School of Computing
CA400 Technical Specification

Project Title: **xDrive Private Cloud Solution**
Student Names: **Michael Collins, Ikenna Festus**
Student IDs: **16313231, XXXXXXXX**

Project Supervisor: **Stephen Blott**

Completion Date: **16/05/2020**

Table of Contents

1. Glossary

- **Javascript -** Main programming language uses.
- **MongoDB -** NoSQL Database used.
- **NodeJS -** Javascript framework used for server.
- **ExpressJS** -Web Application framework.
- **ReactJS -** Front-End framework.

1. Motivation

Motivation for this project came during our common interests in privacy and security. Between this and the lack of truly private and transparent cloud storage solutions we decided on giving our attempt in the field. We initially set out with the intention of focussing on security or privacy. During our search for a project we came across several solutions that provided high levels of security. However, a common trend we found among these solutions is that they were not open-sourced, and thus they could not be self-hosted. This flaw caught our attention as we both wanted something truly private while still having an emphasis on security.

After Researching the market some more, we began to narrow our focus. We started to set out requirements. We wanted to build a solution that was not only secure, but truly private. For us this meant that the user could host the service on their own machine to ensure that they truly own their data. We wanted the service to be easy to use for non-technical users and also free, as we believe that privacy should not be held behind a paywall.

2. Research

For this section we will comment on the technologies used and why we used them. We will also note any significant findings during the research phase of our project.

We began to research this idea and found that though it was an idea that is making a huge income for companies such as Google and Microsoft, not many people felt the need to re-invent their own versions of a file storage system. This meant that resources were highly limited. With this project we had to research the correct methods to implement the application. We were phased with various factors such as security, accessibility and scalability.

We decided upon using .Net frameworks for the implementation. However, we realised that implementing that framework would mean that we will be limited to users of Windows operating system. That being said, we agreed upon using Javascript libraries such as React, Redux and Express. These JavaScript frameworks seemed best suited for our application as it offers flexibility and allows for a powerful and secure web interface for our users. Once we understood the libraries/frameworks we will be using, we began thorough research on best practices, studied the similar ideas in the world today (such as icloud, oneDrive and GoogleDrive), while also researching students' needs with regards to a file hosting/ Storage system.

3. Design

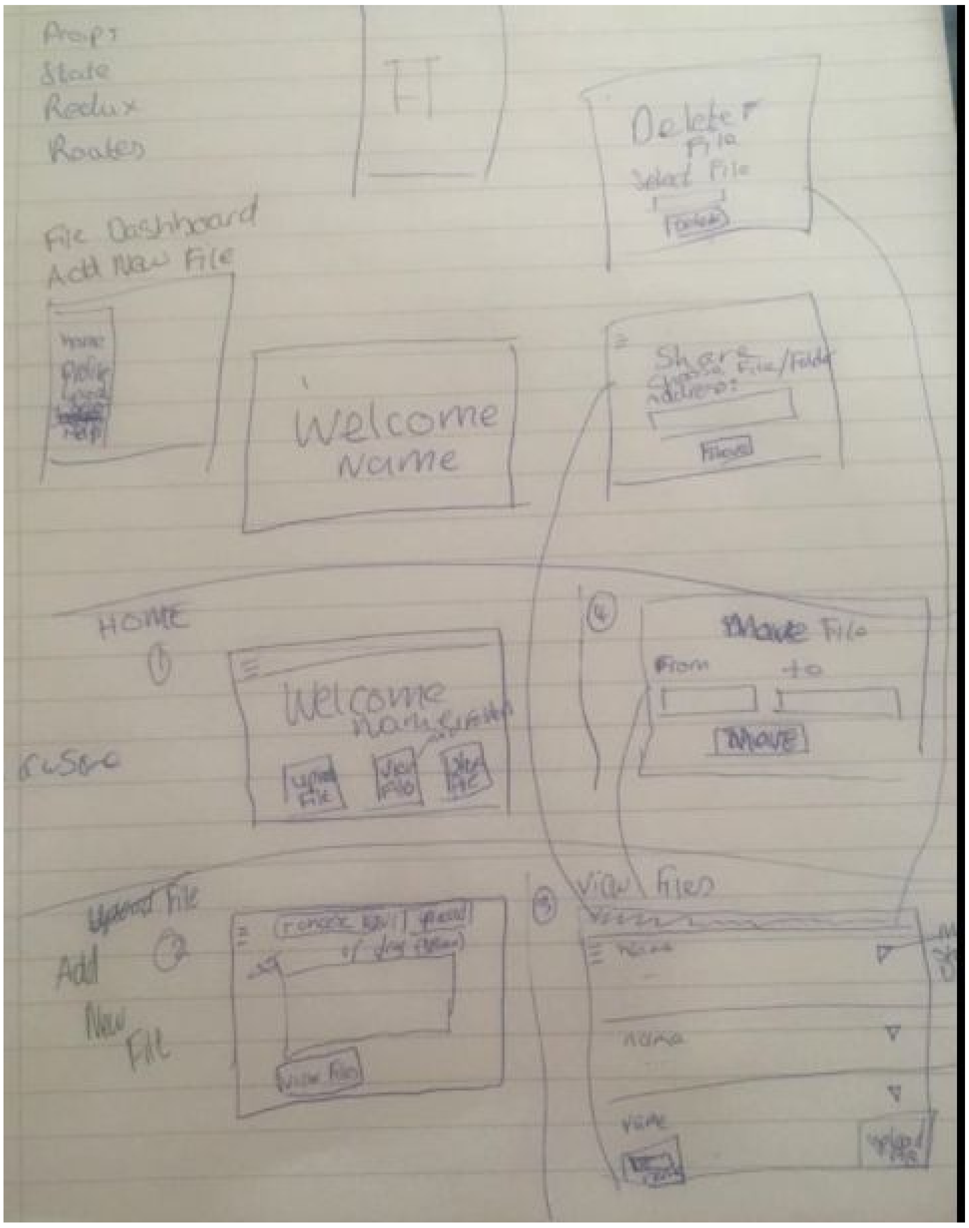


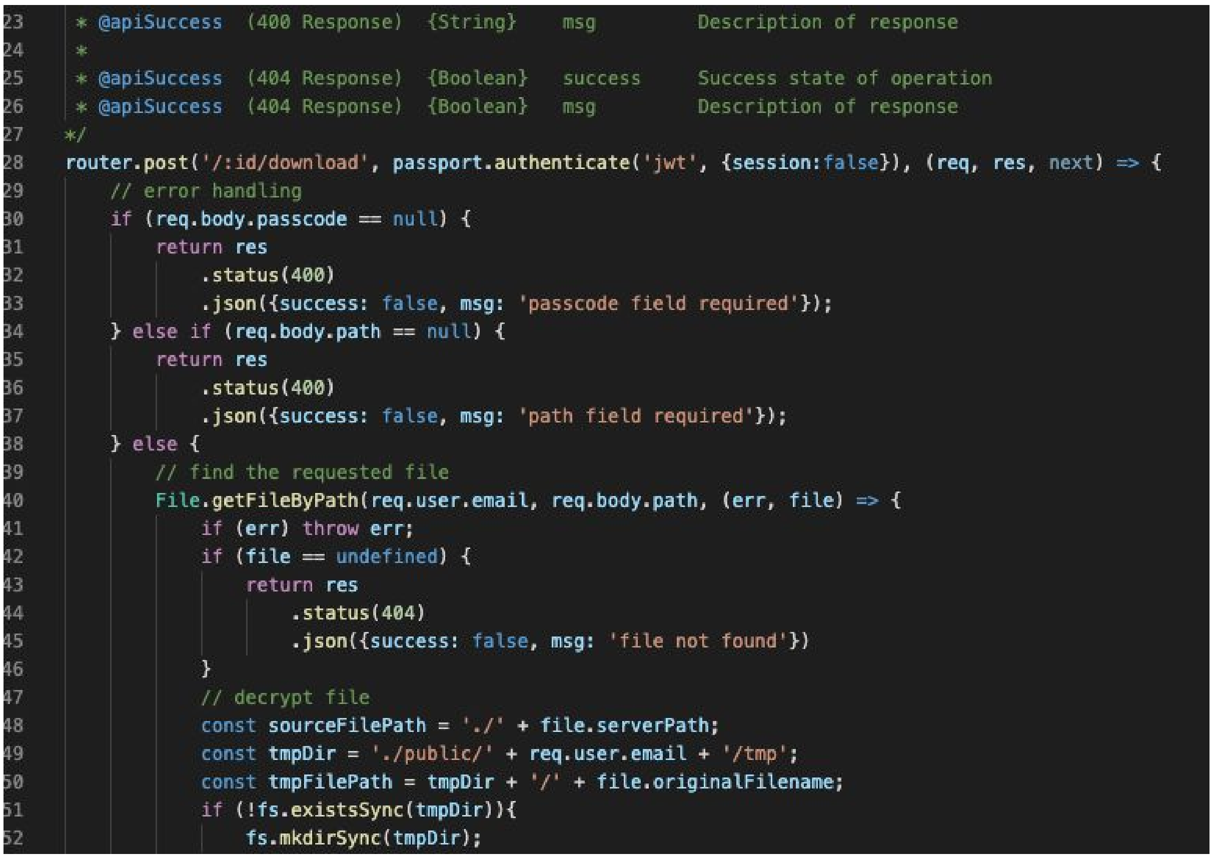
Image of User Interface design prior to coding.

4. Implementation

The aim of this project is to choose a complicated objective, create a full back to frontend project and use as many tools as possible to broaden our knowledge in the field of Software Engineering.

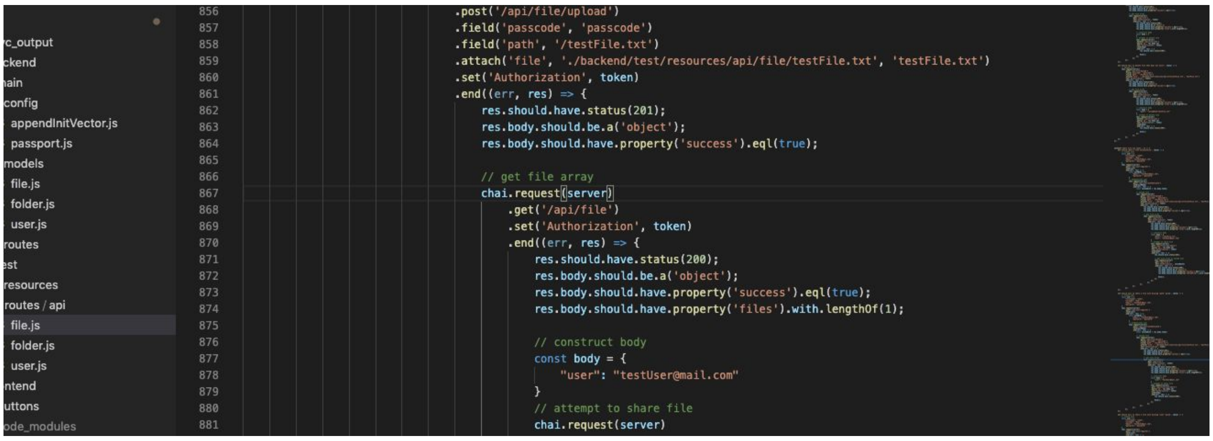
These goals led to the use of many different components while creating these web applications. As mentioned previously, this is a full back to front end application. The backend was completed using express and MongoDB as our primary database. Intrinsic coding meant that security would be implemented using encryption to ensure that user data is not jeopardized both back to frontend. We use React for the frontend with enabled an easy to use user interface for it’s users. We implemented this and added accessibility, simplicity and calmness to the user experience using important software development principles such as those we learned in previous years of the course such as Jakob Nielsen's 10 Usability Heuristics for UI Design, and Bruce Tognazzini's Principles of Interaction Design, and Ben Shneiderman's The Eight Golden Rules of Interface Design

5. Sample Code



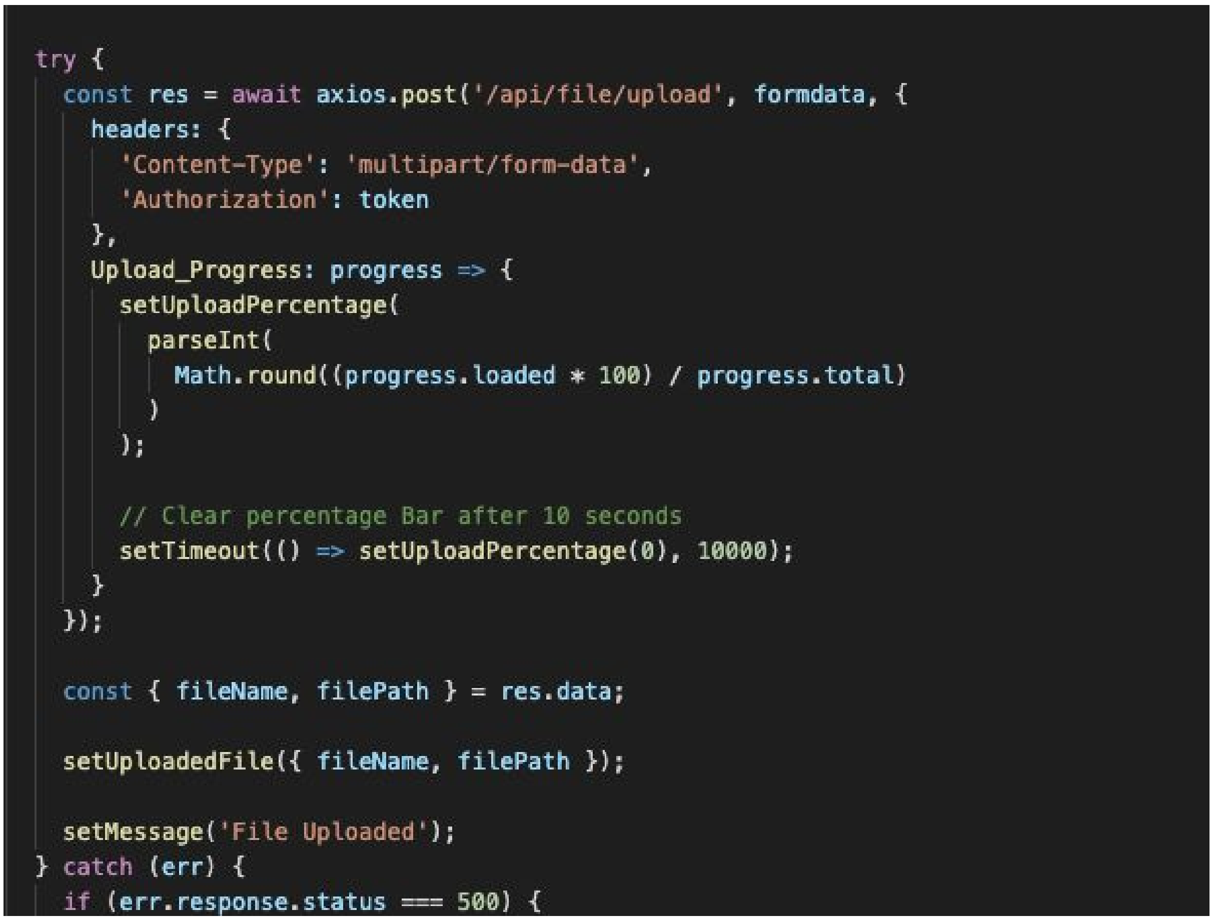
```
 * @apiSuccess  (400 Response)  {String}    msg        Description of response
 *
 * @apiSuccess  (404 Response)  {Boolean}   success    Success state of operation
 * @apiSuccess  (404 Response)  {Boolean}   msg        Description of response
*/
router.post('/:id/download', passport.authenticate('jwt', {session:false}), (req, res, next) => {
    // error handling
    if (req.body.passcode == null) {
        return res
            .status(400)
            .json({success: false, msg: 'passcode field required'});
    } else if (req.body.path == null) {
        return res
            .status(400)
            .json({success: false, msg: 'path field required'});
    } else {
        // find the requested file
        File.getFileByPath(req.user.email, req.body.path, (err, file) => {
            if (err) throw err;
            if (file == undefined) {
                return res
                    .status(404)
                    .json({success: false, msg: 'file not found'})
            }
            // decrypt file
            const sourceFilePath = './' + file.serverPath;
            const tmpDir = './public/' + req.user.email + '/tmp';
            const tmpFilePath = tmpDir + '/' + file.originalFilename;
            if (!fs.existsSync(tmpDir)){
                fs.mkdirSync(tmpDir);
```

Example of backend code for downloading a file.



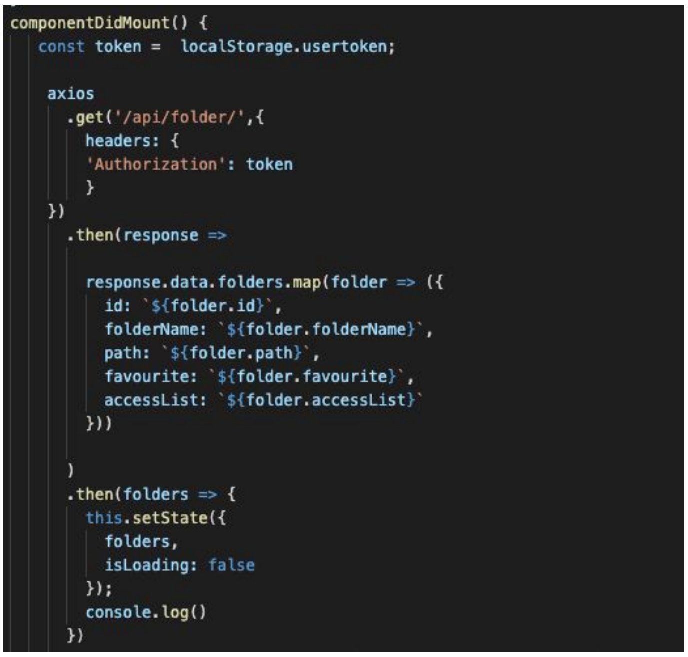
```
.post('/api/file/upload')
.field('passcode', 'passcode')
.field('path', '/testFile.txt')
.attach('file', './backend/test/resources/api/file/testFile.txt', 'testFile.txt')
.set('Authorization', token)
.end((err, res) => {
    res.should.have.status(201);
    res.body.should.be.a('object');
    res.body.should.have.property('success').eql(true);

    // get file array
    chai.request(server)
        .get('/api/file')
        .set('Authorization', token)
        .end((err, res) => {
            res.should.have.status(200);
            res.body.should.be.a('object');
            res.body.should.have.property('success').eql(true);
            res.body.should.have.property('files').with.lengthOf(1);

            // construct body
            const body = {
                "user": "testUser@mail.com"
            }
            // attempt to share file
            chai.request(server)
```

Example of 98% coverage test of file system.



```
try {
  const res = await axios.post('/api/file/upload', formdata, {
    headers: {
      'Content-Type': 'multipart/form-data',
      'Authorization': token
    },
    Upload_Progress: progress => {
      setUploadPercentage(
        parseInt(
          Math.round((progress.loaded * 100) / progress.total)
        )
      );

      // Clear percentage Bar after 10 seconds
      setTimeout(() => setUploadPercentage(0), 10000);
    }
  });

  const { fileName, filePath } = res.data;

  setUploadedFile({ fileName, filePath });

  setMessage('File Uploaded');
} catch (err) {
  if (err.response.status === 500) {
```

Example of frontend file upload system using React.

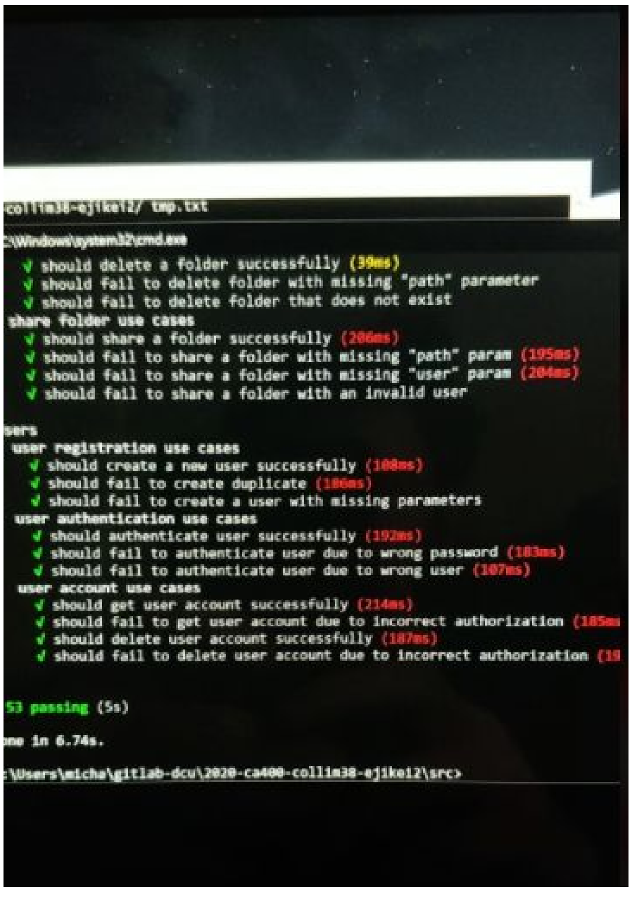


```
componentDidMount() {
  const token =  localStorage.usertoken;

   axios
     .get('/api/folder/',{
       headers: {
       'Authorization': token
       }
   })
     .then(response =>

       response.data.folders.map(folder => ({
         id: `${folder.id}`,
         folderName: `${folder.folderName}`,
         path: `${folder.path}`,
         favourite: `${folder.favourite}`,
         accessList: `${folder.accessList}`
       }))

     )
     .then(folders => {
       this.setState({
         folders,
         isLoading: false
       });
       console.log()
     })
```

Example of handling user data from the backend.

---

6. Results



```
collim38-ejikei2/ tmp.txt
:\Windows\system32\cmd.exe
    √ should delete a folder successfully (39ms)
    √ should fail to delete folder with missing "path" parameter
    √ should fail to delete folder that does not exist
  share folder use cases
    √ should share a folder successfully (206ms)
    √ should fail to share a folder with missing "path" param (195ms)
    √ should fail to share a folder with missing "user" param (204ms)
    √ should fail to share a folder with an invalid user

sers
  user registration use cases
    √ should create a new user successfully (108ms)
    √ should fail to create duplicate (186ms)
    √ should fail to create a user with missing parameters
  user authentication use cases
    √ should authenticate user successfully (192ms)
    √ should fail to authenticate user due to wrong password (183ms)
    √ should fail to authenticate user due to wrong user (107ms)
  user account use cases
    √ should get user account successfully (214ms)
    √ should fail to get user account due to incorrect authorization (185ms
    √ should delete user account successfully (187ms)
    √ should fail to delete user account due to incorrect authorization (19


 53 passing (5s)

one in 6.74s.

:\Users\micha\gitlab-dcu\2020-ca400-collim38-ejikei2\src>
```

Image of passed test on backend Component with 98% Coverage.

---

7. Future Work

This project has an endless point. Due to time constraints many components had to be extracted. With a project of such magnitude, one can only get a plethora of experience. I do believe that all components envisioned can be implemented by us even beyond the deadline of the project.