CA400
User Manual

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: 7th May 2021

0. Table of Contents

1. Overview

Synche is a file management system that features a multipart file transfer tool for fast uploading.

Synche provides quick and reliable uploads to those that wish to store data on their own private server. It facilitates multiple concurrent users by allowing users to create accounts on a server. Once the Synche client and server are installed, Synche tracks files belonging to a user and allows them to create, delete, and manage both files and directories.

The Synche interface can be accessed through either the CLI or the Synche GUI.

2. Dependencies

2.2 Languages and Tools

- **Go**
  - **Version:** v1.16.2
  - **Usage:** Synche client, Synche server, Synche SDK

- **GNU Make**
  - **Version:** v4.1
  - **Usage:** Building the Synche client and server

- **Docker**
  - **Version:** v19.03.14
  - **Usage:** Synche server

- **React**
    - **Version:** v17.0.2
    - **Usage:** Synche GUI

- **Node.js**
    - **Version:** v16.1.0
    - **Usage:** Synche GUI

- **Tailwind**
    - **Version:** v2.1.2
    - **Usage:** Synche GUI

2.3 Direct Dependencies

These are the direct dependencies of our application only. Some of these libraries may have other dependencies that are not listed below. Go OpenAPI libraries have not been included.

- **sqlmock**
    - **Version:** v1.5.0
    - **Description:** a mock library implementing SQL/driver. Its purpose is to simulate SQL driver behaviour in tests, without needing a real database connection.
    - **Source Code:** https://github.com/DATA-DOG/go-sqlmock
    - **Usage:** Unit and integration testing.

- **jwt-go**
    - **Version:** v3.2.0

- **Description:** A library that supports the parsing and verification, generation, and signing of JWTs.
- **Source Code:** https://github.com/dgrijalva/jwt-go
- **Usage:** Server-side authentication.

- ### structs
  - **Version:** v1.1.0
  - **Description:** A library that contains various utilities to work with Go structs.
  - **Source Code:** https://github.com/fatih/structs
  - **Usage:** Allows the user to input initialisation configuration.

- ### fsnotify
  - **Version:** v1.4.9
  - **Description:** A library that utilizes golang.org/x/sys rather than `syscall` from the standard library.
  - **Source Code:** https://github.com/fsnotify/fsnotify
  - **Usage:** Allows the user to input initialisation configuration.

- ### Go-MySQL-Driver
  - **Version:** v1.6.0
  - **Description:** A MySQL-Driver for Go's database/sql package
  - **Source Code:** https://github.com/go-sql-driver/mysql
  - **Usage:** Database management.

- ### imohash
  - **Version:** v1.0.0
  - **Description:** A library for fast, constant-time hashing.
  - **Source Code:** https://github.com/kalafut/imohash
  - **Usage:** Validating file integrity.

- **go-homedir**
  - **Version:** v1.0.0
  - **Description:** A library for detecting the user's home directory so that the library can be used in cross-compilation environments.
  - **Source Code:** https://github.com/mitchellh/go-homedir
  - **Usage:** Configuration.

- **go-cache**
  - **Version:** v2.1.0
  - **Description:** A library for an in-memory key:value store/cache similar to memcached that is suitable for applications running on a single machine.
  - **Source Code:** https://github.com/patrickmn/go-cache
  - **Usage:** Caching.

- **logrus**
  - **Version:** v1.8.1
  - **Description:** A structured logger that is compatible with the standard library logger.
  - **Source Code:** https://github.com/sirupsen/logrus
  - **Usage:** Clear and legible logging.

- **afero**
  - **Version:** v1.6.0
  - **Description:** A filesystem framework providing a simple, uniform and universal API interacting with any filesystem.
  - **Source Code:** https://github.com/spf13/afero
  - **Usage:** File management.

- **cobra**
    - **Version:** v1.1.3
    - **Description:** A library for creating powerful modern CLI applications.
    - **Source Code:** https://github.com/spf13/cobra
    - **Usage:** CLI.

- **pflag**
    - **Version:** v1.5.0
    - **Description:** A flag package for implementing POSIX/GNU-style --flags.
    - **Source Code:** https://github.com/spf13/pflag
    - **Usage:** CLI and binding server ports.

- **viper**
    - **Version:** v1.7.1
    - **Description:** A configuration solution designed to handle all types of configuration formats.
    - **Source Code:** https://github.com/spf13/viper
    - **Usage:** Configuration.

- **testify**
    - **Version:** v1.7.0
    - **Description:** A library that provides testing packages.
    - **Source Code:** https://github.com/stretchr/testify
    - **Usage:** Testing.

- **go-swagger**
    - **Version:** v0.27.0
    - **Description:** A tooling ecosystem for developing APIs with the OpenAPI Specification (OAS).

- **Source Code:** https://github.com/go-swagger/go-swagger
- **Usage:** Validating user passwords.

gorm

- **Version:** v1.21.8
- **Description:** An ORM library.
- **Source Code:** https://gorm.io/
- **Usage:** Database management.

3. Installation

3.1 Go

Go must be installed. If you are not using a Linux system, follow GoLang's guide to installation for the operating system that your machine uses.

Install the Go compiler and associated tools on Linux, run the command below. This will extract the archive you downloaded into /usr/local, creating a Go tree in /usr/local/go:

```
rm -rf /usr/local/go && tar -C /usr/local -xzf
go1.16.3.linux-amd64.tar.gz
```

Next, add /usr/local/go/bin to the PATH environment variable using the following command. You can do this by adding the following line to your /etc/profile (for a system-wide installation) or $HOME/.profile:

```
export PATH=$PATH:/usr/local/go/bin
```

Important note: Changes made to a profile file may not apply until the next time you log into your computer. If you wish to apply the changes immediately, you can run the shell commands directly.

3.2 GNU Make

Make is required to build the Go executable binaries. There are alternative ways to install Make if you are not using a Linux system, such as chocolatey for Windows OS. For more information on Make, refer to the GNU website.

To install Make on a Linux system, run the below commands:

```
apt update
apt install make
```

3.3 Node.js

Node.js is required for the GUI. If you are not using Linux, follow the Node.js installation guide for your operating system.

To install Node.js on Linux, run the following commands using sudo.

```
apt update
apt install nodejs
apt install npm
```

3.4 Tailwind

Tailwind is required for the GUI. If you are not using Linux, follow the Tailwind installation guide for your operating system.

Note: `npm` (installed in the step above) is required to install Tailwind.

To install Tailwind on Linux, run the following commands:

```
npm install -D tailwindcss@latest postcss@latest autoprefixer@latest
```

Another note: Tailwind does not automatically add vendor prefixes to the CSS it generates. It may be recommended to install an autoprefixer to handle this.

3.5 Docker

It is recommended to run the server in a Docker container. This is due to the likelihood of conflicts occuring with the database dependencies.

To install Docker, run the below commands:

```
apt update
apt install docker.io docker-compose -y
```

If you prefer not to use Docker, you will need to install MariaDB and set up a user. These user details will need to be entered into the Synche config, the details of which will be explained in the configuration section. You can do so by running the following commands:

```
apt update
apt install mariadb-server
mysql -u root -p
CREATE USER 'user'@localhost IDENTIFIED BY 'password';
GRANT ALL PRIVILEGES ON 'yourDB'.* TO 'user'@localhost;
```

Replacing 'yourDB' with the name of the database you wish to use for Synche.

4. Setup

Firstly, you must clone the repository via HTTPS by running the following command:

```
git clone
https://gitlab.computing.dcu.ie/collint9/2021-ca400-collint9-coynemt2
.git
```

Note: you must clone the repo into the directory that you set up your PATH environment variable to point at. Assuming that you installed Go with the default path variables, you must be in the directory: `/home/user/go`

Next, navigate to the directory containing the source code with the below command:

```
cd 2021-ca400-collint9-coynemt2/src
```

From this directory, you can build the Synche dependencies using the `Makefile`. To do so, run the following command from the `src` directory:

```
make generate
```

Now you can build the executable binaries using:

```
make build
```

You will now find the client:

```
2021-ca400-collint9-coynemt2/build/synche
```

If you do not want to run the server in a Docker container, you will also now find the server executable in:

```
2021-ca400-collint9-coynemt2/build/synche-server
```

Note: If you are using the server executable instead of Docker, it is recommended to move `server` to the below folder (or anywhere else on your PATH):

```
usr/local/bin
```

As previously mentioned, it is recommended to run the server within a Docker container. Prior to running Docker, it is recommended to update the server configuration file if needed. You can move this file anywhere within your system. It can be found in the following location:

```
2021-ca400-collint9-coynemt2/src/synche-server.yaml
```

You will need to update the docker-compose file. Here, you need to ensure that the path under `services: synche: volumes:` is the full path to your configuration file. The docker-compose file can be found in the following location:

```
2021-ca400-collint9-coynemt2/src/docker-compose.yaml
```

To start Docker run:

```
systemctl start docker
```

It's always good to ensure that Docker is running, to check this run:

```
docker ps
```

Once docker is running, execute the following command to create the containers:

```
docker-compose up -d
```

Your server and database will now be running within the Docker containers.

5. Configuration

5.1 Server

Below illustrates the default server config:

```
config:
 database:
   address: 127.0.0.1:3306
   driver: mysql
   name: synche
   password: password
   protocol: tcp
   username: username
 ftp:
   certfile: ""
   hostname: 127.0.0.1
   keyfile: ""
   passiveports: 52013-52114
   port: 2121
   publicip: ""
   welcomemessage: Welcome to the Synche FTP server
 server:
   basepath: /v1/api
   chunkdir: /root/synche/data/chunks
   host: 127.0.0.1
   port: 9449
   storagedir: /root/synche/data/storage
 synche:
   debug: false
   dir: /root/synche
   verbose: false
```

This allows you to have complete control over where and at what ports each service is running. The only values that need to be updated in order for Synche server to run are `config: database: username` and `config: database: password`. All other values are customisable for the user.

5.1.2 CLI

If you choose to run the serve through the CLI instead of through Docker, run:

```
synche-server init
```

This prompts you to enter the server configuration values via the CLI. The only values that must be updated are the database username and password. You can press the enter button to accept a default value. Each configuration field will be printed with it's default value in brackets beside it. Below illustrates what this should look like:

```
INFO[0000] Leave the input blank (press enter) at any time to use the
default/current value.
INFO[0000] ==== Synche configuration ====
      > Dir (/home/user/synche):
      > Verbose (false):
      > Debug (false):
INFO[0016] ==== Server configuration ====
      > Port (9449):
      > Host (127.0.0.1):
      > BasePath (/v1/api):
      > ChunkDir (/home/user/synche/data/chunks):
      > StorageDir (/home/user/synche/data/storage):
INFO[0021] ==== Database configuration ====
      > Name (synche):
      > Username (root):
      > Password ():
      > Protocol (tcp):
      > Address (127.0.0.1:3306):
      > Driver (mysql):
INFO[0025] ==== Ftp configuration ====
      > Port (2121):
      > KeyFile ():
      > Hostname (127.0.0.1):
      > CertFile ():
      > PublicIp ():
      > PassivePorts (52013-52114):
      > WelcomeMessage (Welcome to the Synche FTP server):
```

5.2 Client

The default client configuration is as shown below:

```
config:
  chunks:
      sizekb: 1024
      workers: 100
  server:
      basepath: /v1/api
      host: 127.0.0.1:9449
      https: false
  synche:
      datadir: /home/user/synche/data
      debug: false
      dir: /home/user/synche
      verbose: false
```

As you can see, the chunk size and worker values are configurable. This allows you to optimise your upload speeds by configuring these values to suit a multitude of variables such as your hardware and network connection speeds.

5.2.1 Client CLI

Similar to the server, upon running the client for the first time through the CLI, you can run the following command:

```
synche init
```

This prompts you to enter the server configuration values via the CLI. All the default values are fully valid, so you may change them to reflect your personal preferences if you wish. This will create and write the configuration file to disk for you. To accept the default values, simply press the enter button.

The default values shown via the CLI are illustrated below:

```
INFO[0000] Leave the input blank (press enter) at any time to use the
default/current value.
INFO[0000] ==== Synche configuration ====
      > Debug (false):
      > Dir (/home/user/synche):
      > DataDir (/home/user/synche/data):
      > Verbose (false):
INFO[0004] ==== Chunks configuration ====
      > Workers (10):
      > SizeKB (1024):
INFO[0006] ==== Server configuration ====
      > Host (127.0.0.1:9449):
      > BasePath (/v1/api):
      > Https (false):
```

6. Usage

6.1 Client CLI

All file paths and directory paths relative from the Synche storage directory that you specified in your configuration file.

Commands should be entered with the following format:

```
synche [command] [-flag] [arg] [...]
```

6.1.1 Open the GUI via CLI

Open the UI in a browser, provided that it is running.

```
synche ui
```

6.1.2 Create a User

Create a user on the server the client is current connected to. This will prompt the user to enter an email address, name, and password to be associated with their account on the server.

```
synche new user
```

6.1.3 Delete a user

Delete a user and all files and directories owned by them on the server. This will prompt the user for their email address and password before deleting any information.

```
synche delete user
```

6.1.4 Upload a file

Upload a file from specified by filename in your current working directory.

```
synche put [filename] [directory-path]
```

```
synche put [filename] -d [directory-id]
```

Options:

Flag	Usage
-n	store the file on the server with this name instead
-s	size (in KB) of each chunk
-w	number of workers. This affects how many chunks are uploaded in parallel

Aliases:

```
upload
```

6.1.5 Retrieve file

Retrieve a file from the server. The file should be specified by file path by default.

```
synche get [file-path]
```

Options:

Flag	Usage

-o	output location. This specifies where the file is downloaded to. It can be a full file path or directory

Aliases:

```
download
```

6.1.6 Move a file

Move a file from a one specified directory to another, file paths are used by default. This can also be used to rename a file by simply specifying the new filename on the new path. Specifying files by file IDs and directories by directory IDs is optional.

```
synche mv [current-file-path] [new-file-path]
```

To move a file specified by file ID:

```
synche mv -i [file-id] [new-file-path]
```

To move a file specified by file ID and directory ID:

```
synche mv -i [file-id] -d [directory-id]
```

Aliases:

```
move
```

6.1.7 Delete a file

Delete a file specified by a file path by default.

```
synche rm [file-path]
```

To delete a file specified by the file ID:

```
synche rm -i [file-id]
```

Aliases:

```
del
```

```
delete
```

6.1.8 Make a directory

Make a directory on the server specified by the path to the new directory by default. Specifying the parent directory by ID instead of by path is optional.

```
synche mkdir [directory-path]
```

To make create a directory in a parent directory specified by the parent directory ID:

```
synche mkdir [directory-name] -d [directory-id]
```

Aliases:

```
md
```

6.1.9 Retrieve the contents of a directory

Retrieve the contents of a directory specified by directory path by default. Specifying the directory by ID is optional. This will return information regarding the following:

- Directory name
- Directory ID
- Information on the files present. This includes file name, file size, and file ID.
- Information on the directories present. This includes name and file count.

```
synche ls [directory-path]
```

To retrieve based on a directory ID:

```
synche ls -d [directory-id]
```

Aliases:

```
list
```

```
dir
```

6.1.10 Move a directory

Move a directory specified by the path to its current location and path to its new location by default. This can be used to rename directories. Specifying directory IDs instead of paths is optional.

```
synche mvdir [current-directory-path] [new-directory-path]
```

To move a directory specified by directory ID.

```
synche mvdir -d [directory-id] -o [output-directory-id]
```

6.1.11 Delete a directory

Delete a directory specified by its path by default. Specifying the directory by ID is optional.

```
synche rmdir [directory-path]
```

To delete a directory specified by ID:

```
synche rmdir -d [directory-id]
```

6.2 Client GUI

To start the Synche GUI you first must be in the `src//gui` directory from the repository's root. Once you are in the `gui` directory, you need to create a new .env file and add the following to it:

```
NEXT_PUBLIC_BASE_URL=<protocol>://<ip address>:<port><base API path>
```

Filling in the parameters above with your specific values. The default example is as follows:

```
NEXT_PUBLIC_BASE_URL=http://127.0.0.1:9449/v1/api
```

Once you have done that, run:

```
npm install
# followed by either
npm run dev
# or
npm run build
npm start
```

This will start the server on http://localhost:3000 by default.

6.3 Server

Server commands are accessed through the CLI. There are only two commands needed.

A comprehensive guide to the API can be found on Synche SwaggerHub. It is highly recommended that you read the written documentation here to understand the API endpoints.

6.3.1 Start the API server

Start the Synche API server to begin making requests.

```
synche-server serve
```

6.3.2 Generate HTTPS and TLS certs

Generate HTTPS and TLS certificates and keys to enable HTTPS and securely host the server.

```
synche-server makecerts [server-ip-addresses] -o [output-directory]
```