

Project: *AFFORDABLE File Manager*

Documentation: *Overview, Decisions, User Guide*

Client: *Alex Campbell*

Team: *Hannah Seabert, Ryan Smolik, Chase Tiberi*

Table of Contents

Table of Contents	2
Project Overview	3
Code Breakdown	4
Additions to the Opuscapita File Manager:	4
Sidebar + UI:	5
AWS Connectivity	6
File Viewer	7
Dropzone UI	9
Dropzone dropping into the correct directory	10
Additional Database Tables Created	11
Common MySQL Queries	11
JavaScript Wrapper for Common MySQL Queries	12
GENERAL DECISIONS	12
Decision to switch from Warpdesign file explorer → OpusCapita file manager	12
Addition of Dropzone Support	14
File Upload	15
Preview File	17
Rename File	17
Delete File	18
Set-up for the Current Project Deliverables	19
Next Steps	20

Project Overview

The AFFORDABLE project is a platform dedicated to reducing barriers to affordable health care. The AFFORDABLE team plans to provide people who are in need of affordable health care an easier way to access and communicate with organizations that provide financial aid or free clinic services. The process of attaining this aid can be difficult as the organizations that do provide the services often require a lot of paperwork, and this can cause the process to be bogged down as people in need of health care may have problems organizing and providing all of these documents. AFFORDABLE will expedite this process making it easier for people to organize their documents and access these organizations, as well as making it easier for organizations to validate these documents and provide the aid as soon as possible.

Our contribution to the AFFORDABLE project will be a document organizer that will allow people to upload, organize, modify, and share their documents. Users will be able to upload any necessary forms such as tax-information, identification, medical records, or any other documents they may need to provide to receive health care. After uploading their documents users will be able to organize them into different folders, and modify elements of the documents such as their names. Furthermore, documents can be shared with health care providing organizations or charities, allowing these organizations to validate the documents and make a decision on the aid to be provided. It ultimately will act as a drop box for health care related documents. This project will be a web based application that for the time being will be focused on desktop use only.

Code Breakdown

Additions to the Opuscapita File Manager:

- Share button added to UI
 - How to add a button to the UI:
 1. Create a new file in the
`/node_modules/@opuscapita/react-filemanager-connector-node-v1/lib/capabilities` folder
 2. Require this file in the index.js of the capabilities folder mentioned in step 1
 - Make it into two variables following the way it is done for the other variables in that file
 3. Add the second variable you make into the capabilities array in the file
 4. In order for the button to work within the file you made in Step 1 you must export some props - look at the pre-existing capabilities to see what is necessary
 - It may be best to copy and paste the contents of one of the other files in capabilities and then edit what you want to change
- View Button Added to UI
 - The process for adding is the same as share above
 - The handler calls on a hidden button to click that handles showing the modal box to display the document viewer
 - This might not be the best way to do this

- Making a button not show up when right-clicking on folders
 - In the return statement of the file you made for the button (ex. Share.js, view.js)

there will be a variable for `shouldBeAvailable`, you have to set it to the code below

```
shouldBeAvailable: function
shouldBeAvailable(apiOptions) {

    var selectedResources = getSelectedResources();

    return selectedResources.length > 0 &&
    !selectedResources.some(function (r) {

        return r.type === 'dir';

    }) && selectedResources.every(function (r) {

        return r.capabilities.canDownload;

    });

},
```

Sidebar + UI:

- App.js changed from a functional component to a Class component so that it can have state. State used to change the styling of the file-manager and sidebar on the click of the sidebar button (three lines in top right of filemanager). Clicking on sidebar button changes sidebar between a class that shows nothing (`.hidebar`) and one that shows it (`.active`), while file-manager switches between a class in which it spans the whole page(`.manager`) to only spanning 79.5% of the screen (`.smallManager`). These classes are

found in App.css. The function that controls the sidebar is in the App.js and is called moveBar().

- The Sidebar itself contains an image of the AFFORDABLE logo and a UL with links to other places in the application. The css for these are the classes .logo and .sideBarList and can be found in app.css, and the file for the sidebar is a functional component called sidebar.js.
- The UI styling is based on the AFFORDABLE style guide:
https://docs.google.com/presentation/d/1KoTrjGvUjwJ3lW6rgMySMHnApmfZ8G7tpN55pe4_VD4/edit#slide=id.g386105ff5_2_75

AWS Connectivity

- Need to npm install aws for Node.js
- Upload:
 - Upload process located in the opuscapita server folder in the file uploadOrCreate.js
 - Start by importing the AWS api and javascript FS to read files
 - Instantiate a S3 object by giving it the public and secret access key from your AWS account
 - Create a file stream using FS
 - Read the file using FS and then create S3 Parameters for the bucket you want to send the file to, the name of the file in S3 (key) and the file stream we created before

- Use the s3 upload function
- Delete
 - Delete process located in remove.js in the opuscapita server folder
 - Start by importing the AWS api
 - Configure your AWS object by giving it the public and secret access key from your AWS account
 - Set the region for your bucket in the AWS object
 - Create the S3 instance and give it the parameters bucketname and key, this will identify the file you want to delete and what bucket to delete it from
 - Call the s3 delete function
- Rename
 - Delete process located renameCopyMove.js in the opuscapita server folder
 - Start by importing the AWS api
 - Instantiate a S3 object by giving it the public and secret access key from your AWS account
 - Set the region for your bucket in the AWS object
 - Create a copy of the S3 object you want to rename with the name you want to change it to using the s3 copy function
 - After this delete the original copy using the s3 delete function

File Viewer

- The react file viewer uses the react file viewer node_module

- In order to use it you have to add a `<FileViewer>` tag and then specify the file type with a `fileType` prop and the file with a `filePath` prop
 - The component handles unsupported file types by itself and will display an appropriate message
 - You also must import `FileViewer` from `'react-file-viewer'`
- To connect this with AWS you have to give the component the file type and its aws url
 - To make sure you will be able to view these you must change the CORS policy on the bucket

```
<?xml version="1.0" encoding="UTF-8"?>

<CORSConfiguration
xmlns="http://s3.amazonaws.com/doc/2006-03-01/">

  <CORSRule>

    <AllowedOrigin>http://localhost:3000</AllowedOrigin>

    <AllowedMethod>GET</AllowedMethod>

    <AllowedMethod>POST</AllowedMethod>

    <AllowedHeader>*</AllowedHeader>

  </CORSRule>

</CORSConfiguration>
```

- The viewer is located in a modal pop-up, this pop-up is from react-modal an installable node_module

- To use this you must import Modal from 'react-modal'
- You surround the content you want in the modal with a <Modal>
- There are various props you can send the modal, view the documentation to see what
 - <https://www.npmjs.com/package/react-modal>
- To open the viewer you click the eye icon and to close it you click anywhere on the page that is not the modal window that pops-up
 - The eye icon calls handleOpenModal() to open the file viewer
 - Once the modal is open clicking anywhere on the webpage that is not the file viewer will close the modal by calling handleCloseModal()
 - You must pass this to the <Modal> in the prop onRequestClose
 - Furthermore for the isOpen prop in the modal I made it part of App.js' state and handleOpen/CloseModal is what changes this to true or false

Dropzone UI

- The Dropzone UI is referring to the image that shows up when a document is dragged over it
- This works by passing the <Dropzone> tag props to handle onDragOver, onDragExit, and onDrop
 - After passing these props, in src/Dropzone.js you have to specify in the event handlers which prop goes to which event
 - All possible events are located in the documentation for the viewer

- <https://github.com/felixrieseberg/React-Dropzone-Component#server-example>
- You specify this in the const eventHandlers variable in the render() method
- The functions passed are located in App.js and are showDropUI() and hideDropUI()
 - onDragOver calls showDropUI()
 - onDragExit and onDrop call hideDropUI()

Dropzone dropping into the correct directory

- In order to make the dropzone drop into the correct directory I had to move the djsConfig into App.js and pass it to the dropzone as a prop
- The reason for this is i needed to set the parentId to a prop so that it would set it to the correct directory
 - To get the correct directory whenever the file manager changes folders the api is called in using the function in api.js that is located on lines 65-98
 - I made it so this function writes to a hidden div on the page with the parentId
 - Whenever onDrop is called it now calls the function changeCurrId() which updates the state for the currId of the directory
 - changeCurrId() takes what is in the innerHtml as its value

Additional Database Tables Created

In order to maintain the necessary information related to file storage and sharing, three tables were added to the existing AFFORDABLE database.

The first table added is called 'files'. This table contains the following fields:

'File_id' → Primary Key, 'User_id', 'Pathname', 'Url', 'EncryptionType',
'FileType', 'Size', 'DateAdded'.

The second table is called 'permissions'. This table contains the following fields:

'File_id' → Primary Key, 'User_id' → Primary Key, 'PermissionType'.

The third table is called 'encryptions'. It contains the following fields:

'File_id' → Primary Key, 'Password'.

Common MySQL Queries

Common MySQL queries were created to handle the following processes: file upload, file modification, and file deletion. To handle file upload, 'AddFileToFilesDatabase' and 'AddPermissionToFile' queries were created. To handle file modification, 'UpdateFilePermissions' was created. Finally, to handle file deletion, 'DeleteFilePermission' and 'DeleteFileFromFilesDatabase' were created.

JavaScript Wrapper for Common MySQL Queries

In order to expedite the process of calling common queries, a JavaScript wrapper function was created to handle the following processes: file upload, file modification, file deletion. This wrapper function is found in the file *storedProcedures.js*.

GetQueries is a function that takes one parameter, *qType*, which is representative of one of the aforementioned processes. Depending upon the process type, *GetQueries* returns the necessary queries to carry out the process.

ExportModule is a function that takes one parameter, *obj*, which is the object to export. Within this function, *module.exports* was used to export the object in order to mirror the methodology in the existing AFFORDABLE application.

ObjToExport is the wrapper function that leverages both *GetQueries* and *ExportModule* in order to determine the common queries needed and subsequently export them.

GENERAL DECISIONS

Decision to switch from Warpdesign file explorer → OpusCapita file manager

Initial project plans included working with the Warpdesign file explorer; however, persistent build issues across multiple code versions prompted a change in library. After researching potential options, we decided to move forward with the OpusCapita file manager. In

addition to being highly rated, OpusCapita uses a React framework and supports connections to various file storage options.

As we began working with OpusCapita, we ran into a few roadblocks with server requests being blocked on the local host. We were able to work through these issues by using the terminal commands below:

```
cd filemanager-demo
node

let config = {
  fsRoot: ".",
  rootName: 'Root folder',
  port: process.env.PORT || '3020',
  host: process.env.HOST || 'localhost'
};

let filemanager =
require('@opuscapita/filemanager-server');
filemanager.server.run(config);

// LEAVE THAT TERMINAL OPEN AND OPEN A NEW SHELL

cd filemanager-demo
npm start
```

Addition of Dropzone Support

While OpusCapita did not initially provide dropzone support, our client and our group mutually felt that it would be an important addition to the application. Dropzone support creates a more user-friendly experience and increases ease of use.

In order to add dropzone support to the existing application, we leveraged [dropzone.js](#) as a resource and guide. The application was supplemented with this dropzone support following the procedure below:

- The dropzone component was based off of Felix Riseberg's React-Dropzone-Component, provided in GitHub at <https://github.com/felixrieseberg/React-Dropzone-Component>.
- Example implementations of the component were explored in the projects's *examples* folder.
- Various CORS issues were encountered that were resolved by adding the 'Cache-Control' header to the Access-Control-Allow-Headers in *filemanager-server/router/index.js*
- Once the CORS issues were resolved, this led to a 500 internal server error message.
- *'file'* and *parentId: 'Lw'* were added as parameters in the djsConfig object in the dropzone component as requested.
- Issues within the file manager's middleware and the use of the Busboy node-module led to issues where an API request was needed with the word 'files'

instead of the already written ‘file.’ A modification to the file manager’s code allowed us to circumvent this issue and allow for drag-and-drop upload to be successful.

User Guide




Figure 1. Image of AFFORDABLE File Manager that user will see upon navigating to the file manager page.

File Upload

Once inside the AFFORDABLE file manager, there are various methods to upload a file. The first is deemed the ‘traditional’ method, in which the user uploads a file by clicking on a button. The second is deemed the ‘dropzone’ method, in which the user uploads a file by dragging and dropping the file over the UI.

Traditional

To upload a file by the 'traditional' method:

1. Click on the upload button  **Figure 2.** Image of upload button.
2. Select the file you would like to open when the default file manager appears

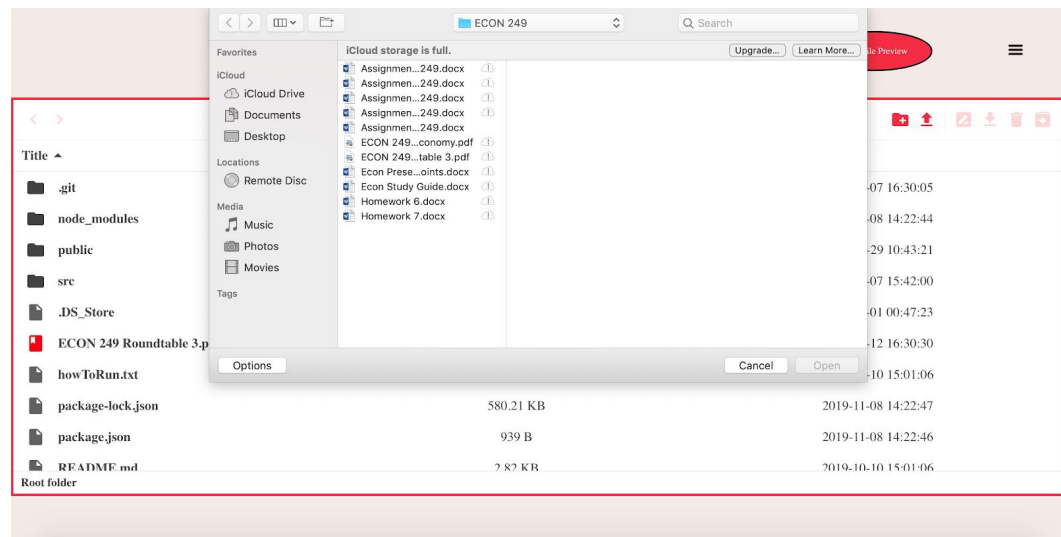


Figure 3. Image of AFFORDABLE file manager and default file manager application to demonstrate the result of clicking on the upload button.

Dropzone

To upload a file by the 'dropzone' method:

1. Click on desired file in your default file manager/on your desktop
2. Drag and drop file over the filemanager
3. Refresh the page to see that the file has been uploaded and is now located in the manager

Preview File

Right-click method

1. Right-click on the file you would like to view
2. Select “View” from the dropdown menu

Icon Method

1. Select the file you would like to view
2. Click on the eye icon in the upper right hand corner

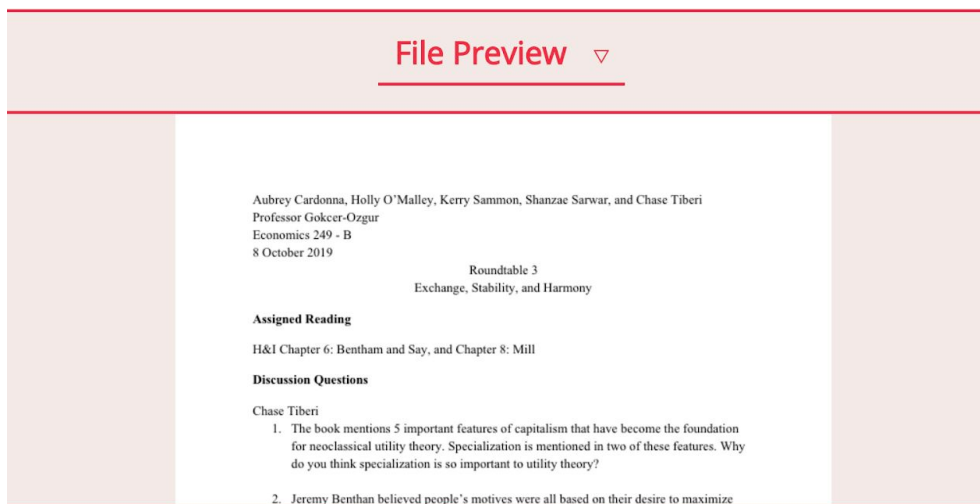


Figure 4. Image of AFFORDABLE File Preview window.

Rename File

Right-click method

3. Right-click on the file you would like to view
4. Select “Rename” from the dropdown menu

Icon Method

3. Select the file you would like to view
4. Click on the pencil icon in the upper right hand corner

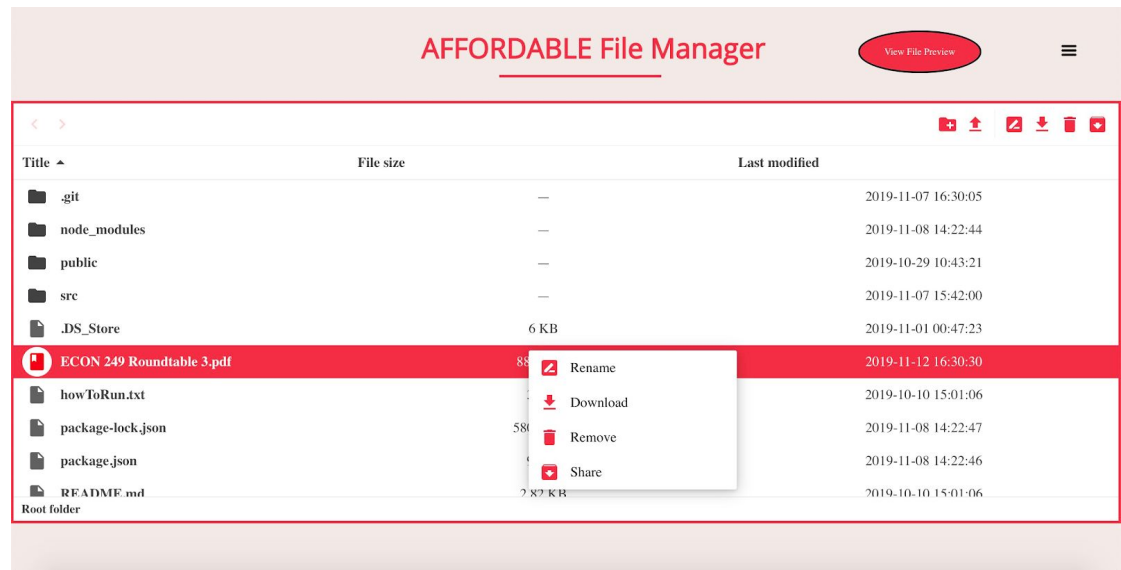


Figure 5. Image of dropdown menu in AFFORDABLE File Manager.

Delete File

Right-click method

5. Right-click on the file you would like to view
6. Select “Remove” from the dropdown menu

Icon Method

5. Select the file you would like to view
6. Click on the trash icon in the upper right hand corner

Set-up for the Current Project Deliverables

1. Open the develop branch from GitLab Repository.
2. In the terminal navigate to the directory of the repository.
3. Enter the command: `node` to enter the node shell.
4. Then enter the following command to launch the file-manager server:

```
let config = {  
  fsRoot: ".",  
  rootName: 'Root folder',  
  port: process.env.PORT || '3020',  
  host: process.env.HOST || 'localhost'  
};  
  
let filemanager =  
  require('@opuscapita/filemanager-server');  
filemanager.server.run(config);
```

If you encounter difficulties launching the server, visit:

<https://github.com/OpusCapita/filemanager>, for more information.

5. Once the file-manager server is started, keep that terminal window open, and open a new terminal window or tab.
6. In this new window or tab, navigate to the directory of the repository and run: `npm start`.
7. Once this is run, the UI should open in your default browser window and be able function.

Next Steps

- Next steps for the project include incorporating our component with the AffordableServerOSU component.
- The OpusCapita file-manager needs to be incorporated into the AffordableServer so when the AffordableServer launches, the OpusCapita server also launches.
- The JavaScript SQL wrappers need to be incorporated into the AffordableServer to execute the SQL stored procedures for the database queries when the changes are made to the user files in the UI.