**Implementation**

**Why we chose NodeJS**

The server application was built using NodeJS. One of the main advantages of NodeJS is the ability of building fast and scalable network applications. It is able to handle big amounts of simultaneous requests due to its non-blocking and asynchronous architecture. Each connection is handled in the same thread using non-blocking I/O calls which means that one request does not have to wait until the previous one is finished to be executed. This is crucial for the efficiency of the server application as writing files to disk or transferring them through HTTP can take long amounts of times. So performing various requests at a time will decrease the response time for each request.

Another great advantage of NodeJS is the Node Package Manager (NPM). This package manager allows to reuse components and helps with managing the versions and dependencies of the components. NPM is public and anyone can publish their modules there for other people to use.

Our team also considered that NodeJS uses JavaScript which will also be used in the desktop application making it easier for all the team members to adapt from one to the other.

**API description**

The server application provides end points for the client side applications to send or receive information. There are a total of 8 end points which will be explained in this section although further explanation can be found in the ‘docs.md’ file inside the server project folder. These 8 end points have been implemented in 2 different files ‘/src/routes/users.js’ for all the user account related functionality and ‘/src/routes/files.js’ for the file related functionality.

A new user account can be created by calling the sign up function which has the end point /user/signup. This function requires a POST request with the parameters ‘username’ and ‘password’. The function checks that the username has not already been taken by another user and saves the username and a salted hash of the password in the database.

User log in is done by calling the /user/login end point this function requires a POST request with the ‘username’ and ‘password’ the hash of the password is recalculated and compared to the one stored for that user. If the authentication is successful a session cookie will be sent in the response. The client is expected to use this cookie to identify himself later.

Users can logout of the system by calling the logout function at the end point /user/logout. This function receives a GET request from an authenticated user (the session cookie must be included in the request) and will invalidate that session and logout the user.

To delete a user account the user has to send a GET request will being logged in to the /user/delete end point. This function will invalidate that session and delete his user entry in the database.

The rest of the API functions deal with the file synchronization part of the system. For all of them the user must be authenticated by sending the session cookie in the headers of each request.

In order for the user to know all the information about the files that he has uploaded to the server the end point /file/getInfo can be called with a GET request that will return a json array with all the information about that users files. This function accepts a ‘filename’ parameter which acts as a filter to only get the file information for one file. The file information includes filename, server version and the hash of that file among other useful information.

The push function implements a lot of our system functionality. It can be called through a POST request and receives 5 parameters depending on what the user wants to do. The first parameter is ‘file’ this is an actual file that the user wants to upload to the server. If ‘file’ is included the user either wants to upload a new file or update an existing one. To distinguish between these two scenarios the database is checked to see if this user already has a file with that filename. If it is a new file the information about the file will be stored in the database and the file stored in the file system. In the case that the file already existed the hash will be checked to detect if the file has been modified. If it has not been modified there is no need to save the file so an error will be sent to the user. If the hash is different the server must ensure that the version of the file that the user modified was the latest in the server to avoid conflicts. In order to do this the user must include the ‘version’ parameter with the version number of the file that he modified and the server will check if it is the latest version. If it is not the latest version the server will respond with the error code 409 Conflict indicating that there is a conflict which he will have to resolve. If there is no conflict the server will save the new information for that file, increase the version number and update the file in the file system. The other main functionality of this function is to delete files from the server. To do this the ‘file’ parameter will be ignored, the ‘delete’ must be set to “true” and the ‘filename’ parameter to the name of the file to be deleted. The server will check the version number provided in the request and if there are no conflicts delete the file from the file system and the database. The ‘force’ parameter is used to ignore the version and therefore the conflicts. This is useful to resolve conflicts in the case where the user wants to overwrite the latest version with his version.

((((Probably should include a flow chart of the logic and make references to it in the paragraph))))

Once the user gets the information from the files he has stored in the server he might want to download them in order to do this he can use the download function located in the end point /file/getFile. This function receives a GET request with the parameter ‘filename’ which indicates the file the user wants to download. If the user has previously uploaded a file with that filename the file will be sent back in the response.

In the case of updating a file and receiving a response stating that there is a conflict the user can decide to discard his changes and download the latest version, use the push function with the ‘force’ flag set to true or use the get differences function located in the /file/getDiff end point. This function receives a POST request with the ‘version’ and the ‘file’ that has the conflict. The server checks that indeed there is a conflict with that file by checking the version numbers and checks if the file is a text or binary file. If it is a text file it then generates a diff file similar to what ‘git diff’ would generate. The content of this file is sent on the response and the client can easily resolve the conflicts in the file and then use the push function normally to update the file taking into consideration the latest version of the file.

**Libraries**

One of the greatest advantages of NodeJS is the huge number of modules available through NPM which speed up development and help make code more reusable.

The main module that the application requires is ExpressJS [1], express provides functionality to set up a http server and handle the requests.

MongooseJS [2] has been used as a object modeling tool for MongoDB, it allows for easy creation of models for the different objects stored in the database and provides an asynchronous way to make the database interactions.

Morgan [3] has been used to log all the requests done to the server and all the possible error that can occur.

HTTP-errors [4], helps creating meaningful HTTP errors for express.

PassportJS [5], allows to authenticate users in different ways including google, facebook or twitter accounts. In our case we used it together with connect-mongo [6], express-sessions [7] and cookie-parser [8] to do the authentication of users with accounts in our own database.

Express-fileupload [9], helps managing files uploaded through HTTP making it easier to save them and collect information about them.

To do the SHA256 hashes of the files the application uses bcryptJS [10].

Finally in order to generate the diff files when handling conflict isbinaryfile [11] is being used together with jsDiff [12].

[1] [https://expressjs.com](https://expressjs.com/)

[2] [https://mongoosejs.com](https://mongoosejs.com/)

[3] [https://github.com/expressjs/morgan#readme](https://github.com/expressjs/morgan" \l "readme)

[4] [https://github.com/jshttp/http-errors#readme](https://github.com/jshttp/http-errors" \l "readme)

[5] <http://www.passportjs.org/>

[6] [https://github.com/jdesboeufs/connect-mongo#readme](https://github.com/jdesboeufs/connect-mongo" \l "readme)

[7] [https://github.com/expressjs/session#readme](https://github.com/expressjs/session" \l "readme)

[8] [https://github.com/expressjs/cookie-parser#readme](https://github.com/expressjs/cookie-parser" \l "readme)

[9] [https://github.com/richardgirges/express-fileupload#readme](https://github.com/richardgirges/express-fileupload" \l "readme)

[10] [https://github.com/dcodeIO/bcrypt.js#readme](https://github.com/dcodeIO/bcrypt.js" \l "readme)

[11] [https://github.com/gjtorikian/isBinaryFile#readme](https://github.com/gjtorikian/isBinaryFile" \l "readme)

[12] [https://github.com/kpdecker/jsdiff#readme](https://github.com/kpdecker/jsdiff" \l "readme)