Decentralized social data sharing in the browser

By

Michael Fakhry 100879357

Supervisor: Dr. Babak Esfandiari

A report submitted in partial fulfillment of the requirements
of SYSC-4907 Engineering Project

Department of Systems and Computer Engineering
Faculty of Engineering
Carleton University

April 7, 2017

Abstract

The Network Management and Artificial Intelligence (NMAI) Laboratory at Carleton University (CU) have several data sharing applications which were implemented using Java technology, mainly Tomcat (Java servlet container) and the Java Runtime Environment (JRE). The Java technology, used to deliver these applications, can sometimes be perceived by users as non-user-friendly during install time, mainly because it requires a significant amount of effort and understanding of the Java environment when things do not work as expected.

My task is to build a peer-to-peer (P2P) framework that can deliver such applications directly to the browser without using any third-party software such as plugins or extensions. Thus, reducing the install time required to the time it takes to download the scripts and HTML needed to load an application within a browser window and eliminating the install effort needed entirely. A system model which is known as Distributed Social Data Sharing (DS2), developed at Carleton University, was used as the basis for the framework design and Application Programming Interface (API). We use this framework and API, implemented using JavaScript and browser technologies, to deliver a sample application which runs directly in the browser without any other programs, plugins, or extensions. The main obstacles faced during this project were directly a result of the technology limitations and the partially incomplete browser implementations of those technologies that are currently available for use within browsers.

Acknowledgments

I would like to acknowledge the guidance of my supervisor Dr. Babak Esfandiari, who was instrumental in the successful completion of this project. His honest, continuous, and accurate feedback throughout the project were one of the large contributing factors in making this project a rewarding experience. He continually suggested alternative paths whenever I faced a blocking issue opening the way for new ideas and different solutions.

Table of Contents

List of Figures

List of Tables

Chapter 1

1.0 Introduction

P2Pedia is a distributed wiki, developed by NMAI Laboratory at CU, where each user is in complete control of the articles they share with their peers and store on their local machine. P2Pedia allows several articles on the same topic to exist, avoiding unnecessary time wasting edit wars sometimes observed on Wikipedia. Tomcat and JRE were used to deliver P2Pedia to the user. For a few semesters, students in LAWS2908 used P2Pedia for notetaking during tutorials. During this period, the largest obstacle faced by students was the install effort, and technical knowledge required to install an application served using this Java technology. [1] Furthermore, it could also drive potential users away from a web application simply due to its install requirement in general. Thus, giving rise to the need for a change that allows non-technical users the ability to run such applications with minimal effort. In this case, the change comes in the form of considering the latest browser technology to implement a JavaScript P2P framework capable of delivering such applications.

Therefore, the objective of this project is to implement a P2P framework in the browser that eliminates the need to install an application to participate in a P2P community. This project accomplishes this objective by using existing libraries that make use of implementations of the latest web technologies that permit P2P connectivity and data storage. Another primary goal is to achieve this without using any browser-specific plugins, add-ons, extensions, or external processes. In an ideal world, this would also be done without any server interaction at all, but the state of the current networking technologies and protocols does not permit such server-less communication between browsers. Therefore, it is required to perform several networking tasks with the help of an external server known as a signalling server. Keeping in mind that the server is only being used for connection establishment rather than for relaying data between two users. Thus, once a connection is established, there is no longer a need for the signalling server.

This report will first discuss the organizational details of this project in chapter 2. Chapter 3 will present the background material on the network model and the requirements for this project while Chapter 4 will discuss the currently available browser technologies. Chapter 5 is dedicated to presenting the design, and implementation of the P2P framework. Chapter 6 presents an example application built using the P2P framework. Chapter 7 will briefly look at open source contributions that occurred as result of work on this project. Finally, Chapter 8 will conclude the report.

Chapter 2

2.0 The Engineering Project

2.1 Health and Safety

The health and safety guide posted on the SYSC4907 course website is not applicable to this project since no university laboratories were involved in the completion of this work.

2.2 Engineering Professionalism

Regular meetings and discussions were used to define the requirements of this project and track the progression of tasks with continuous feedback from the supervisor. These meetings would resemble the regular meetings of an agile team working in a tech company responsible for delivering a product or several features for a customer. The guidelines and instructions for the course deliverables, posted on the course website, were all followed while all deadlines were met on time without any delay. These guidelines would represent the company protocols and procedures while the deliverables would represent project details and documentation to the customer.

2.3 Project Management

The Agile software development principles were used to develop this product. There were several iterations incrementally improving the P2P framework, adding more features as time progresses to arrive at the current release. There were also several face-to-face meetings throughout the project to track progress.

Yodiz, an agile project management tool was used briefly to organize one-week sprints. [2] Yodiz was later abandoned due to its superfluous nature for a small project with a few team members working on a limited number of tasks over a short period.

2.4 Individual Contributions

This section lists the contributions made towards each stage leading to the final product and the final report.

2.4.1 Project Contributions

The following table summarizes the contributions made towards each milestone of the final product.

Table 1 Individual contributions

Milestone	Michael Fakhry	Himanish Kaushal
Research	Browser P2P connectivity Browser data storage	
MVP (Minimum Viable Product)	P2P connection establishment Text transfer	File transfer
Browser P2P framework	Design Implementation	-
Application using browser P2P framework	MongoIdbDs2 and simple stamp collection	

2.4.2 Report Contributions

The entire report was written by Michael Fakhry.

Chapter 3

3.0 Requirements

The following subsections will present the basis of the peer to peer model based on which the requirements for this project were derived.

3.1 Background: Decentralized Social Data Sharing $(DS)^2$

Decentralized Social Data Sharing is a general system model that describes social connections between users and links between data resources shared by those users. [3]

The following figure demonstrates the main concepts of the $(DS)^2$ system model:

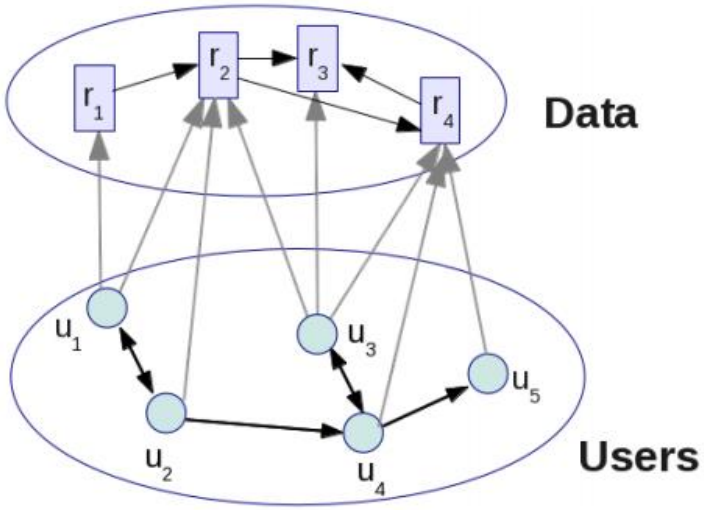

Figure 1 Decentralized Social Data Sharing graph representation [3]

Figure 1 shows the following two graphs:

1. Social graph: The graph of users u_1 to u_5 and their connections to each other.
2. Data graph: The graph of structured data resources r_1 to r_4 and their links to one another.

The figure also shows that multiple users could store a resource regardless of origin, which is typical in a P2P network. Operations in this model are further simplified into two types of operations, data management operations and community operations [3]. Each user can perform any of these operations. The data management operations are described next [3].

- Publish – publish a resource so that users can find it.
- Delete – delete a published resource so that users can no longer find it.
- Copy – copy a resource that is published by another user so that this user also publishes it.
- View – view a published resource.
- Query – query users for a resource that meets the specified search parameters.

The community operations are described next [3].

- Connect - Connect to another user with the given address.
- Disconnect – Disconnect from a user with the given address.

3.2 Background: Schema-based P2P file sharing

Earlier file sharing applications implemented a P2P network model (example: Napster and various Gnutella implementations) that relied on file discovery through filename only and no other attributes. Therefore, finding a file required knowledge of at least a part of the filename beforehand, which is not necessarily possible in all cases. So, to improve file discovery within the P2P network, the proposed idea is to use a file descriptor object that is used to provide additional information about the file. This descriptor object is created whenever a new file is to be shared on the network and would be propagated along with the file whenever the file is sent to another user. This descriptor object would facilitate file discovery through a set of pre-determined fields. [4]

One implementation of this idea is known as Universal Peer to Peer (UP2P), which will be discussed in more detail for the remainder of this section.

A schema in simple terms is the recipe used to write metadata, another term for the XML-based file descriptor object, within the UP2P platform. A user can decide to create a schema at any time and share it on the network. A schema could itself contain many attributes describing the metadata and files being shared including but not limited to schema name, file attributes, description, security, and P2P network protocol. The schema could also contain references to stylesheets describing how the XML metadata is viewed, created for new files and which attributes should be used as search parameters within queries. [4]

Other users can perform a search for schemas within their known peers, another term that is often used to refer to connected users within the network. If they like what they see in a schema, they can join it by downloading it and therefore becoming a part of the group that uses this schema. A group of users that uses a specific schema is termed as a community. A user can belong to several communities at the same time but can only search for files within a single community since each community will have its searches indexed per different attributes that are based on the search information specified in the schema. Any user, whose part of the community, can share a file with the rest of the community by using the schema to create the metadata for that file and make it available so that other peers can find it when searching within the community. If a user likes a search result, the user can retrieve the file for which the search result belongs. The file is downloaded to the local file system for viewing or sharing with other peers. [4]

3.3 Requirements

This section will discuss the various requirements for the developed framework, both functional and non-functional.

3.3.1 Functional requirements

The operations discussed in the peer to peer system model, presented in section 3.1, will be used as the basis for the functional requirements of the framework designed and implemented for this project. Therefore, the framework should be able to perform the following tasks:

- Establish a new connection between two browsers.
- Close an existing connection between two browsers.
- Publish files so that they are queryable and accessible by other peers.
- View published files.
- Delete published files.
- Receive files from connected peers.
- Propagate queries across the network.

In this realization of the DS^2 network model, the shared resource is split into two main parts which was done in a similar fashion to the way it is done in UP2P, described in section 3.2. The two parts are:

1. The contents of the shared information (also known as file contents).
2. The metadata describing these contents.

Each of these parts will be stored separately since the file content is usually binary data while the metadata is usually structured data that requires indexing for querying.

The following use case diagram illustrates the various operations that can be performed by a peer.

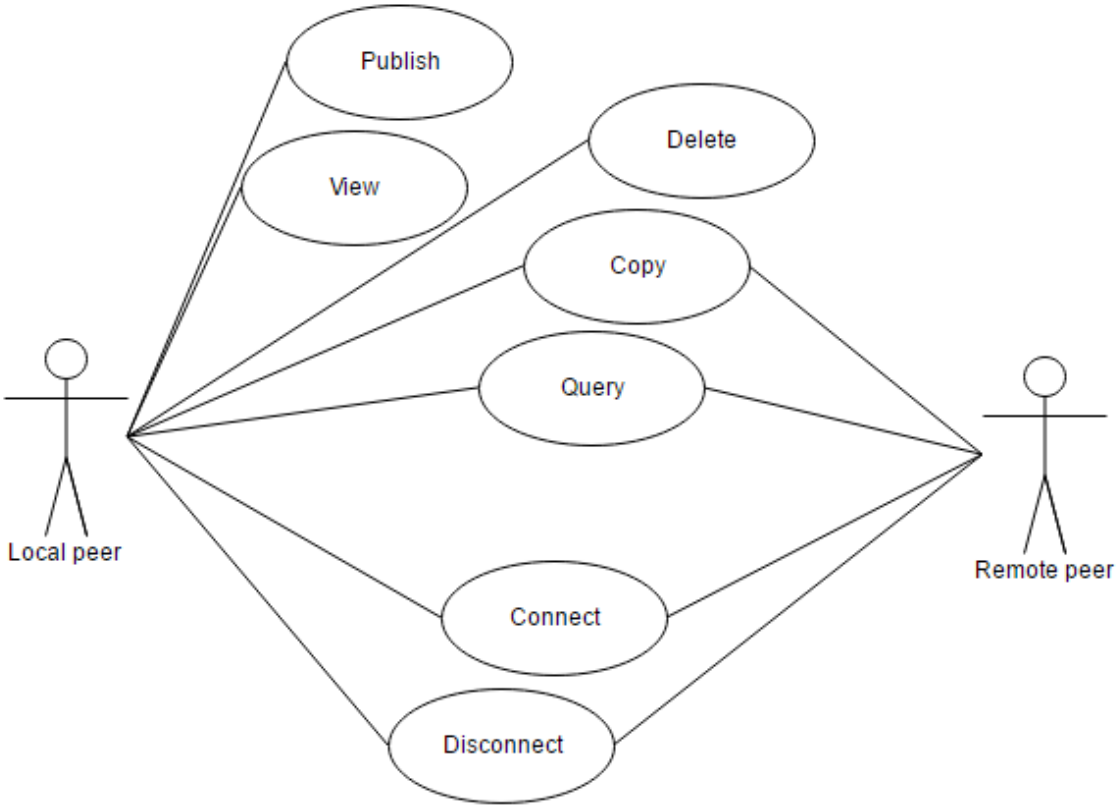

Figure 2 use case diagram

The rest of this section will discuss the various actions seen in figure 2 in more detail.

Use case: Connect
Description: Connect to a remote peer.
Primary actor: Local peer
Secondary actor: Remote peer
Precondition: None
Postcondition: None
Basic flow:

1. Local peer provides the identity of the remote peer.
2. The system starts establishing the connection to the peer.
3. The system returns upon connection establishment.

Use case: Disconnect
Description: disconnect from a remote peer.
Primary actor: Local peer
Secondary actor: Remote peer
Precondition: The remote peer was connected using the Connect use case initiated from either the local peer or the remote peer.
Postcondition: None
Basic flow:

1. Local peer provides the identity of the remote peer.
2. The system starts closing the connection to the peer.
3. The system returns upon closing the connection.

Use case: Publish
Description: Make a file and its metadata available to other peers in the network and store them locally.
Primary actor: Local peer
Secondary actor: None
Precondition: None
Postcondition: None
Basic flow:

1. Local peer provides the file contents and metadata to publish.
2. The system creates a hash of the file contents.
3. The system stores the file contents and metadata using the hash as a key.
4. The system returns the hash of the file contents to the local peer.

Use case: Copy
Description: Copy the contents and metadata belonging to a file.
Primary actor: Local peer
Secondary actor: Remote peer
Preconditions: File contents and metadata were stored using the Publish use case at the remote peer. Remote peer is connected using the Connect use case initiated by the Local peer or the remote peer.
Postcondition: None
Basic flow:

1. Local peer provides the hash of the file they want to copy and the identity of the remote peer that published the file.
2. The system initiates the file download (file contents and metadata) from the remote peer.
3. The system stores the downloaded the file contents and metadata using the hash as a key.
4. The system returns upon completion of the download.

Use case: View
Description: View a locally stored file.
Primary actor: Local peer
Secondary actor: None
Precondition: File contents and metadata were stored using the Publish use case or the Copy use case.
Postcondition: None
Basic flow:

1. Local peer provides the hash of the file they want to view.
2. The system retrieves the file from storage.
3. The system returns the retrieved file to the local peer.

Use case: Delete
Description: Delete a locally stored file and its metadata.
Primary actor: Local peer
Secondary actor: None
Precondition: File contents and metadata were stored using the Publish use case or the Copy use case.
Postcondition: None
Basic flow:

1. Local peer provides the hash of the file they want to delete.
2. The system deletes the file from storage.
3. The system returns upon completion of the delete.

Use case: Query
Description: Query the metadata stored locally and remotely at connected peers.
Primary actor: Local peer
Secondary actor: Remote peer(s)
Precondition: None
Postcondition: None
Basic flow:

1. Local peer provides the query and the number of hops that the query should make away from the local peer.
2. The system simultaneously processes the query locally and sends it to the connected peers.
3. The system collects all the query hits from all nodes.
4. The system returns the query hits to the local peer.

3.3.2 Non-functional requirements

The following are the non-functional requirements identified for this project:

- Modularity through object oriented approach.
- Ease of extensibility through simple, clear, and well-defined APIs.
- Minimize the number of stateful and coupled objects in design to reduce the dependencies between objects and operations.
- Use asynchronous APIs and libraries whenever available/applicable.
- No browser add-ons, extensions or plugins are allowed for this project.
- Use APIs recommended by World Wide Web Consortium (W3C) for storage and peer to peer connectivity.
- Minimal server interaction and reliability on server operations.

Chapter 4

4.0 Web technologies

This chapter is dedicated to discussing the various technologies available within the browser environment for P2P applications. The following three topics will be the focus of this chapter:

1. P2P connectivity
2. File access on the filesystem
3. Data storage (Binary and structured data)

4.1 Browser P2P connectivity using WebRTC

Currently, there is only one available protocol that allows for P2P communication between browser clients, which is known as Web Real-Time Communications (WebRTC). This protocol was developed to exchange audio, video, and data between peers for teleconferencing but it can be used for simple data connections between peers without video or audio feeds. The protocol also provides Network Address Translator (NAT) traversal techniques making it easier to establish a connection with a peer behind a NAT.

The offer/answer model is used to establish a connection, which means that a mechanism is needed to exchange the offer and the answer from one peer to the other even before a direct connection is established. The specification for this web technology does not specify how the offer and answer are exchanged but leaves it up to the individual applications to implement a mechanism for exchanging offers and answers. In most cases, this means that a server will be used to transfer the offer and answer. This server is usually known as a signalling server. This setup is far from the ideal P2P connection between two peers without any server interaction, but unfortunately, that is the current state of P2P connectivity within a sandboxed environment such as a browser behind a NAT. [5]

The WebRTC specification is still a work in progress meaning that the APIs could change at any given time, but this also implies that support for this technology is not necessarily available in all browsers, which is a huge disadvantage to a modern web app looking to use this browser technology. Since WebRTC is the only technology available for browser P2P connectivity without any other options, it is the technology that was used for this project.

4.2 Networking stack

Every P2P framework requires a network layer to establish connections and send data between peers. In this section, several networking layers from different applications are examined to determine the best fit for the JavaScript framework discussed in this report. All the discussed frameworks use WebRTC to establish the connections to other peers.

4.2.1 WebP2P

WebP2P has the simplest network layer which tries to start as many connections as possible using an off the shelf signalling service rather than implementing its signalling servers. It offers a simple approach to networking but does not provide the required amount of control over connectivity that is needed by the DS2 system model. [6]

4.2.2 WebTorrent

WebTorrent uses servers as torrent trackers which are also used to establish connections between peers. So, the server acts as a torrent tracker and a WebRTC signalling server. The architecture of this network layer is not relevant to the use case of the DS2 model mentioned in section 3.1 due to its reliance on torrent trackers which means that it is not a suitable network layer for this framework. [7]

4.2.3 Libp2p

InterPlanetary File System (IPFS) has a dedicated networking stack called Libp2p which was designed and implemented to work in the browser and Node.js. It offers support for several transports (mounted over WebRTC), encryption in transit, ping, multiplexing a connection and offers a dedicated signalling server written in Node.js. A development server is deployed, which can be used by any applications that are currently in development. It provides an application with complete control over outgoing connections and has the possibility of control over incoming connections. It also offers a peer discovery mechanism using the signalling server and the option of bootstrapping a node. [8]

4.2.4 Summary

Out of all the networking stacks considered, libp2p offers a complete networking stack with several desirable features including a deployed signalling server that can be used for development of new applications. Therefore, it will be used as the network layer for this framework.

4.3 Browser file access using File and FileReader objects

The File object can be used to represent a file on the client's storage. A File object can only originate from an HTML input element, which means that it needs user interaction for the creation of a File object. [9] This object allows JavaScript code to read the file using the FileReader object [10].

Both objects fall under the same specification which is known as the File API. This specification is again still a work in progress. The File API provides the only mechanism to read files in the browser, which means that it is the technology that will be used for reading files in the browser. [9, 10]

4.4 Browser file storage

This section will briefly discuss the various file storage technologies available in the browser. The technologies examined will include local storage, WebSQL, IndexedDB and the Filesystem API.

4.4.1 local storage

The local storage API offers a way to store key-value pairs persistently in a client's browser. For simple applications, this could a great way to store data, but it has a major limitation which is the limited amount of data stored. Each implementation of the technology might have different limits, but it is usually in the range from 1 MB to 10 MB while the specification suggests a limit of 5 MB. This technology is one of the currently recommended technologies for use by W3C and therefore should have full support in major browser vendors. [11]

4.4.2 WebSQL

WebSQL is a deprecated relational database technology that uses a variant of SQL for querying the database. Browser support is currently limited and will be decreasing over time since W3C is no longer working on a specification for this technology. Therefore, it is not considered a good choice for building new applications since its support will continue to diminish over time and will not be implemented by any new browsers. [12]

4.4.3 IndexedDB

IndexedDB is the latest database technology in the browser. It uses a transactional model with asynchronous operations. One of the main disadvantages of IndexedDB is the complex API needed to perform simple operations, but several libraries exist that simplify the API by performing the low-level actions required to configure and run those simple operations. [13]

This technology is also one of the technologies currently recommended by W3C.

4.4.4 FileSystem API

The FileSystem API offers a representation of a filesystem, located in a sandboxed part of the client's hard drive so that applications can store and read files in as similar fashion to desktop applications. This greatly simplifies the storage of files, but the technology is not yet implemented in all browsers and may never be until a standard is finalized and accepted. [14]

This technology is currently considered experimental and is not on any standard's track and therefore, should not be used by new applications until the standard is accepted or at least started the process.

4.4.5 Summary

After considering all the alternatives, the best fit for file storage in P2P applications is the FileSystem API, but since there is no guarantee that it will be standardized, it is a technology that is currently too risky to make a commitment towards using. Therefore, leading us to the next best alternative, which is IndexedDB due to its large storage capabilities and recommendation by W3C.

IndexedDB is also the clear choice for storing metadata since it is usually structured data that requires indexing for optimal searching.

Chapter 5

5.0 DS2

This chapter will present the design and implementation details of the browser-based P2P framework built for this project.

5.1 Overview

This section aims to provide a complete overview of the browser-based P2P framework through providing a description, the code location, installation requirement and usage details.

5.1.1 Description

A library that is used to represent a node in a peer to peer network. It is capable of the following functionalities:

- Assign the node a persistent identity.
- Establish connections to other peers using their identity.
- Offer complete control over which peers are connected to the node.
- Store files and their metadata persistently.
- Propagate queries throughout the network.
- Copy files stored by other peers.

5.1.2 Code location

The code is currently located in the GitHub repository:

https://github.com/michaelfakhri/ds2

The library is also available from npm through the name “ds2” and from UNPKG CDN through https://unpkg.com/ds2/, which is a CDN that uses npm registry as it’s storage medium.

5.1.3 Install and Usage

Install and usage instructions can be found in APPENDIX A.1 and A.2.

5.1.4 Browser support

The currently supported browsers are:

- Chrome
- Firefox
- Opera

The browser support is limited due to the lack of implementation of WebRTC in the other major browsers, which is the only limiting factor in extending browser support.

5.1.5 Development environment

Npm [15] and Node.js [16] environments were used to develop and test this library.

5.2 Design

This section will first introduce the main idea behind the design followed by the details of each module.

In this design, storage is split into two main units: the file storage unit and the metadata storage unit. The file storage is implemented by the DS2 library while the metadata storage unit is implemented, based on a pre-defined interface, by the consuming library or application. The rationale behind this decision is to avoid inheriting limitations imposed by the library or the storage mechanisms used for the metadata storage unit. A very good example to the type of limitations is the sample library MongoIdbDs2 discussed in chapter 6, which lacks the ability to search within text blocks.

Th design consists of the following components:

- DS2 Class: This class implements the API which translates the functionality to lower classes.
- RequestHandler Class: This class implements the query tracking and monitoring functionality.
- ConnectionHandler Class: This class implements the P2P networking features used by the DS^2 system. This includes establishing connections, closing connections, forwarding resources to peers and propagating queries around the network following the implemented algorithm.
- DatabaseManager Class: This class is responsible for interacting with storage to store the configuration, file, and metadata. It also provides a mechanism for querying metadata through the MetadataHandler interface described next.

- MetadataHandler Interface: This interface will be used by the DatabaseManager to interact with the metadata storage unit implemented by the consuming application.

The following UML diagram illustrates how the classes and interfaces are connected to each other:

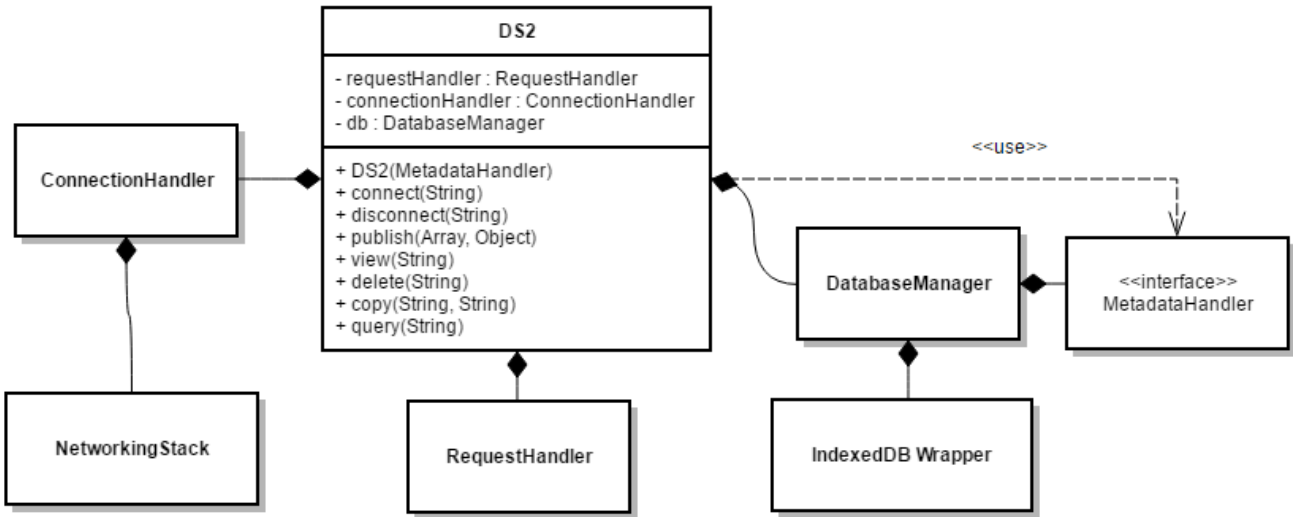

Figure 3 UML diagram of simplified classes and their relationships

Figure 3 shows the four main components of the design along with their dependency on storage and network functionalities provided by external open-source libraries that make use of browser APIs.

The following figure shows how this design fits into an application with respect to the other components of the application.

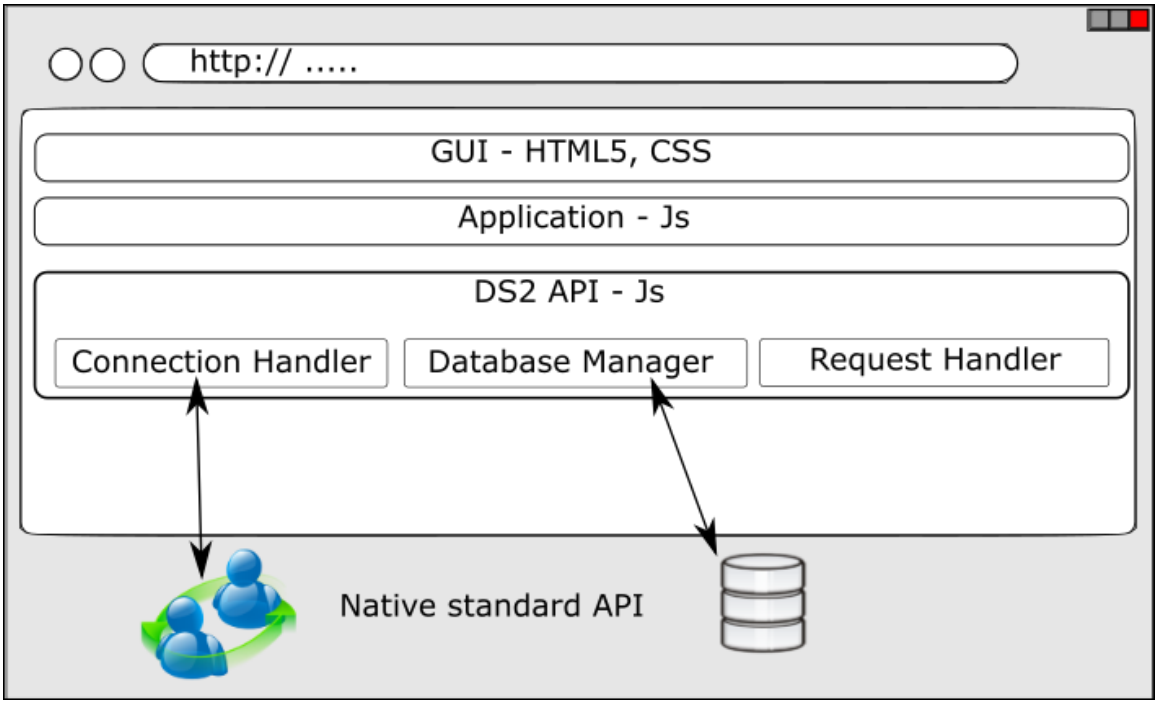

Figure 4 Overview of application using the DS2 library [17, 18]

Figure 4 shows the design's four main classes and how they interact with the browser. The Connection Handler and Database Manager have been simplified to include their internal dependencies which make use of browser APIs.

5.3 Implementation

This section will present and discuss the details of the implementation of the design introduced in section 5.2. More specifically, the following topics will be discussed in this chapter:

1. Peer identity
2. Storage
3. Connections and data transfer (query and file transfer)
4. UML diagram of final implementation
5. Event based design implementation details
6. Challenges
7. accomplishments

5.3.1 Peer Identity

In a typical peer to peer application, each peer has an identity, which is used by other peers for identification and connection establishment. In this implementation, the same identity is used by DS2 and the underlying networking stack, Libp2p. The identity of a peer is the base64 encoded SHA-256 hash of the public key of a RSA key pair used for identification and encryption purposes by the underlying networking stack. This Identity is stored using IndexedDB in the browser and is protected by the same-origin policy implemented by web browsers.

The IP of a user could also have been used as the identity used to establish connections, but due to the dynamic nature of modern IP assignment techniques, it is more reliable to use a pre-determined identity stored in the browser for each application. This allows the application to be used across networks rather than the original implementation limiting a peer's known identity to a single static IP.

5.3.2 File and Metadata storage

As mentioned earlier in section 5.2, file storage is implemented in this library while the consuming application implements metadata storage. This has the advantage that the query structure and capabilities are defined and controlled by the consuming application. This allows the consuming application to define its own query processing capabilities by using a library that fits the application's requirements. Therefore, allowing more flexibility by using different query languages/frameworks and database structures.

5.3.3 Connection establishment

Although the underlying networking stack, libp2p, implements a peer discovery mechanism. A decision was made to avoid using this mechanism to ensure that the implementation stays true to the community-based model of the system derived from the implementation of UP2P. This means that peers wanting to connect to each other are expected to exchange identity information in some other form of communication before being able to establish a connection to one another.

Once a connection is established between two peers, queries, and files can be multiplexed together through a single connection. This eliminates the need for an FTP server to serve files to other peers.

5.3.3.1 Query propagation

Queries in this implementation are propagated by flooding the network through the connected peers in a similar fashion to the Gnutella protocol. The following figure demonstrates query propagation in this implementation.

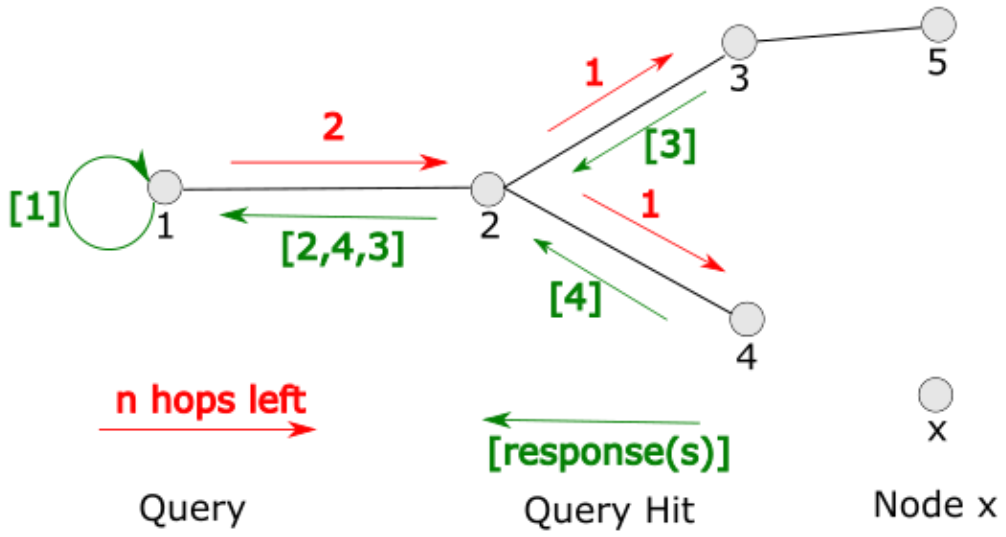

Figure 5 Query propagation

Figure 5 shows all the hops that a query makes from the node that initiated the request. As the query is passed from one peer to the next, the number of hops left is decremented until it reaches 0. At this point, the query is not propagated to further nodes. Each node keeps track of the number of responses it expects after propagating the query to other nodes. Then, the node collects all the responses it receives and sends them back to the node that sent it the query after the last expected response is received.

5.3.3.2 File data propagation

Files can only be downloaded if a direct connection between two nodes exists. The following figure demonstrates the copying of a file from one node to another:

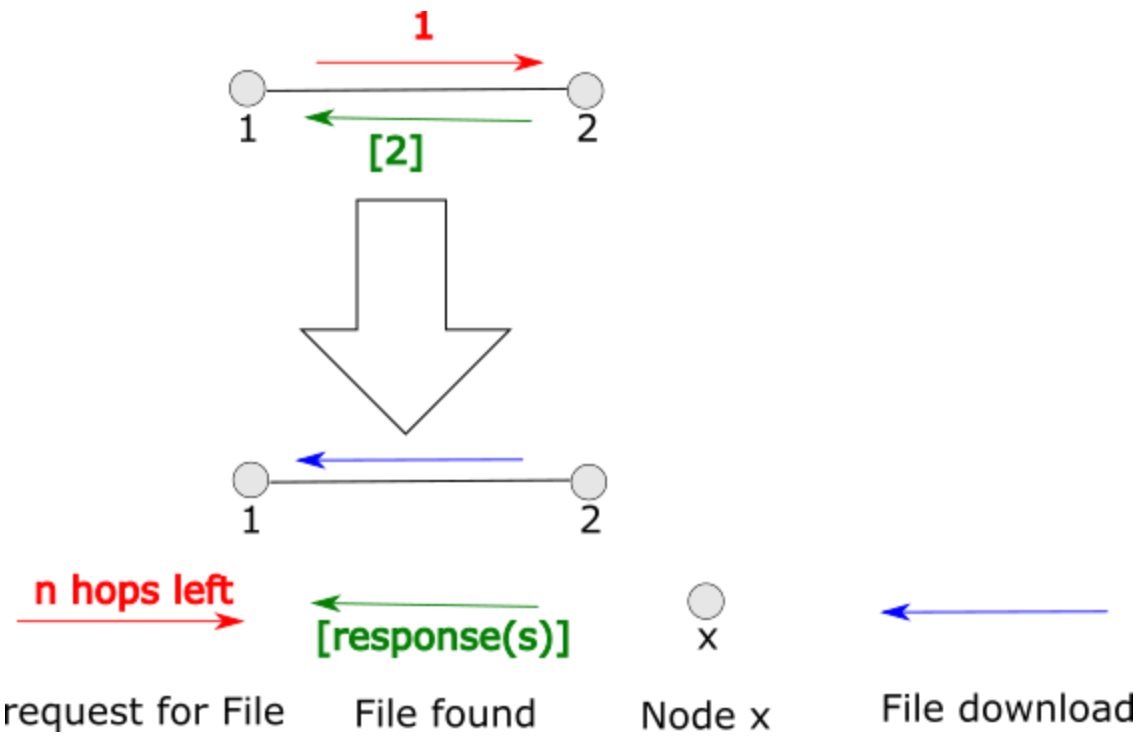

Figure 6 Requesting a file

Figure 6 shows node 1 requesting a file from node 2. Node 2 sends back a response indicating that it can send the file along with the metadata belonging to that file. Once the response is sent, the file transfer starts until the entire file is transferred to node 1.

5.3.4 UML class diagram

This section presents the UML class diagram and its dependencies to the reader. First, a brief overview of the external libraries used for this project will be included to supplement the UML class diagrams to follow.

The following classes are used from external libraries:

- PeerId: The identity of the current peer/node that is used by the networking stack. [19]
- Libp2p: A networking stack that offers P2P connectivity in the browser. [20]
- PullStream: A library that converts data into a stream with different processing stages. This library is used to perform the different tasks, such as hashing followed by storage, on the data as it passes through the different stages of the stream. [21]
- PullBlobStore: A library that offers a PullStream compatible storage medium for binary data. This library is used for file storage. [22]
- Deferred: A Promise implementation that offers more flexibility than native implementations and offers the ability to convert functions implemented using the callback paradigm into a function that returns a promise. [23]

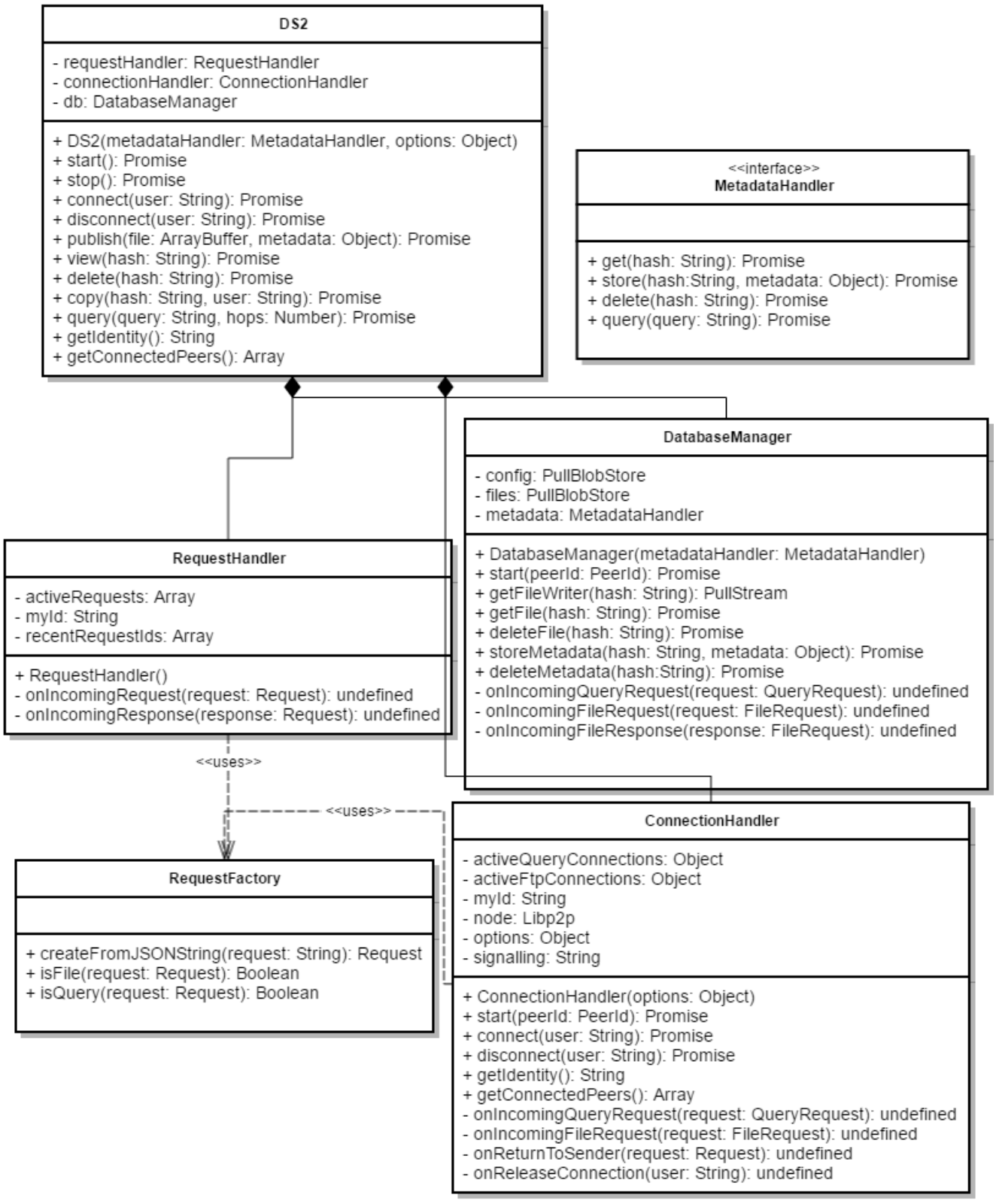

Figure 7 DS2 UML class diagram part 1

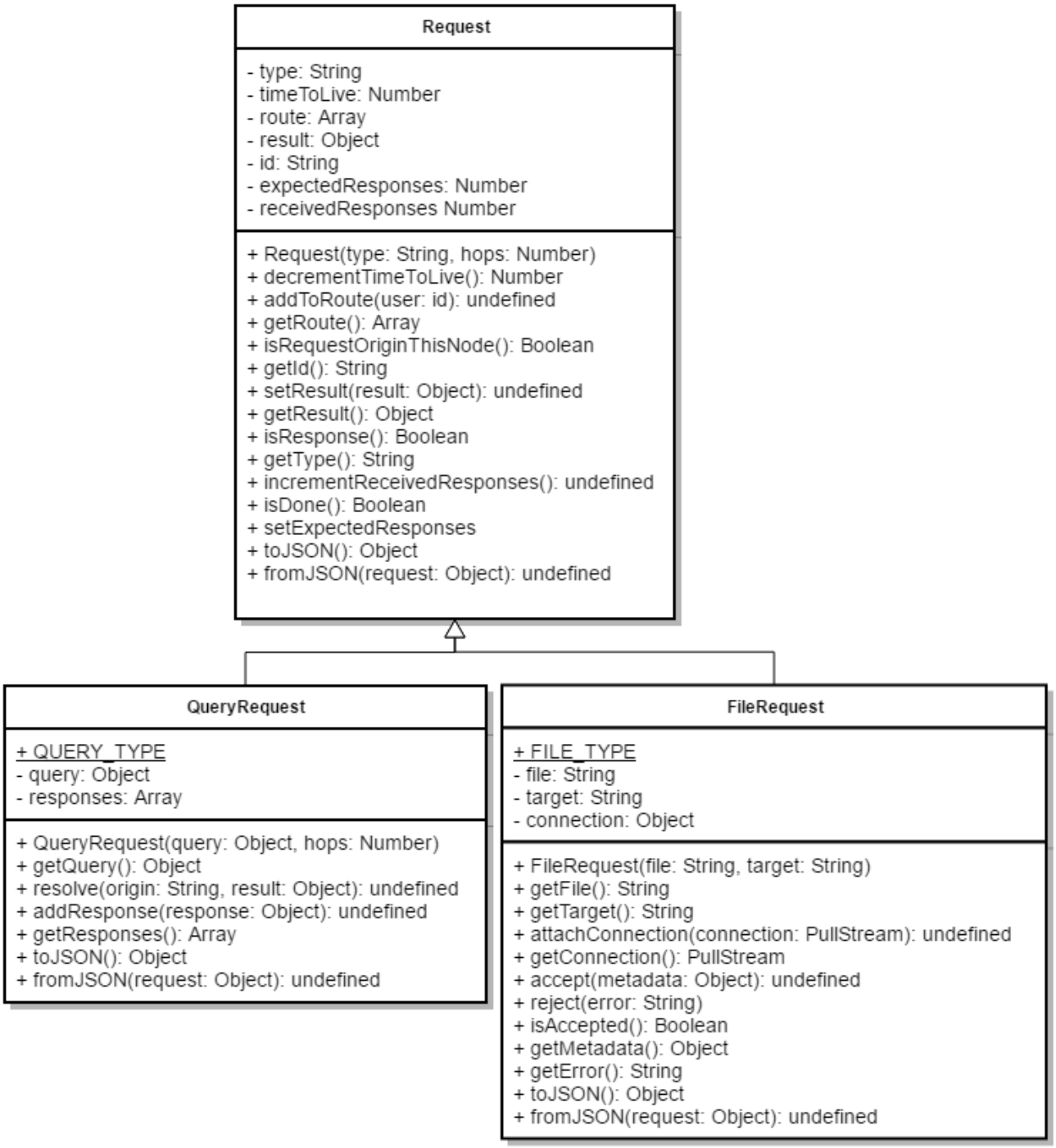

Figure 8 UML class diagram part 2

Figure 7 and 8 present the main classes and their relationships to one another. It is important to note that JavaScript is not easily modelled using the UML class diagrams due to fundamental concepts in the language's design. Therefore, simplifications have been made to make this more comprehensible and reader friendly.

A more detailed description of the API can be found in APPENDIX A.4 and a few examples in APPENDIX A.5.

5.3.5 Event-based querying and file request processing

From figure 8, we can see that a request can either be a query request or a file request. A request can be initiated by any node which in turn triggers a series of events in the network. This means that whenever a request is received, its processing is done through a series of events. Each class can attach a listener to any of these events and perform its actions every time the event is triggered. This section presents the various events triggered for each type of request.

5.3.5.1 Query request

A query request is processed symmetrically regardless of the origin being the API or the network. The following figure shows the events triggered at each node when a query request is received.

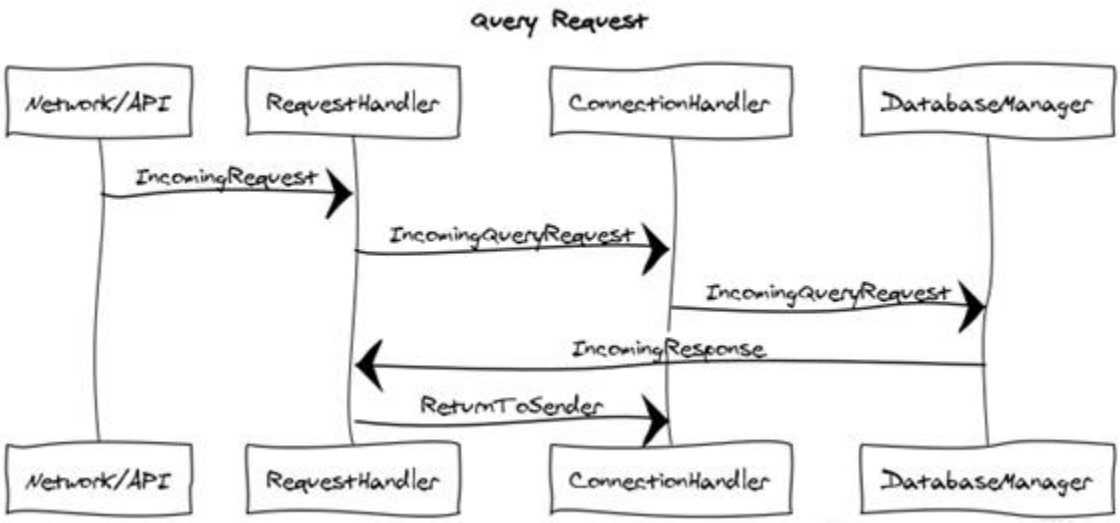

Figure 9 QueryRequest processing events

The following table describes each of the events seen in figure 9:

Table 2 QueryRequest processing events in detail

Event name	Triggered by	Cause
IncomingRequest	Network / API	A request was received from the network or the API.
IncomingQueryRequest	RequestHandler	A request was identified as a query and should be processed by all classes.
IncomingResponse	DatabaseManager	The query was resolved, and the result is now available.
ReturnToSender	RequestHandler	All the expected responses are received. It is time to return the query result to the requesting node.

5.3.5.2 File request

A file request is processed asymmetrically in three stages. The three stages are:

1. The file request is processed at the node that initiated the request
2. The file request is then processed at the node that the request was intended for.
3. The response of the file request is processed at the node that initiated the request.

The following three figures and tables present and describe these three stages.

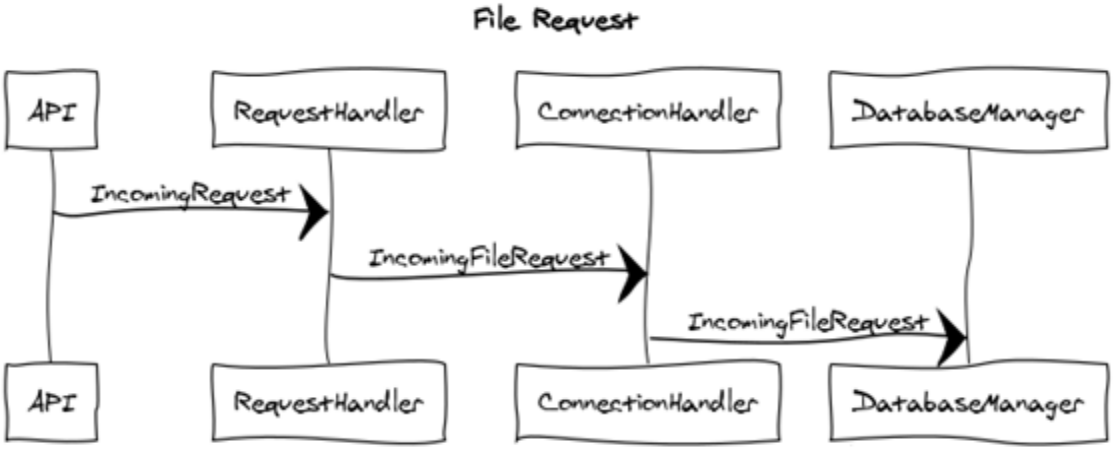

Figure 10 FileRequest processing events (stage 1 at requesting node)

The following table describes each of the events in the first stage:

Table 3 FileRequest stage 1 processing events in detail

Event name	Triggered by	Cause
IncomingRequest	API	A request was received from the API/UI.
IncomingFileRequest	RequestHandler	A request was identified as a file request and should be processed by all classes.

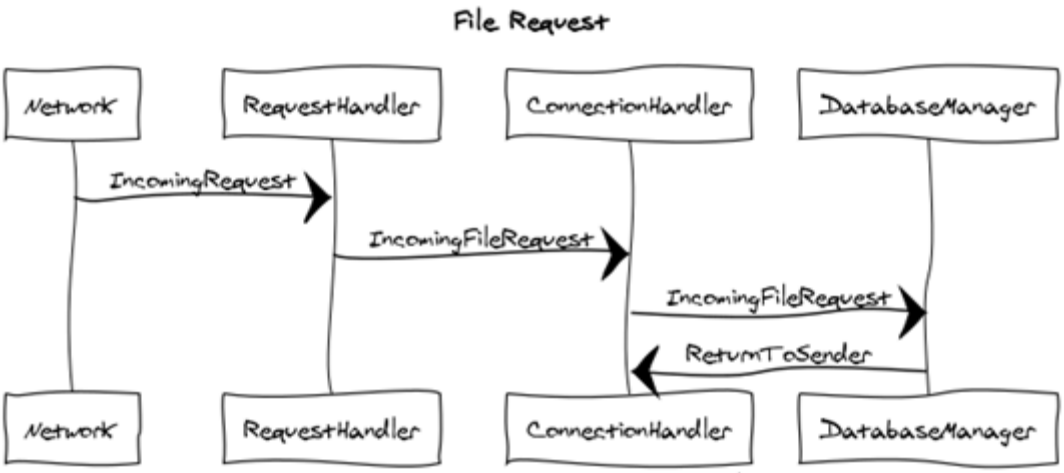

Figure 11 FileRequest processing events (stage 2 at target node)

The following table describes each of the events in the second stage:

Table 4 FileRequest stage 2 processing events in detail

Event name	Triggered by	Cause
IncomingRequest	Network	A request was received from the network.
IncomingFileRequest	RequestHandler	A request was identified as a file request and should be processed by all classes.
ReturnToSender	DatabaseManager	The file request was processed. Either the transfer has started, or an error has been issued.

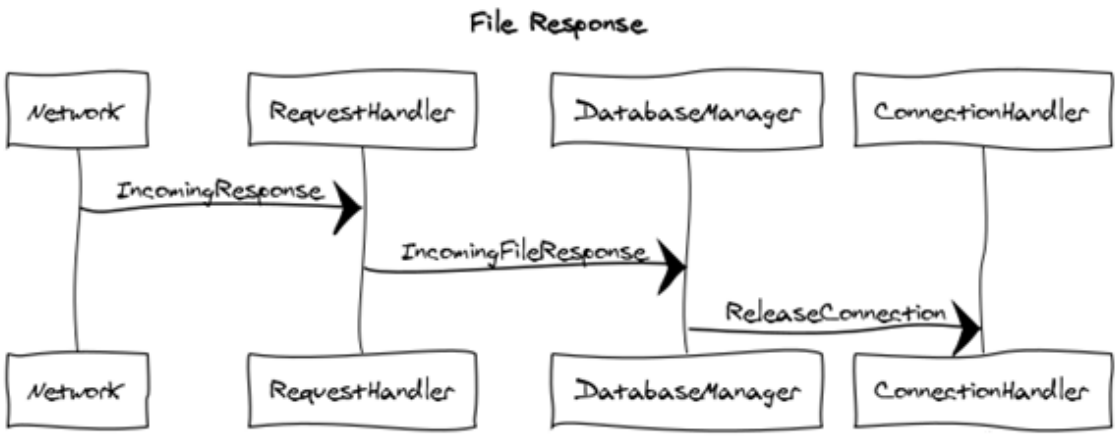

Figure 12 FileRequest processing events (stage 3 at requesting node)

The following table describes each of the events in the third stage:

Table 5 FileRequest stage 3 processing events in detail

Event name	Triggered by	Cause
IncomingResponse	Network	A response was received from the network.
IncomingFileResponse	RequestHandler	A response was identified as a response to a file request and should be processed by the DatabaseManager.
ReleaseConnection	DatabaseManager	Either the file transfer is complete, or an error has occurred. The ConnectionHandler will stop expecting the file from the user.

5.3.6 Implementation challenges

The following subsections discuss the challenges faced during the implementation phase. All the challenges discussed in this section are a direct result of the current state of the technology and are not in any way the shortcomings of the research done for this project.

5.3.6.1 Signalling server

As mentioned in section 1.0 and 4.1, the signalling server is needed to enable the offer/answer model used by WebRTC to establish a connection without the two peers being initially connected to each other. This means that the P2P network will deviate from the ideal P2P network due to the interaction with a server.

5.3.6.2 Single Page Application (SPA) requirement

The redirection of a webpage disconnects the connection to the database and other peers. This means that for an application to fully function without any disruptions, it is required to operate without any actual refresh or reload of a browser session. This leads to the requirement that the application should be a single page application operating from a single page load. This means that the application needs to be built with more recent JavaScript frameworks which have been known among the community to have a steep learning curve. This slightly extends the time to market for an application built using this framework.

5.3.7 Implementation accomplishment

This section presents the major implementation accomplishment for this framework.

5.3.7.1 Freedom from a specific query syntax

One of the main advantages of this DS2 implementation is the freedom from a specific query language. This is partly because IndexedDB does not support a specific query language but rather offers simple queries through cursors on indices using its JavaScript API. This means that it is vital to ensure that this framework implementation is future-proof and can handle any query language as more IndexedDB query options become available.

5.4 Testing

Testing in the browser was done using Karma [24], a test runner capable of launching browsers, building a test suite, and running the test suite. The test framework used is the mocha [25] testing framework while the assertion library used is the browser compatible library Chai [26].

The testing procedure is described below:

1. Start a local signalling server through which the required signalling for WebRTC is done.
2. Run all functional and unit tests in a browser.
3. Stop the local signalling server.

Unit testing was used to test the individual modules expected behaviours while functional testing was used to verify that the application meets the functional requirements listed in section 3.3. There were also some integration tests to ensure the individual modules were instantiated as expected. For all tests, a simple mock MetadataHandler was implemented to allow testing of the entire library and the DatabaseManager without implementing an actual MetadataHandler. The most challenging obstacle faced when writing tests occurred because of the npm environment used for module(package) management. The algorithm implemented by npm does not always guarantee the packages installed, which lead to a few unexpected breakages which were eventually resolved. Another challenge was the testing of several nodes in a single browser window, which was fixed by adding a way to instantiate a node or one of the other three main modules using a custom PeerId object passed through the start method. Another possible fix that was not considered at the time could have been to use the options object in the constructor of the DS2 library. Stress testing of the transport was left to the networking stack libp2p. The networking stack is currently in alpha testing with a few applications using it.

Currently, all testing is automated, but there were several manual tests performed along the way. By the end of the project, all the testing had become automated through the usage of Continued Integration (CI).

Testing instructions can be found in APPENDIX A.3.

Chapter 6

6.0 MongoIdbDs2 and stamp collection example

This chapter will present an example application built using this framework

6.1 Overview

This section aims to provide a complete overview of the browser-based P2P application through providing a description, the code location, installation requirement and usage details.

6.1.1 Description

The class MongoIdbDs2 is an extension of the DS2 class through inheritance. It adds a query syntax and metadata storage to the DS2 framework. The query syntax allows for exact text matches or JSON based numeric operations. This library only serves to change the signature of the constructor and the query method making the new syntax for instantiating a node easier. Other than changing the API slightly and instantiating the implemented MetadataHandler internally, this module does not accomplish anything else. It was created to stay true to the non-functional requirements listed in section 3.3.2.

The stamp collection is a distributed stamp collection proof of concept where each user can connect to other users, upload their stamp collection, view their stamp collection, search for other stamps within their peers.

6.1.2 Code location

The MongoIdbDs2 code is currently located in the GitHub repository:

https://github.com/michaelfakhri/mongo-idb-ds2

The library is also available from npm through “mongo-idb-ds2” and from UNPKG CDN through https://unpkg.com/mongo-idb-ds2/, which is a CDN that uses npm registry as it’s storage medium.

The stamp-collection example is currently located in the GitHub repository: https://github.com/michaelfakhri/stamp-collection and can be directly run by using the following link: https://michaelfakhri.github.io/stamp-collection

6.1.3 Install and Usage

Install and usage instructions can be found in APPENDIX B.1 and B.2.

6.1.4 Browser support

The currently supported browsers are:

- Chrome
- Firefox
- Opera

6.1.5 Development environment

Npm [15] and Node.js [16] environments were used to develop and test the MongoIdbDs2 library.

6.2 Design and implementation

The dexie-mongoify library [27] was used to add JSON based query syntax to the DS2 library. The details of this JSON based query syntax can be found in APPENDIX B.5.

The following figure shows the UML class diagram for the MongoIdbDs2 class.

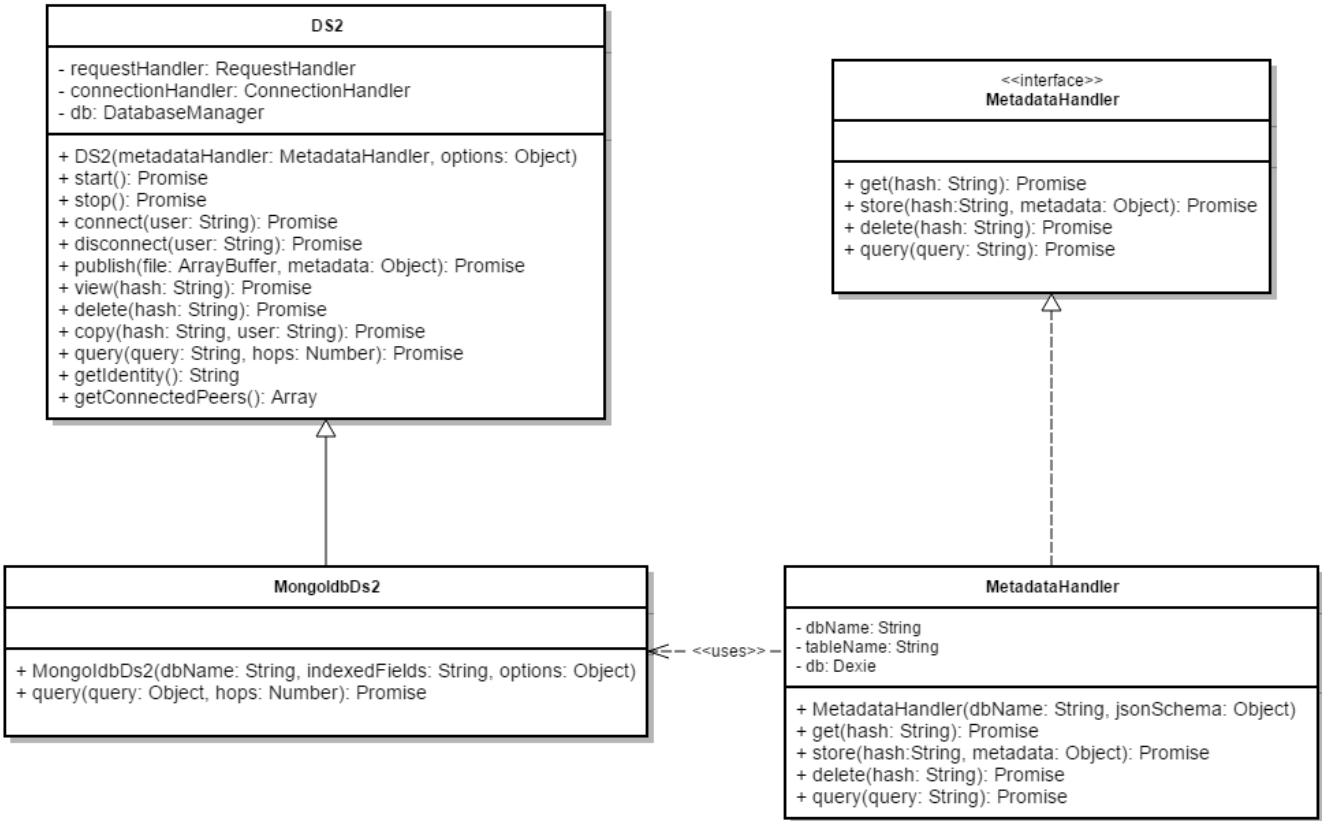

Figure 13 MongoIdbDS2 UML class diagram

Figure 13 shows that the only difference between the DS2 library and the MongoIdbDs2 library is that the constructor no longer requires a MetadataHandler object and that the query function signature now takes a JSON based query rather than simply a text query.

A more detailed description of the API can be found in APPENDIX B.4 and a few examples in APPENDIX B.6.

The stamp collection was created as a proof of concept rather than a designed product of this project; therefore, it lacked the proper design cycle, but there was a significant amount of effort put into implementing and refining it for the poster fair.

The following are the highlights for this implementation of the web application:

- AngularJS was used for routing and templating the different tabs so that it acts like a Single Page Application (SPA).
- jQuery for Document Object Model (DOM) and Cascading Style Sheets (CSS) manipulation.
- Bootstrap framework for tables and visuals.

6.3 Testing

The same exact testing setup used for the DS2 library is used (section 5.4) for the MongoIdbDs2 library and is listed below for reference.

Testing in the browser was done using Karma [24], a test runner capable of launching browsers, building a test suite, and running the test suite. The test framework used is the mocha [25] testing framework while the assertion library used is the browser compatible library Chai [26].

The testing procedure is described below:

1. Start a local signalling server through which the required signalling for WebRTC is done.
2. Run all functional and unit tests in a browser.
3. Stop the local signalling server.

Testing of this library mainly consisted of unit tests involving the query syntax and integration tests verifying the library works as expected when instantiated. The rest of testing was delegated to the DS2 library since all the tests written for the DS2 library are still valid for the MongoIdbDs2 library. There was no need to mock anything for this library other than using multiple PeerId objects to instantiate multiple nodes in a single browser session in a similar fashion as described in section 5.4.

Testing instructions can be found in APPENDIX B.3.

Manual alpha testing of the stamp collection proof of concept was done throughout the development of the proof of concept. Unfortunately, there was not enough time to use a JavaScript framework to automate the testing of the proof of concept.

Chapter 7

7.0 Related work

This chapter will discuss the initial work done towards the minimum viable product (MVP) and the open source contributions that occurred as a direct result of this project's requirements.

7.1 Initial work

At the start of the project, a minimum viable product was used to determine the feasibility of P2P connectivity and P2P file transfer in the browser. WebRTC was used for P2P connectivity while file transfer was done by chunking a file and sending it through the WebRTC transport. The MVP also experiments with the selenium WebDriver End-to-End (E2E) testing framework, which was considered for the DS2 P2P framework but was later abandoned in favour of the setup described in section 5.4. The rationale for the change was that Karma offers a more modular approach to testing, separating the browser launcher(karma), testing framework (mocha) and the assertion library (Chai).

The initial investigation and experimentation with WebRTC and E2E test automation using selenium WebDriver is available at the GitHub repository:

https://github.com/michaelfakhri/BasicWebRtcUsingPubNub

7.2 Contributions to open source work

This section will briefly present the various open source bug reports and fixes that were submitted because of this project.

7.2.1 Bug reports

The following table summarizes the bug reports were submitted to their respective repositories:

Title	Repository	URL
Not all tests are run in the Firefox browser launched in TRAVIS	libp2p/js-libp2p-webrtc-star	https://github.com/libp2p/js-libp2p-webrtc-star/issues/79
Websocket stays connected even after webrtc-star listener is closed	libp2p/js-libp2p-webrtc-star	https://github.com/libp2p/js-libp2p-webrtc-star/issues/51
Error when trying to run libp2p module in the browser	ipfs/js-libp2p-ipfs-browser	https://github.com/ipfs/js-libp2p-ipfs-browser/issues/118

The first bug report was simply an administrational bug report while the following two came as a direct result of understanding and using the Libp2p networking stack for this project.

7.2.2 Bug fixes

The following table summarizes the bug fixes that were submitted as a result of working on this project and using the libp2p networking stack:

Title	Repository	URL
Added error handling to transport.dial functionality when there is one multiaddr	libp2p/js-libp2p-swarm	https://github.com/libp2p/js-libp2p-swarm/pull/133
Added signaling error handling to webrtc-star transport	libp2p/js-libp2p-webrtc-star	https://github.com/libp2p/js-libp2p-webrtc-star/pull/39
A few BUGFIXES and new tests to cover those scenarios + multiple signalling servers!	libp2p/js-libp2p-webrtc-star	https://github.com/libp2p/js-libp2p-webrtc-star/pull/89

The first two bug entries were vital to the operation of the P2P framework while the third entry included a few bug fixes and a new feature. The first two entries in the previous table were successfully merged while the third entry is pending review.

Chapter 8

8.0 Summary and Recommendations

The problem that this project focused on solving was eliminating the install requirement for P2P applications built using the system model known as $(DS)^2$. This was done by using the latest browser technologies to implement a JavaScript framework capable of delivering such applications to the user with minimum effort. The stamp collection proof of concept served to demonstrate the success of the implemented solution.

Overall, this project proved to be a great learning experience. Challenges ranged from the simple non-documented JavaScript API to the more complex unstable/unreliable environments used for JavaScript development and debugging. Implementing for several browser vendors and browser versions seems to be the most challenging aspect of web development.

A few suggested projects for the continuation of this work are available below:

- Create two types of nodes, one that runs in the browser only for light users and one that uses a daemon that is available all the time for more experienced users and users that wish to be online all the time. This is possible to do using the same codebase with minimal modifications.
- Using the daemon based approach, attempt to design, and implement a P2P e-learning system such as cuLearn.
- Might not be practical but attempt to study the feasibility of running a P2P network using mobile phone browsers or applications.
- A command line based library could be built using JavaScript with the same code base. Another possibility is a P2P version of git.
- Create a boilerplate repository so that future applications can just use that as their starting point to create a new demo or proof of concept.

Abbreviations

CDN	Content Delivery Network
CI	Continuous Integration
CU	Carleton University
DS^2	Distributed Social Data Sharing
JRE	Java Runtime Environment
MVP	Minimum Viable Product
NAT	Network Address Translation
NMAI	Network Management and Artificial Intelligence
OO	Object-Oriented
P2P	Peer to Peer
UP2P	Universal Peer to Peer
W3C	World Wide Web Consortium
WebRTC	Web Real-Time Communication
SPA	Single Page Application
STUN	Session Traversal Utilities for NAT
GUI	Graphical User Interface

References

[1] S. Engineering, "P2Pedia - Network Management and Artificial Intelligence", Carleton.ca, 2017. [Online]. Available: https://carleton.ca/nmai/research-projects/universal-peer-to-peer/p2pedia/. [Accessed: 02- Apr- 2017].

[2] "Agile Project Management Software for Agile Development and Issue Tracking | Yodiz", Yodiz.com, 2017. [Online]. Available: http://www.yodiz.com/. [Accessed: 02- Apr- 2017].

[3] 2017. [Online]. Available: http://sce.carleton.ca/~adavoust/A_Davoust_PhD_Thesis_2015.pdf. [Accessed: 02- Apr- 2017].

[4] A. MUKHERJEE, B. ESFANDIARI and N. ARTHORNE, "U-P2P: a peer-to-peer system for description and discovery of resource-sharing communities", *22nd International Conference on Distributed Computing Systems Workshops*, pp. 701-704, 2002.

[5] "WebRTC API", Mozilla Developer Network, 2017. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API. [Accessed: 02- Apr- 2017].

[6] "WebP2P by ShareIt-project", Shareit-project.github.io, 2017. [Online]. Available: https://shareit-project.github.io/WebP2P.io/. [Accessed: 02- Apr- 2017].

[7] "WebTorrent - Streaming browser torrent client", Webtorrent.io, 2017. [Online]. Available: https://webtorrent.io/. [Accessed: 02- Apr- 2017].

[8] "libp2p/libp2p", GitHub, 2017. [Online]. Available: https://github.com/libp2p/libp2p. [Accessed: 02- Apr- 2017].

[9] "File", Mozilla Developer Network, 2017. [Online]. Available: https://developer.mozilla.org/en/docs/Web/API/File. [Accessed: 02- Apr- 2017].

[10] "FileReader", Mozilla Developer Network, 2017. [Online]. Available: https://developer.mozilla.org/en/docs/Web/API/FileReader. [Accessed: 02- Apr- 2017].

[11] 2017. [Online]. Available: https://developer.mozilla.org/enUS/docs/Web/API/Web_Storage_API/Using_the_WebStorage_API. [Accessed: 02- Apr- 2017].

[12] 2017. [Online]. Available: https://developer.mozilla.org/enUS/docs/Web/API/IndexedDB_API/Basic_Concepts_Behind_IndexedDB. [Accessed: 02- Apr- 2017]

[13] "IndexedDB API", Mozilla Developer Network, 2017. [Online]. Available: https://developer.mozilla.org/en/docs/Web/API/IndexedDB_API. [Accessed: 02- Apr- 2017].

[14] "FileSystem", Mozilla Developer Network, 2017. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/FileSystem. [Accessed: 02- Apr- 2017].

[15] "npm", Npmjs.com, 2017. [Online]. Available: https://www.npmjs.com/. [Accessed: 02- Apr- 2017].

[16] N. Foundation, "Node.js", Nodejs.org, 2017. [Online]. Available: https://nodejs.org/en/. [Accessed: 02- Apr- 2017].

[17] "Peer To Peer Icon - Free Large Torrent Icons - SoftIcons.com", Softicons.com, 2017. [Online]. Available: http://www.softicons.com/application-icons/free-large-torrent-icons-by-aha-soft/peer-to-peer-icon. [Accessed: 02- Apr- 2017].

[18] "Database Icon - Marmalade Icons - SoftIcons.com", Softicons.com, 2017. [Online]. Available: http://www.softicons.com/toolbar-icons/marmalade-icons-by-icojam/database-icon. [Accessed: 02- Apr- 2017].

[19] "libp2p/js-peer-id", GitHub, 2017. [Online]. Available: https://github.com/libp2p/js-peer-id. [Accessed: 02- Apr- 2017].

[20] "libp2p/js-libp2p", GitHub, 2017. [Online]. Available: https://github.com/libp2p/js-libp2p. [Accessed: 02- Apr- 2017].

[21] "pull-stream", Pull-stream.github.io, 2017. [Online]. Available: https://pull-stream.github.io/. [Accessed: 02- Apr- 2017].

[22] "ipfs/js-idb-pull-blob-store", GitHub, 2017. [Online]. Available: https://github.com/ipfs/js-idb-pull-blob-store. [Accessed: 02- Apr- 2017].

[23] "medikoo/deferred", GitHub, 2017. [Online]. Available: https://github.com/medikoo/deferred. [Accessed: 02- Apr- 2017]. [24]

[24] "Karma - Spectacular Test Runner for Javascript", Karma-runner.github.io, 2017. [Online]. Available: https://karma-runner.github.io/1.0/index.html. [Accessed: 03- Apr- 2017].

[25] "Mocha - the fun, simple, flexible JavaScript test framework", Mochajs.org, 2017. [Online]. Available: https://mochajs.org/. [Accessed: 03- Apr- 2017].

[26] "Chai", Chaijs.com, 2017. [Online]. Available: http://chaijs.com/. [Accessed: 03- Apr- 2017].

[27] "YurySolovyov/dexie-mongoify", GitHub, 2017. [Online]. Available: https://github.com/YurySolovyov/dexie-mongoify. [Accessed: 03- Apr- 2017].

APPENDIX A DS2

A.1 Installation for development

This library can be fully installed for development using the following steps:

1. Download and install npm >= 3.0.0 and node.js >= 4.0.0
2. Clone the repository from GitHub (https://github.com/michaelfakhri/ds2)
3. run command “npm install”

A.2 Usage in application

This library is available through the following sources:

- npm registry `ds2`
- UNPKG `https://unpkg.com/ds2/dist/index.min.js`

A.3 Testing

These tests can be run using the following step (do this only after installation is complete, see A.1):

Run command “npm test”

A.4 API Documentation

Constructor

`DS2(metadataHandler, options)`
returns an instance of this module.

Where:

- `metadataHandler :MetadataHandler` - Any object that implements the Metadata Handler interface.
- `options : Object` - The available options are listed below:
  - `useEcncryption : Boolean` - true/false default: false
  - `signalling : String` - default: '/libp2p-webrtc-star/ip4/127.0.0.1/tcp/15555/ws/ipfs/' The address of the signalling server (libp2p-webrtc-star) in Multiaddr (https://github.com/multiformats/multi-addr) format. Another example is '`/libp2p-webrtc-star/dns/blooming-atoll-60728.herokuapp.com/wss/ipfs/`'.

Methods

`Promise start([peerId])`
returns a promise that resolves when network layer and storage modules are started.

Where:

- `peerId :PeerId (https://github.com/libp2p/js-peer-id)` - An optional parameter that is used to instantiate the libp2p networking layer with a custom peer identity. This is mainly used for testing.

`Promise stop()`
returns a promise that resolves when network layer and storage modules are stopped.

`Promise connect(user)`
returns a promise that resolves when the peer is connected.

Where:

- `user : String` – The identity of the user that the networking layer should connect to.

`Promise disconnect(user)`
returns a promise that resolves when the peer is disconnected.

Where:

- `user : String` – The identity of the user that the networking layer should disconnect from.

`Promise publish(file, metadata)`
returns promise that resolves to the hash of the file after the file and metadata are stored in their respective storage units.

Where:

- `file : Array` - The file contents to store in storage so that other users can request this file contents.
- `metadata : Object` – The metadata of the file contents provided, which will also be stored using metadata storage provided in the constructor.

`Promise delete(hash)`
returns a promise that resolves when the file and its metadata are deleted from the data storage unit.

Where:

- `hash : String` – The hash of the file contents to delete from data storage unit.

`Promise copy(hash, user)`
returns a promise that resolves when file is downloaded. A user must be connected using connect(user) before calling this function.

Where:

- `hash : String` – The hash of the file contents to download from another peer.
- `User : String` - The identity (public key) of the user to download the data from

`Promise view(hash)`
returns a promise that resolves to the file stored in the data storage unit. The promise is rejected if hash does not exist in the data storage unit.

Where:

- `hash : String` – The hash of the file contents to view from the data storage unit.

`Promise query(query, hops)`
returns promise that resolves when query responses are received from all connected peers. This query string is sent to all connected peers. Each peer will do three things: 1) pass it to the MetadataHandler object passed to the `constructor` when this class was instantiated. 2) Decrement hops and forward the query to all connected peers if the number of hops left is greater than 0. 3) collect all responses that come in from the peers that received this query from this node. Send back the responses once the last response is received.

Where:

- `query : String` – A query of the metadata storage unit.
- `hops : Integer` – The number of hops this query should make away from this node.

`Array getConnectedPeers()`
returns a list of the connected peer.

`String getIdentity()`
returns the identity of a node.

A.5 Detailed examples

The following code snippets show several examples of the API being used to accomplish the tasks set out in the system model described in section 3.1.

```
// DS2 is the implemented P2P framework while MetadataHandler is implemented by the
application using DS2


// Create a new instance and start it

let node = new DS2(new MetadataHandler())

node.start()

.then(() => {

  // Node has started

})


// After node is started, you can connect to node known as ‘QmUserID’

node.connect(‘QmUserID’)

.then(() => {

  // User is connected

})

.catch((err) => {

  // Operation failed with error err

})


// After node is started, you can publish a file with contents fileContents of type
ArrayBuffer and with metadata object {fileMetadata: ‘this is some metadata’}. The ArrayBuffer
is usually created using a FileReader invoking readAsArrayBuffer method on a File Object
obtained from a HTML5 submitted form.

node.publish(fileContents : ArrayBuffer, {fileMetadata: ‘this is some metadata’})

.then((fileHash) => {

  // Do something with hash of file contents

})

.catch((err) => {

  // operation failed with error err

})
```

```
// After node is started, you can view a previously publish file with hash ‘QmFileHash’
node.view(‘QmFileHash’)
.then((fileContetns) => {
  // Do something with file contents
 })
.catch((err) => {
  // Operation failed with error err
})

// After node is started and user ‘QmUserID’ is connected, you can copy file and metadata with
hash ‘QmFileHash’. After the file is copied, the file is viewed
node.copy(‘QmFileHash’, ‘QmUserID’)
.then(() => node.view(‘QmFileHash’))
.then((fileContents) => {
  // Do something with new received file contents
})
.catch((err) => {
  // Operation failed with error err
})

// After node is started, you can query the locally stored metadata to get the results that
match ‘query’. From this class’s point of view, query is a String that will be passed to
MetadataHandler which will process it and return the results.
Node.query(‘query’, 0)
.then((localQueryResults) => {
  // Do something with local query results
})
.catch((err) => {
  // Operation failed with error err
})
```

```
// After node is started and at least one connection to another peer is established, you can query the metadata stored locally and 2 hops away from this node to get the results that match ‘query’. The ‘query’ is passed to the MetadataHandler instance of each of the peers that are at a maximum of two hops away.

Node.query(‘query’, 2)

.then((localAndRemoteQueryResults) => {

  // Do something with local and remote query results

})

.catch((err) => {

  // Operation failed with error err

})
```

APPENDIX B MongoIdbDs2

B.1 Installation for development

This library can be fully installed for development using the following steps:

1. Download and install npm >= 3.0.0 and node.js >= 4.0.0
2. Clone the repository from GitHub (https://github.com/michaelfakhri/mongo-idb-ds2)
3. Run command “npm install”

B.2 Usage in application

This library is available through the following sources:

- npm registry `mongo-idb-ds2`
- UNPKG `https://unpkg.com/mongo-idb-ds2/dist/index.min.js`

B.3 Testing

These tests can be run using the following step (do this only after installation is complete, see A.2):

Run command “npm test”

B.4 API Documentation

Constructor

`MonfoIdbDS2(aDBName, indexedFields, options)`
returns an instance of this module.
Where:

- `aDBName : String` - The name of the database to store the metadata.
- `indexedFields : String` - A comma separated list of the indexed fields in the database. The only requirement is that the first field is 'hash' that is used internally by this module.
- `options : Object` - The available options are listed below:
  - `useEcncryption : Boolean` - true/false default: false
  - `signalling : String` - default: '/libp2p-webrtc-star/ip4/127.0.0.1/tcp/15555/ws/ipfs/' The address of the signalling server (libp2p-webrtc-star) in Multiaddr (https://github.com/multiformats/multi-addr) format. Another example is '`/libp2p-webrtc-star/dns/blooming-atoll-60728.herokuapp.com/wss/ipfs/`'.

Overridden Methods

`Promise query(query, hops)`
returns promise that resolves when query responses are received from all connected peers.
Where:

- `query : Object` – An object that follows the JSON query syntax documented here (https://github.com/YurySolovyov/dexie-mongoify/blob/master/docs/query-api.md).
- `hops : Integer` – The number of hops this query should make away from this node.

B.5 Query syntax

Supported mongo query operators:

- $eq - equals
- $gt – greater than
- $gte – greater than or equal
- $lt – less than
- $lte – less than or equal
- $ne – not equal
- $in – in array
- $nin – not in array
- $exists – key exists
- $all – all in array
- $size – equal size of array
- $elemMatch – combines different queries operators together on same field
- $not – inverse the query

Supported logical operators:

- $and
- $or
- $nor

B.6 Detailed examples

```
// Create a new instance and start it

let node = new MongoIdbDs2('stamps', 'hash, name, country, year')

node.start()

.then((() => {

  // Node has started

})

// After node is started, you can query the locally stored metadata to get the results that
have a year field that is less than 500 or greater than 2000. From this class's point of view,
the query is a JSON object that will be stringified and then passed through the processing
stages of the DS2 library.

node.query(({
    $or: [
        { year: { $lt: 500 } },
        { year: { $gt: 2000 } }
    ]
}, 0)

.then((localQueryResults) => {

  // Do something with local query results

})

.catch((err) => {

  // Operation failed with error err

})
```