Companion File Browser

FINAL REPORT

SD-DEC19 Team 11
Client: Buildertrend (Kirit Chandran)
Adviser: Mai Zheng

Team Members:
Christopher Bui
Brian Chodur
Luke Stoll
Bei Zhao
Zhenyu Zhao

Email:
sddec19-11@iastate.edu

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

Revised: 12/10/2019 V1

Table of Contents

List of Figures and Definitions **3**
List of Definitions 3
Figures 3
Introduction **4**
Executive Summary 4
End Product and Other Deliverables 4
Requirement Specification **5**
Functional Requirements 5
Non-functional Requirements 6
Use Cases 6
Project Design **8**
Objective of the Task 8
System Constraints 8
Design Plan and Description 9
System Analysis 13
Implementation Details **14**
Back-end Design Implementation 14
Front-end Design Implementation 15
Desktop Implementation 16
Technologies used 16
Software Techniques 17
Testing, Validation, and Evaluation **18**
Interface Specifications 18
Hardware and Software Used 18
Functional Testing 18
Non-Functional Testing 19
Process 19
Results 19
Conclusion **20**
Appendix I - Operation Manual **21**
Prerequisite 21
Source Code 21
Application Setup 21
Appendix II - Postman API Testing **22**
Appendix III - Web Application UI **28**

List of Figures and Definitions

List of Definitions

API: Application Programming Interfaces

UI: User Interface

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

BTconnect Application: Buildertrend's proprietary desktop application.

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

Figures

Figure 1: Use Case Diagram 7

Figure 2: System Design Block Diagram 9

Figure 3: File System Design 10

Figure 4: Database Schemas 12

Figure 5: Postman Example API Request 14

Figure 6: Home Page UI 15

1. Introduction

1.1. Executive Summary

Buildertrend has a companion desktop application that users can install on their local machine that allows a user to edit files locally. This project is to expand the functionality of this current application to include a file browser where users can upload and download files. The project will provide user easy collaborations, error recovery and high accessibility

Although there are a lot of public file sharing web applications such as Dropbox, Google Drive, OneDrive and so forth, from Buildertrend's points of view, having their own file sharing software will secure their sensitive files from third party software. An additional benefit is that it allows Buildertrend to create custom add ons without costing a lot of extra money. From a similar point of view, using a third party's file sharing application may cost more as well. Therefore, owning a file sharing web applications may also reduce Buildertrend's budget on non-proprietary software.

The necessary operational environment for this project will be under a computer with Internet accessibility. The users could choose two ways to interact with the system which is using the web application in a browser, or a desktop application that Buildertrend currently offers.

The intended users are clients under Buildertrend who will want cloud file storage with sharing functionality across a team or company. Intended uses will be the ability to create an account, upload files to the cloud and download files.

1.2. End Product and Other Deliverables

Throughout the last two semesters, we have generated a design document, project plan and many weekly reports. A project poster was also created that summarized the work over the past two semesters. The main deliverables are:

- A web application, which has current features; register account, log in, log out, upload files, delete files,download files, rename files, create folders, rename folders, delete folders, view file details, view file version histories, add comments to file, edit file comments, delete comments and view deleted files.

- Expanded capabilities' desktop application which can sync folder, upload files, download files and send out notifications.

2. Requirement Specification

2.1. Functional Requirements

1. **A new web application that has the following:**
    - Web interface to create users.
    - Web interface to create different user types with view/add/edit permissions.
    - Web interface to set permission roles for other users (view, edit, download, delete files).
    - Web interface that allows users to upload files up to 50 MB in size with all file formats supported.
    - Web interface that allows users to browse all uploaded files and history of each file.
    - A new page that can view file information.
    - A history page that allow users to see all previous versions made to the file with a link to download that version of the file.
2. **Expand the capabilities of the btconnect application:**
    - The newly created sync folder when viewed in windows explorer should have a folder with the Buildertrend logo on the top right of it.
    - Once the folder is created, it should display all the files that are currently on the web-application that the user has access to.
    - The files should not be downloaded by default, they should be in a grayed-out state and give the user the option to download them,
    - If a user downloads and opens an existing file, that file in the web-application should have an indication that it is being edited.
    - The user should be able to indicate that they have finished editing the file and want to sync it back to the web application.
    - The user should have the capability to drag and drop any file type into the folder.
    - New version alert.
3. **Build a WebAPI interface that the btconnect application will connect with to send and receive data.**

2.2. Non-functional Requirements

1. **Maintainability** - Code and documents should be easy for future developers to understand.
2. **Usability** - The application should be user-friendly and easy to use without a detailed explanation about how the application works.
3. **Security** - The documents and user information uploaded into the system should only be accessed by people who have permission.
4. **Availability** - The application should be available at any point as long as there is an internet connection.
5. **Data Integrity** - The application should display and send accurate data all the time.

2.3. Use Cases

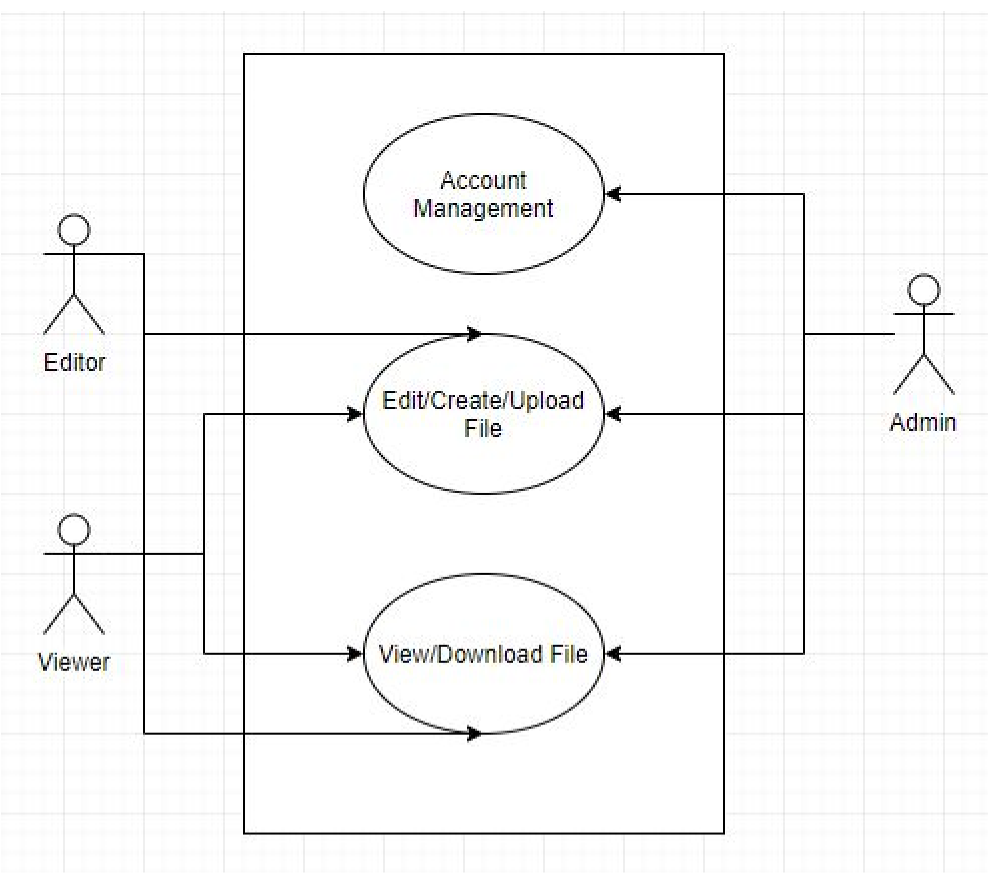


Figure 1: Use Case Diagram

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 download access to all files belonging to the company.

To secure the companies' access, an access code is used to manage the whole process. Once a new company added into the system, Buildertrend could create an access code for that company to register the highest level access account for that company's ID. This account will be able to create an access code and set up permission level for new users who are going to register accounts for the company.

3. Project Design

3.1. Objective of the Task

The desired outcome of the project is to extend the current desktop application with a web application. The scope of the web application and the expanded capabilities of the desktop application is defined by functional requirements described above. Both applications should have all these functional requirements and non-functional requirements implemented in the final prototype.

3.2. System Constraints

1. **Labor Resource:** The team has fixed amount of people who have their own strengths. Except classes, many members of the team are working, finding jobs or applying for graduate schools. The balance between school, work and the like have been difficult throughout the past two semesters.
2. **Technical**:Buildertrend chose the technology that they would like to use in the product which are react.js frontend framework, SignalR to communicate between desktop and web application and C# as the basic language.
3. **Time:** The product must be completed before the final presentation of the Senior Design class in the second semester.
4. **Integration with Pre-existing Technology:** Buildertrend supplied a desktop application to expand the capabilities. Understanding their application was also a difficult topic for the team to overcome.

3.3. Design Plan and Description

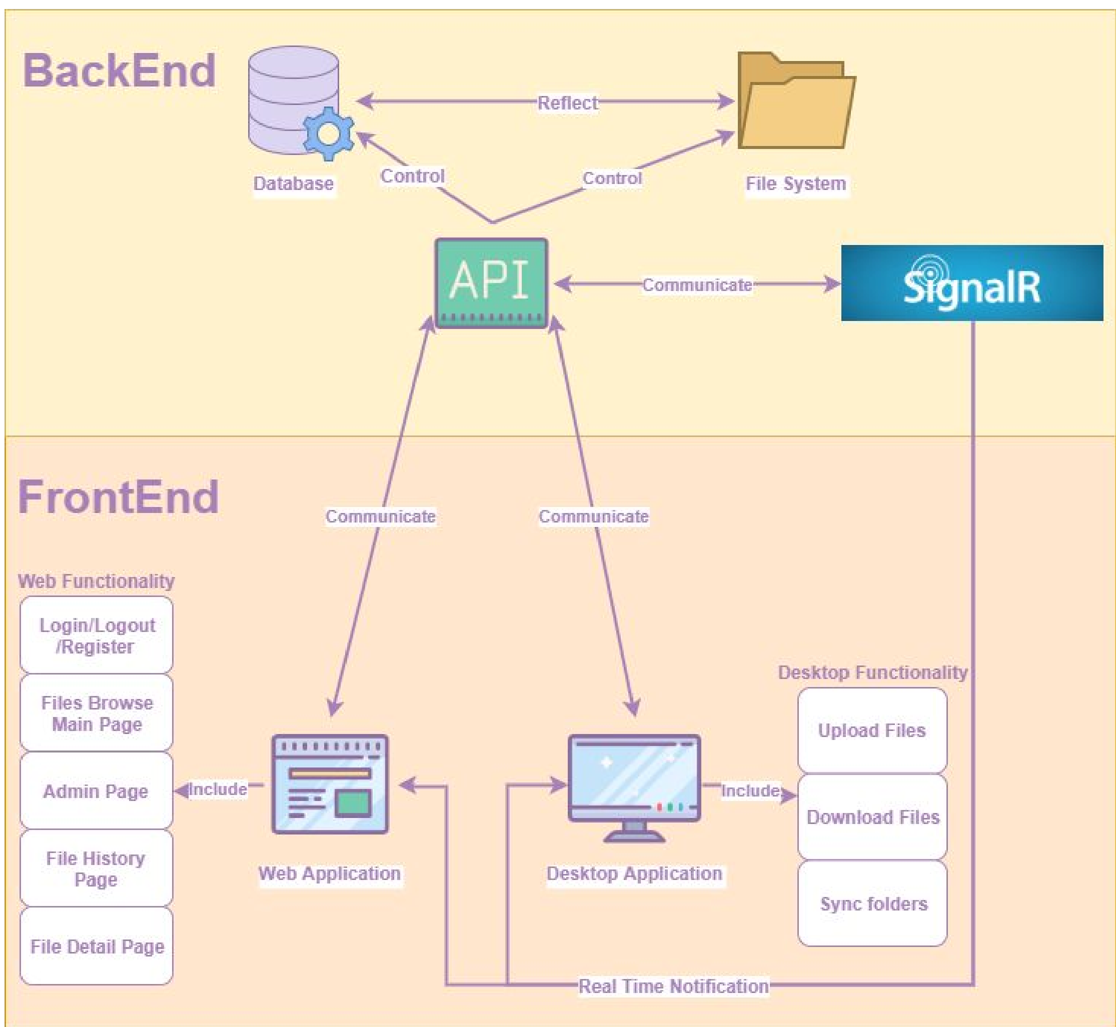


Figure 2: System Design Block Diagram

Figure 2 shows the overall system design diagram for the system. There are a total of six different portions in the project. The first portion of the project is the database. Microsoft SQL Server is used as the database holder. The database could reflect the files, folders and users information. The changes in the database are based on different API calls. The second portion of the project is the file system. The local file system is used to simulate the company folders. The third portion of the project is the Restful API. The RestAPI is used to control the information in the database and file system, send data to the display in the web and desktop application and set up communication protocols to the SignalR service. The fourth portion of the project is SignalR. SignalR is used to send real time notifications to the web and desktop application and return reactions in the web and desktop application back to the Restful API. These four portions constitutes the backend section in the project.

The fifth portion of the project is a web application. It includes all web user interfaces required by the functional requirements, which are login/logout/register pages, file browse main page, admin page, file history page and file detail page. It also communicates with Restful API and receive real time notification sent by SignalR. The sixth portion of the project is a desktop application. It includes part of expanded capabilities required by functional requirements, which are upload files, download files and sync folders. Same as the web application above, it communicates with Restful API and receive real time notification sent by SignalR as well. These two portions constitutes the frontend section in the project.

File System Design

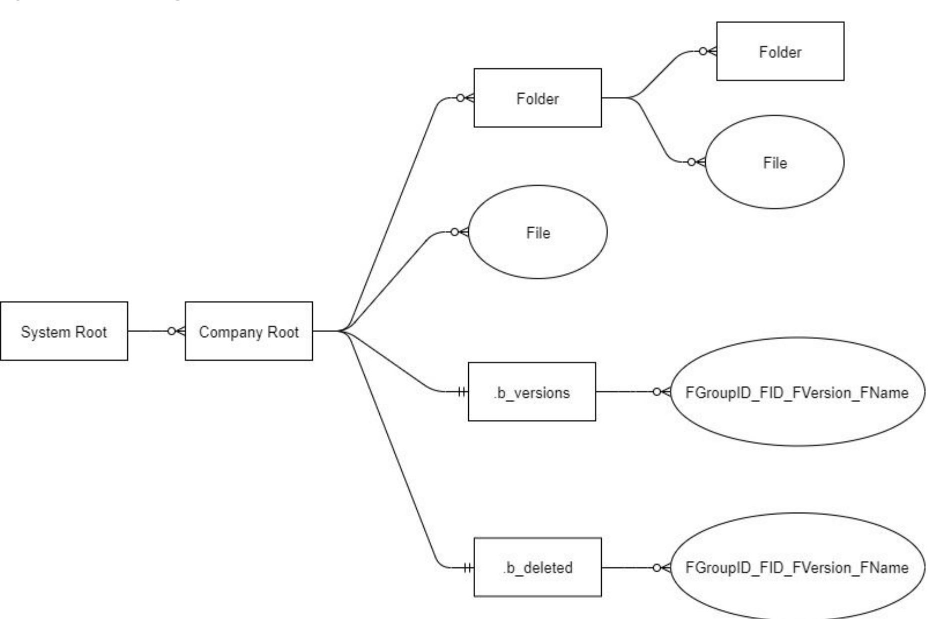


Figure 3: File System Design

There will be a folder that represents the system storage root folder. In the System Root, there will be multiple folders, and one unique folder which will represent one company in the system, all users under the company can view the company folder.

As a feature, the system will allow users to roll back the older versions of the file, or recover deleted files. Therefore, in the Company Root Folder there will be a hidden folder call .b_versions which stores older versions of the file. The file name in the version folder will contain the fileGroupID, FileID, FileVersion, and the FileName to avoid naming collision. Upon switching the file version, the application must first save the current file in the version folder then identify the old version file to move it to the folder that the current file previous located. At the end, the application will update the database accordingly.

There is a folder name .b_deleted which stores all the deleted files and folders. The deleted file/folder will be renamed to fileGroupID, FileID, File Version, and the file name. When time comes to delete a file, the application just simply moves the file to the deleted folder with generated deletion name, and updates the database accordingly. To recover the file, it simply puts the file back in the original spot, if there is a naming collision, it will append the timestamp of the current time.

The reason that file system is designed this way using a filegroup is to better keep track of the series of a file version. In particular, the cases where after file A with multiple versions got deleted, user creates another file B with the same name as file A. The file B should not able to roll back to the versions that belongs to file A.

API Calls

Following are the main API calls used in the system. They can be fetched by the web and desktop frontend to manage the account information, files, and folders information based on users' requests.

Controller Name	**Action Name**
AccessCodeController	● Create ● List
AccountController	● Register ● Login ● Init
AdminController	● Users ● UpdateAccessLevel ● DeleteUser
CommentController	● CreateNew ● List ● Edit ● Delete
CompanyController	● CreateNew
DeletionController	● ListDeleted ● RecoverFile ● RecoverFolder
FileController	● Download ● UploadFiles ● DeleteFiles ● Rename ● Move ● Checkout ● Checkin
FolderController	● CreateNew ● Delete ● List ● Rename

PermissionController	● Check
ValuesController	● Get
VersionController	● List ● Recover

Database Schemas

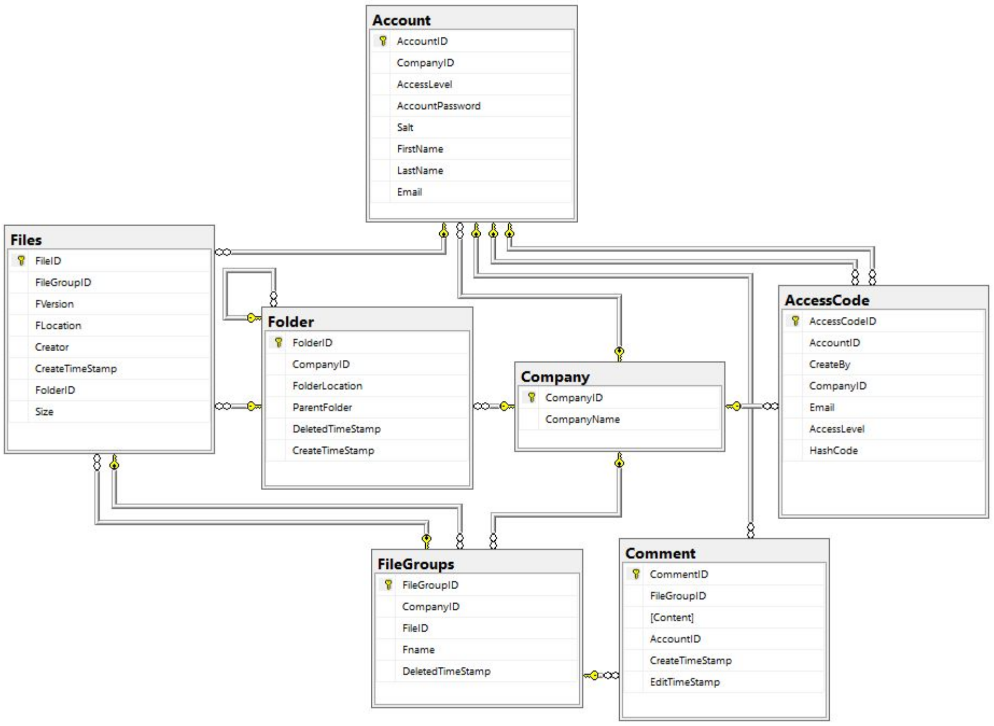


Figure 4: Database Schemas

The current design of backend uses database for information API call (list files/folders, create/list/edit comments) to avoid as much file system IO operation as possible.

7 types of records:

Company, Account, AccessCode, Folder, FileGroups, Files:

• Company keeps track of basic company information such as CompanyName

• Account which contains the login information, name, permission level and which company the account belongs to

• AccessCode is needed to register an account, it contains information such as company, permission level of the new account, and the creator of the AccessCode

• Folder contains information about which Company belongs to, parent folder, folder location, and delete timestamp indicate the folder deletion time

• FileGroups are responsible to keep track of file versions. It contains which company it belongs to, information about the current version file and a deletion timestamp which indicates the file deletion time

• The File contains information about which FileGroups it belongs to, current location, file versions, create a timestamp, the creator, and the size of the file

3.4. System Analysis

Backend Analysis:

For the database, Microsoft SQL Server was chosen as the database holder because its installation is streamlined, it has policy-based management to detect security policies that are non-compliant and lower cost of ownership, which can be easy and safe for Buildertrend to install and use. For the system access safety, Json Web Tokens were used as the authentication technique because it has a fast response time, simple to use and can be utilized across different services. For the real time notification, SignalR was used to communicate between Restful API and applications because it is not only in the requirement of Buildertrend but also is available for a variety of platforms and easy to set up. This is suitable to the web and desktop application development and will help the client to continue developing the projects.For the web API, Restful API is used because it has a clear separation between the client and the server for developers to manage, which also improves the portability of the interface to other types of platforms, such as developing both desktop and web application in the project.

Front-end Analysis:

For the framework of the application, react.js is used because it is stable while small changes in child structure will not affect their parents, it ensures faster rendering and includes a lot of useful format packages, which saves a lot of time for the UI developer. Since React was the chosen page application framework for the rendering technique, client side rendering was used because once the page loads, all subsequent navigation within the site will be fast as no more pages needed to be loaded from the server. This is especially important for the project as users may want to edit multiple files in one site. Increasing navigation speed after the site is loaded will make the clients finish their work faster.

4. Implementation Details

4.1. Back-end Design Implementation

API:

For the API, a general .Net Framework API template was followed. For every module, such as files, folders, and users, a controller is created that handles all endpoints regarding the module. Within each endpoint, the corresponding engine is called and this engine will finally call the handler for the module. This design separates the business logic from the controllers into their own classes. The general flow follows as below:

Web Browser sends request -> API Endpoint -> Engine -> Handler -> Database

Most of the APIs require a bearer token to access the API. The frontend should get the token from the Login API calls. The token has a valid time, after calling an API, the tokens valid time will automatically extend (extend only works within one backend application cycle, which means that the token will fail if the backend restarted on the extension). The token contains information about the userID and the validation time. There are only a few APIs in Account that do not require a bearer token. Refer to Figure 5 below.

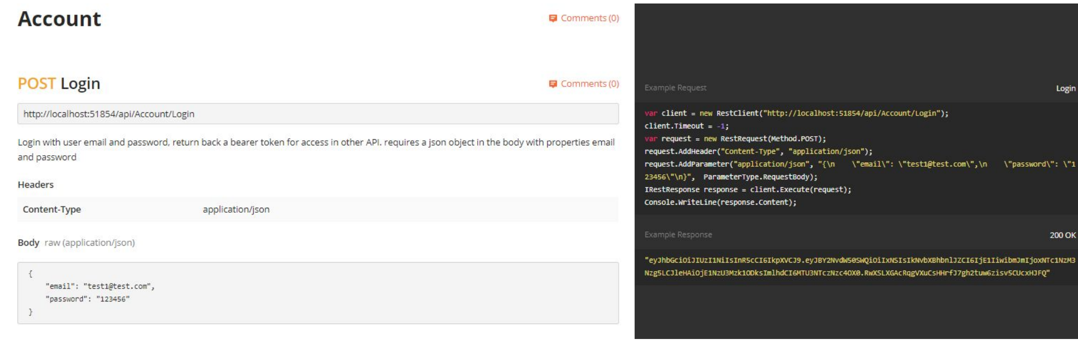


Figure 5: Postman Example API Request

SignalR Hub:

For real-time notifications, a SignalR hub class is created. This class overrides two functions for when users connect and disconnect. Using this, it can separate each user into a grouping by company. The structure to hold this information is a dictionary so it can quickly look up each company’s active user.

Within this SignalR hub, multiple events are defined that will trigger events that each user is listening for. For example, an event called FileInEdit is defined. This event will send every user in the company the file information that is now being edited.

For the web and desktop applications, once they establish a connection, they can override how they want to handle the incoming events. With this, an event handler function is defined for FileInEdit and creates a notification.

4.2. Front-end Design Implementation

Redux Store

Redux was used as a global state store for any components that may need data and do not share any common relation. This was used to store whether the user is authenticated, a current list of files displayed, current folders, and the current folder id. The reason behind storing this was to allow the SignalR component, real time notifications, to be able to understand the current state the user is at.

Components

Many react components were created to separate each functionality into its own class. This allowed the team to have minimal code and high reusability. The components are broken down into main bigger components called the page components. As an example; DeletionPage is the component that contains the necessary components for that page. The other pages were Home, Vision, Admin, and few other components were separate. Each page contained its own list and item component. So DeletionPage contained a deletion list, deletion file and deletion folder. The plan was to originally create one big list but was changed. This was changed to insure that each component has only a limited number of responsibilities and enforcing the single responsibility principle. Refer to Appendix III for additional images of components.

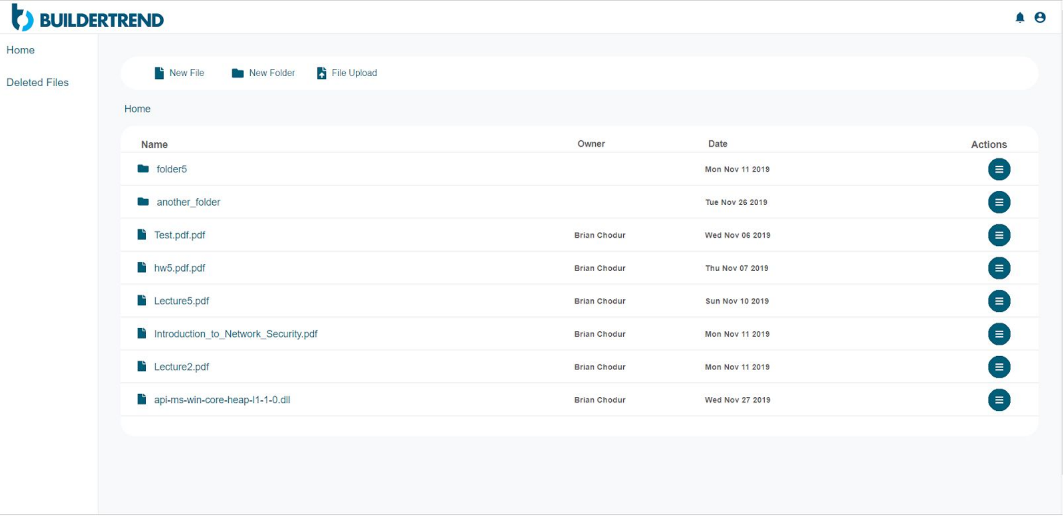


Figure 6: Home Page UI

Entity

To be able to easily parse data from the server, a shared model was created that is the same on the API and react application. This ensures the data expected will always be in

the correct format, allowing it to send information to the server as well, without the need to manually map each object.

Handlers

Handlers are an abstraction of all of the API requests to the server. To maintain low coupling, similar API requests are grouped into their own handlers. For example, all requests handling files will have its own handler. This allows for easy maintainability when changes need to be made, as changes would only need to be made in one place versus every place that calls the endpoint.

The handlers also use promises and async methods to insure accurate data back and to avoid undefined data. TypeScript is an async language meaning that instead of line by line TypeScript will not wait for one line to finish before the other. This is a problem for getting information from the server so to combat this problem, async methods are used. Promises are also used as a safeguard for if the server doesn't return anything.

4.3. Desktop Implementation

There is limited desktop functionality completed including; login, create sync folder, download/upload file and create folder. The base infrastructure for calling an API is completed.

It was a challenge to find the best way to let users interact with the application, especially for download and upload in the context of Windows file explorer. The solution is to add button to the right click menu. This was challenging because the application would need to edit the Windows Registry via Microsoft Win32 module.

There are two steps:

1. Add registry subkey(menu button) to all types of files under HKEY_CLASSES_ROOT*\shell
2. Add registry subkeys (core function/sub menu button) to HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\CommandStore\shell

The desktop application also has a btRequestHelper that uses httpclient to communicate to the backend API. The purpose of btRequestHelper is to make calling the API easier by providing the name of the controller, action, parameter, get or post request, and httpcontent if needed.

For Apply custom folder icon, the application needed to change the desktop.ini in order to change the folder icon.

4.4. Technologies used

A brief breakdown of the technologies used can be broken down to front-end and back-end technologies.

Front-End

- React - Library to build the user interface
- Redux - Global application state store

Back-end:

- .Net Framework - Rest API framework portion
- SQL Server - Database
- SignalR - Socket Server for live updates

4.5. Software Techniques

Data Structure

- HashSet and Dictionaries used for quick lookup for caching.
- Arrays are used to store the lists of different types of data by mapping method.
- Optional used for nullable primary type

Programming Concept

- Objects are used to store different types of data sent from HTTP requests.
- Inheritance are used when two frontend classes have duplicate methods.
- Backend API Controllers separated into own classes
- Abstraction for engines and handlers to remove business logic from controllers
- Optional parameters and overload functions/methods in the function to reduce duplicate code
- Using hashing to generate the json web token and user password storage (with salt) for better security.
- Throw and catch exception in both frontend and backend for better software flow control
- Http status code to communicate the state of the api calls.

Algorithm

- BFS used for pulling information to the sync folder in the desktop application.

5. Testing, Validation, and Evaluation

5.1. Interface Specifications

There are no specific hardware interfaces that were used. Per requirement, users will need a browser application that can access the web such as Google Chrome. SQL Server will be used for database storage, React.js will be used for the frontend user interface, and .Net framework will be used for the API.

For testing, various browser clients were used on local machines, and for software; Postman was used for testing the restful API, and mock data was used for testing the frontend.

5.2. Hardware and Software Used

There will be no hardware tests as there were no hardware components. The servers were run on a local machine.

As for software testing, it can be split into two parts: front-end and back-end testing. Mock data was used for the front-end and Postman for the back-end.

Mock Data - Web framework for testing JavaScript. This is used to ensure the components for the frontend are rendered correctly without needing to render every component. This is done 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 the API. This tests various types of requests that are mocked from a client and outputs the response from the API. This is to ensure the routes are handled correctly

5.3. Functional Testing

Integration Testing: Postman & Mock data

Using postman, a saved request call to the API is created. With each saved request, specific data in each request can be saved to test the results and make sure they will always be the same. This ensures are API endpoints are functional and pure, results are always the same with the same input. Mock data supplies hard coded data to ensure the front-end components render correctly and maintain their state throughout the lifecycle of the application.

System Testing: End-to-End testing

Here, each major task finished was dedicated to test all systems working together. Previous test data from the database was used to ensure data was the same for each test. This allowed the team to test if components are rendered correctly. Afterwords, the functionality of the new task is tested, for example, file rename. If this works as expected end-to-end, other functionality is tested to ensure every component still works.

Acceptance Testing: BuilderTrend feedback

The code will be shipped to BuilderTrend for feedback.

5.4. Non-Functional Testing

Compatibility Testing

Not many tests were implemented here as Buildertrend will need to integrate the application themselves into their existing systems. Specific tools such as .Net Framework and React were used such that the same tools allow for easy integration.

Performance Testing

Google Chrome offers great analytical tools for performance and speed tests. Loading times of the web application were under ~3.0 seconds which seemed to put it above average loading. For the api endpoint calls, optimized database queries were used to maintain under 200 ms. This number retains user attention and allows the feeling of responsiveness.

Security Testing

Security testing was done by testing different users making actions against their current access level. Each API endpoint that requires a valid user ensures the request contains a valid secure token. This then proceeds to do checks on whether the user has the correct access level to perform the action.

5.5. Process

The process we used can be broken down into four main steps. Each member selected a given task to develop and used a form of test driven development. This is where tests were created based on the functional requirements. The goal was to develop against these tests until they all pass and create more tests until each task has 100% code coverage. Once testing passed, each task was integrated with another task and began testing components working together.

5.6. Results

File upload and download can vary between different internet connection speeds. This made it hard to test accurately how well the API performs. Overall, all planned test cases passed and covered all predicted edge cases. Exceptions are also moved up to the API that send a response back to the user with a server error.

6. Conclusion

After many iterations on system design and architecture, the file companion application is finished. Although every deliverable was not met for the desktop extension portion, the current progress allows for Buildertrend to either easily integrate the changes into their own systems or use the designs as a reference. This was considered a successful project as all core features have been implemented, tested, and ready to release to Buildertrend.

Some issues that pushed back development were multiple iterations of the back-end systems during development. The design documents from the first semester were great on paper until development started. The back-end was heavily dependant on the file system instead of the database. This pushed back the development for connecting the front-end and back-end. With this issue, the back-end was completely redesigned mid-development.

7. Appendix I - Operation Manual

7.1. Prerequisite

- SQL Server 2017
- Microsoft Visual Studio 2019
- Node.js
- Knowledge on how to send HTTP Requests and necessary tool (browser or Postman)

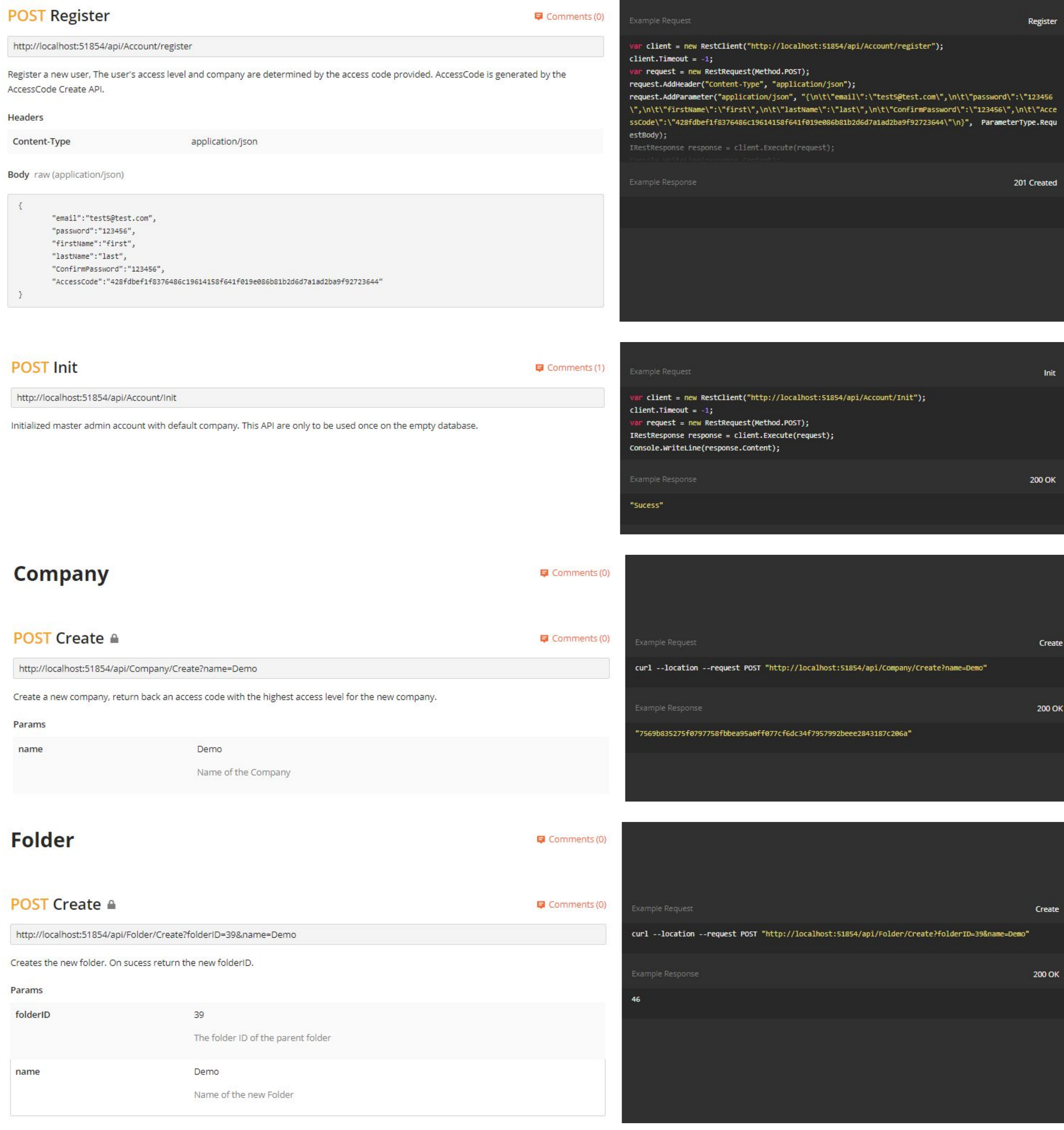
7.2. Source Code

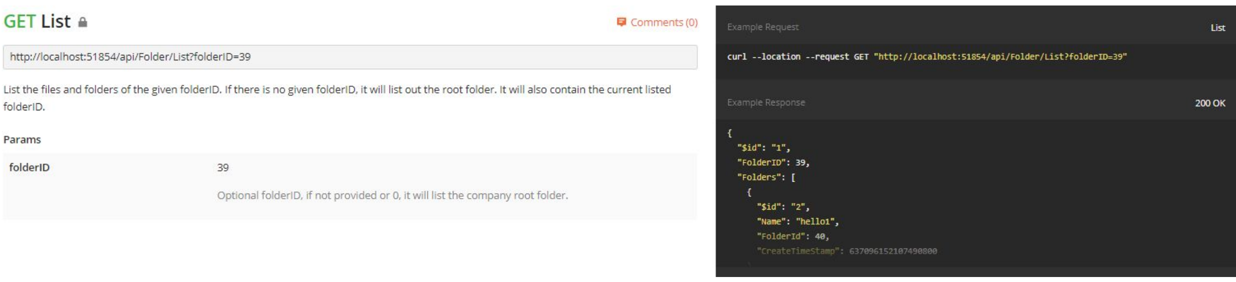
The link to download or clone the repository is:
https://git.ece.iastate.edu/sd/sddec19-11/tree/master

7.3. Application Setup

1. Microsoft SQL server setup
    a. Create a new instance of Server with a database called “fileDB”
    b. Runs all the sql script inside of the folder call Database. This sets up the initial database table and relationships
2. Setup frontend
    a. Open a terminal, preferably Node
    b. Navigate to /CompanionApp.Web/ClientApp/
    c. Execute the commands
        i. npm install (Installs dependency packages)
        ii. npm start (Run the frontend for the web application)
3. Setup backend
    a. Open Visual Studio 2019 and open nuget package console
        i. Execute the command “Update-Package -reinstall” to install the package dependencies for the web api
    b. Create an ADO.NET Entity Data Model that is pointing to the database created in step 1(Documentation is stored in the team google drive)
    c. To host the application, select the play button in Visual Studio 2019
4. Send an HttpPost request to /Account/Init
    a. This generates the super user account that is able to use the web application to create companies and users
5. Navigate to localhost:3000
    a. Login as admin@Buildertrend.com, nimba2019 for the master account

8. Appendix II - Postman API Testing





GET List

Comments (0)

http://localhost:51854/api/Folder/List?folderID=39

List the files and folders of the given folderID. If there is no given folderID, it will list out the root folder. It will also contain the current listed folderID.

Params

folderID	39 Optional folderID, if not provided or 0, it will list the company root folder.

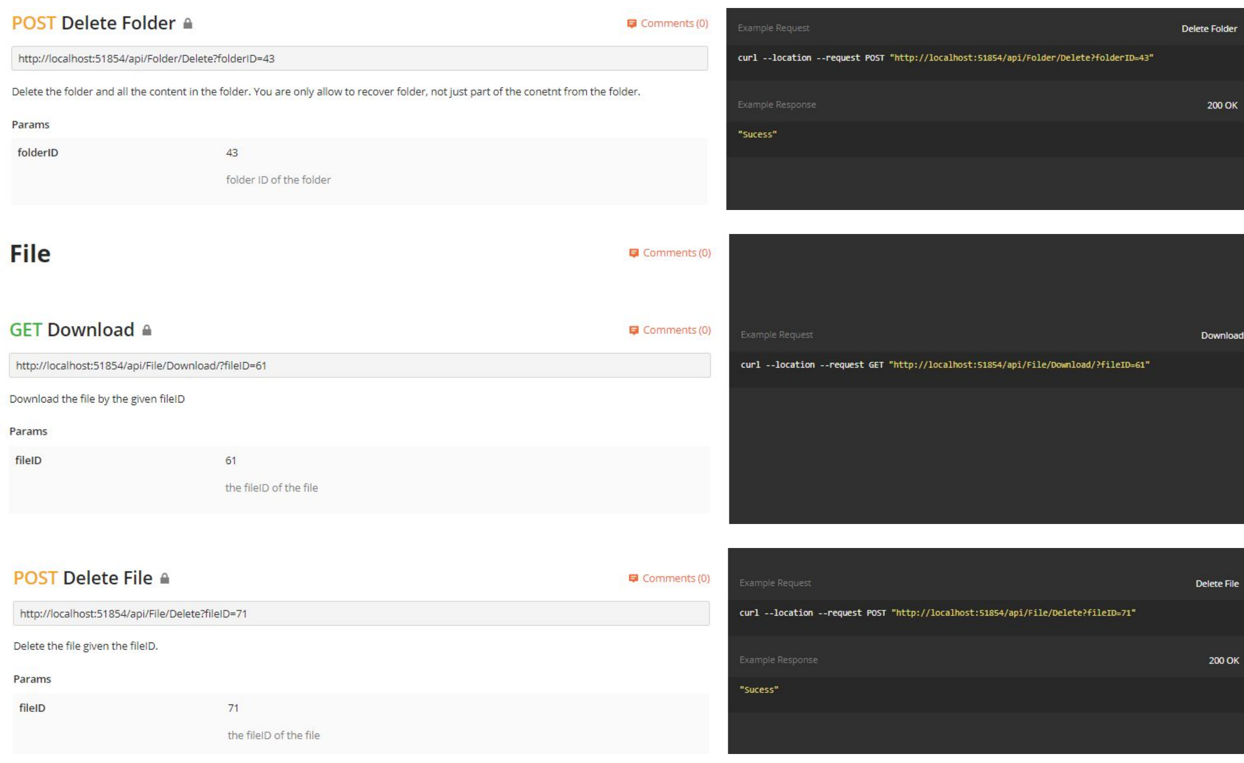
Example Request — List

```
curl --location --request GET "http://localhost:51854/api/Folder/List?folderID=39"
```

Example Response — 200 OK

List


```
{
  "$id": "1",
  "FolderID": 39,
  "Folders": [
    {
      "$id": "2",
      "Name": "hello1",
      "FolderId": 40,
      "CreateTimeStamp": 637096152107490800
    }
  ],
  "Files": [
    {
      "$id": "3",
      "Name": "newName.txt.txt",
      "Size": 12,
      "FileId": 61,
      "Created": 637107253387421700,
      "Version": 4,
      "UserName": "first last"
    }
  ]
}
```

POST Delete Folder

Comments (0)

http://localhost:51854/api/Folder/Delete?folderID=43

Delete the folder and all the content in the folder. You are only allow to recover folder, not just part of the conetnt from the folder.

Params

folderID	43 folder ID of the folder

Example Request — Delete Folder

```
curl --location --request POST "http://localhost:51854/api/Folder/Delete?folderID=43"
```

Example Response — 200 OK

```
"Sucess"
```

File

Comments (0)

GET Download

Comments (0)

http://localhost:51854/api/File/Download/?fileID=61

Download the file by the given fileID

Params

fileID	61 the fileID of the file

Example Request — Download

```
curl --location --request GET "http://localhost:51854/api/File/Download/?fileID=61"
```

POST Delete File

Comments (0)

http://localhost:51854/api/File/Delete?fileID=71

Delete the file given the fileID.

Params

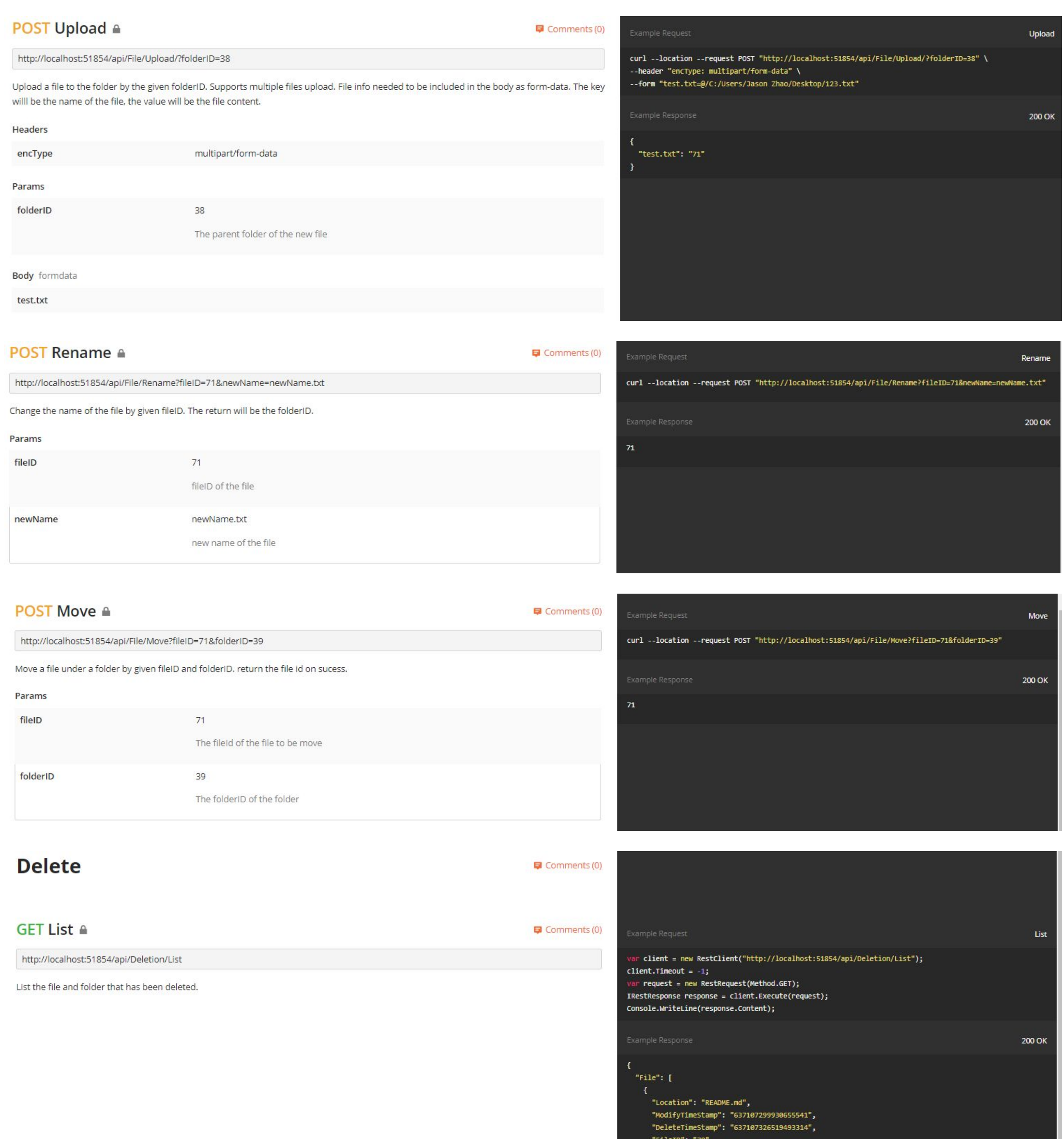
fileID	71 the fileID of the file

Example Request — Delete File

```
curl --location --request POST "http://localhost:51854/api/File/Delete?fileID=71"
```

Example Response — 200 OK

```
"Sucess"
```



POST Upload

Comments (0)

http://localhost:51854/api/File/Upload/?folderID=38

Upload a file to the folder by the given folderID. Supports multiple files upload. File info needed to be included in the body as form-data. The key willl be the name of the file, the value will be the file content.

Headers

encType	multipart/form-data

Params

folderID	38 The parent folder of the new file

Body formdata

test.txt

Example Request — Upload

```
curl --location --request POST "http://localhost:51854/api/File/Upload/?folderID=38" \
--header "encType: multipart/form-data" \
--form "test.txt=@/C:/Users/Jason Zhao/Desktop/123.txt"
```

Example Response — 200 OK

```
{
  "test.txt": "71"
}
```

POST Rename

Comments (0)

http://localhost:51854/api/File/Rename?fileID=71&newName=newName.txt

Change the name of the file by given fileID. The return will be the folderID.

Params

fileID	71 fileID of the file
newName	newName.txt new name of the file

Example Request — Rename

```
curl --location --request POST "http://localhost:51854/api/File/Rename?fileID=71&newName=newName.txt"
```

Example Response — 200 OK

```
71
```

POST Move

Comments (0)

http://localhost:51854/api/File/Move?fileID=71&folderID=39

Move a file under a folder by given fileID and folderID. return the file id on sucess.

Params

fileID	71 The fileId of the file to be move
folderID	39 The folderID of the folder

Example Request — Move

```
curl --location --request POST "http://localhost:51854/api/File/Move?fileID=71&folderID=39"
```

Example Response — 200 OK

```
71
```

Delete

Comments (0)

GET List

Comments (0)

http://localhost:51854/api/Deletion/List

List the file and folder that has been deleted.

Example Request — List

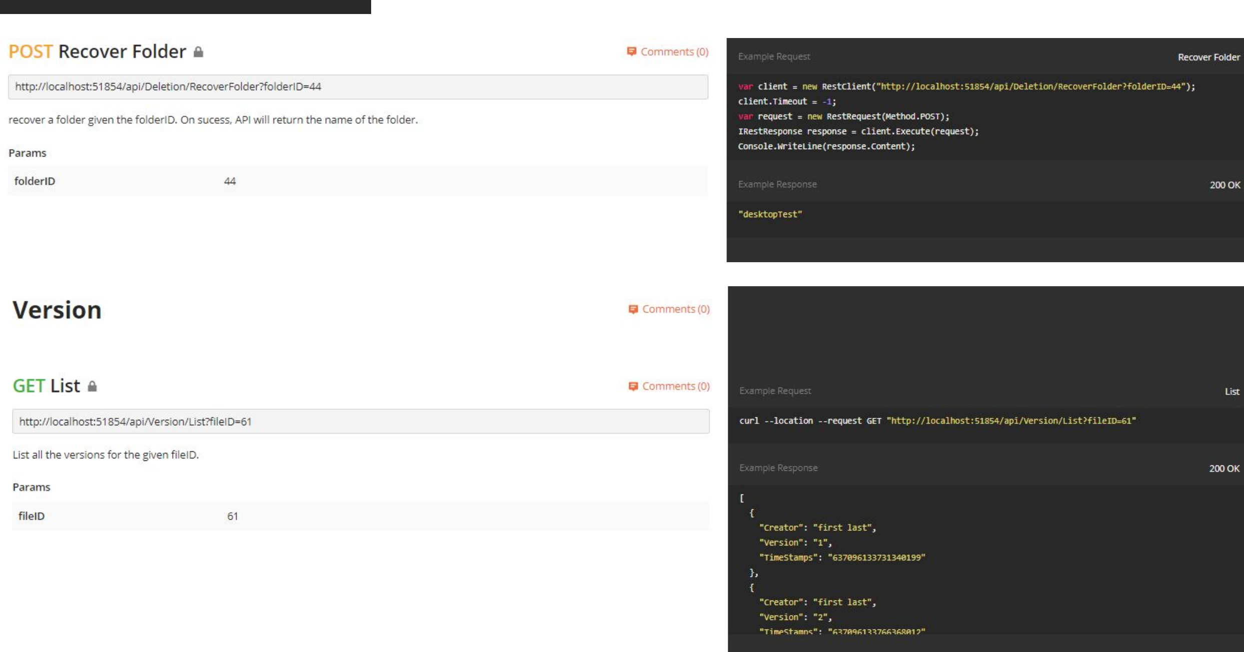
```
var client = new RestClient("http://localhost:51854/api/Deletion/List");
client.Timeout = -1;
var request = new RestRequest(Method.GET);
IRestResponse response = client.Execute(request);
Console.WriteLine(response.Content);
```

Example Response — 200 OK

```
{
  "File": [
    {
      "Location": "README.md",
      "ModifyTimeStamp": "637107299930655541",
      "DeleteTimeStamp": "637107326519493314",
      "FileID": "70"
    }
  ],
```

List


```
{
  "File": [
    {
      "Location": "README.md",
      "ModifyTimeStamp": "637107299930655541",
      "DeleteTimeStamp": "637107326519493314",
      "FileID": "70"
    }
  ],
  "Folder": [
    {
      "Location": "hello1\\hello1\\hello1",
      "DeleteTimeStamp": "637096153641054219",
      "FolderID": "41"
    },
    {
      "Location": "hello2",
      "DeleteTimeStamp": "1575762692",
      "FolderID": "43"
    },
    {
      "Location": "desktopTest",
      "DeleteTimeStamp": "637107673117854103",
      "FolderID": "44"
    }
  ]
}
```

POST Recover Folder

Comments (0)

http://localhost:51854/api/Deletion/RecoverFolder?folderID=44

recover a folder given the folderID. On sucess, API will return the name of the folder.

Params

folderID	44

Example Request — Recover Folder

```
var client = new RestClient("http://localhost:51854/api/Deletion/RecoverFolder?folderID=44");
client.Timeout = -1;
var request = new RestRequest(Method.POST);
IRestResponse response = client.Execute(request);
Console.WriteLine(response.Content);
```

Example Response — 200 OK

```
"desktopTest"
```

Version

Comments (0)

GET List

Comments (0)

http://localhost:51854/api/Version/List?fileID=61

List all the versions for the given fileID.

Params

fileID	61

Example Request — List

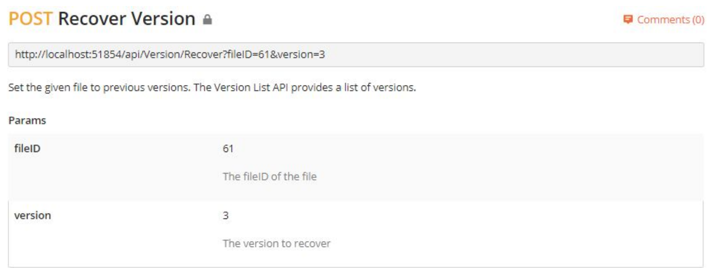
```
curl --location --request GET "http://localhost:51854/api/Version/List?fileID=61"
```

Example Response — 200 OK

```
[
  {
    "Creator": "first last",
    "Version": "1",
    "TimeStamps": "637096133731340199"
  },
  {
    "Creator": "first last",
    "Version": "2",
    "TimeStamps": "637096133766368012"
```

List

```
[
  {
    "Creator": "first last",
    "Version": "1",
    "TimeStamps": "637096133731340199"
  },
  {
    "Creator": "first last",
    "Version": "2",
    "TimeStamps": "637096133766368012"
  },
  {
    "Creator": "first last",
    "Version": "3",
    "TimeStamps": "637107253327905830"
  },
  {
    "Creator": "first last",
    "Version": "4",
    "TimeStamps": "637107253387421643"
  }
]
```



POST Recover Version

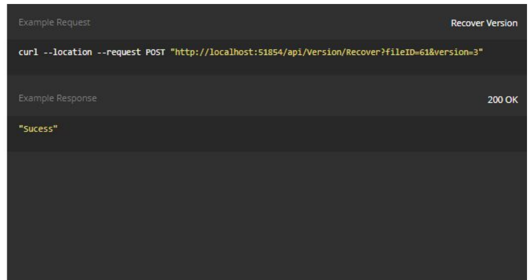
Comments (0)

http://localhost:51854/api/Version/Recover?fileID=61&version=3

Set the given file to previous versions. The Version List API provides a list of versions.

Params

fileID	61 The fileID of the file
version	3 The version to recover

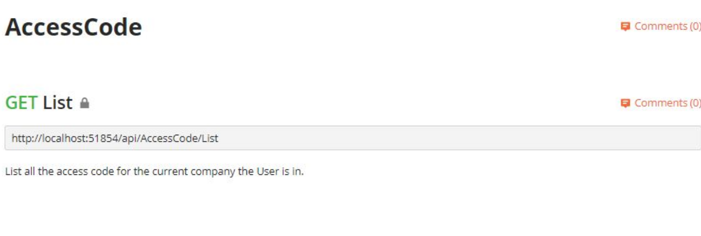


Example Request — Recover Version

```
curl --location --request POST "http://localhost:51854/api/Version/Recover?fileID=61&version=3"
```

Example Response — 200 OK

```
"Sucess"
```



AccessCode

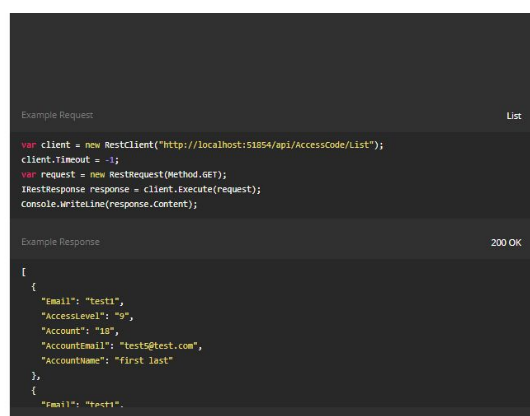
Comments (0)

GET List

Comments (0)

http://localhost:51854/api/AccessCode/List

List all the access code for the current company the User is in.

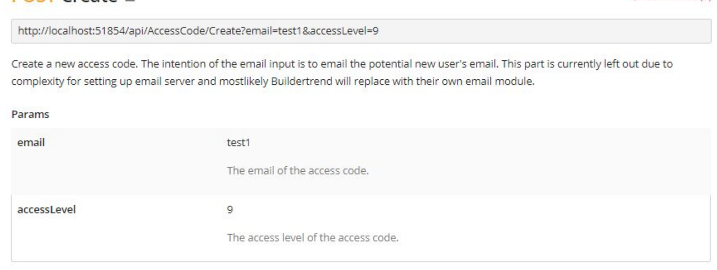


Example Request — List

```
var client = new RestClient("http://localhost:51854/api/AccessCode/List");
client.Timeout = -1;
var request = new RestRequest(Method.GET);
IRestResponse response = client.Execute(request);
Console.WriteLine(response.Content);
```

Example Response — 200 OK

```
[
  {
    "Email": "test1",
    "AccessLevel": "9",
    "Account": "18",
    "AccountEmail": "test5@test.com",
    "AccountName": "first last"
  },
  {
```



POST Create

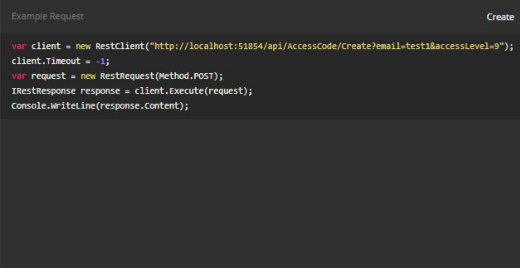
Comments (0)

http://localhost:51854/api/AccessCode/Create?email=test1&accessLevel=9

Create a new access code. The intention of the email input is to email the potential new user's email. This part is currently left out due to complexity for setting up email server and mostlikely Buildertrend will replace with their own email module.

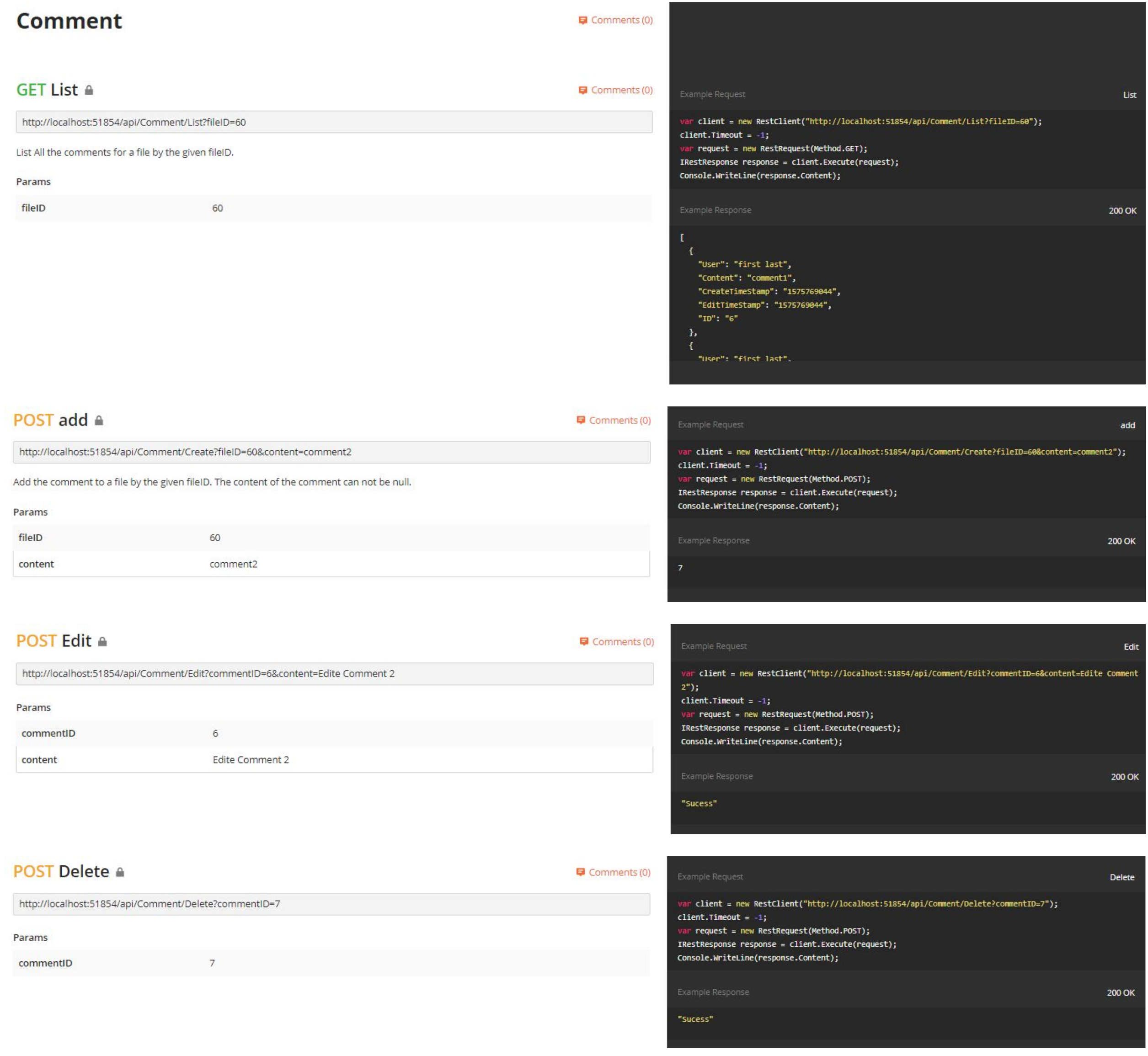
Params

email	test1 The email of the access code.
accessLevel	9 The access level of the access code.



Example Request — Create

```
var client = new RestClient("http://localhost:51854/api/AccessCode/Create?email=test1&accessLevel=9");
client.Timeout = -1;
var request = new RestRequest(Method.POST);
IRestResponse response = client.Execute(request);
Console.WriteLine(response.Content);
```



Comment

Comments (0)

GET List

Comments (0)

http://localhost:51854/api/Comment/List?fileID=60

List All the comments for a file by the given fileID.

Params

fileID	60

Example Request — List

```
var client = new RestClient("http://localhost:51854/api/Comment/List?fileID=60");
client.Timeout = -1;
var request = new RestRequest(Method.GET);
IRestResponse response = client.Execute(request);
Console.WriteLine(response.Content);
```

Example Response — 200 OK

```
[
  {
    "User": "first last",
    "Content": "comment1",
    "CreateTimeStamp": "1575769044",
    "EditTimeStamp": "1575769044",
    "ID": "6"
  },
  {
    "User": "first last",
```

POST add

Comments (0)

http://localhost:51854/api/Comment/Create?fileID=60&content=comment2

Add the comment to a file by the given fileID. The content of the comment can not be null.

Params

fileID	60
content	comment2

Example Request — add

```
var client = new RestClient("http://localhost:51854/api/Comment/Create?fileID=60&content=comment2");
client.Timeout = -1;
var request = new RestRequest(Method.POST);
IRestResponse response = client.Execute(request);
Console.WriteLine(response.Content);
```

Example Response — 200 OK

```
7
```

POST Edit

Comments (0)

http://localhost:51854/api/Comment/Edit?commentID=6&content=Edite Comment 2

Params

commentID	6
content	Edite Comment 2

Example Request — Edit

```
var client = new RestClient("http://localhost:51854/api/Comment/Edit?commentID=6&content=Edite Comment
2");
client.Timeout = -1;
var request = new RestRequest(Method.POST);
IRestResponse response = client.Execute(request);
Console.WriteLine(response.Content);
```

Example Response — 200 OK

```
"Sucess"
```

POST Delete

Comments (0)

http://localhost:51854/api/Comment/Delete?commentID=7

Params

commentID	7

Example Request — Delete

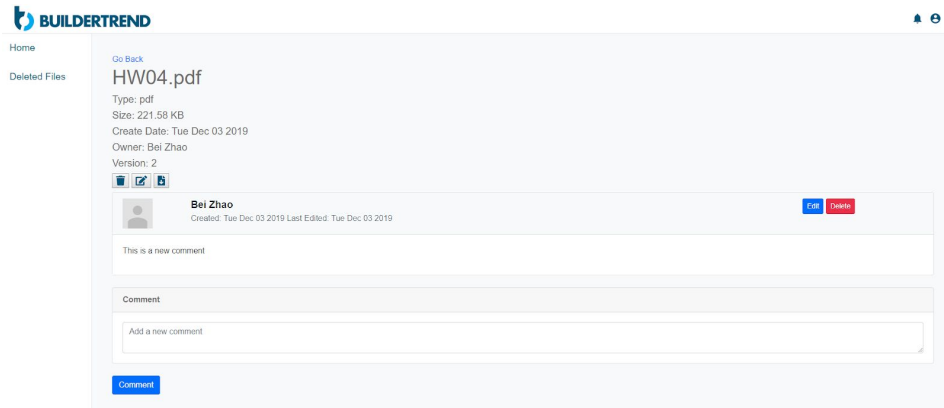
```
var client = new RestClient("http://localhost:51854/api/Comment/Delete?commentID=7");
client.Timeout = -1;
var request = new RestRequest(Method.POST);
IRestResponse response = client.Execute(request);
Console.WriteLine(response.Content);
```

Example Response — 200 OK

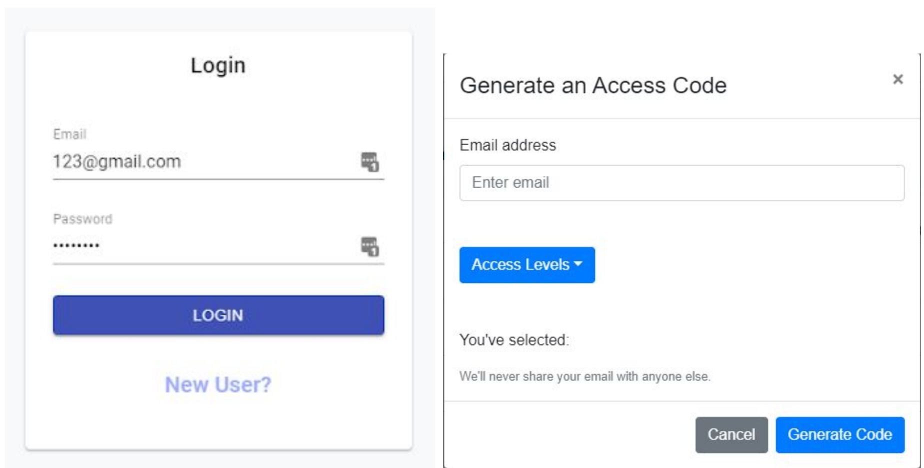
```
"Sucess"
```

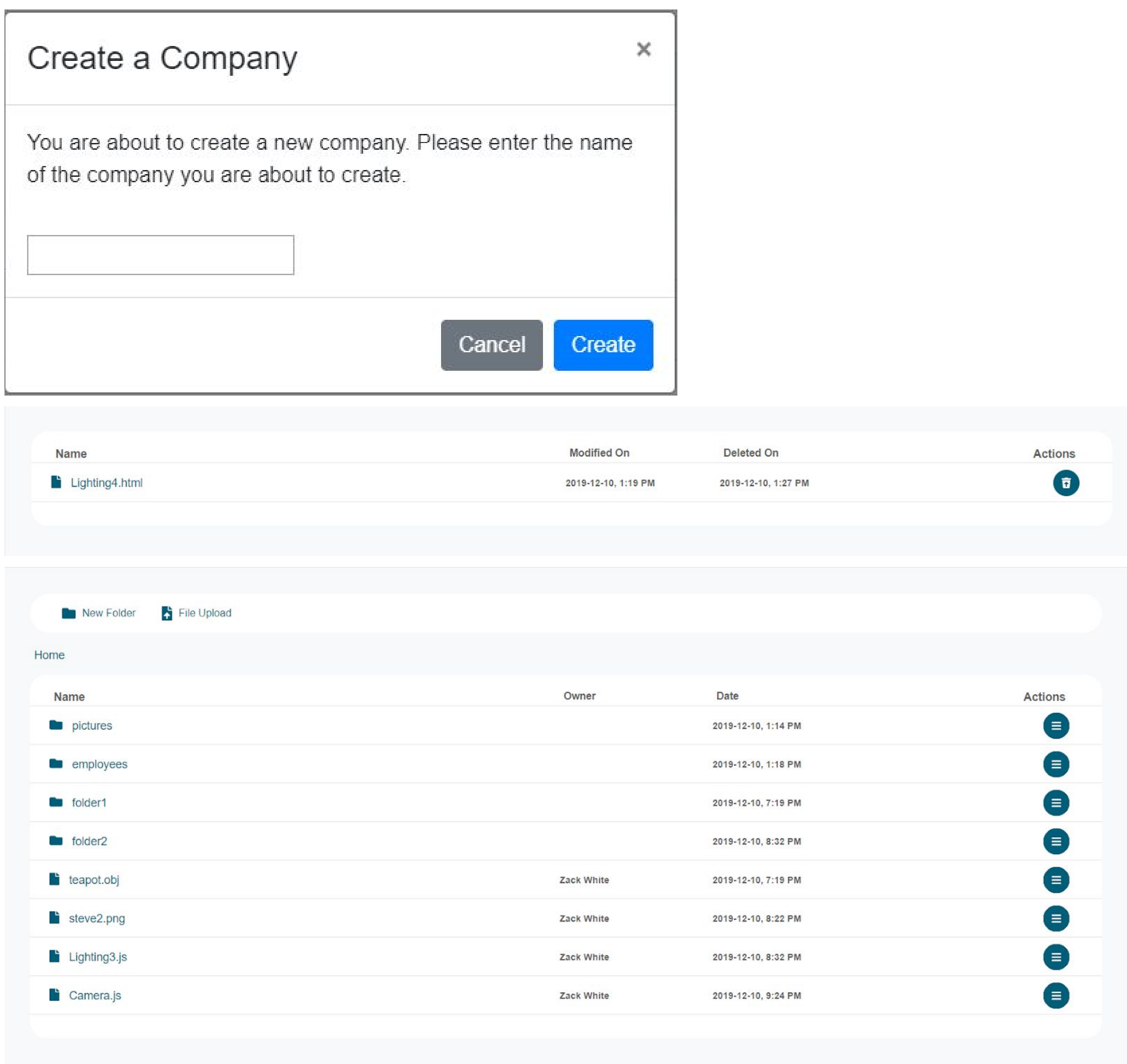
9. Appendix III - Web Application UI

Frontend Design - File Details Page



Frontend Design - Login Page




Create a Company
×
You are about to create a new company. Please enter the name of the company you are about to create.
Cancel
Create
Name
Modified On
Deleted On
Actions
Lighting4.html
2019-12-10, 1:19 PM
2019-12-10, 1:27 PM
New Folder
File Upload
Home
Name
Owner
Date
Actions
pictures
2019-12-10, 1:14 PM
employees
2019-12-10, 1:18 PM
folder1
2019-12-10, 7:19 PM
folder2
2019-12-10, 8:32 PM
teapot.obj
Zack White
2019-12-10, 7:19 PM
steve2.png
Zack White
2019-12-10, 8:22 PM
Lighting3.js
Zack White
2019-12-10, 8:32 PM
Camera.js
Zack White
2019-12-10, 9:24 PM