

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

Thomas Smith, tcs1g20
March 8, 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

Abstract	i
Statement of Originality	i
Acknowledgements	iii
1 Problem Statement	1
1.1 The Problem	1
1.2 Goals	1
1.3 Scope	1
2 Background Research	3
2.1 BitTorrent	3
2.2 Ethereum	4
3 Literature Review	5
3.1 Blockchain-Based Cloud Storage	5
3.2 P2P File Sharing	6
4 Design	7
4.1 Stakeholders & Requirements	7
4.2 Design Considerations	9
4.2.1 Blockchain	9
4.2.2 Distributed File Sharing	11
4.3 Limitations	12
5 Project Management	13
5.1 Risk Assessment	13
5.2 Work to Date	13
5.3 Plan of Future Work	13
6 Implementation	16
6.1 Components	16
6.1.1 Backend	16
6.1.2 Persistence	18
6.1.3 User Interface	18
7 Testing	20
7.1 Overview	20
7.2 Unit Testing	20
7.3 Integration Testing	21
7.4 Validation Testing	21

7.5	Benchmarking	21
7.6	Evaluation	22
References		23

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 [9, 2]
- (b) being vulnerable to censorship from governments,
The Chinese version of Steam is heavily censored [15]
- (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 be broken down into two distinct components:

1. **On-Chain** This component will consist of a set of Solidity Smart Contracts written for the Ethereum blockchain that will allow users to view metadata about and purchase games. It will be tested using a local test-net like Ganache using TypeScript. It will later be deployed to the Ethereum test-net to showcase the application in a live network.
2. **Off-Chain** This component will be what users will actually run. Each user will join a peer-to-peer network in which they can upload and download games off of other users. This will interface with the blockchain to allow users access to game metadata. See Section ?? for details about how this will be tested.

For both of these, a series of acceptance tests, that directly correlate to individual requirements, will be run and include a series of integration tests to show that my ap-

plication can meet the requirements and goals I set out. A more detailed description is given in Section ??.

Chapter 2

Background Research

2.1 BitTorrent

It is 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 [5, 12] 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 [16].

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 blocks¹, 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’* [5] and that *‘the majority of users disconnect from the network within a few hours after the download has finished’* [12]. This paper [11] looks at how the use of multiple trackers for the same content and DHTs can be used to boost availability.

¹nodes 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 [18]	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 [7]	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 [6]	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 [3]	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 [8]	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 [14]

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 [1]	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 [12]	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 [10, 4]	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 [13]	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 [13]	Gnutella nodes form an overlay network by sending <i>ping-pong</i> messages. When a node sends a <i>ping</i> message to their peers, each of them replies with a <i>pong</i> message and the <i>ping</i> 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 download 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
<i>Must</i>	
F-M1	Store game metadata on the Ethereum blockchain
F-M2	A node must download 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
<i>Should</i>	
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 Downloadable Content (DLC) for games
<i>Could</i>	
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
<i>Must</i>	
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
<i>Should</i>	
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
<i>Could</i>	
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

Data

The first consideration is what kind of data we are going to be storing and where is it going to be stored.

Data	Size	Location	Explanation
Game Metadata	100 – 200B	Ethereum	<p>This data represents information about the game that help identify it, such as its title, developer, version, root hash, etc. This information should allow for the unique identification of every game uploaded, whilst remaining minimal.</p> <p>Due to the minimal amount of storage required and the fact that every user should be able to discover every game, this data is best stored on the blockchain.</p>
Game Hash Data	S_g/B ¹	IPFS	<p>This data will be the hash tree of a given game and will be used by a player to verify the contents of each block of data they download as well as the expected structure of the applications contents.</p> <p>Due to this data's moderate size and the fact that not every user will need to view every game's hash tree, there is no need to upload this to the blockchain as this will add an unnecessary expense in publishing games. IPFS is ideal for this as we do not need to restrict access to the data but still need to easily share it and we can simply include the IPFS ID within our game metadata stored on the blockchain.</p>
Game Data	<i>avg.</i> 60GB	Peers	<p>This will be the actual data for the game that is used to run it. To play a game, a user will need access to all of its data. More information about this is in Section 4.2.2.</p> <p>An important property for uploaded games is that all users do not have access to all games without having payed for them first so we shouldn't store them on public platforms like the blockchain or IPFS. Section 4.2.2 will discuss how this can be achieved using a distributed peer-to-peer file sharing model.</p>

4.2.1 Blockchain

Type of Blockchain

To satisfy (NF-M1) and (NF-M2), we will need to use a public blockchain, which will benefit our project by:

¹where S_g is the size of the game and B is the shard size

- 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**).

Data to Store

To satisfy (**F-M1**), the data stored on the blockchain will be used for the identification games and will consist of the following fields, where *italic* fields will be automatically-generated for the user:

Name	Description
<i>For each game</i>	
<i>title</i>	The name of the game.
<i>version</i>	A version number of the game.
<i>release date</i>	When the game was released.
<i>developer</i>	The name of the developer releasing the game.
<i>previousVersion</i>	The root hash of the most previous version of the game if it exists.
<i>price</i>	The price of the game in Wei
<i>uploader</i>	The Ethereum address of the developer.
<i>root hash</i>	The root hash of the game that uniquely identifies it and is based upon its contents.
<i>IPFS address</i>	The ID of the hash tree of IPFS that can be used to download the hash tree before starting a download of a game.
<i>Other</i>	
<i>library</i>	A mapping for storing all games uploaded to the network, where a game's root hash is the key used to find its information.
<i>gameHashes</i>	Solidity doesn't allow us to enumerate maps so we will also store a list of hashes for all games uploaded.
<i>purchased</i>	A mapping which allows us to easily check if a user has purchased a game.

Table 4.4: All the data to be stored on the Ethereum blockchain

Purchasing Content

Users will purchase content from developers over Ethereum using Ether (**F-M9**) and this will be recorded on the blockchain (**F-M10**). Any user can see which other users have purchased the game users can prove this between each other using their public/private keys.

4.2.2 Distributed File Sharing

Hash Tree

The hash tree of a given directory is a tree object that stores information about its structure and contents. This is used to tell users what information they need to download, where it goes and what its contents should be. For every file, the hash tree stores a series of SHA-256 hashes.

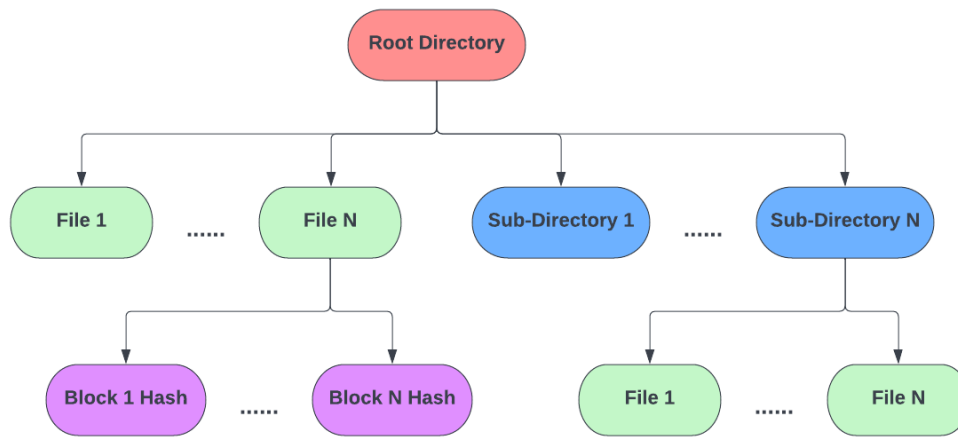


Figure 4.1: The structure of a hash tree

Uploading Content

For a developer to upload their game (**F-M5**) they must provide the required metadata outlined in Section 4.2.1 as well as the location of the game in storage. A hash tree is then produced of the game and this is uploaded to IPFS, and the metadata about the game is uploaded to Ethereum.

Downloading Content

Like mentioned in Section 4.2, it is impractical to store the game's data on the blockchain or even IPFS. Instead we will consider ideas from decentralised file-sharing networks, like BitTorrent, to facilitate the distribution of content.

Games are content addressable and are identified by their root hashes, which are stored on the blockchain and are calculated from the hash tree. Users will send messages using this hash to identify other users who are also interested in the same content (**F-M3**), so they can share data. When two nodes connect to share content the node seeking content will:

1. send their ethereum address along with an encrypted message to prove their address. The address is then looked up on the purchase list stored in the game's entry on the blockchain (**F-S2**).
2. Request individual shards from the node using the shard's hash (**F-M2**).
3. Use the hash tree, which has been fetched from IFPS, to verify the shard's contents (**F-M7**).

4. Send an encrypted encrypted certificate that the sender can use to prove their contribution.
5. Repeat this for an arbitrary number of shards.

Updating Content

To satisfy (**F-M4**), developers will perform the same steps outlined in Section 4.2.2 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. Additionally, this will include the restriction that only the original uploader can upload an update for their game (**NF-S2**).

Each version is considered as 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 and any changes will be considered as a future extension to this project.

Downloadable Content

UNFINISHED

Proving Contribution

When a user successfully downloads a shard of data, they will reply with a confirmation certificate to prove that they have downloaded the game. Each Ethereum address is related to a public/private key pair so a confirmation certificate will be encrypted by a given node's private key to prove authenticity. A user will prove their contribution (**F-S1**) by sending a collection of certificates to the uploader, who will validate them and reward the user accordingly.

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.

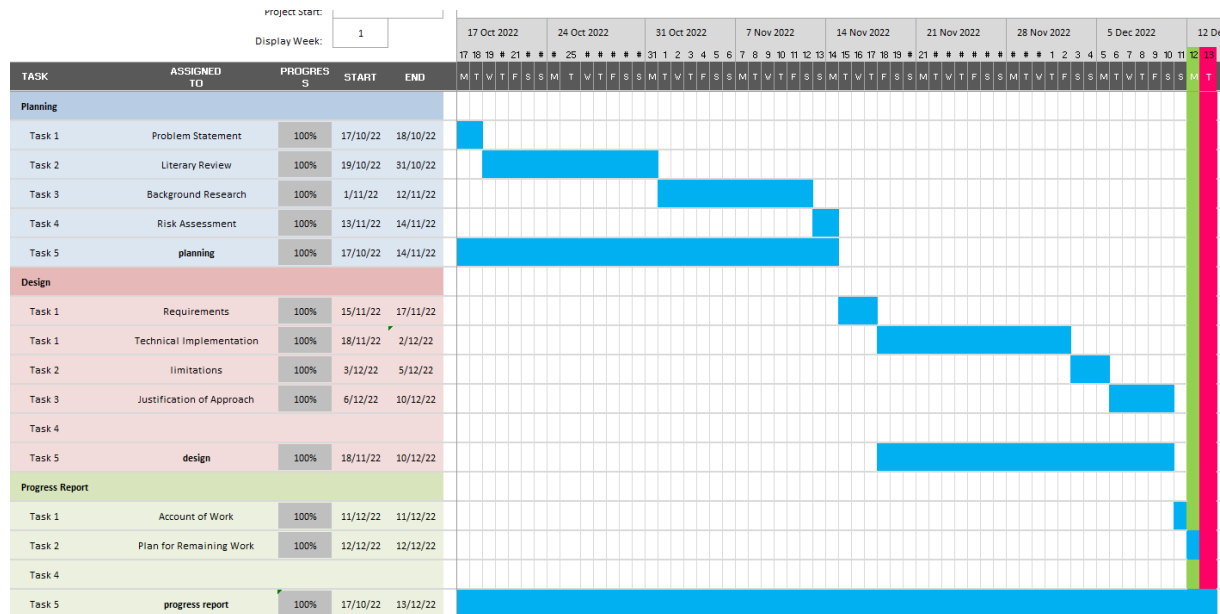


Figure 5.1: A Gantt chart for my work up until the progress report.

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.

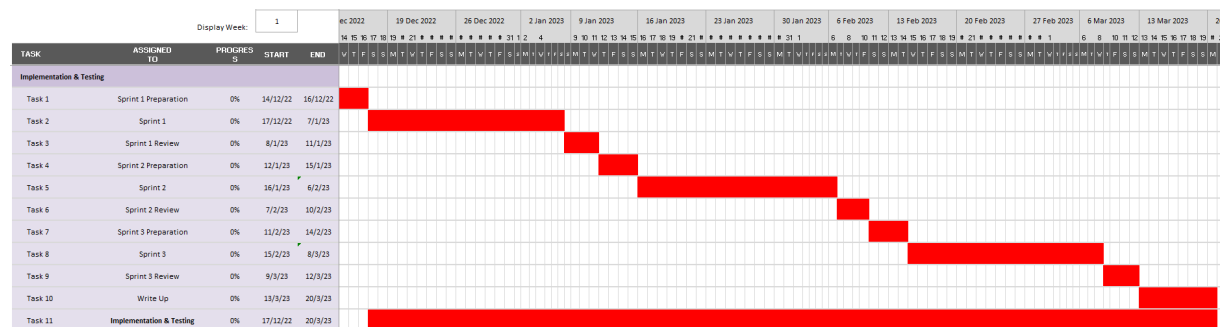


Figure 5.2: The plan for my three sprints

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?

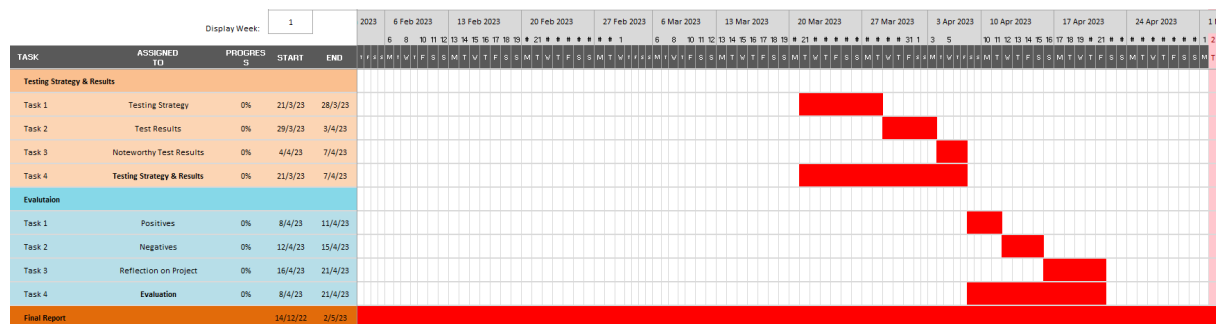


Figure 5.3: The plan for my testing and evaluation phases

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.

Chapter 6

Implementation

6.1 Components

The diagram in Figure 6.1 shows the main components of the application and their relationships. The following sections will discuss these in detail and provide a justification as to their existence.

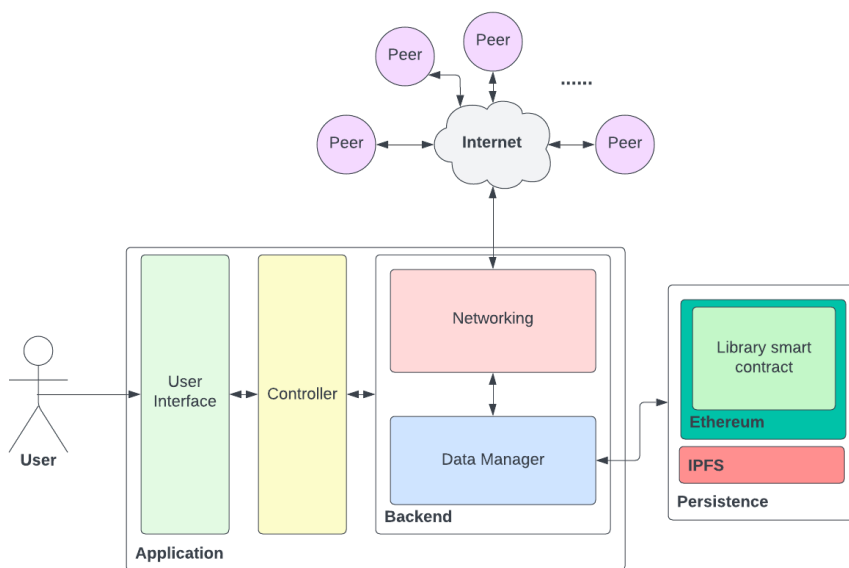


Figure 6.1: The layers of the application

6.1.1 Backend

The backend for this application was written using Google's Go <https://go.dev/> due to its simple syntax, high performance, strong standard library and excellent third parties for interacting with the Ethereum blockchain. This component can be broken down into two major parts:

- **Networking** the creation and maintenance of network connections with other peers over the internet and with the Ethereum blockchain.
- **Data Manager** the management of local data and processing of data received and to be uploaded by the networking part.

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 in the network and will communicate by sending structured messages to each other. Section 6.1.1 describes these commands in detail.

Address Verification To verify the Ethereum address of each peer, and thus which games they own and are allowed to be sent the data of, we will request a signed message using the peer's Ethereum private key (See the `VALIDATE_REQ` and `VALIDATE_RES` commands).

When two peer's initiate a connection they will need to perform a handshake that will allow them identify each other according to their Ethereum addresses.

Message Handling The main responsibility of this section is to respond to requests sent by the Data Manager by sending and tracking messages to other peers to gain the necessary information. 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 will typically come as part of a request/response pair involving the sharing of information between peers.

Message Format	Description
LIBRARY	Request that a peer sends their library of games in the form of a BLOCK message.
GAMES; <i>[hash₁]</i> ; <i>[hash₂]</i> ;...	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; <i>[gameHash]</i> ; <i>[blockHash]</i> ;	The user will request a block of data off of a user by sending the root hash of the game and the hash of the block being requested. The response will be a SEND_BLOCK message.
SEND_BLOCK; <i>[gameHash]</i> ; <i>[blockHash]</i> ; <i>[data]</i> ;	The user sends a block of data in response to a BLOCK message.
VALIDATE_REQ; <i>[message]</i>	The user is requesting for this message to be signed using the receiver's Ethereum private key. This is used to verify the receiver's identity and owned collection of games.
VALIDATE_RES; <i>[signedmessage]</i>	The user responds to a VALIDATE_REQ message with a signed version of the given message. From this signature, the receiver can determine the address and public key of the user.
ERROR; <i>[message]</i>	An error message that can be used to prompt a peer to resend a message.

Data Manager

The data manager has several responsibilities:

1. Track the user's owned games and know which are installed and which aren't.
2. Interact with the Library contract deployed to the Ethereum blockchain to discover, upload and purchase games.
3. Fetch shards of game data to send to other peers.
4. Interact with IPFS to upload/fetch a game hash tree and store assets.
5. Sending requests to the networking layer to find and retrieve blocks of data from peers in the network.

6.1.2 Persistence

The persistence layer is about storing public data that can be accessed by any user of the application at any point in time.

Ethereum

The smart contract will be written in Solidity <https://docs.soliditylang.org/en/v0.8.19/>, and will have to be able to store the data mentioned in Section 4.2.1 as well as perform the following functions:

- **uploadGame** Upload a new game to the blockchain provided it doesn't already exist. Once uploaded, emit an event to let all peers aware of the new game.
- **purchaseGame** A peer purchases a game they don't already own and transfer ETH to the original uploader and are added to the list of authorised users.

The go-ethereum package <https://geth.ethereum.org/> will allow us to interact with the ethereum blockchain and Abigen <https://docs.avax.network/specs/abigen> will allow us to compile any smart contracts to Go code. This will allow us to interact with our smart contract on ethereum using a set of Go functions.

IPFS

Storing data on the Ethereum platform is costly to the uploader and adds a greater storage burden on other nodes in the Ethereum network. As such, we only want to store the minimal amount of data required for the identification of a given game on Ethereum and can leverage IPFS for the storage and distribution of larger pieces of public data.

This includes:

- **Hash Tree** The tree representation of the game's data contents. Only users who purchase the game will need this.
- **Assets** Digital assets such as artwork, description, promotional work, etc.

6.1.3 User Interface

Frontend

This application will have a GUI where users can interact with the platform and will need to include the following pages:

- **Library** The user's collection of owned games, where they can view details of each game owned as well as manage their download status.

- **Store** Where user's can find new games that have been uploaded by other users and purchase them.
- **Upload** Where user's can fill in details about a new game and have it be processed and uploaded to the blockchain.
- **Downloads** Where user's can track all of their ongoing downloads.
- **Peers** Where users can manage their list of connected peers.

Wails <https://wails.io/> was chosen to create the GUI because it allows us to use a reactive web-based framework. This meant building the UI was straight-forward and allowed me to use tools that I was familiar with; namely: Vue.js v3 <https://vuejs.org/>, <https://pinia.vuejs.org/>, and SCSS <https://sass-lang.com/>.

Controller

The Wails package allows us to write controller functions in Go that are callable from our frontend code. These functions can be used to fetch data and trigger events in the backend code. This package also allows us to emit events that can be watched by the frontend; this will allow us to easily create a reactive user interface.

Chapter 7

Testing

7.1 Overview

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 group contextually.

Tools

Below are the different tools I used to test my application and a justification as to why they were included.

Tool/Package	Justification
Go testing	The testing package is part of Go's standard library and will be sufficient to produce most test cases and also includes support for benchmarking and fuzzing tests.
testify by stretchr	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 7.1: The tools used for testing my project

7.2 Unit Testing

Unit tests will be written to test individual functions within the application and how they work on their own. This includes cases with:

- well formed inputs, where the function should produce the correct result, and
- illegal inputs, which should result in errors being raised.

7.3 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 messages and may reply with the wrong contents at random.

7.4 Validation Testing

To ensure that this application meets the requirements as described in Section 4.1, each requirement will be given a specific automated test that aims to prove the completeness of the requirement.

7.5 Benchmarking

Strategy

Benchmarking tests will be used to demonstrate how the scalability and performance of my application in a live environment, by tracking how it performs in a series of increasingly expensive tasks, ran using a consistent set of hardware outlined in the test specification.

Tests & Results

Generating Hash Trees

The generation of hash trees as described in Section 4.2.2, can be an expensive computational task depending on two factors:

- the number of files (or leaf nodes) present in the hash tree,
- the average size of files included.

Whilst varying the number of worker threads performing the algorithm affects the time it takes to complete, it will not be investigated as the same computational effort is still required regardless of the number of threads performing it.

Varying File Count For each run, we will create a project with N files of a constant size $S = TBD$ and run the hash tree algorithm using a pool of 5 workers.

Run No.	File Count	Time (seconds)	Hash Tree Size (KB)
1			

Table 7.2: *How varying file count affects hash tree generation time and size*

Varying Average File Size For each run, we will create a project with $N = TBD$ files all of size S and run the hash tree algorithm using a pool of 5 workers.

Run No.	avg. File Size (MB)	Time (seconds)	Hash Tree Size (KB)
1			

Table 7.3: *How varying the average file size affects hash tree generation time and size*

7.6 Evaluation

Bibliography

- [1] BENET, J. IPFS - content addressed, versioned, p2p file system.
- [2] BROWN, A. Valve defends taking 30 per cent cut of steam sales in response to lawsuit.
- [3] CHEN, Y., DING, S., XU, Z., ZHENG, H., AND YANG, S. Blockchain-based medical records secure storage and medical service framework. 5.
- [4] HOWARD, J. H., KAZAR, M. L., MENEES, S. G., NICHOLS, D. A., SATYANARAYANAN, M., SIDEBOTHAM, R. N., AND WEST, M. J. Scale and performance in a distributed file system. 51–81.
- [5] KAUNE, S., RUMÍN, R. C., TYSON, G., MAUTHE, A., GUERRERO, C., AND STEINMETZ, R. Unraveling BitTorrent’s file unavailability: Measurements and analysis. In *2010 IEEE Tenth International Conference on Peer-to-Peer Computing (P2P)*, pp. 1–9. ISSN: 2161-3567.
- [6] LI, J., WU, J., AND CHEN, L. Block-secure: Blockchain based scheme for secure p2p cloud storage. 219–231.
- [7] LI, J., WU, J., CHEN, L., AND LI, J. Deduplication with blockchain for secure cloud storage. In *Big Data*, Z. Xu, X. Gao, Q. Miao, Y. Zhang, and J. Bu, Eds., Communications in Computer and Information Science, Springer, pp. 558–570.
- [8] MANZOOR, A., LIYANAGE, M., BRAEKE, A., KANHERE, S. S., AND YLIANTTILA, M. Blockchain based proxy re-encryption scheme for secure IoT data sharing. In *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pp. 99–103.
- [9] MARKS, T. Report: Steam’s 30% cut is actually the industry standard.
- [10] MORRIS, J. H., SATYANARAYANAN, M., CONNER, M. H., HOWARD, J. H., ROSENTHAL, D. S., AND SMITH, F. D. Andrew: a distributed personal computing environment. 184–201.
- [11] NEGLIA, G., REINA, G., ZHANG, H., TOWSLEY, D., VENKATARAMANI, A., AND DANAHER, J. Availability in BitTorrent systems. In *IEEE INFOCOM 2007 - 26th IEEE International Conference on Computer Communications*, pp. 2216–2224. ISSN: 0743-166X.
- [12] POWELSE, J., GARBACKI, P., EPEMA, D., AND SIPS, H. The bittorrent p2p file-sharing system: Measurements and analysis. In *Peer-to-Peer Systems IV*, M. Castro and R. van Renesse, Eds., Lecture Notes in Computer Science, Springer, pp. 205–216.

- [13] SAROIU, S., GUMMADI, P. K., AND GRIBBLE, S. D. Measurement study of peer-to-peer file sharing systems. In *Multimedia Computing and Networking 2002*, vol. 4673, SPIE, pp. 156–170.
- [14] SHARMA, P., JINDAL, R., AND BORAH, M. D. Blockchain technology for cloud storage: A systematic literature review. 1–32.
- [15] STEAMDB. Steam china officially launched.
- [16] WANG, L., AND KANGASHARJU, J. Measuring large-scale distributed systems: case of BitTorrent mainline DHT. In *IEEE P2P 2013 Proceedings*, pp. 1–10. ISSN: 2161-3567.
- [17] WANG, S., ZHANG, Y., AND ZHANG, Y. A blockchain-based framework for data sharing with fine-grained access control in decentralized storage systems. 38437–38450. Conference Name: IEEE Access.
- [18] YUE, D., LI, R., ZHANG, Y., TIAN, W., AND PENG, C. Blockchain based data integrity verification in p2p cloud storage. In *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 561–568. ISSN: 1521-9097.