CA400
Functional Specification

Project Title: Synche

Student 1 Name (ID): Tara Collins (15416228)
Student 2 Name (ID): Theo Coyne-Morgan (17338811)
Project Supervisor: Dr. Stephen Blott
Date Submitted: 4th December 2020

0. Table of Contents

1. Introduction

1.1 Overview

Synche will be a faster and more reliable alternative to cloud storage solutions such as Google Drive or Dropbox. It will be a file management system that provides a fast file uploading tool and easy file management on the user's server. Synche will orchestrate these fast file uploads by providing a tool that splits files and concurrently uploads the individual parts to the server, known as *multipart file transfer*. Using multiple connections between the client and the server Synche will achieve faster file upload speeds over the standard method of using a single connection.

Synche will allow users to transfer and share large files in an agile manner. Currently, the multipart uploading tools that are available are expensive, proprietary, and cater to large businesses. An existing example is the AWS S3 multipart upload tool. The bridge that allows individual users and smaller businesses to access fast multipart file transfer, data loss mitigation and private self-owned storage does not exist. Synche aims to be that bridge.

The universal benefits of using Synche are the following:

- **Faster data transfer**

  By using multipart uploading, higher data transfer throughput can be achieved for large files (>100 MB).

- **Data loss from connection failure mitigation**

  Files are uploaded in small parts so that, in the event of a connection failure, parts that have been uploaded previously are still on the server and can be used to resume the upload when the connection is reinstated.

- **Pausing and resuming uploads**

  Uploads can be paused and the non-transferred data segments can be uploaded at another time.

- **Complete first-party server ownership**

  Cloud storage such as OneDrive etc. may be a privacy or security concern for some people, and by owning your own server you can be in full control.

- **No proprietary third-party software**

The software is open-source, meaning there are no subscription fees, it is not tied to any company and users have full control over the software and their files.

- **Cheaper storage costs**

  Storage solutions such as AWS S3 can become very costly for huge amounts of data storage. By owning your own server you can dramatically cut down the storage costs.

The below table illustrates how much money a user may save by using Synch and renting a private server instead of using a service such as AWS S3.

Provider	**Storage Space**	**Total Monthly Cost**
AWS S3	8 Terabytes	155.76 EUR
Hetzner*	8 Terabytes	45.24 EUR
BuyVM*	8 Terabytes	45.47 EUR

* *with Synche*

1.2 Business Context

Synche will be a creator-sponsored open-source project.

1.3 Glossary

API	An application programming interface (API) is a set of functions and procedures allowing the creation of applications that access the features or data of an operating system, application, or other service.
AWS	Amazon Web Services (AWS) is a subsidiary of Amazon that provides cloud computing platforms and APIs, on a metered pay-as-you-go basis.
CLI	A command line interface (CLI) is a terminal program that handles input from a user via text based commands.
GUI	A graphical user interface (GUI) is a form of user interface that allows users to interact with an underlying computer system via visual indicator representations.
OS	An operating system (OS) is the software that manages and communicates with a computer's hardware.
S3	Simple Storage Service (S3) is a service offered by Amazon Web Services that provides object storage through a web service interface.
Metadata	A set of data that describes and gives information about other data. In relation to this project, 'metadata' will be used to describe the information pertaining to a file such as its on-disk size, location, hash, modification date, upload date and permissions.

2. General Description

2.1 Product / System Functions

Synche will be a file management system. Synche's core feature will be a multipart file uploading tool that splits files into parts and concurrently uploads the parts to a storage server.

The interface will allow the user to enter the file or files that they would like to upload and the location of the server that they would like to upload the files to. Before the upload process begins, the user will have to provide their authentication details.

Once the client has been authenticated, Synche will split the files that the user has listed and assign a hash to each file part. Hashing the file parts will allow Synche to reassemble them correctly on the server side, and avoid duplication. In the case where the connection drops while an upload is in progress, hashing will allow individual parts to be cached on the server side until the upload is reinitiated or resumed. This is unlike other file management services, such as *Google Drive* that do not cache file uploads. A connection failure using a currently existing cloud storage solution, the user must completely restart the upload process. For large files, this may be incredibly costly in terms of time. Synche aims to mitigate this potential time loss completely.

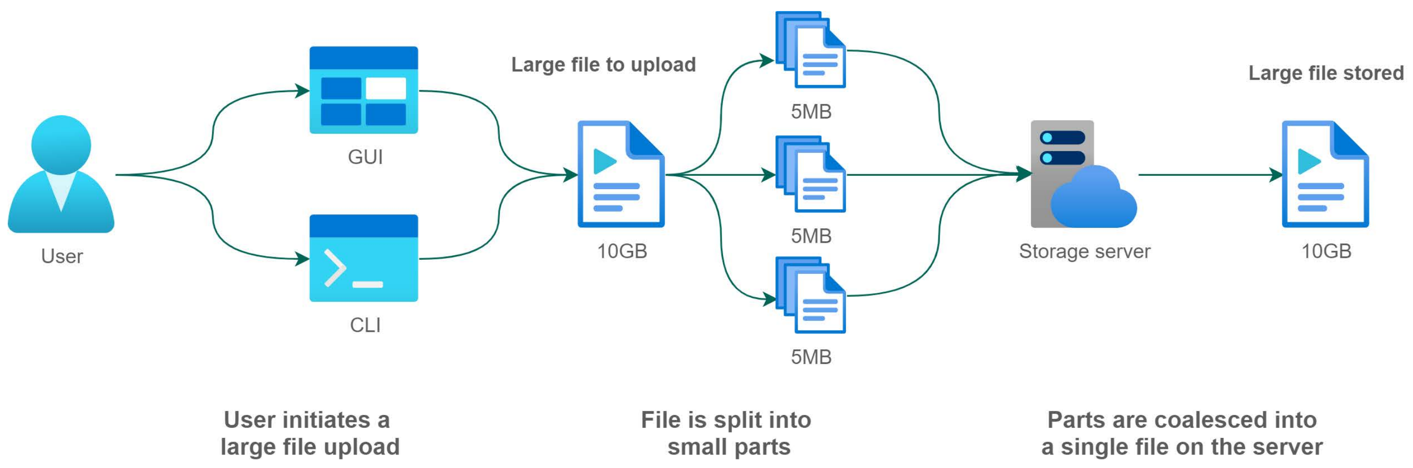
Once the files are divided into parts, Synche will begin concurrently transferring the file parts to the server. Once all the file parts for an individual file are received on the remote server, Synche will coalesce the file parts into a composite, whole file that exactly resembles the original local file. The user will see the full files on both the client side and the server side, they will not need to be concerned with the underlying file part operations.

Synche will be a full file storage and management system and thus it will allow the user to download files, list files, move files, and delete files on their server. Serving file data back to the user from the storage server will be private, over an authenticated connection.

2.2 User Characteristics and Objectives

Our expected users are individuals that have basic knowledge and understanding of servers and server software and are comfortable using a CLI. They will not need to understand the underlying operations of the multipart file uploading process, they will however, need to have a basic knowledge of Linux based servers in order to deploy the Synche server software. We expect that users will be individuals that work or have worked with tech in some form or have tech related hobbies. Proposed users may be system administrators, web developers, or development and operations engineers. We expect that users would be interested in Synche as a secure and private file transferring tool because it does not require the use of proprietary third party applications or servers.

Small to medium businesses (SMB) are also in the Synche target audience. For example, a video production company could benefit from the faster transfer times to get footage to editors - see section 2.2.1. An SMB that deals with storing and/or analysing data would also benefit from Synche, e.g. a medical research company - see section 2.2.1. These users would be accustomed to frequently handling, transferring, and sharing large files such as video files or datasets.


User
GUI
CLI
Large file to upload
10GB
5MB
5MB
5MB
Storage server
Large file stored
10GB
User initiates a large file upload
File is split into small parts
Parts are coalesced into a single file on the server

2.2.1 Example User Usage Scenarios

Scenario 1:

User details	Usage Scenario - For a digital agency
Background/ Occupation: Videographer **Technical skills:** Moderate	This user is an individual that works for a videography agency. The user captures high-definition video footage daily, which needs to be given to the editing team as quickly as possible. The amount of footage he needs to deliver quickly, exceeds 10 Gigabytes in size. Both the videographer and the editing team want the footage stored in a secure location that they can easily access. By using Synche, the videographer can quickly upload their footage onto their private business server and avoid having to deliver a physical copy of the video footage to the editors. This would save both parties a large amount of time and money as they will not need to pay for a cloud service to share their files. If the agency decided to use a server located in their offices, Synche will save time twofold, as the editing team would not have to download the data from an external cloud storage provider.

Scenario 2:

User details	Usage Scenario - As a cloud alternative
Background/ Occupation: Independent user **Technical skills:** Low	An independent user has sensitive or private information that they want to upload to their own private server. Their main concern is the security of their information. This user has private documents that they want to back-up on a server that they already own. They do not want to use a service such as Google Drive as they currently pay for their own server and want to ensure that their information is truly private. By using the proposed software, the user has full ownership over their data. If the user lives in a location with a medium-slow internet connection, they may transfer their data quicker and would not need to be as concerned about loss of data in the event of a connection failure.

2.3 Operational Scenarios

Below are the primary uses cases for Synche.

Use Case 1	**User creates an account**
Description	User creates an account on their server.
Success End Condition	The user details are stored in the server database
Failed End Condition	The user is unable to create an account
Actors	User, server provider
Steps and Actions	**1.** User opens/starts the Synche user interface **2.** User enters registration details **3.** Synche receives the account registration request. **4.** Synche verifies and stores user registration details **5.** User enters login details **6.** Synche receives the login request **7.** Synche verifies user login details **8.** Synche displays an information message informing the user that an account has been created successfully
Variations and Branching Actions	**4.** User registration details cannot be verified. User is prompted to re-enter registration details. **7.** Synche cannot verify the user login details. The user is prompted to re-enter their login details.

Use Case 2	**User uploads a file**
Description	User initiates a file upload request, expects a file to be transferred to a specified server
Success End Condition	Full file data is on the specified server
Failed End Condition	Full file data is not on the specified server
Actors	User, server supplier
Steps and Actions	**1.** User specifies a file and a server location using the CLI or user interface **2.** Synche requests authentication details **3.** User provides authentication details **4.** Synche verifies authentication details **5.** Synche splits the file into multiple parts while hashing each part **6.** Synche uploads each file part concurrently to the specified server **7.** Synche reassembles the file on the specified server **8.** Synche displays file upload confirmation for the user
Variations and Branching Actions	**4.** Synche is unable to verify the authentication details and denies the user's request to upload the file **6.** Synche is unable to upload the file parts. Synche displays file upload failure status to the user

Use Case 3	**User deletes a file**
Description	User sends a request to delete a file
Success End Condition	The requested file is deleted
Failed End Condition	The requested file is not deleted
Actors	User, server provider
Steps and Actions	**1.** User specifies a file that they want to be deleted **2.** Synche receives the request **3.** Synche verifies that the user has the correct permissions to delete the file **4.** Synche deletes the file and its associated stored data from the server **5.** Synche displays the file status to the user
Variations and Branching Actions	**3.** Synche cannot verify the user's permissions and the request is denied

Use Case 4	**User moves a file**
Description	User sends a request to move a file to a different location on the server
Success End Condition	The requested file is moved
Failed End Condition	The requested file is not moved
Actors	User, server provider
Steps and Actions	**1.** User specifies a file that they want to be moved and the location to move it to **2.** Synche receives the request **3.** Synche verifies that the user has the correct permissions to move the file in both locations **4.** Synche moves the file on the server, and updates the database records to reflect the changes **5.** Synche displays the file status to the user
Variations and Branching Actions	**3.** Synche cannot verify the user's permissions and the request is denied

Use Case 5	**User downloads a file**
Description	User sends a request to download a file stored on the server
Success End Condition	The requested file data is provided and the user receives the correct file
Failed End Condition	The requested file is not found, incorrect file is provided or Synche fails to authenticate the request
Actors	User, server provider
Steps and Actions	**1.** User specifies a file that they want to download **2.** Synche receives the request **3.** Synche verifies that the user has the correct permissions to download the file **4.** Synche prepares the file data and sends it back to the client **5.** Synche displays the file download progress to the user
Variations and Branching Actions	**3.** Synche cannot verify the user's permissions and the request is denied **4.** The server cannot find the file and displays a message to the user

Use Case 6	**User requests file information**
Description	User requests information about files stored on the server.
Success End Condition	The user is returned the requested information
Failed End Condition	The user is returned incorrect information, or no information.
Actors	User, server provider
Steps and Actions	**1.** User opens/starts the Synche user interface **2.** User sends an info request using the client **3.** Synche receives the request **4.** Synche authenticates the user and checks they have correct permissions to view the file(s) **5.** Synche retrieves the requested data, e.g. file list or file metadata from the SQL database **6.** Synche responds to the client with the requested data **7.** Synche displays the requested data to the user
Variations and Branching Actions	**4.** Synche cannot authenticate the and the request is denied

Use Case 7	**User views their statistical dashboard**
Description	Information regarding the user's file upload history and metrics are displayed to the user
Success End Condition	The user views their statistical summary
Failed End Condition	The statistical summary for the user is not retrieved and the user does not view it
Actors	User, server provider
Steps and Actions	**1.** User opens/starts the Synche user interface **2.** User selects the option to view their statistical summary **3.** Synche receives the request **4.** Synche retrieves the user's data **5.** Synche analyses the user's data **6.** Synche displays the user's statistical summary to the user
Variations and Branching Actions	**4.** Synche cannot retrieve the user's data and displays the failure to the user

2.4 Constraints

2.4.1 Technical Constraints

Speed - The primary aspect on which Synche will be judged is the speed at which it transfers files to the remote server. Synche aims to provide a faster uploading tool than current major cloud storage providers. The client will need to split files and perform multipart file uploading as fast as possible to compete with existing cloud storage providers.

Security - User authentication needs to be managed securely to ensure user data is protected. Data transfer such as transferring file parts to the server needs to be secure so that intercepted parts do not reveal user data.

Language - Both of the team members are relatively new to Golang. They both have a few months experience writing in Golang, but do not have experience in writing in Golang on a large scale. It is incredibly important that both team members invest adequate time in learning Golang before beginning the backend development which will use Golang.

Cost - Synche will require one remote storage server to test and run the server software. As the primary function of Synche is to upload large files quickly, the server will need to have an adequately large amount of storage space and a fast network connection. Since we do not need more than one server, cost should not be too much of a constraint. However, time should be spent to find the most cost-effective server provider for our project so that it does not become an issue.

2.4.2 Time Constraints

Deadline - The project deadline is in May 2021. With these time constraints, we aim to get a fully functional version of Synche operating. This is why we have listed additional features that we will implement if the core features are completed ahead of the deadline. The core features that we have listed provide the most important functionality. Ideally, we would like to implement all the features that we have listed but the time restraints may inhibit the additional features being developed.

3. Functional Requirements

3.1 Minimum Functional Requirements

Rank / ID	**1**
Description	The client must be able to connect to the Synche server API.
Criticality	If the client cannot connect to the server API, users will not be able to interact with the system.
Technical Issues	Users must be able to create a Synche storage server without excessive prior technical knowledge.
Dependencies	N/A

Rank / ID	**2**
Description	The server must listen and respond to the client requests.
Criticality	In order for any actions to be performed on the server, the server application must respond to user requests sent by the client.
Technical Issues	Ensuring the server software runs continuously and that the server can receive incoming connections.
Dependencies	N/A

Rank / ID	**3**
Description	Securely upload files using multipart file transfer.
Criticality	This is a primary function of the Synche system, and is what distinguishes it from other cloud storage solutions.
Technical Issues	If data is not split into the optimal number of segments, or not uploaded using the right amount of connections, it may result in the file being transferred slower than the traditional single connection method.
Dependencies	1, 2, 5, 6

Rank / ID	**4**
Description	Synche must store and serve uploaded data as composite, whole files.
Criticality	The server must allow users to download and access their previously uploaded files *exactly* as they were prior to being uploaded. If users cannot access their stored files, the system is useless.
Technical Issues	Re-assembling files must not take an excessive amount of time and should not affect the performance of the system.
Dependencies	1, 2, 3, 5, 6

Rank / ID	**5**
Description	Add a server to the client; configure a new remote server and store the connection details on the client.
Criticality	In order for the client to connect a server, it must first allow the user to configure its details such as it's IP address in the client.
Technical Issues	We must make sure that clients cannot connect to other servers they do not own.
Dependencies	1, 2

Rank / ID	**6**
Description	Register a new user account.
Criticality	File permissions, server access rights and stored information rely on accounts to authenticate users.
Technical Issues	User data must be stored and accessed securely to ensure there are no security holes in the application.
Dependencies	1, 2

Rank / ID	**7**
Description	Authenticate a user.
Criticality	Users must be authenticated for the system to apply file and action restrictions.
Technical Issues	The login system must be immune to brute-force attacks.
Dependencies	1, 2, 5, 6

Rank / ID	**8**
Description	Display information about stored files on the remote server with the client.
Criticality	A direct representation of the files, and file structure on the remote server is essential for users to interact with the server.
Technical Issues	Information should be transmitted and stored efficiently, without lacking any crucial information.
Dependencies	1, 2, 5, 6, 7

Rank / ID	**9**
Description	Remove a file from the remote storage server via the client. Must remove all the stored file parts and relevant stored information.
Criticality	Without file removal functionality, users would run out of space or be forced to manually remove file data which could be detrimental to file management within Synche.
Technical Issues	It is extremely important that Synche does not unintentionally lose track of file data, as this would result in lost storage space.
Dependencies	1, 2, 6, 7

Rank / ID	**10**
Description	Notify the user of a completed upload.
Criticality	Users should be notified both when the file transfer completes, so they are not waiting indefinitely.
Technical Issues	N/A
Dependencies	1, 2, 3, 6, 7

Rank / ID	**11**
Description	Move a file on the server via the client.
Criticality	Managing the location or structure of files on the remote server is not critical, but it is important so that users have full control over their files.
Technical Issues	Adding support for multiple drives and creating pseudo-directories are two hurdles that will need to be overcome.
Dependencies	1, 2, 3, 6, 7, 8

Rank / ID	**12**
Description	Update and delete user account details via the client.
Criticality	Managing users and account details is necessary so users and admins can change passwords and remove users from the database.
Technical Issues	Differentiating regular user accounts from administrator accounts.
Dependencies	1, 2, 6, 7

Rank / ID	**13**
Description	Allow multiple user accounts on the same server.
Criticality	Within a business context, many users may be dissuaded from using Synche if they are unable to have full control over what users access what files on their server.
Technical Issues	Differentiating regular user accounts from administrator accounts.
Dependencies	1, 2, 5, 6, 7, 12

Rank / ID	**14**
Description	Allow the admin account to alter other account details
Criticality	This is not critical but would likely be useful in scenarios that there are multiple long-term users on the same server.
Technical Issues	Incorporating account hierarchy may result in account access rights issues.
Dependencies	1, 2, 5, 6, 7

3.2 Additional Functional Requirements

Rank / ID	**15**
Description	Analyse the user file transfer history and display a statistical summary.
Criticality	A statistical summary is not critical to the functionality of the system, however it would provide a useful insight to users
Technical Issues	Gathering and storing the relevant user data.
Dependencies	1, 2, 7, 8

Rank / ID	**16**
Description	OAuth2 user authentication integration with Google accounts
Criticality	This is not critical.
Technical Issues	The user must be correctly redirected to a web browser log-in to generate the access token JSON.
Dependencies	6

Rank / ID	**17**
Description	Cross-platform GUI desktop application.
Criticality	While this would expand our user base, it is not critical.
Technical Issues	May be difficult to test the GUI on multiple operating systems.
Dependencies	N/A

Rank / ID	**18**
Description	Automated server-deployment with Docker
Criticality	Making the server deployment painless and fast for users with any level of technical experience would expand the user base.
Technical Issues	Testing containerized server software
Dependencies	N/A

Rank / ID	**19**
Description	Display file transfer statistics on a Grafana dashboard.
Criticality	This would provide interesting statistics to the user but it is not required.
Technical Issues	Integrating our stored data with Grafana.
Dependencies	1, 2, 5, 6, 7

3.3 Non-functional Requirements:

Faster upload speed

It is essential that Synche is capable of providing faster upload speeds than the traditional single-connection upload. If too many connections are opened between the client and the server, the network may become overloaded. If this happens and all available bandwidth is consumed, network requests will be sent slower, and subsequently, the file upload speed will decrease. This could cause the multipart file upload rate to be less than a single connection standard file upload. A sub-optimal number of concurrent connections could cause performance issues on the server side as well, as the server will need to process many incoming requests at once. We need to ensure that we avoid this, as fast upload speeds are the core improvement Synche has over existing cloud storage solutions.

Maximum speed file download

As outlined above, fast file transfer speeds are an important feature of Synche. We need to ensure that our service provides the fastest download speed possible for the user.

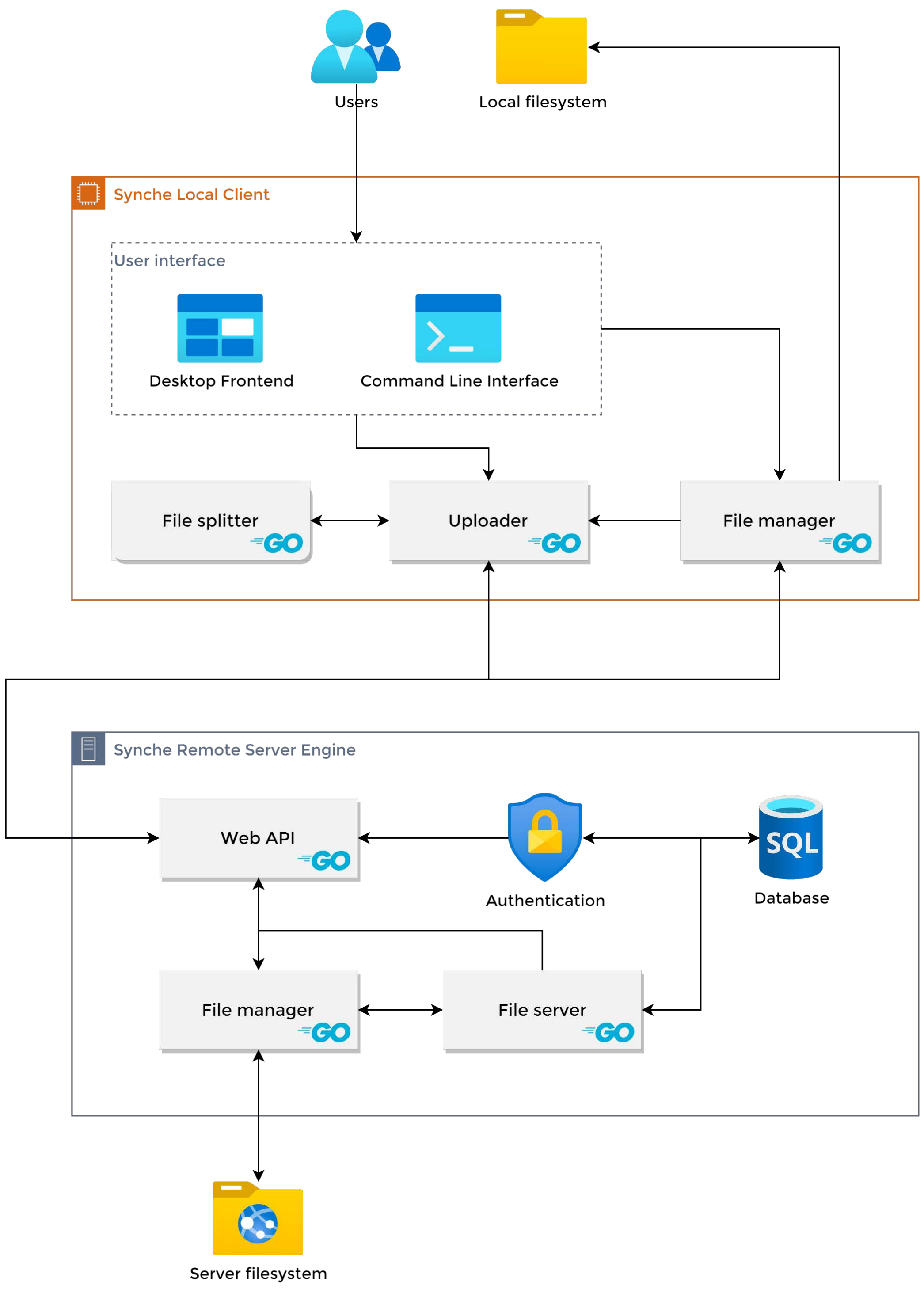
Efficient file storage

We must ensure that files are stored efficiently and unused file parts are removed from the server. We expect that the users will be uploading large files so moving the files and storage fragmentation will need to be avoided.

Security

As Synche will be handling and overseeing the storage of personal user data, privacy and security are critical. To ensure data privacy, files will be transferred over an SSL connection (HTTPS). Stored files will be kept in a web-inaccessible location and only provided when the user requests to download them. Downloading files must also be secure and private, and to achieve this, file transfers to the client will only be performed whilst the user is authenticated. Data for user accounts will be stored in an SQL database on the remote server. The industry standard password encryption and user data security will be taken to ensure the highest level of protection against attacks.

4. System Architecture



The above simplified overview of the Synche system illustrates the components and internal parts of the Synche cloud storage solution that provides a multipart file upload tool and file storage server.

Synche will consist of two main components; a user client and a server storage system. The client's core purpose is to provide an interface for the end-user which will communicate with the server API, upload, list and delete files on the remote server.

4.1 Client

The core feature of the client is to efficiently split files into multiple parts and transfer the parts to the server in parallel. The client will also allow users to authenticate with the server API, initiate file uploads and list, delete, download and manage files on the remote server.

- **Command line interface**

  A feature-rich command line interface (CLI) written in Go. The CLI will be the default method of interacting with the server API.

- **Desktop frontend**

  In addition to the CLI, A cross-platform graphical user-interface (GUI) will be provided using go-astilectron. Go-astilectron is a Golang API wrapper for Electron which allows us to create a cross-platform user-interface with HTML/CSS/JS.

4.2 Server

The server stores files, authenticates users and provides an API for interacting with the Synche server software.

- **Web server**

  NGINX will be used to reverse proxy the Go web application. The required set-up scripts for the web server will be containerized with the Synche server software for easy deployment.

- **Web API**

  A web API written in Go will allow the client to interact with the server, authenticate, upload, list and retrieve files.

- **Authentication**

  Users must authenticate themselves via the web API before they can transfer local files and manage files on the server.

- **Database**

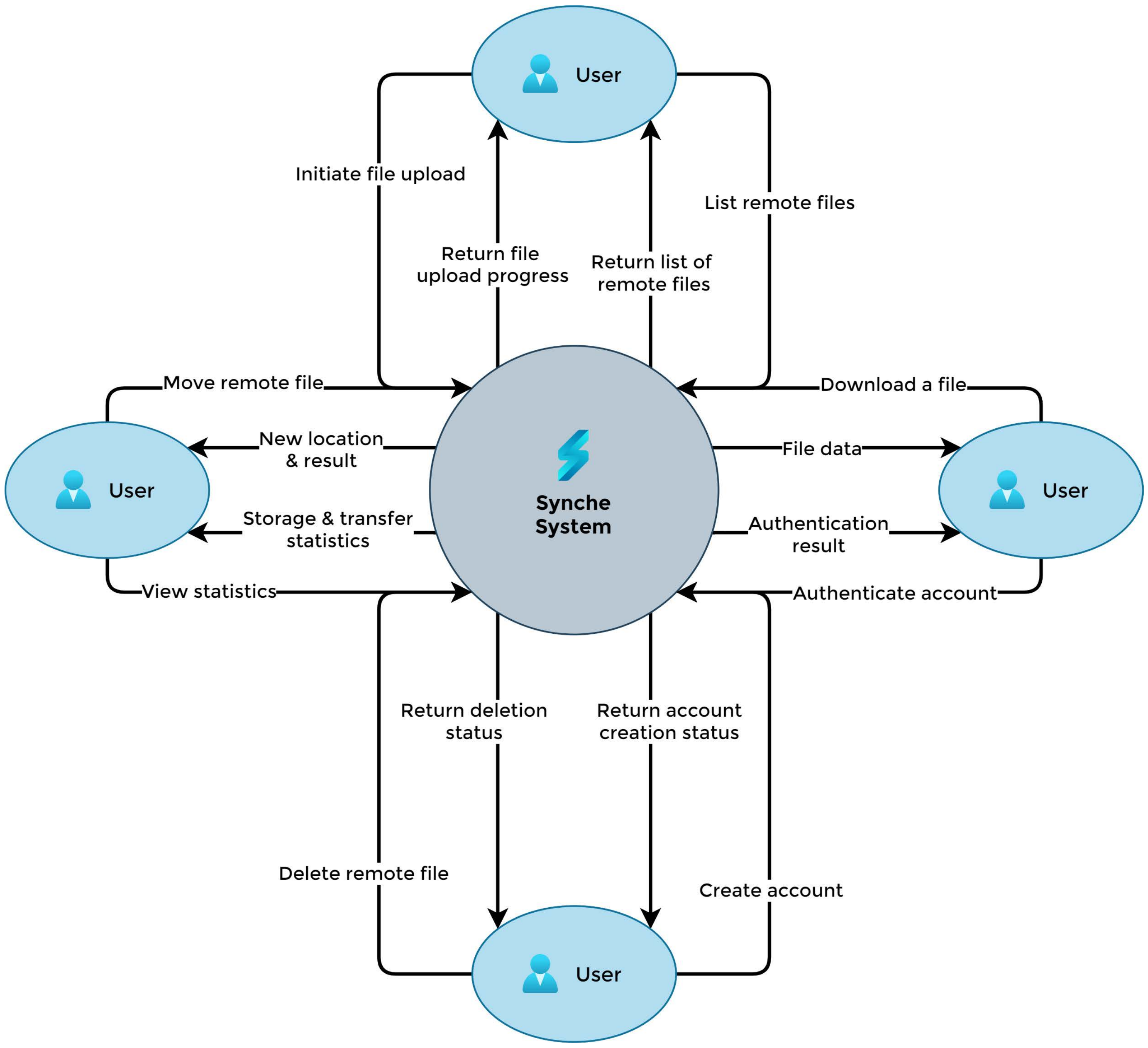
  An SQL database to persist file and user information.

- **Server file system**

  A remote server with allocated storage space.

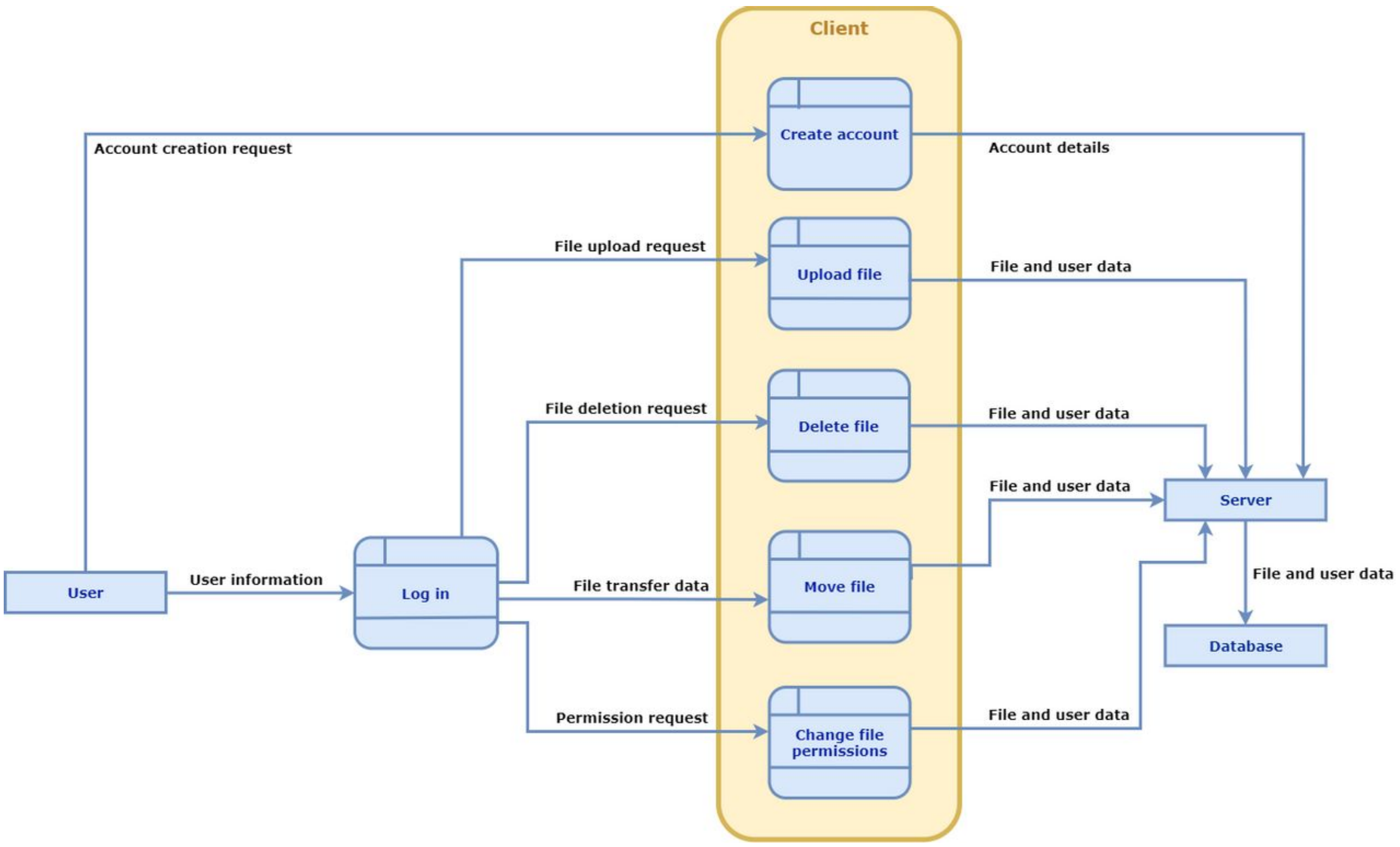
5. High-Level Design

5.1 Context Diagram



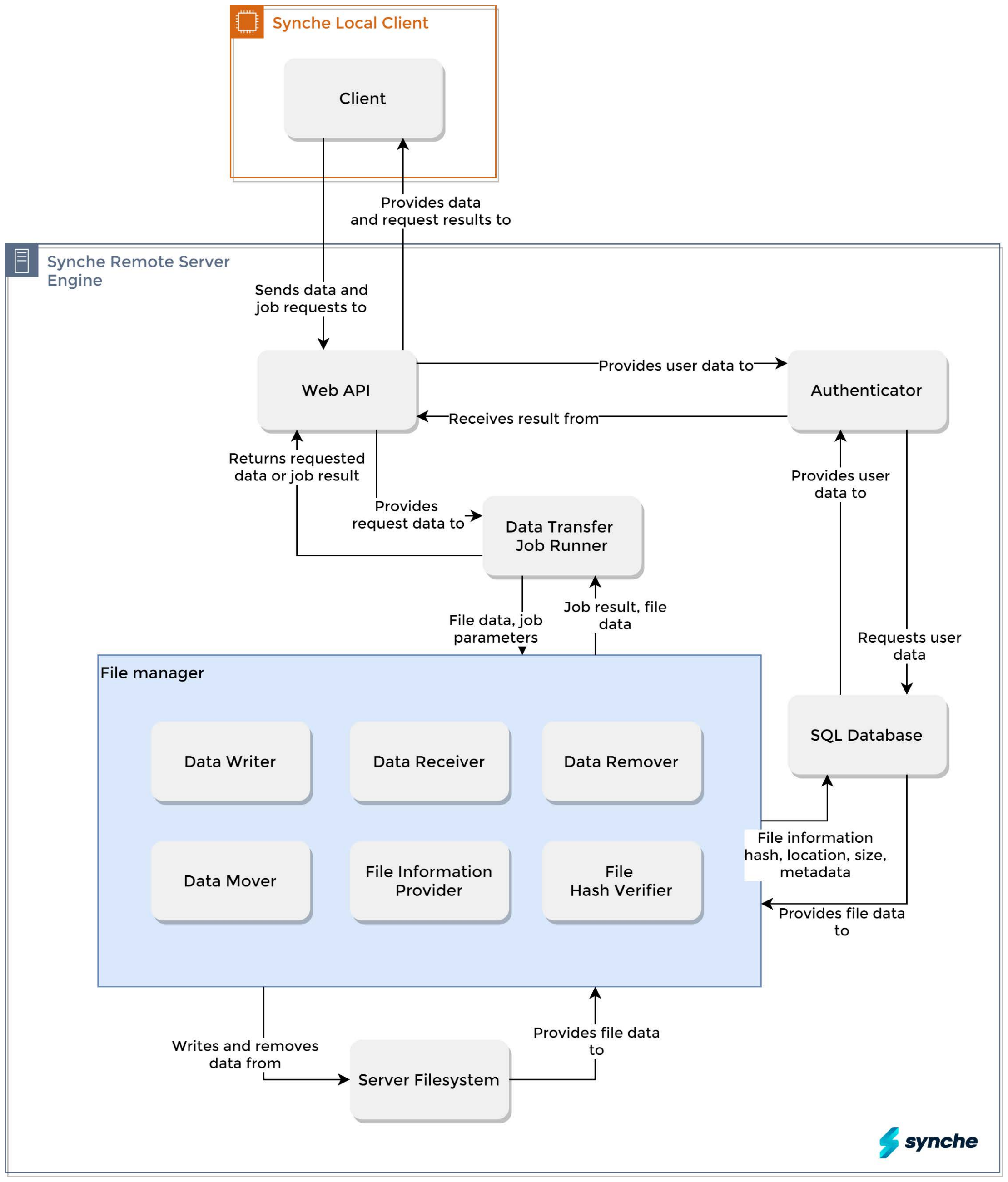
This context diagram illustrates the actions users can perform with the Synche system, and the associated response they will receive for each action.

5.2 User Data Flow Diagram



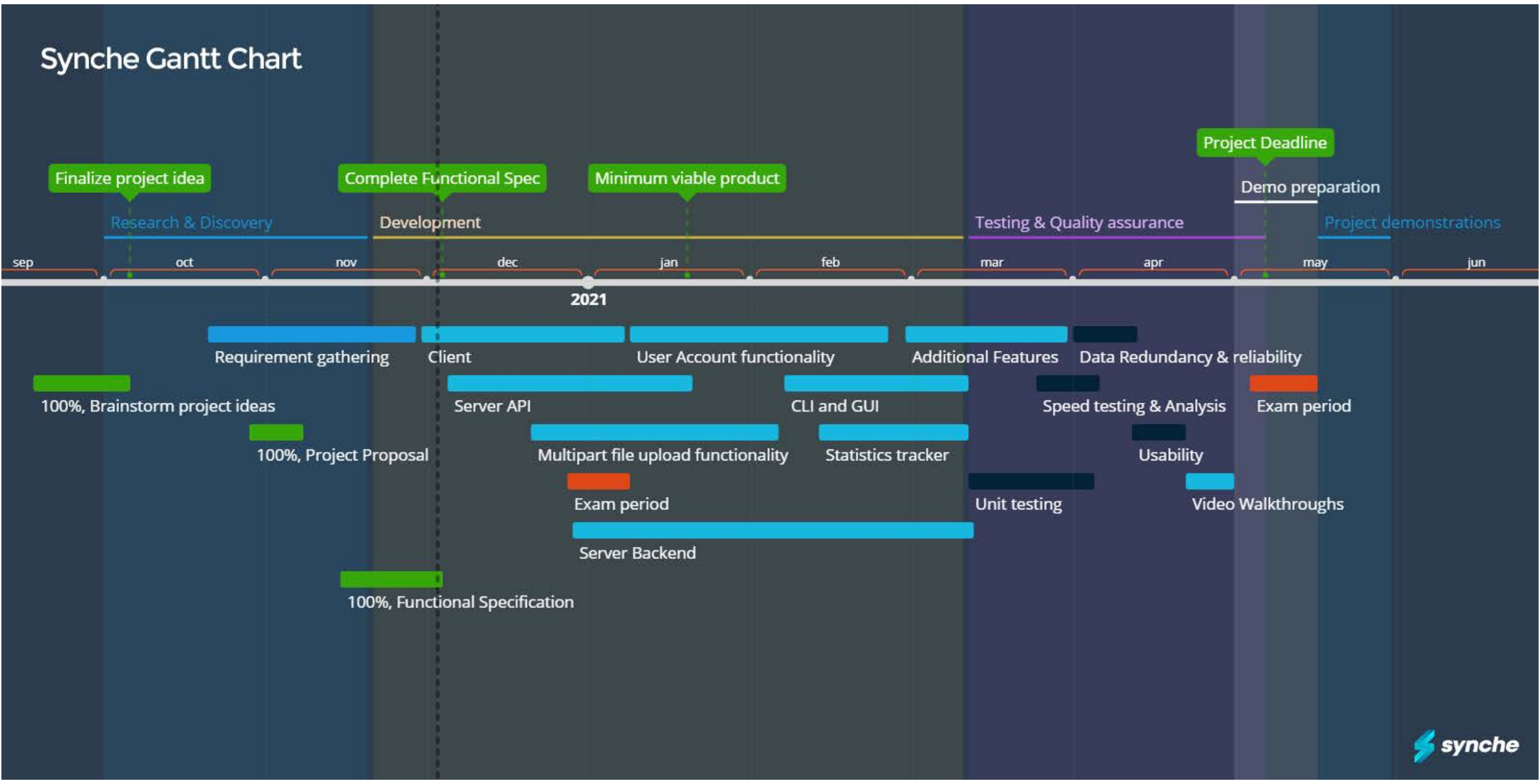
The user data diagram shown above shows the transfer of data from a user's perspective. Users will have one primary connection to the Synche storage system and this will be done via the local client.

5.3 Server Data Flow Diagram



In the diagram above, the flow of data from components on the server end is shown. The client will handle user requests e.g. uploading a file, and will perform the segmentation of the file, and computing a hash for the file parts before sending the data and user credentials to the server. The server must handle these requests by first authenticating the user before executing any server-sided actions, or receive any file data.

6. Preliminary Schedule



Shown above is the preliminary Gantt chart schedule for the development project. We aim to have a working product in February, and to spend the additional time debugging, testing and adding the additional features listed above in the document.

7. Appendices

- **Information on Golang:** (https://golang.org/)
- **Go-astilectron:** (https://github.com/asticode/go-astilectron)
- **Electron:** (https://github.com/electron/electron)
- **Information on Grafana:** (https://grafana.com/grafana/)
- **Sequential vs single transfer methods:**
  (https://www.aurigma.com/docs/iu/UploadingFilesSequentiallyAndConcurrently.htm)
- **AWS S3 multipart file upload:**
  (https://aws.amazon.com/blogs/aws/amazon-s3-multipart-upload/)