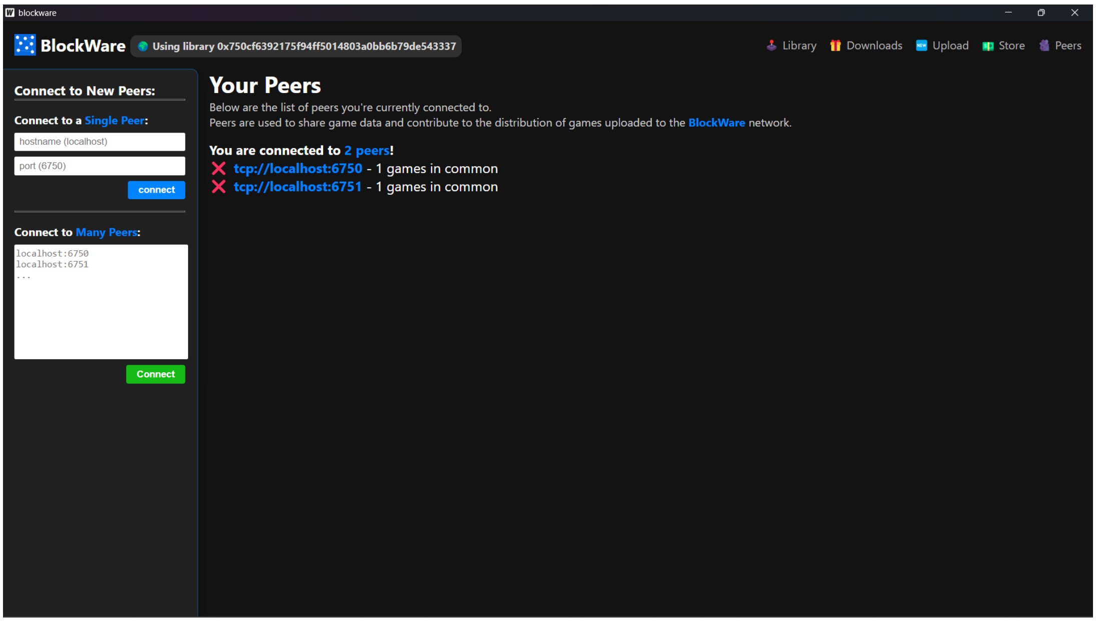

Figure A.4: *The page where users can manage their connections to peers with whom they will download game data off of.*

Appendix B

Code Snippets

B.1 Example Tests

Unit Test

The following shows the example test structure for the tests about a function that handles an incoming SEND_BLOCK message received from another peer. For each test we include a long comment above it detailing each test case included within the function below such that the structure of the test cases matches the structure of the nested tests.

```
/*
function: handleSEND_BLOCK
purpose: receive a block of data from a peer and insert it into storage

Test cases:
success
  #1 => single message received
  #2 => all messages to download a file received

failure
  illegal arguments
    #1 => invalid game hash
    #2 => invalid shard hash
    invalid data
      #1 => wrong length
      #1 => wrong content

  unexpected data
    #1 => download for game not started
    #2 => block not needed for download
*/
func TestHandleSEND_BLOCK(t *testing.T) {
  // all tests setup
  ...
  // end all tests setup
  t.Run("success", func(t *testing.T) {
    t.Run("single message received", func(t *testing.T) {
      ...
    })

    t.Run("all messages to download a file received", func(t *testing.T) {
      ...
```

```
    })
  })

  // failure tests
  ...
  // end failure tests

  // all tests teardown
}
```

Listing B.1: *An example test case used for the handleSEND_BLOCK function*

User Walkthrough 2

Here we show an example of how user walkthrough 2 from Section 6.4 is carried out. In this test, one peer P_1 will download a piece of data G_1 off of another peer P_2. Both peers will interact with the Sepolia test-net where appropriate and will communicate wirelessly in the same local network.

1. P_1 and P_2 own game G_1.
   This is a given and the steps to reach this aren't necessary for this test. For this run, P_2 will upload the game and P_1 will purchase it
2. P_1 connects to P_2
   P_1 heads to the Peers page and enters in the details for P_2 and hits connect. P_1 and P_2 form a TCP connection.
3. P_1 and P_2 exchange Ethereum addresses.
   The logs show us that both P_1 and P_2 send and receive a VALIDATE_REQ and VALIDATE_RES command and that an address was successfully retrieved.
4. P_1 starts a download for G_1.
   P_1 hits the download button in the game's library entry.
5. P_1 sends requests for blocks to P_2.
   From the logs we can see P_2 receives requests for data with the BLOCK command.
6. P_2 queries the smart contract to verify that P_1 owns G_1.
   The logs that P_2 makes a query to the smart contract and finds that P_1 owns the game.
7. P_2 will respond to P_1 with the requested data.
   The logs show P_1 receiving SEND_BLOCK responses from P_2 with their requested data.
8. P_1 will verify each block of data received using its hash.
   The logs show that after receiving a block its contents are checked against its hash.
9. P_1 will have a full download of G_1
   Using the diff command, we can see that G_1 has been downloaded by directly comparing the original and downloaded file.