SP-114 Data Lake Implementation

Final Report

CS 4308, Section 02, Spring 2026

Team Members: Caleb Cox, Bryce Wishart

Instructor: Sharon Perry, 04/20/2026

Website Link

GitHub Link

Stats:

Lines of Code: 3,492

Project Components: 4

Total Work Hours: 156

Status:

This project is 100% complete and working as intended

Table of Contents

1.0 Overview

This document details the planning and development of a miniature data lake for storing and filtering data. Each section of the document provides different details on the factors and considerations that were assessed to create the final product. The overview section is intended to provide readers with a description of the project and what it means.

A data lake is a form of large data storage that involves potentially hundreds of thousands of files. The main difference between data lake and other data storage methods, however, is that database schema are applied to data housed within the data lake on read rather than on write. For example, a data lake can intake text files, images, and videos, but it will only filter this data upon a user request.

As a result, data lakes can often store much wider ranges of data compared to something like a data warehouse, which applies schema as data enters the structure. This makes a data lake much more scalable to a large amount of data and allows for more flexibility in what metrics the user can choose to analyze.

This project creates a miniature version of a data lake to prove the flexibility of the structure in analyzing data. Our project uses a variety of tools and programs detailed later in the document to create a functional data server that stores hundreds of thousands of car listings across the state. These are then sanitized and filtered to allow for SQL queries to further filter car listings toward a particular statistic.

With these clean data records, someone who wants to analyze a topic like retail sales across different stores, car sales by location, or website accesses by time can easily filter data to suit their needs. Our project looks at the car listings that are imported to gather pricing trends across the state.

2.0 Requirements

2.1 Scope

The Data Lake Project has several main constructions needed to ensure proper functionality. Firstly, data lake must have a system to intake raw data from various formats. This raw or bronze zone should encompass where all input data is first run through. Our data is sourced through the Carfax website by scraping data on car listings.

Secondly, this data lake needs some form of storage for the massive amounts of data that are streaming in. This is where the object store is used. With an object store, the raw data can be checked, cleaned, and organized into a standard format or several formats for ease of access. This is the silver layer, and it is here that our project will add schemas to the data to provide structure.

The final main construction of the Data Lake Project is the query engine. This part allows the user to directly pull data from the object store using the file formatting system applied. Here, data can be collated into a proper report for analysis purposes. This gold zone is where the data is put into its most organized state.

2.2 Constraints

The main constraint of the Data Lake project is the tools that we can use. This project is made using open-source tools like MinIO or the Apache suite of software for data analytics. If these tools ever lose support in some fashion, this project will need to be moved to other data manipulation tools that are still in service.

To keep familiarity with the tools, this project is also going to be created using Python as the primary programming language for the main programs.

2.3 System Features

2.3.1 Data Acquisition

2.3.1.1 Description

This part of the Data Lake system manages intaking inputs from a source and creating an entry for the object store. This function encompasses the raw data and the bronze layer of medallion architecture.

2.3.1.2 Functional Requirements

- Connect to input stream.
- Read input data (text files, application logs, website clicks, etc.)
- Create key for input to use as access when in database.
- Place data into proper folders for type of input data.
- If input does not validate, system moves on to another input.
- Connect to object store to send input data.

2.3.2 Data Storage

2.3.2.1 Description

Data storage in the data lake is an object store tool that can hold massive amounts of data with varying types of formats. The data must be accessible both locally and across the Internet.

2.3.2.2 Functional Requirements

- Intake data from the data acquisition component
- If no folder exists for the data type being added, create new folder to store data.
- Sort the incoming data into the proper storage folders.
- Allow programs with valid credentials to access the folders and read, write, or transfer files.

2.3.3Data Formatting

2.3.3.1 Description

Data formatting in this data lake system includes both the file format used to access the data from the object storage, as well as the file format used to structure the raw data. With this feature, data becomes more structured and enters the silver zone of medallion architecture.

2.3.3.2 Functional Requirements

- Connect to the object store.
- Create schema according to user specification.
- Sort data by the created schema.
- Access the data according to the query given by a user.
- Display all valid data that conforms to the specified query.
- Any errors in displaying data should be documented by the system in a secondary log.

2.3.4 Data Querying

2.3.4.1 Description

This final system feature allows for the querying of data stored in the object store using query programs. The data is accessed by a program which sorts the cleaned data to find what files fit the necessary schema. Once done, these listings are added to a new file and written to a file that can be read by the user.

2.3.4.2 Functional Requirements

- Take in user SQL input for data.
- Send user input through the data format portion of the system.
- Filter listings to only those that fit the criteria.
- Create a new file that holds all files that fit the criteria.
- If data cannot be retrieved, send an error message to the user.

2.4 Interface Requirements

2.4.1 Software Requirements

The Data Lake project has several components that contribute to the completion of the software. Each section needs to connect with each other to allow for data to flow seamlessly from one program to another. The input stream should connect from the Carfax website source and send the packaged data into the data store.

From the MinIO object store, the data should be retrievable so that programs can grab the data to clean the datasets for organization according to specified schema. Parquet should be the format system used in the program to access and clean the data. Once cleaned, this data is parsed with SQL queries to create a new file with the proper listings.

2.5 Analysis

2.5.1 Use Cases

There are many diverse types of users or businesses that may want access to a data lake for gathering and filtering data. Some examples include:

- A web developer wants to analyze the traffic on their website and how often users stay active.
- A business wants to check sales figures for specific products or during specific times of the year.

2.5.2 Data Flow

In this data lake, data will first enter the system from the data stream connected to the source website. At this time, the data can be in any format, from excel files to text documents or even images or website clicks. From the data stream, these pieces of data are organized into folders for easier control over data flow.

After checking and validating the data, it is moved into the object store. At this point, the data can be further organized using a standard format to create schemas that structure the data. This structured form of the data is what the end user can access when they make SQL calls to the data store.

3.0 Design

3.1 Dependencies

Our data lake is going to be built on a Windows installation, so the assumption for end users is that they own a computer that is running Windows or can run Windows programs. Each of the tools used to create the components of the data lake will also be using Windows installations.

Many of the tools that will be used to create the data lake are from the Apache Software Foundation, which is a group providing open-source computing software for communities. The assumption in using these tools is that they will work with each other to connect the separate components of the data lake.

On the user side, it is assumed that they have knowledge of SQL queries and working with databases, as pulling data from the data lake requires knowledge of database systems and how to manipulate and pull data from them.

3.2 Development Methods

The primary development method employed for this project is the Agile method. From the requirements we determined in the specification document, we will find the most key features that need to be implemented.

Once found, our team will begin a sprint to develop and implement a few chosen features of the final product. Following each sprint, the team will meet and discuss what needs to be implemented next and what might need for revision. Afterwards, another iteration of planning and sprinting can begin.

3.3 Architectural Strategies

This section details the various strategies that we used in creating our data lake. As mentioned in earlier sections, the programming aspect of the project will be managed using the Python programming language. The reasoning behind the use of this language is familiarity and ease of use for team members.

Another part of our data lake is the user's input. The end user of the database will not be able to directly enter raw data stored in the system, as the query engine is only going to be used for system output. If any data requires introduction into the data storage, it should be added through the data acquisition component of the architecture.

3.3.1 Apache Parquet

Apache Parquet is an open-source data file format that is used for retrieving data from a database. A data lake requires some form of formatting for data retrieval, since the data format is also how structure can be added to the raw data through schema. There are several options for file formats, but we chose Parquet for its speed.

This file format is column-oriented; meaning data is stored by column instead of by row. As such, the Parquet queries only read the user's specified columns, speeding up the time it takes to retrieve the data. The goal of our project is to create a repository holding substantial amounts of data, so having a file format that can scale to higher volumes of data while keeping speed in querying is imperative.

3.3.2 MinIO

MinIO or the Alstor object server, is a large-scale data storage system that can be deployed through local computers or the cloud to hold data. This was chosen for use in creating our project since it is primarily used in analytics and has support for Python application connections.

MinIO carries support for single-node deployment for free licenses, meaning our project can use one computer or drive as a storage system for data. Versioning is another aspect of the server which allows for a correct audit of how data is managed in the data store.

3.3.3 Future Support

Since this project is a collaborative effort among students, much of its final design will remain unchanged following the conclusion of the project. If any team member wishes to continue iterating on the project, the final package will remain available for members to fork and create updates to continue support on the data lake.

3.4 System Architecture

A data lake needs several different systems working in tandem to create a functional data storage that can be accessed and analyzed. Our team partitioned the creation of the data lake into four major sections: Data Acquisition, Data Storage, Data Formatting, and Data Querying.

The data acquisition subsystem encompasses all functions and libraries that are used to create a program that pulls in raw data as a stream and transfers it to the data storage subsystem. Data storage encompasses all programs that deal with storing raw data, and some minor structures with folders for storing diverse types of data.

From the storage segment, the data formatting section should pull data from the data storage section and apply the necessary schema to the data to gather all valid data that can be used for

analysis. Finally, the data querying section will take the structured data from data formatting and place it into clean reports that are easy for the end user to read and gather information from.

Below is a figure of the connection between users and functions of the system.

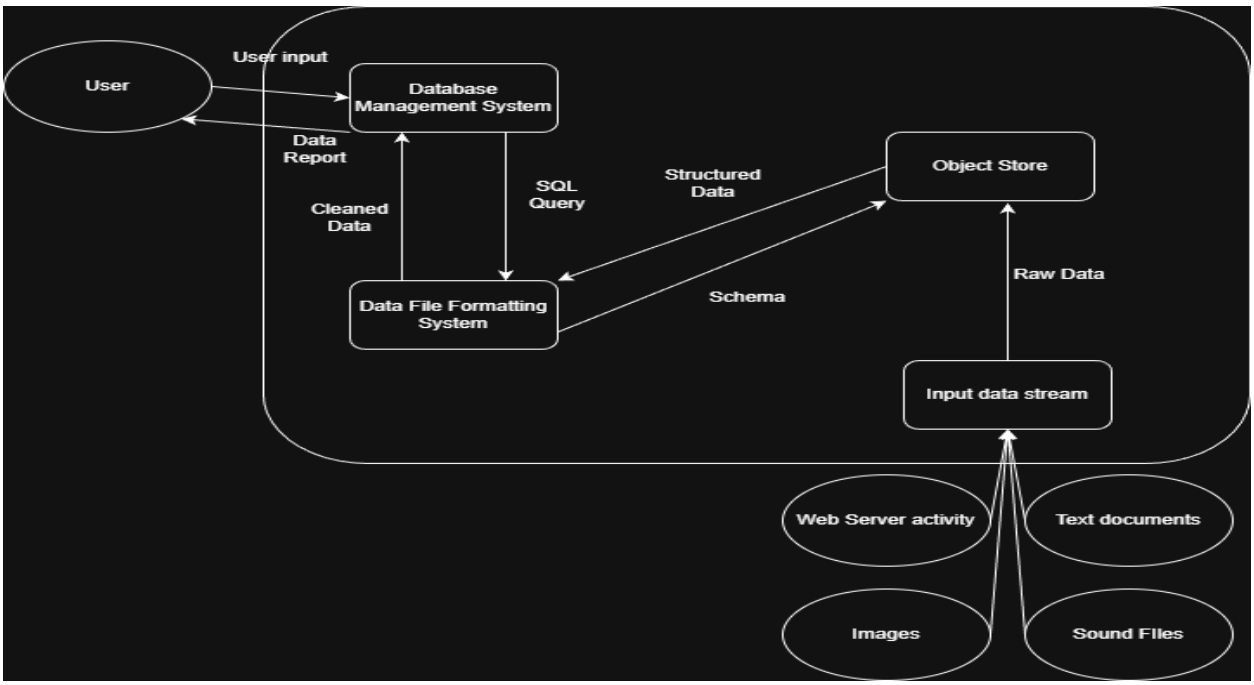

Figure 1:A diagram that shows the connection between users and the system of the data lake

3.5 Detailed System Design

3.5.1 Classification

- Input data stream- subsystem.
- Object store- subsystem.
- Data file formatting- subsystem
- Database management system- subsystem
- Raw data- file
- Schema- function
- Apache Parquet-package
- MinIO- package

3.5.2 Definition

- Input data stream- The functions and packages used for gathering raw data from an outside source.

- Object store- The group of components that encompass the storage of raw data. Packages like MinIO as well as raw data files are included in this subsystem.
- Data file formatting-The packages, functions, and classes used to apply structure to the raw data stored within the object store. Apache Parquet and the functions used for Schema are included in this subsystem.
- Database management system- The group of components used to send queries to the object store and receive structured data reports.
- Raw data- All files brought into the system by the input data stream.
- Schema- The structures applied to raw data that filters it into more refined data sets that can be arranged into a single report.
- Apache Parquet- Package that gives methods to convert data and manage schema.
- MinIO- Object storage system that can be connected to a supported input stream.

3.5.3 Constraints

- Input data stream- The data that streams into this component should be unstructured, meaning it can be of any type. Additionally, raw data must not be stored in this subsystem; it must be sent to the object store after validation.
- Object Store- While multiple users can interact with the object store when running queries, if the query results in a structure change for the data, lockouts must be enforced so that the system supports up-to-date state information of all stored data.
- Data file formatting- Data brought into this part of the system must be in the same file format, so that schema application and data modification can be applied to create a thoroughly cleaned set of data for analysis.
- Database management system- The program should use SQL queries to create new tabled data that fulfills the necessary schema and labels it accordingly.

4.0 Development

One key aspect of this project is the data that we collect and organize through database schema. We decided to use automotive information as a starting point for information gathering, so we sourced data from the Carfax website, pulling data on available cars. Once we gathered enough data, we could transfer the data to the object store.

The data sourcing program was created using Python, with two python files acting as the pipeline. The first one acts as the core logic for gathering data, with the second script inheriting some functions of the first one to add more cars from Carfax, print them to an output JSON file, and detail the progress of each data write.

To store the car listings, we created another script which writes the listing information into a Python dictionary, which has a max size of 512 megabytes. Once reached, a new file is created, and car listings are directed to the new output file. After enough car listings have been gathered, we have another program which connects to the object store using an access key to send all the necessary data into the raw zone of the object store.

Another component to this project is the object store that will function as a repository for all the data gathered, and a place where applications can pull data from for analytics. The MinIO object store is our solution to create a server to store data, as it is an open-source node deployment service. With a free license, our team only has access to a single server node to store data on, which we chose to deploy on a Windows installation. A server was created with several buckets or folders to store data, with incoming data being stored in the data lake raw bucket until they are organized into cleaner schema.

The next application that is needed for our data lake is the formatting program that uses Apache Parquet. To create this, we have several python scripts that access the MinIO object store server and pull the raw data from the bronze layer bucket. Once the files are pulled, the program goes through the file, which is a list of JSON files, and creates a table holding each of the attributes listed in the JSON files for the cars. After being tabled, the now clean dataset is sent back to the object store under a new bucket for cleaned datasets.

Finally, there is the gold layer of the data lake. When creating this, we have a program that accesses the silver layer bucket for the cleaned data tables, then sorts through every Parquet file stored in the bucket to create a curated list of cars that fulfills criteria such as market performance, depreciation, and price changes.

Some challenges our project faced included time spent waiting for data retrieval. Due to a limit on pulling data from Carfax, it took several days before we were able to pull enough data that could be used in our object store. Additionally, there were issues with connecting to the object store initially with the reverse proxy, as network firewalls prevented the data transfer.

4.1 Database Connections

The data lake project has one central object store that serves as a database for several types of data stored in their original formats. Our database was created using the MinIO Alstor server program. Once deployed, this program can be configured to store data on allocated drives using folders known as buckets that can hold multiple versions of the same piece of data. We configured the object store to run from a team member's local computer with an attached external drive that can hold a large quantity of data.

Connecting this central database to our other programs is a key point of the data lake project, and a focal point of development. The Alstor server allows for the connection of an application that is running libraries compatible with Amazon S3, another data storage service. Since our project is a smaller local deployment, the applications for sending and receiving data from the server will just need to have the network address of the store set up for data to be transferred to the server.

To allow the address of our store to be visible outside of the local network, we had to create a network tunnel that could funnel requests from a public hostname to the local network address. This was carried out by using a reverse proxy service known as LocalXpose, which allows for the creation of network tunnels to manage requests.

When connecting to the server, a program needs to implement the appropriate library for the programming language they are using. In python, there is a MinIO library that can be used to create an endpoint for the server. This command would look like:

Client =Minio(Endpoint, access_key, secret_key, use_https)

In the above example, the object needs 4 parameters, the endpoint, which is the hostname of the server, in our case that would be sp114web.loclx.io, the access key or username of the user trying to connect, the secret key or password of the user, and the https flag. If the user has valid credentials in the system, then they can access items within the object store using commands such as "put" or "get" to add or retrieve data from the server.

4.2 Setup and Deployment

The first part of setting up the data lake is starting up the instance of our object store. By configuring the environmental variables beforehand, the only thing needed to start the server is running the minio application in a command line window while pointing to the folder of our implementation. Once the object store is running, then the data from Carfax can be transferred from a computer onto the object store.

After transferring all the raw data into the appropriate zone in the object store, our other applications for cleaning the data and creating curated lists can be run. Once complete, these can be pulled from the minio store and read using software that can sort through the parquet files.

5.0 Test Plan

5.1 Objectives

Testing in this project focuses on the validation of user credentials to access the object store website, ensuring that users can access the data. Once it has been verified that user credentials can login to the store, making sure the object store is accessible outside of the local network is another key focus of test validation.

The data lake implementation features a heavy emphasis on the collection and organization of data for business analytics. As a result, ensuring that the data is properly transformed at each level of the object store is crucial in supporting a functional database.

5.2 Scope

There are several features that will be evaluated over the course of the creation of this project. These include:

- Object Store Connectivity
- Object Store Storage
- Scraper API Access
- Raw Data Formatting
- Data Querying
- User Credentials for Object Store
- Policy Access in Object Store
- Data Transferring from User to Object Store
- Database Schema for Analyzing Car Listings

5.3 Test Cases

5.3.1 Object Store Credentials

- Description: This test case requires the tester to enter the hostname of the object store into an internet browser, then enter a valid user credential set on the login screen. Once logged in, the tester should try to create a new user credential.
- Expected Result: The tester is redirected to the object store dashboard. The account should be allowed to create new users.

5.3.2 Network Access

- Description: The object store is connected to by a computer or mobile device on a separate wireless network than the one the object store is hosted on, then the tester tries to login using a valid credential.
- Expected Result: The tester is redirected to the object store dashboard.

5.3.3 Data Transfer

- Description: The scraper program will pull data from a specified location on the host computer, then send the raw JSON file to the proper bucket in the object store
- Expected Result: A JSON file holding car listings will appear within the “lakeraw” bucket of the object store.

5.3.4 Data Transformation

- Description: The raw car listings from the earlier file are accessed by a program that runs the listings through Apache Parquet to organize them into tables, then writes the tabled listings into a new bucket.
- Expected Result: The Parquet file will appear in a new bucket labeled “lakesilver” in the object store.

5.3.5 Data Querying

- Description: The tables from the “lakesilver” bucket are accessed by a querying program, which collects all listings that fulfill the necessary criteria for analysis. Once complete, these listings are copied and grouped into a new table, then sent to the object store in the final bucket layer.
- Expected Result: A new data table file is added to the “lakegold” bucket in the object store, with much fewer listings than the files in the “lakesilver” bucket, and with each listing having similar characteristics.

5.4 Procedures

5.4.1 Sign In

- Step 1: Enter server hostname.
- Step 2: Enter TestUser credentials.
- Step 3: Navigate to the Access tab.
- Step 4: Click Add User button.

- Step 5: Enter DataLakeReader and TestRead for Username and Password
- Step 6: Give the user read-only privileges.
- Step:7: Save the user settings
- Step 8: Logout of the TestUser account and login to the DataLakeReader account.
- Expected: TestUser can create new accounts, and users with limited privileges, like DataLakeReader, can only access specific policies, like reading objects from the buckets.

5.4.2 Network Access

- Step 1: Open a computer or mobile device connected to a different wireless network than the object store.
- Step 2: Enter the server hostname.
- Step 3: Enter the TestUser credentials.
- Expected: User is redirected to the object store dashboard.

5.4.3 Data Transfer

- Step 1: Create a new JSON file holding some information to store, like car listings from a website.
- Step 2: Run the file uploading python program with the JSON as data.
- Step 3:Wait for the program to finish running.
- Step 4:Log in to the object store.
- Step 5: Navigate to the “lakeraw” bucket.
- Expected: The JSON file should be found within the “lakeraw” bucket.

5.4.4 Data Transformation

- Step 1:Run the silver layer program with access to the “lakeraw” bucket.
- Step 2: Wait for program completion.
- Step 3:Log in to the object store.
- Step 4: Navigate to the “lakesilver” bucket.
- Expected: The JSON file should become a Parquet file and now be in the “lakesilver” bucket.

5.4.5 Data Querying

- Step 1: Run the gold layer program with access to the “lakesilver” bucket.
- Step 2: Wait for the program to be completed.
- Step 3: Log in to the object store.
- Step 4: Navigate to the “lakegold” bucket.
- Expected: A new, smaller parquet file should be found within the “lakegold” bucket.

5.6 Environment

The testing environment for this project requires the use of at least two computers on different networks. This will ensure that the network service portion of the data lake is functioning as intended. One computer should be running the MinIO object store database for holding data, while the other computer should have access to both an internet browser, and a python program that tries to connect to the server API of the object store.

Additionally, the computer hosting the object store should have a reverse proxy enabled for the server's local hostname, and an external storage drive for hosting the test contents. The reverse proxy allows the server to be reachable on the wider Internet by other computers, like the second computer for testing. For testing purposes, the external storage drive creates an easier diagnostic process for checking server files to find errors and allows for the storage of as much test data as possible.

To confirm the structuring of raw data into tables and queries, external software for reading Parquet table files is needed.

5.7 Test Data

Test data breaks down into two major categories for the data lake: object store user accounts, and raw data sourced from websites across the Internet. For the first category, our object store server has multiple test accounts for the purpose of validation of user privileges and credentials. The first account is the TestUser account, which has the credentials of an administrator of the object store: being allowed to read and write to buckets, create new users, set policies, and run diagnostics on the server.

Later accounts that are assessed are given less privileges in the object store, to assess the functionality of policy limitations, and whether these accounts can still function at their assigned level. Users like DataLakeReader and DataLakeWriter are given only the privilege to access buckets to read data or write in new data, respectively. Once testing is completed, these sample accounts are disabled, to prevent any security risk.

For the test data used in the second category, our test data was small samples from the larger batch of car listings that are stored on the server. After pulling some listings from Carfax, these were transferred across the Internet to the object store. Then, one or two of the JSON files sent over were run through the silver layer Parquet transformation program to assess if the data transformation is working as expected. Once the data is tabled, it is run through the gold layer program to ensure that our project is organizing data according to the specified schema.

This test data helps to confirm several aspects of our project, from the functionality of the network accessibility to the execution of data treatment and transformation into tables.

5.8 Software Test Report

Requirement	Pass/Fail	Severity
User Login	Pass	Critical
Network Access	Pass	Minor
Data Transfer	Pass	Critical
Data Transformation	Pass	Moderate
Data Querying	Pass	Moderate

6.0 Version Control

Our main way of implementing version control for this project was using GitHub. This allowed us to branch off the project to add each new program or change existing applications without affecting the main branch. As a second form of version control, we regularly back up the server files of the object store on a separate drive in case an issue pops up that prevents the server from being accessed.

Additionally, for the data stored within the object store, MinIO supports the versioning of objects in the server. This means that anytime an object is accessed and changed, the earlier version is kept with an older id while the server will point to the new id. This allows for correct auditing of data access and alteration to keep the integrity of the data intact.

7.0 Conclusion

In summary, this project is an application that pulls in data from websites such as Carfax, then uses a raw data storage server to hold the data. Afterwards, layered architecture is applied to the data to clean the datasets and then query to create tables of data that display the price changes and depreciation curve of car listings.

Once the data is fully cleaned and queried, the final tables of data can be opened by analysts to check the ongoing market trends for a specific business field, in this case car sales. As shown in the report, this way of analyzing data is ideal for businesses or users that may want to check the trends of multiple different fields of interest.

8.0 Appendix

8.1 Project Time Sheet

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1	**Project Name:**	Data Lake Implementation																			
2	**Report Date:**	4/25/2026																			
3						*January*				*February*				*March*				April			
4	**Phase**	**Tasks**	**Complete%**	**Current Status Memo**	**Assigned To**	**W1**	**W2**	**W3**	**W4**	**W1**	**W2**	**W3**	**W4**	**W1**	**W2**	**W3**	**W4**	**W1**	**W2**	**W3**	**W4**
11		Explore datasets and plan aggre	100%		Caleb					3											
12		Pull raw data from Online	100%		Bryce					10	10	4									
13		Design Frontend program	100%		Caleb						4	10	3								
14		Develop and test working proto	100%		Bryce							10	10								
15	Development	Review prototype design	100%		Caleb.Bryce								2								
16		Rework requirements	100%		Caleb								3								
17		Create development document	100%		Caleb									2							
18		Test product	100%		Bryce									5	10						
19	Presentation	Presentation preparation	100%		Caleb, Bryce										2	4					
20		Website preparation	100%		Bryce										2	4	4				
21		Present project in class	100%		Caleb, Bryce											1	1	1			
22	Final report	Update requirements and proto	100%		Caleb												10	10	10		
23		Draft Final report	100%		Caleb, Bryce														1	2	3
24		Final report submission to	0%																		2
25																					
26				**Total work hours**	**156**	**0**	**0**	**0**	**13**	**13**	**14**	**24**	**18**	**7**	**14**	**9**	**15**	**11**	**11**	**2**	**5**
27																					
28																					
29																					
30	*Legend*																				
31	Planned																				
32	Delayed																				
33	Number	Work: man hours																			

9.0 References

AIStor Object Store Documentation. (n.d.). *Operations*. AIStor Object Store Documentation. https://docs.min.io/enterprise/aistor-object-store/operations/

Morley, L. (2026, January 27). *Building a modern data lake using Open-source tools*. OpenMetal IaaS. https://openmetal.io/resources/blog/building-a-modern-data-lake-using-open-source-tools/

Overview. Parquet. (2025, November 12). https://parquet.apache.org/docs/overview/

What is a data lake? - introduction to Data Lakes and analytics - AWS. (n.d.). https://aws.amazon.com/what-is/data-lake/