Companion File Browser

PROJECT PLAN

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: 04/16/2019 V2

Table of Contents

List of Figures

List of Tables

List of Definitions

API: Application Programming Interfaces

UI: User Interface

Desktop Application: Software that runs on a user's desktop.

Web Application: Software that runs on a server and accessible through a web browser.

1 Introductory Material

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 Statement

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

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

1.3 Operating Environment

The necessary operational environment for this project will be a computer with Internet accessibility. The user may choose one of two ways to interact with the cloud system, either by using web application in a browser, or a Windows desktop application that Buildertrend currently offers.

1.4 Intended Users and Intended Uses (two paragraph +)

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

Intended uses are the creation of an account, upload files to the cloud, view file history, and download files.

1.5 Assumptions and Limitations

Assumptions:

- User has basic knowledge of English and how to use a computer.
- User has consistent Internet accessibility.

Limitations (upon client request):

- Backend programming languages needs to be C#
- Backend framework needs to be ASP.NET
- Frontend framework needs to be React

1.6 Expected End Product and Other Deliverables

Web application – December 2019:

The end product will be a 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, and 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 high level of detail as well as document how the web API handles requests.

2 Proposed Approach and Statement of Work

2.1 Objective of the Task

The objective of this project is to create a web application that will allow users to be able to upload any type of file and also be able to view or modify these files by downloading and re-uploading the modified version all while maintaining sync with the desktop application.

2.2 Functional Requirements

Functional Requirements

- The user shall be able to upload files up to 50 MB in size
- The user shall be able to view the contents of the files by downloading first
- The user shall be able to set permission roles for other users (view, edit)
- The user shall be able to browse all uploaded files and see the history of each file
- The user shall be able to download previous versions of a file

Unwanted Behavior Requirements

- If the user does not have the specified user type while trying to access a file, a warning modal shall display informing insufficient user type role.
- If the hosted server does not have sufficient storage space, the web application shall display a warning modal informing insufficient storage.

2.3 Constraints Considerations

Some constraints would be the chosen language for the project cannot be changed. It would be easier to implement a full stack application for example using mongo, express, react, and node.

Non-Functional Requirements

- The web application UI shall be built in react.js
- The web API shall be built in C#
- The max file size shall be no greater than 50 MB

2.4 Previous Work And Literature

No previous literature work done for the given project.

Similar products on the market are Dropbox and Google Drive. These allow the ability to upload files to a server and store them and obtain access by downloading them later on. Another similar feature is the ability to edit these files, although with the Companion File Browser, editing will be done through downloading and modifying on the user desktop then uploading the file unlike Google Drive.

Through previous work, our group has experience with developing API's to be able to upload and download files very similar to Dropbox and Google Drive. On previous work, we researched ways we can store files remotely on a server to handle user uploads and the process of finding that file and allowing a user to download that file. C#, or more specifically, the .NET framework has a set of great tools that will make this process easier.

2.5 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.

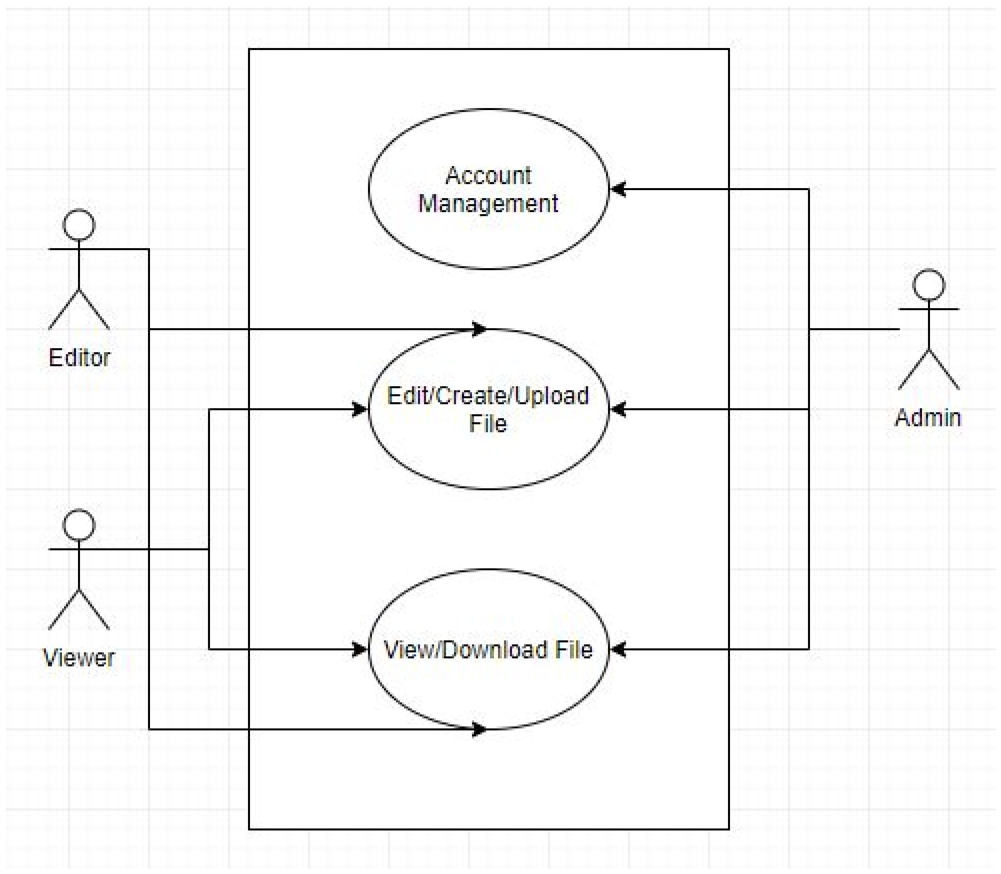


Figure 2.5.1 Use Case Diagram

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.

For our front-end, we will be following a single page web application format. The page components will be accessed through MVC structure and rendered on the server side. The main benefit for rendering on the server is it allows users to see a partial loaded UI. This in turn, allows the user to have some responsiveness as opposed to client side, where the user will not be able to access anything until the rendering is finished.

Please refer to our design document for a more detailed design of our system, which can be accessed on our team website.

2.6 Technology Considerations

The project will mainly focus on developing a web application using React UI for frontend and ASP.NET as backend upon client request. Our team is very comfortable with

JavaScript and many of us have experience with React already. This allows us to be able to develop without having to do too much research on the development tools needed.

The web API being build using C# will not necessarily set us back but the weakness would be if we were using Node, we could develop an API faster and follow similar models that exist on the front-end because the language would be the same. The strength of using a .Net Core C# web API would be ease of reading and understanding. JavaScript has many ambiguous patterns that may throw developers off when reading. A weakness may be that we need dependencies for the web API and have many dlls.

GroupMe was chosen as our main source of communication for the fast group responses. Trello was chosen to create a task board of cards each person can choose and physically see what needs to be done, what is being worked on, and what is finished. Google Drive is being used to create and edit documents.

2.7 Safety Considerations

There are a few safety issues needed to be considered while designing the application. The application would need to authorize users that have correct credentials, and prevent access from a user that has incorrect credentials.

The application would also need to provide access to a user that had a recent login. After user has been idle for a period of time (will be decided on later), the user will be forced to login again.

The user credentials is also a safety issue but Buildertrend will implement their own user security standards into the project after we hand it over to them.

2.8 Task Approach

We should first consider setting up a basic react application. Once that is finished, we can focus on getting a database management system setup such as establishing tables and dependencies. Next, we can begin to work on the web API development in C#. Once we test our API with say, Postman, we can begin connecting our front-end and back-end through API calls.

After our web application is finished we can begin to extend the desktop application to be able to stay in sync with our web application.

2.9 Possible Risks And Risk Management

Some risks that are possible would be hardware limitations of our computers during runtime of our software. The speeds may not be up to par for running multiple test cases. Time is also a risk where we will be given limited time for development.

To overcome these risks, our risk management will be to keep up with our research to avoid roadblocks in development. We will have code reviews to ensure code is up to standard for consistency. Trello will be used to delegate work to ensure we stay on track for our delivery date.

2.10 Project Proposed Milestones and Evaluation Criteria

- Milestone 1: Web interface setup
  - Test: User can access the web application online
  - Test: User can navigate around the web page
- Milestone 2: Database with tables and dependencies established
  - Test: Database calls with test data passing test cases
- Milestone 3: Develop web API for uploading files
  - Test: Ability to upload files to the server
- Milestone 4: Front and back-end connectivity
  - Test: Ability to use the web app to upload files
  - Test: Ability to download files from the web app
- Milestone 5: Desktop extension
  - Test: File sync between web and desktop
  - Test: Upload and download files
- Milestone 6: All functional requirements finished
  - Test: Demonstrate each functional requirement

2.11 Project Tracking Procedures

We will be using a trello board to allocate “To Do” tasks and delegate work to every member. We will also maintain weekly team meetings for at least one hour to cover what is done each week and what each member will be working on.

2.12 Expected Results and Validation

The desired outcome is a working web application that will have each of the functional requirements working as intended. Our high-level test is to demonstrate each requirement as a form of a product use case from step 1 to completion, which will be defined later during the semester.

2.13 Test Plan

For our test plan we will have three main test plans; software testing, functional testing, non-functional testing. A brief overview for them are as follows, software testing will be our main way of testing our software as we develop front-end and back-end. Functional tests will be a series of tests in order, where we proceed only once we finish the previous test. Non-functional tests will be throughout development as well, but ensure quality is met.

2.13a Software Testing

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.

2.13B 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.

2.13C 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

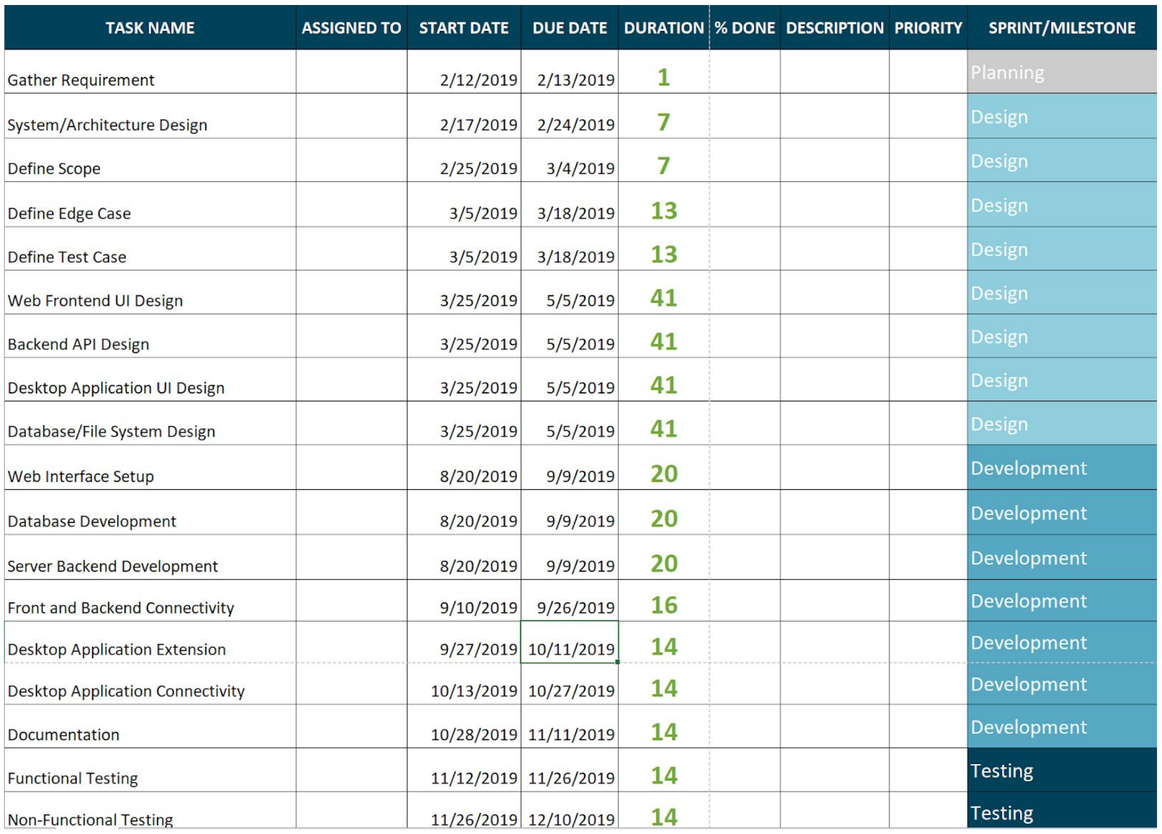
Security testing will be limited as what we can do. Feedback from the company and constant testing of trying to break through the program will be the only thing. Also, do ensure security encryption and decryption will be done through more trusted APIs and technologies then trying to write it from scratch.

2.14 Standards

The app is a cloud base file sharing system meaning the standards that we adopt in the project will be file transfer standards. This program is an extension of Buildertrends existing system, meaning standards for registration will not be focused on as Buildertrend already has a registration system with standards that they follow already. As of right now it is not set in stone on what technologics we will used for file transfer. Some technologies that are being considered and looked into are FTP(File Transfer Protocol), HTTP (HyperText Transfer Protocol), Ajax calls, and other file apis. More research is needed on each standard before we make a definitive answer.

3 Project Timeline, Estimated Resources, and Challenges

3.1 Project Timeline



TASK NAME	ASSIGNED TO	START DATE	DUE DATE	DURATION	% DONE	DESCRIPTION	PRIORITY	SPRINT/MILESTONE
Gather Requirement		2/12/2019	2/13/2019	1				Planning
System/Architecture Design		2/17/2019	2/24/2019	7				Design
Define Scope		2/25/2019	3/4/2019	7				Design
Define Edge Case		3/5/2019	3/18/2019	13				Design
Define Test Case		3/5/2019	3/18/2019	13				Design
Web Frontend UI Design		3/25/2019	5/5/2019	41				Design
Backend API Design		3/25/2019	5/5/2019	41				Design
Desktop Application UI Design		3/25/2019	5/5/2019	41				Design
Database/File System Design		3/25/2019	5/5/2019	41				Design
Web Interface Setup		8/20/2019	9/9/2019	20				Development
Database Development		8/20/2019	9/9/2019	20				Development
Server Backend Development		8/20/2019	9/9/2019	20				Development
Front and Backend Connectivity		9/10/2019	9/26/2019	16				Development
Desktop Application Extension		9/27/2019	10/11/2019	14				Development
Desktop Application Connectivity		10/13/2019	10/27/2019	14				Development
Documentation		10/28/2019	11/11/2019	14				Development
Functional Testing		11/12/2019	11/26/2019	14				Testing
Non-Functional Testing		11/26/2019	12/10/2019	14				Testing

Figure 3.1.1 Project Timeline

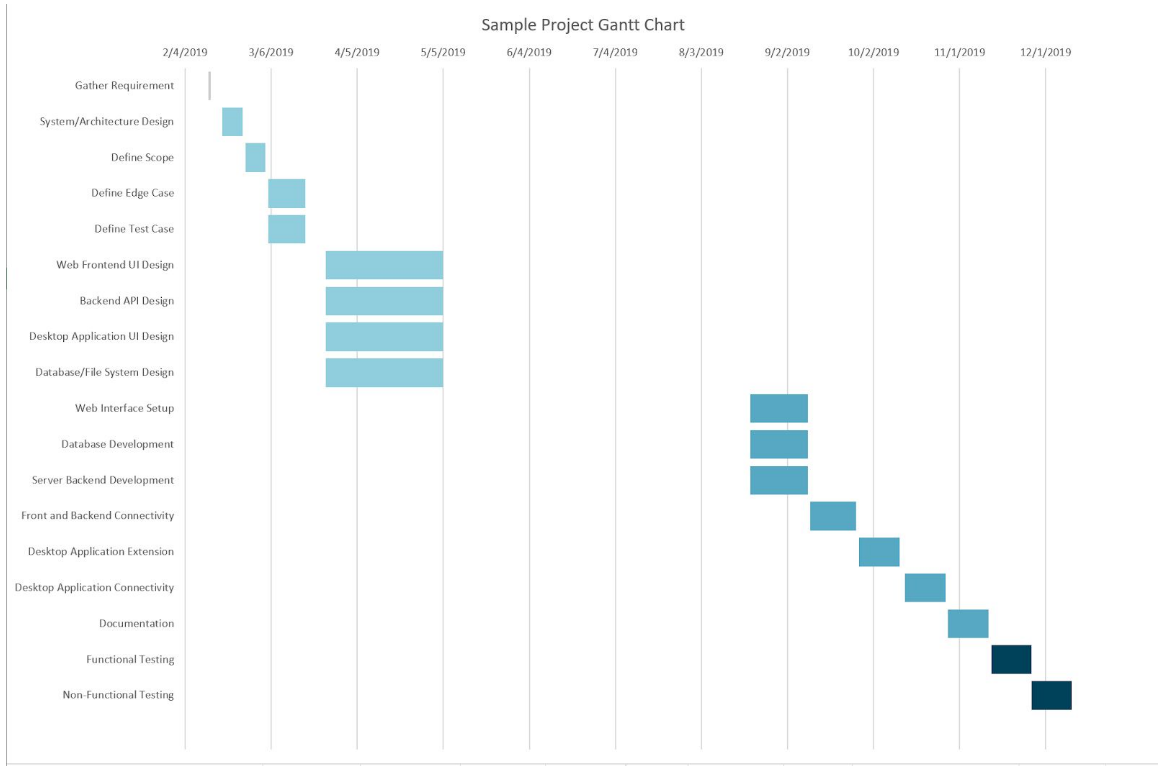


Figure 3.1.2 Project Gantt Chart

The project will be split into four parts, Planning, Design, Development, and Testing. Planning and Design will be will be complete by first semester, where Development and Design will be work on in the second semester.

More specifically, the first semester will be used to understand the project, and finish designing the system. The design should include high level from architecture and low level to class diagram. The design should also include edge cases and test cases; this will help speed up the development and testing.

The first day of second semester will be the beginning of development stage. At the end of the development, we will complete any documentation that is not yet finished. After development, we will spend time work on final testing and possibly perform performance enhancement.

Specific task are split by large system component, web frontend, server backend, database/file system, and desktop application. Most of the task in design and development are based off of system component. There is also task related to connecting system component together.

3.2 Feasibility Assessment

Foreseen challenges of this project could include, synching of files, integration, limited accessibility to libraries, frameworks, and APIs. The web application and desktop application will have to display the same updated information, meaning if you make a change on the web application you also see that change updated on the desktop application which requires synchronization. Making sure for example that if one user is using the web application and uploads a file or edits ones and the other user is using the desktop application at the same time that he/she can see the changes in real time without having to refresh the application could be challenge.

Integration could be a great challenge. Creating a program that can easily integrate into their existing codebase could be a challenge because Buildertrend is a private company. This means we have limited access to their existing code and documentation, furthermore, making a program that can easily integrate into their existing code could prove to be hard.

Buildertrend being a private company means we could have limited access to certain libraries, frameworks, and APIs that one might originally use to ease up on some of the coding. For example there might be a nice library that makes file synchronization easy but the library may not be completely open source and therefore may not be usable to us due to certain legal issues with BuilderTrend.

3.3 Personnel Effort Requirements

TASK NAME	PROJECTED HOURS	SPRINT/MILESTONE
Gather Requirement	12	Planning
System/Architecture Design	24	Design
Define Scope	20	Design
Define Edge Case	20	Design
Define Test Case	28	Design
Web Frontend UI Design	12	Design
Backend API Design	12	Design
Desktop Application UI Design	12	Design
Database/File System Design	12	Design

Web Interface Setup	24	Development
Database Development	24	Development
Server Backend Development	24	Development
Front and Backend Connectivity	40	Development
Desktop Application Extension	40	Development
Desktop Application Connectivity	48	Development
Documentation	48	Documentation
Functional Testing	48	Testing
Non-Functional Testing	48	Testing

Table 1: Projected Hours for Completing Tasks

As shown in Table 1, this lists the projected hours for all parts of our design (Note: this is an initial projection and all estimated times are subject to change). Starting with the gathering requirements task taking 12 hours since we were provided by our client a detailed list of what needed to be accomplished and most of this time is dedicated to outliers in the project that our group has questions on.

Moving into the design sprint: The System/Architecture Design will take 24 hours as coming up with a good design can take time and if done properly will save time in the long run. Both the defining scope and edge cases each will take 20 hours. These are not huge tasks but will take time in order to do it right and make sure for example, that all the edge cases have been found. Defining test cases will take reasonably longer as creating good test cases now can save a lot of time in development and to make sure that testing is accurate. The next four are all related to design and will be done together as to make the design fluid throughout the project. Completing a fluid design will take time especially going back and forth with our client to confirm the design which is why we allotted 48 hours total to complete this.

The next phase of our project is development and will start when school begins in the fall. Overall, development will take longer (a good layout/design helps mitigate this) due to the nature of programming. The first three tasks in development will be worked on simultaneously as we felt those related to each other with each taking 24 hours to complete. These tasks lay the foundation for the rest of the project. Next we have the Front and Backend Connectivity taking 40 hours to complete. This involves linking the front and backend together which could go smoothly or take some time so 40 hours felt like a good compromise. Following this is extending the desktop application to include features that the web app will have. Again, 40 hours should be a good amount of time as we will just be recreating features that we already completed for the web app. Then we have to connect the desktop app to the web app and make sure that information is being updated and passed along correctly. This again could go really well or not, so 48 hours felt reasonable if we ran into a few problems.

The next phase is documentation, good documentation takes time to create and that is our goal which is why we allotted 48 hours to round up all the documentation that we have and put it all in one place that will be easy to understand. After this we will have our testing of the project using the edge/test cases that we created earlier on. We allotted 48 hours for each type of testing because if we found something that isn't correct then we should have time to get it completed.

3.4 Other Resource Requirements

Mainly having the existing code base and documentation for the desktop application will help aid in developing the extension. By having API signatures for the web API, this will help speed up development for the web application.

3.5 Financial Requirements

No financial requirements will be needed.

4 Closure Materials

4.1 Conclusion

The issue of having a place to store files online for multiple people while giving each person different access control will hopefully be solved with the Companion File Browser. The Companion File Browser will allow users to upload files to the cloud and store them giving access to other users based on set permissions to be able to read and/or modify these files. These files will stay in sync with a desktop application as well to allow users to be able to download and upload on either the web or their desktop.

The solution is to create this web application that will communicate with a web API that handles requests from the user and processes these on a server. We expect our solution to meet every functional requirement mentioned above as well as be completed by the end of December of 2019.

4.2 References

At this time there are no references other than meeting notes from client meetings with Kirit Chandran from Buildertrend. References towards development styles and practices will be added throughout the two semesters.

4.3 Appendices

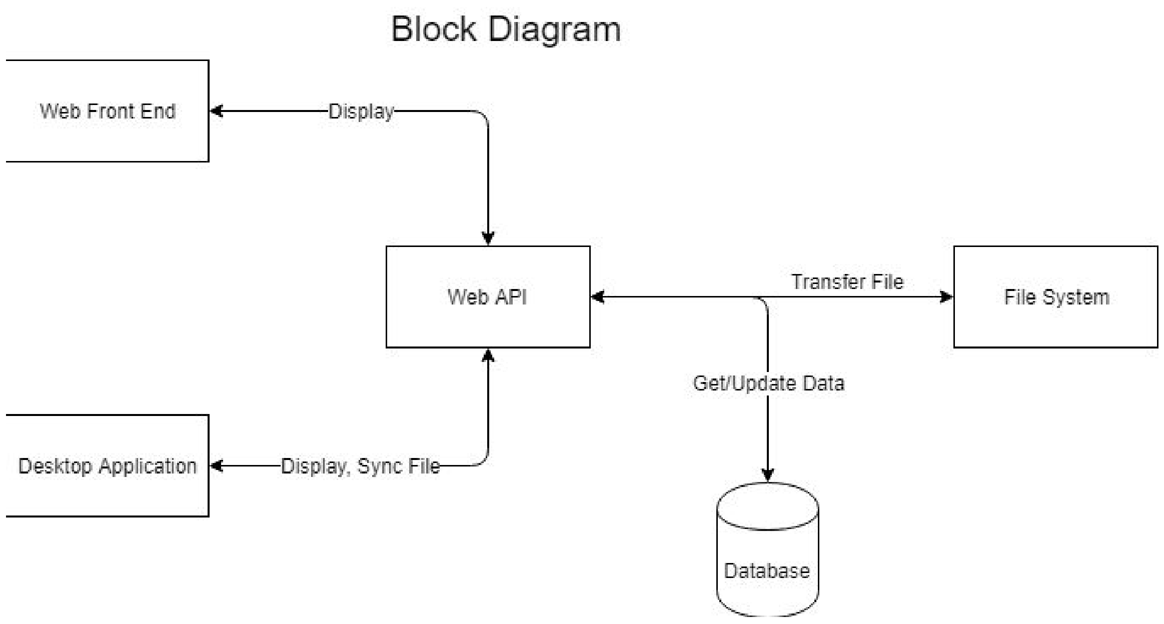


Figure 4.3.1 Block Diagram