

Companion File Browser

DESIGN DOCUMENT

Sd-Dec19 Team 11

Client: Kirit Chandran

Adviser: Mai Zheng

Team Members:

Christopher Bui

Brian Chodur

Luke Stoll

Zhen Zhao

Email: Sddec19-11@iastate.edu

Website: <https://sddec19-11.sd.ece.iastate.edu>

Revised: 05/2/2019 V2

Table of Contents

1 Introduction	3
Acknowledgement	3
Problem and Project Statement	3
Operational Environment	3
Intended Users and Uses	3
Assumptions and Limitations	3
Expected End Product and Deliverables	4
2. Specifications and Analysis	4
Proposed Design	4
Design Analysis	14
3. Testing and Implementation	16
Interface Specifications	16
Hardware and software	16
Functional Testing	16
Non-Functional Testing	17
Process	18
Results	19
4. Closing Material	19
4.1 Conclusion	19
4.2 References	20

List of figures/tables/symbols/definitions

Figure 1: System Architecture (Page 5)

Figure 2: File System Design (Page 6)

Figure 3: Database Relationship tables (Page 7)

Figure 4: Main Page Design Sketches (Page 8)

Figure 5: Registration/Login Sketch (Page 9)

Figure 6: SSR vs CSR (Page 10)

Figure 7: Testing Process (Page 17)

1 Introduction

1.1 ACKNOWLEDGEMENT

Buildertrend will provide the existing desktop application to extend features on. The software team of Buildertrend will also provide the current backend API signature as reference.

1.2 PROBLEM AND PROJECT STATEMENT

Buildertrend would like a a new sub system extension for the current software product. This will be a new feature for the clients of Buildertrend, to provide more effective ways to team collaborations. It will also provide client with error recovery, and high accessibility.

The new extension is a cloud file sharing system that allow users sync and download files on cloud. The system will provide user easy collaborations, error recovery, and high accessibility.

1.3 OPERATIONAL ENVIRONMENT

The necessary operational environment for this project will under a computer with Internet accessibility. The user may choose to use the two ways to interact with the cloud system is using web application in a browser, or a Windows desktop application that Buildertrend currently offer.

1.4 INTENDED USERS AND USES

Intended users are clients under Buildertrend who will want cloud file storage with sharing functionally across team or company.

Intended uses will to be able to create an account, upload files to the cloud, view and download files. The user will be able to modify these files.

1.5 ASSUMPTIONS AND LIMITATIONS

Assumptions:

- All file types will be supported
- User have consistent Internet accessibility.
- Application backend will host on a single server.

Limitations:

- Depending on the number of files, storage space may be an issue for testing
- Files will not exceed 50 MB in size
- Backend programming languages needs to be C#.
- Frontend framework needs to be React.

1.6 EXPECTED END PRODUCT AND DELIVERABLES

Web application – December 2019:

The end product will be web application that allows users to upload files while specifying viewing and editing permissions. Along with this, the user will be able to view, edit, delete these files. A history link of previous versions will be maintained and can revert or download these versions.

Btconnect extension – December 2019:

The end product will be an extension of the existing desktop application that will allow users to view all the files hosted on the web application. These files are in sync together and allow a user to be able to view or modify depending on the permissions set.

Documentation – December 2019:

Documentation will be delivered along with the web application and btconnect extension. This will help aid in understanding the software at a lower level of detail as well as document how the web API handles requests.

2. Specifications and Analysis

2.1 PROPOSED DESIGN

We are limited to possible solutions and need to develop using the current technologies mentioned by our client that is used at Buildertrend which is React for web UI and C# for the web API and backend services needed.

We will develop using ES6 or known as ECMAScript 2015 for better consistency and practices when developing in JavaScript.

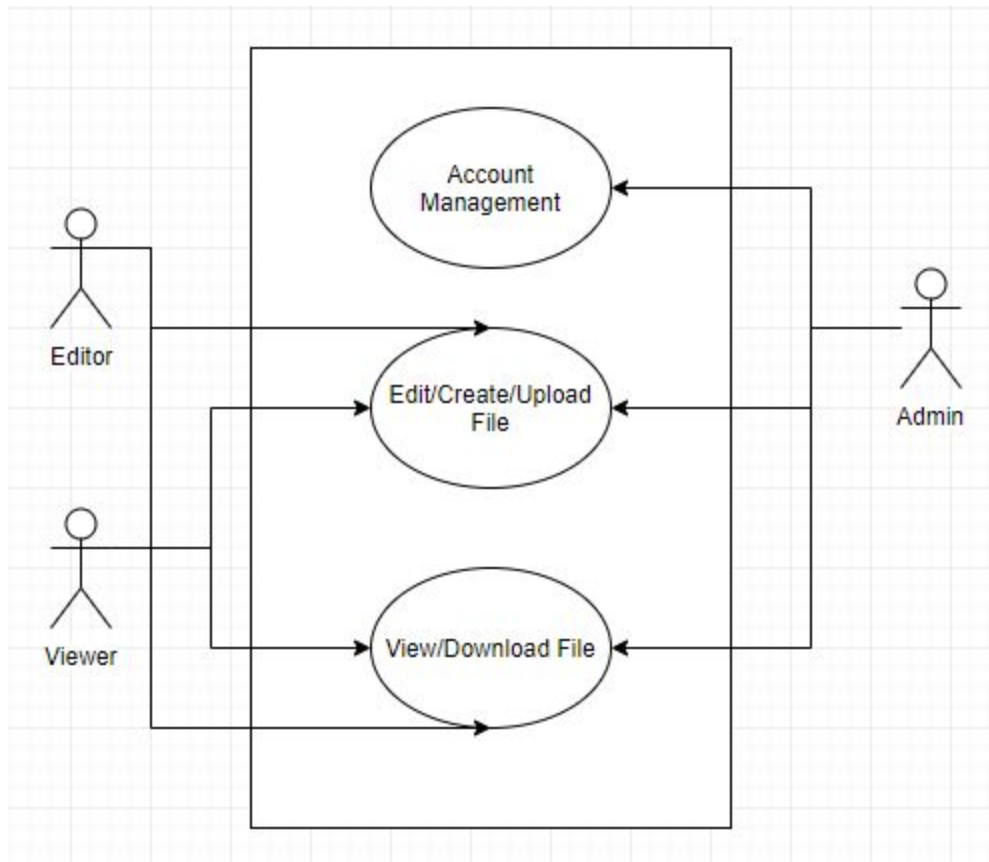


Figure 1 Use Case Diagram

Looking at the use case diagram above, there are three types of users which have access to different features in the application. The admin will have access to all of the features in the application; Account Management, View and Edit File. The Editor will have full access to all files belonging to the company. The viewer will only have view access to all files belonging to the company.

Backend design - System Architecture

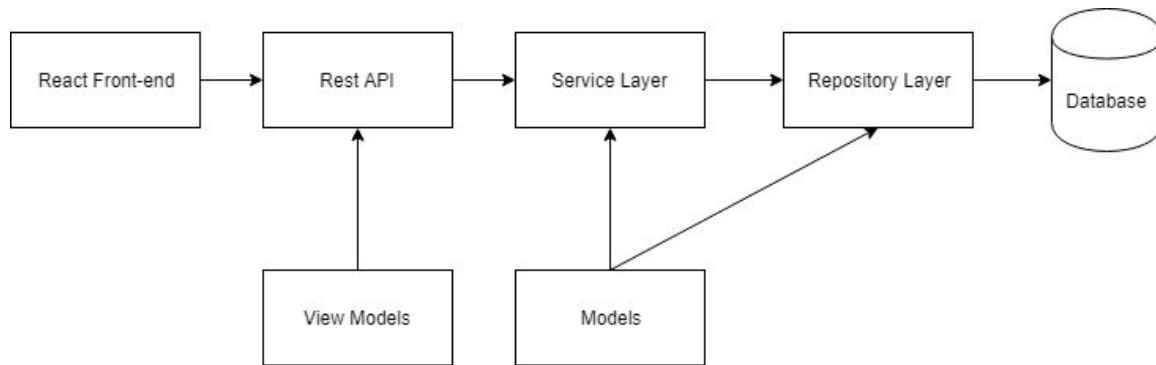


Figure 1: System Architecture

The main goal here is to keep every module to a degree of high cohesion but low coupling. What this means is we want to make sure each module does only the job it is meant while only being dependant on limited modules. This allows for flexibility for modification without the need to change the whole system.

The idea with this design is to allow the frontend to call a restful API service. This API will handle all HTTP requests and forward to the correct service in the service layer. The service layer is where all the business logic will occur. If all success so far in the service layer based on the business rules, we can call the repository layer which is where all database calls will happen. This information will be returned back down the line back to the frontend.

Overall, making a change to one module will require little to no modification of other modules.

Backend design - File System Design

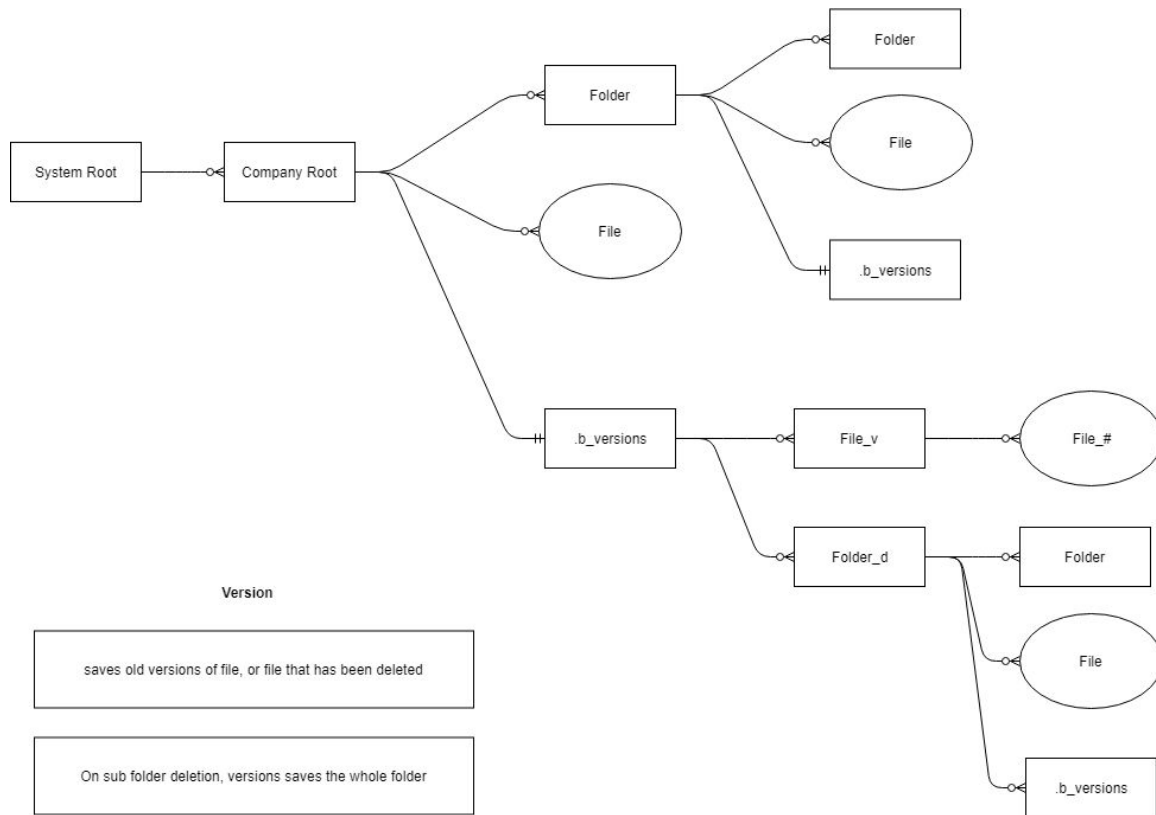


Figure 2: File System Design

There will be a folder that represent as the system storage root folder. In the System Root, there will be multiple folders, and one unique folder will represent one company in the system, all user under the company can view the company folder. The folders under each user root folder will automatically belongs to that company. As a feature, the system will allow user to roll back the older versions of the file, or recover deleted files. Therefore, in the system, for every folder there will be a hidden folder call .b_versions stores older versions of the file, or simply the deleted files.

To more detail about the how history files get stored. There will be a folder corresponding to a file that either has old versions, or has been deleted. The folder will name the same as the file and append “_v” after to represent version folder for that file. As a deleted folder, the whole folder will be move to the history folder of it’s parent’s folder, and append _d after.

Each File/Folder will have custom file attributes with unique file ID and version number, in this case, file can be keep track in the database.

Backend Design - Database Relations

We designed our database tables in respect to the functional and non-functional requirements. To break it down, we have the following tables, Account, Company, File, Folder, and File History.

Account: This table will hold user information and uniquely identified by their ID. To sign up, a user will need their first and last name, email, and password. In our table, we will generate a salt, which is a randomly generated string of characters and prepend passwords then store the hash of the salt + password.

Company: This table will reference each company that has access to the web application. Each company will have many accounts that are users belonging to the company.

File: Users are able to upload files to a remote server so we decided to store each file and have it reference the company the file is under. This is because all users under one company should only see their company's files and not anyone else's file.

Folder: This is the same as our file table except we store sub folders that will contain files. These folders are under one company.

File History: Each file needs a link of all previous history modifications. We added this table to create a list of files that all reference back to only one file which is the current file.

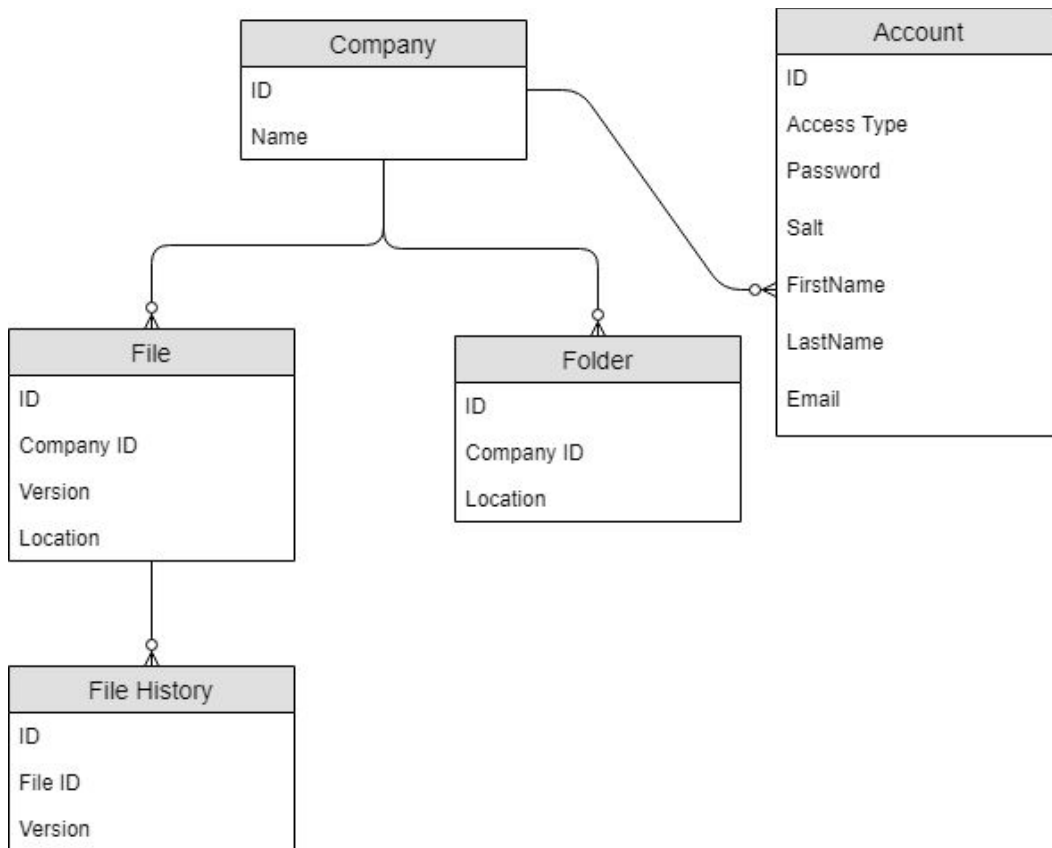
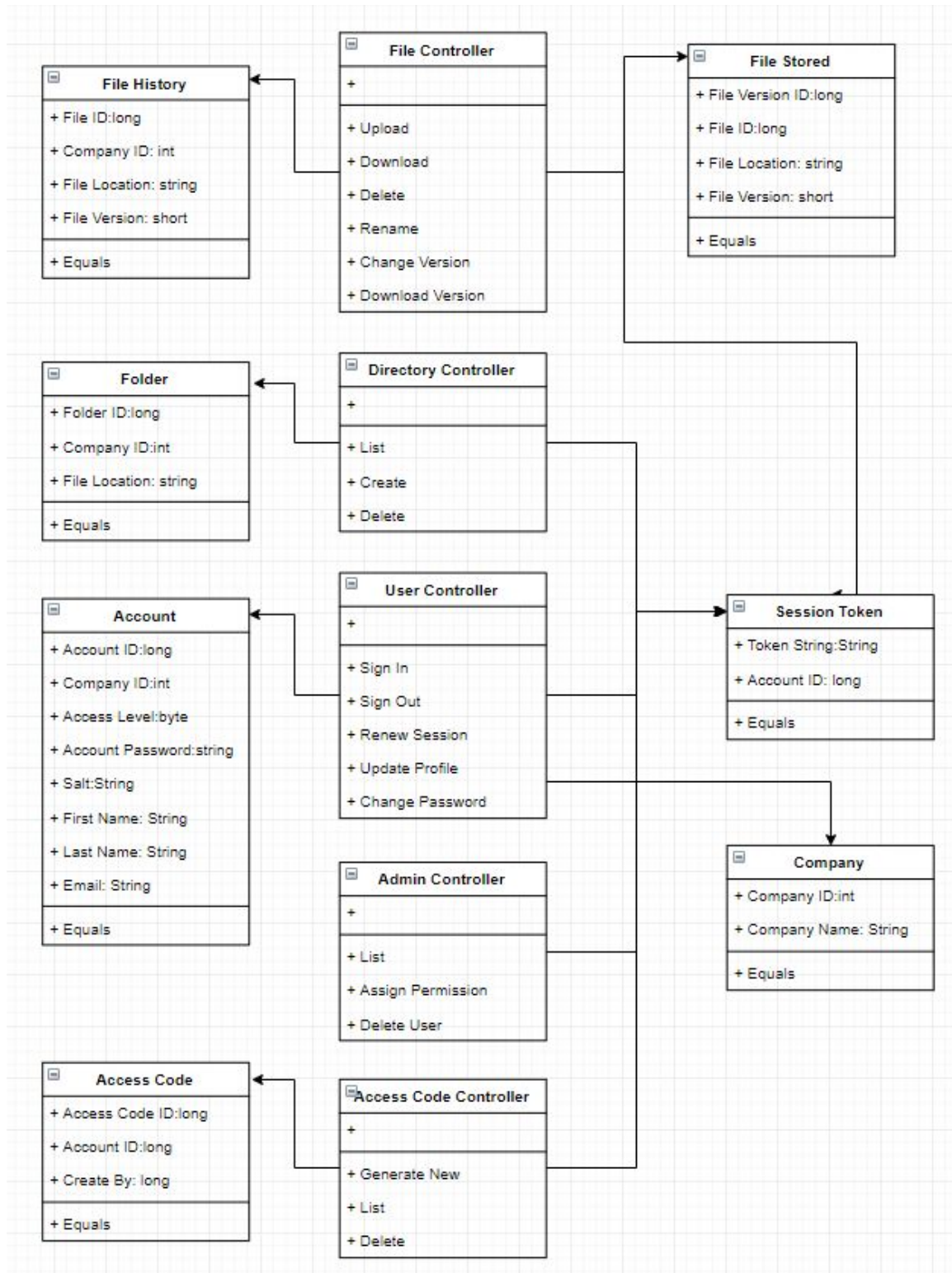


Figure 3: Database Relationship tables

Backend Design - Class Diagram



The class diagram refers to how each class interact each other in the backend. There are three types class in the backend: Controller, Model, and Views. The class diagram above will only talk about Controller and Model, because that most of the views are done in React.js. More specially react component are made in JavaScript, so that all component will be talked about in the frontend design.

There are five controller class handle request from the client. All of the controller when handle request from client, it takes in a session token object to verify the identity of client. Most controller will interact with the database by the model class and a context class comes along with the model.

Backend Design - Concurrency

We need to deal with situation that multiple user is trying to access the same file.

There are few different concurrency related situation below.

- One user trying to read while other user is writing to the same file.
- Multiple users trying to read from the same file.

Where that the first case, when user A trying to write to a file, user A needs to wait until all other user has read the file, then process write. In the meantime, all other user is trying to read will need to wait until the write is done.

The case of multiple user trying to read from the same file should be process immediately.

The methodology to enforce the solution is to use file locks. There will be two type of file locks, read, and write. The read lock will be obtain by read request controller, and write lock will be obtain by the write request controller. When there is read lock held by any controller, the write lock will be on a hold until all read lock is released.

Frontend Design - Main Page

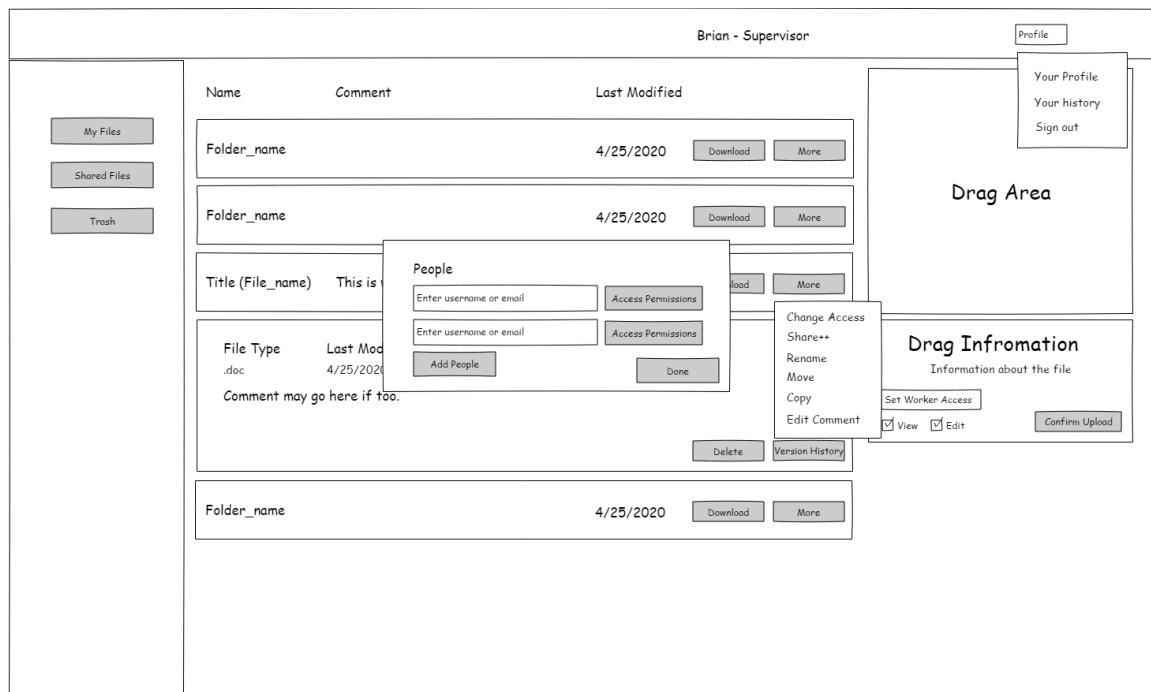
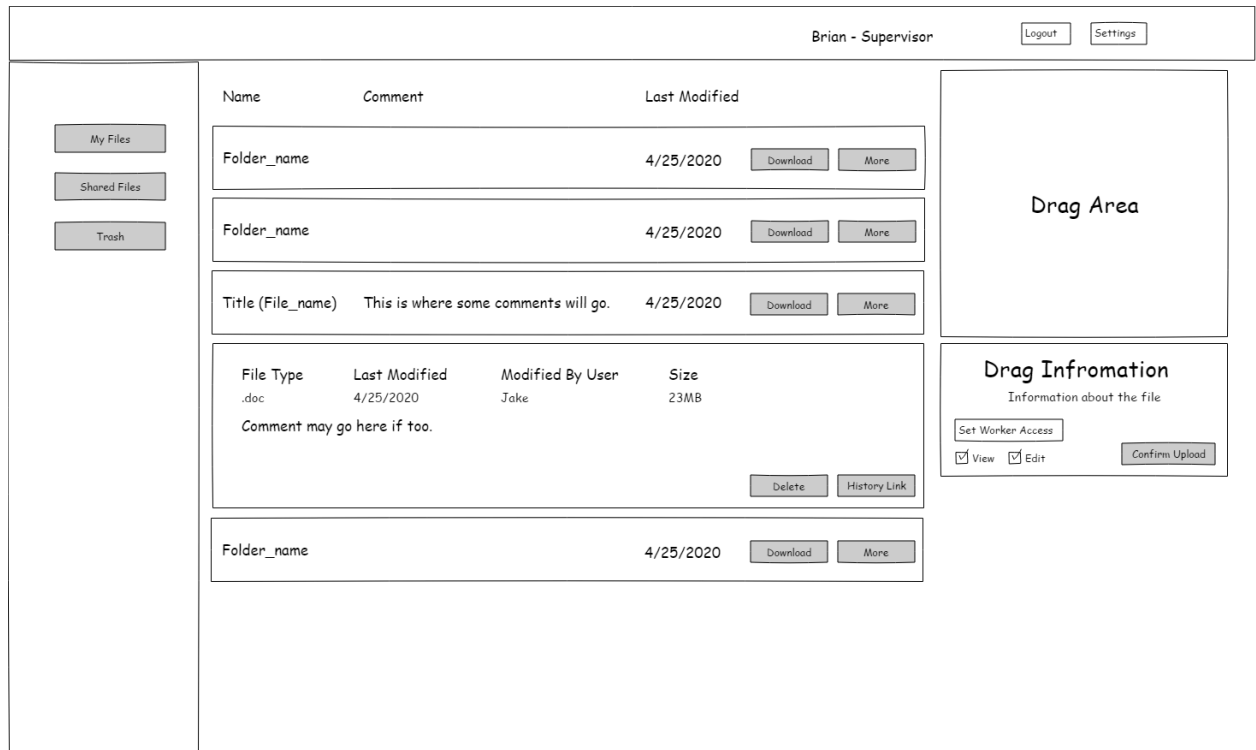
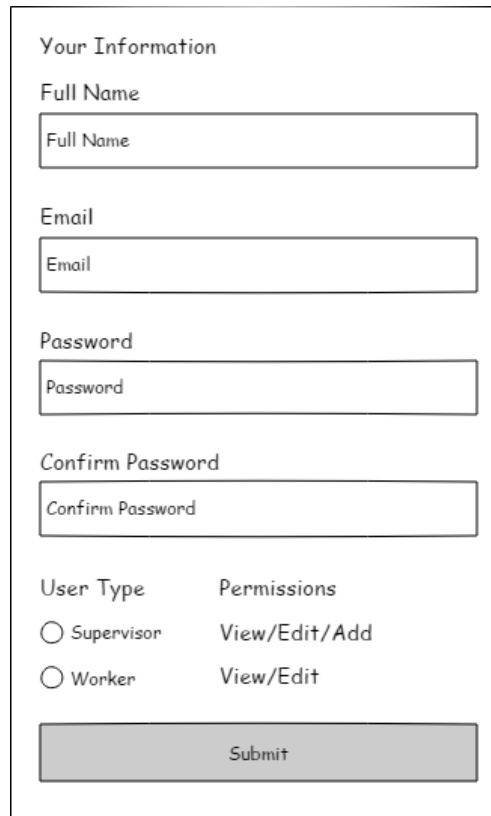


Figure 4: Main Page Design Sketches

Frontend Design - Registration/Login



The sketch shows a registration/login form with the following elements:

- Your Information** (Section Header)
- Full Name** (Label) with a text input field containing the placeholder text "Full Name".
- Email** (Label) with a text input field containing the placeholder text "Email".
- Password** (Label) with a text input field containing the placeholder text "Password".
- Confirm Password** (Label) with a text input field containing the placeholder text "Confirm Password".
- User Type** (Label) with two radio button options:
 - Supervisor
 - Worker
- Permissions** (Label) with two corresponding text labels:
 - View/Edit/Add (aligned with Supervisor)
 - View/Edit (aligned with Worker)
- Submit** (Text) on a grey rectangular button.

Figure 5: Registration/Login Sketch

The user will be asked to first log in or register. After doing so the user will be taken to the main page. The main page has a very similar page to something you would see on google drive or Dropbox. This design is subject to change but for now, it will display the folders and files in a list format with a separate drag area for files uploads. The main page will be made out of different React components. The components as of now will be the side navigation bar, top header, folder list what contains all the folders and files, each folder and file in the list, and the drag area. Clicking on a folder will act like Dropbox meaning the folder list will just update to the folder and files in the folder that you clicked on. As of right now, the folder drag area will upload relative to the folder you're in. So if you have folder 5 open then dragging a file in will place it in folder 5. The second image of the main page are the different dropdown menus.

2.2 DESIGN ANALYSIS

System Architecture:

We have discussed whether we will want a full on MVC application that will consume a restful API or a frontend application that will consume a restful API that connects to a more intricate backend. The idea with the first option is we can use server side rendering while the second option we use client side rendering. Which follows the figure below.

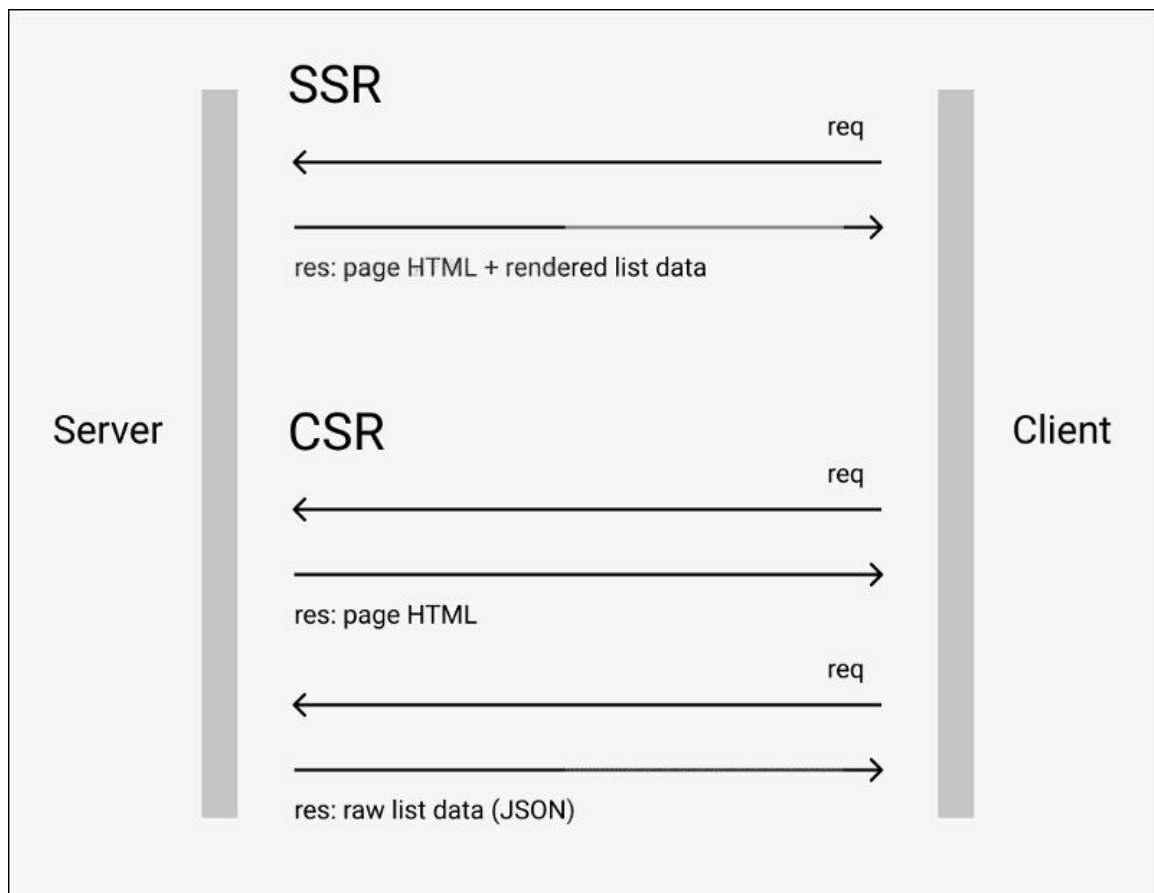


Figure 6: SSR vs CSR (Zhiyentayev, 2019)

The strengths of CSR (Client Side Rendering) is that our application will have few resources to load from the server. This means we don't need to have the server render the component. On the other hand, for SSR (Server Side Rendering) we can make less requests and have the whole page loaded at once. This requires the client to have faster network speed (Zhiyentayev, 2019) which is one of the weaknesses.

As of now, we will be using CSR. This is because of the low amount of page loading that is needed. We can keep a single page layout on the frontend. As for modifications, we may not really have a need for view models since the view models and models will very closely relate without much change on the frontend.

File System Design:

The file system design is completed, but subject to change. We believe this will work as expected. There could be other design might work, but this should be effective when considering the least database query as possible to improve performance.

The strengths of the file system would be better performance accessing files that is owned by the user since there will not be database query needed for access file

The weakness of the file system might be small performance delay when update/moving/delete the file or folders.

Database Design:

The design of our database is incorporate many-to-one and one-to-many principles as well as keep data integrity. This allows us to follow the business rules present and ensure security around them. With our tables now, we can prevent users from different companies from seeing other companys files. User information is also encrypted. By separating folders and files into their own tables we can reduce look up time.

Meeting Requirement - Class Diagram Design:

All the requirement can be satisfy by the current controller design. File controller will mainly handle upload/download different versions of files. The file controller also controls the maximum size of the file that can be upload, which it is one of the non-functional requirements. Where Admin Controller will handle request related to assigning permissions to to user or creating new user.

3 Testing and Implementation

3.1 INTERFACE SPECIFICATIONS

There are no specific hardware interfaces that we will be using. Per requirement, users will need a browser application that can access the web such as Chrome. MySQL will be used for database storage, React.js will be used for frontend user interface, and C# .net core will be used for our restful API.

For testing, we will use various browser clients on our local machines and for software we will use Postman for testing our restful API and for testing our front end we will use Jest.

3.2 HARDWARE AND SOFTWARE

We will not have to test hardware since there are no hardware components. We will run our servers on our local machine.

As for software testing, we can split this into two parts, frontend and backend testing. We will be using Jest for the frontend and Postman for the backend.

Jest - Web framework for testing JavaScript. We will use this to ensure our components for the frontend are rendered correctly without needing to render every component. We do this by defining the expected data to be within the component and compare with the actual data inside the component.

Postman - API development environment to test every endpoint in our API. This will test the with various types of requests that are mocked from a client and output the response we get from our API. We use this to ensure our routes are handled correctly.

3.3 FUNCTIONAL TESTING

Unit Testing: Jest, Postman test individual components

Jest can be used to test each component separately and compare the data the component loads with to the expected data. This will ensure that each component is processing each request and rendering correctly without the need to render the whole frontend and manually compare.

Postman is used to test our API endpoints and compare the expected with actual response back from our API. This is to ensure the data we send to the API is being handled correctly and sending the correct data back to be used.

Integration Testing: Jest to test minimal connected components

We can use Jest for integration testing by using Jest to call Reacts test-utils. This allows us to be able to test components interacting with our API without having to render the whole site and manually enter user interaction. Using Jest, we can simulate user interaction and test the event handlers for these cases. With this, we can compare the expected with actual output of the component data.

System Testing: Jest to test integrated system working together

Jest also allows us to create tests that will mock API requests. We can use this to send a request to our API and test the whole system from a frontend component to our database with everything in between and back. At each step in between, we can setup additional tests to have a point of failure if data mismatch happens and error log the output.

Acceptance Testing: BuilderTrend feedback

Once we have finished system testing, we can either have each member test the system themselves as if they were a client and offer feedback. With this, we can get approval from our client to ensure the system is what is wanted and performs the requirements gathered. On suggestions, we will be able to update and maintain our system and update our tests.

3.4 NON-FUNCTIONAL TESTING

Compatibility Testing

Testing for compatibility will be the biggest challenge as we do not have much access to their existing code. Our project will use the same languages (React and C sharp) to help ensure compatibility. The biggest testing for that will have to come from feedback from the company. The company will have to observe code to see if it can integrate into their existing technologies.

Performance Testing

The performance will be tested by testing how fast certain tasks can be done in the program. These tasks will include startup speed, time to upload a file, download a file and basic navigation throughout the app. In order to test for speed, we will first have to create goals about how fast certain task to perform. Determining these goals will be done by working with the client and getting feedback about how fast a certain task should perform.

Security Testing

Security testing will be limited as what we can do. Feedback from the company and constant testing of trying to break through the program will be the only thing. Also, do ensure security encryption and decryption will be done through more trusted APIs and technologies then trying to write it from scratch.

3.6 RESULTS

File upload and download as well as user ability to see a list of files and selected file details are finished. The results here led to testing the capability of the web application. Some positives are file download and upload work up to par based on speeds. Although, an issue occurs when users are visiting the website. Rendering takes longer than normal and more testing will need to be done before moving on to integrating more components within the web application.

This will not slow development down as other team members are able to still select various tasks that need to be developed, but no integration will occur until all tests are passed.

4 Closing Material

4.1 CONCLUSION

As of now we have designed our backend system as well as basic sketches for what we would want our frontend to be displayed as. We broke our backend system into three main components which are the system architecture, file system architecture, and database design. Our goals here were to keep data flow organized such that every response is predictable and the same for every request that is the same. We ensured our requirements are met such as users being able to store files, we created a system to store files remotely. The best plan as of now is what we mentioned above. We have not done development so we are not entirely sure this design will work as intended, but we plan to make adjustments when we begin development and testing.

As for our frontend design, we have designed the view of the main pages. Our goals here were to try to keep a single-page application and dynamically load in components when changing current views. This allows the webpage to stay responsive and fast loading. Our next step is to break these views into their own separate components when we begin development.

Main Work

The main work we have done is the documentation and design of the program. We have worked on the design for both the frontend and backend of the program. For the backend, we chose a system architecture known as CSR (Client Side Rendering). We have also designed the file system using basic UML notation. The frontend was designed using basic stretching software to create a UI to further see how the program would flow together. Other things that have been gone throughout the project are having basic meetings to get more clarifications and requirements about the program.

Goals and Plan of Action

Right now the goals are to get more flow diagrams of the frontend and create use cases to determine what methods we need in the program. Another goal is deciding how the backend interfaces with the frontend code. To achieve these goals going forward and to ensure a nice structure code base involves more research and meetings with the clients. Continuing to create more diagrams on the backend will also help guide the frontend code of the project too.

4.2 REFERENCES

- [1] Apiumhub. (2019). *Advantages of Test Driven Development* | *Apiumhub*. [online] Available at: <https://apiumhub.com/tech-blog-barcelona/advantages-of-test-driven-development/> [Accessed 2 May 2019].
- [2] Zhiyentayev, T. (2019). *Server-side vs client-side rendering in React apps*. [online] Async-await.com. Available at: <https://async-await.com/article/server-side-vs-client-side-rendering-in-react-apps> [Accessed 15 Mar. 2019].