**Intro**

The current repository for this project can be found at <https://github.com/CentennialTech/GrantsAuditsManagement>. This repo hosts the files for the actual web application as well as tools that will be used in the actual application or can be used standalone. For reference, the prototype can be found at [https://uqdaen.axshare.com](https://uqdaen.axshare.com/)

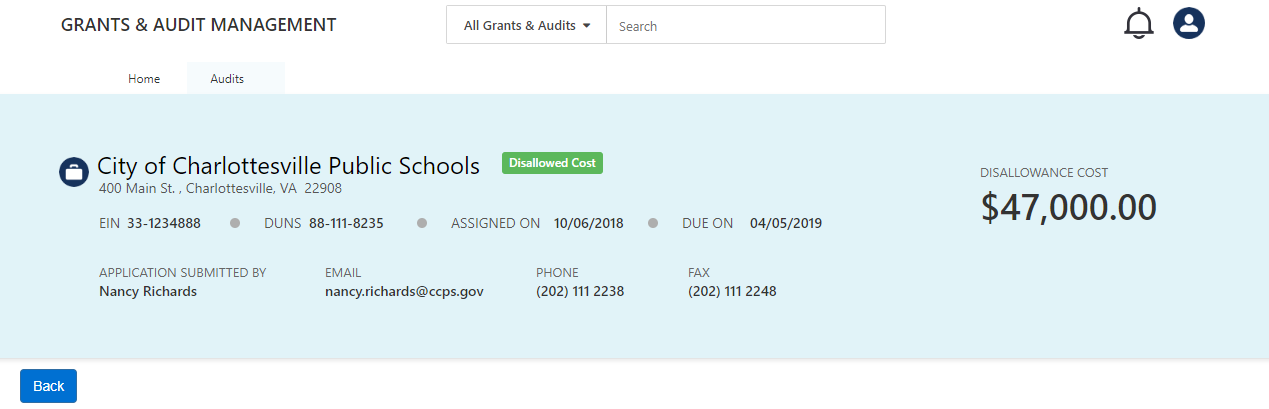
**Folder Composition**

All screens made are put into their own folders. Most of them are named for the use cases they cover. If a screen covers more than one use case, then the first use case it covers is what it is named after. For example, there exists an 08-04 folder, but not one for 05 and 06, as the screen named 04 covers those 3 use cases. Some screens aren’t named after use cases, like “login”, “AuditAdmin-AccessRisk1” and “2”, “assignment”, “dashboard 1-5”, “programmanagerview”, “review-budget”, and “reviewerview”.

Directly inside each screen folder, there exists generally 3 folders, an html or php file, and any additional files needed like .png or .pdf files. The html files can be run by just clicking them of course, but the php files need to be run from a local server. Our solution for this was to use the php-server package in the Atom text editor. The 3 folders in each screen folder are “data”, “files”, and “resources”. The “data” folder includes a document.js file and a styles.css file. Both are used very minimally, if at all. In the “files” folder, there is a folder with an arbitrary name. Inside that folder is a data.js file, used, again, very minimally. The other file is a styles.css file that houses all the main styling used in the screen. Use this file to make style changes. The resources file has two folders named “css” and “scripts”, both of which do not contribute to the screens. The scripts in the “scripts” folder has some javascript files that perform actions like some dropdowns in the screens, but these are not important. This will become apparent why soon.

**Improvements**

First off, it’s important to note that the reason the site architecture looks like this is because we copied the file structure and code from the login and dashboards from the axshare prototype. This, in hindsight, was a very bad idea. Much of the site relies on this drag-and-drop code which makes the site hard to build.

Most of the screens use bootstrap, which makes it easier to develop. Since almost all of the screens use some form of axshare web code as a base, most bootstrap grid code is written in a div with the id=”mainbody”. This is because the axshare code makes the entire page use position:relative;, which means divs and elements are aligned by specifying an exact left and top style. When the mainbody id is used, the div is positioned to it takes up the space on the screen where common user elements are filled. This does not include standard elements copied from the axshare code like the navbar and the audit information bar pictured below. 

Everything in this image is from the axshare code.

Moving forward, if you plan to salvage the code we wrote, try to use what we wrote in the mainbody divs. For the rest, make sure that no axshare code is used moving forward. We heavily encourage having every page screen use the bootstrap grid layout or another viable framework. Just don’t use the axshare code.

Furthermore, we only got some of the html files we made to be php files, so that should happen to all the html files. After that, we advise that all the pages should be in the same directory calling a unified stylesheet, unlike how we did it where each page has its own stylesheet in its own folder.

`Many of the pages we made share the same elements from the UI designs provided to us, but due to not having a standard sheet of elements, elements like buttons which should be similar are not, so everything should be made consistent. Along with that, through development, we somehow realized that we were calling both Bootstrap 4 and Bootstrap 3. Our reliance on both made it impossible to simply remove one of the versions without breaking pages. Therefore, pages should be rewritten using only one version.

We set up a free trial of Azure to host an SQL Server on it with data from FAC to display on some of the screens using php: namely, 08-01, 08-02, 08-03, dashboard4, and 08-07. However, this program manages audits that haven’t been completed yet, which is the opposite of what audits on FAC are. Therefore, a more final version of this product should be able to store audits currently being handled by the program in an SQL Database with accompanying metadata and change the data it needs to.

On the note of retrieving data from FAC, an SSIS package was developed that calls metadata for completed audits from the FAC database and updates the SQL Database with that information. This is included in the repository. Some directory paths may need to be changed for it to function. For actual production, this package would need to be set up in azureto run on a timer in a shared location. However, for the reasons stated in the previous paragraph, this won’t be needed.

I hope this explains everything, but if any answers are needed, I can be emailed: alfredrpk@gmail.com