**Outcomes:**

A user can upload media from their device to be published as content. Once uploaded, it can be sorted into categories and groups. Any type of media can be uploaded including WAVS/MP3s/JPEGS/PDFs etc..

**Why any extension:**

Files could be important contracts, songs, jingles, video etc... The user has ultimate control of where to store and group their uploads. This offers true user flexibility and is something to build on if this project is continued.

**How it works:**

The main screens of the prototype are ‘Categories’, ’Media’ and ‘Playlists’. For all screens, the user will be presented with a populated list of the items they have created. Umbraco is the CMS powering the project behind closed doors. The benefits of using Umbraco are:

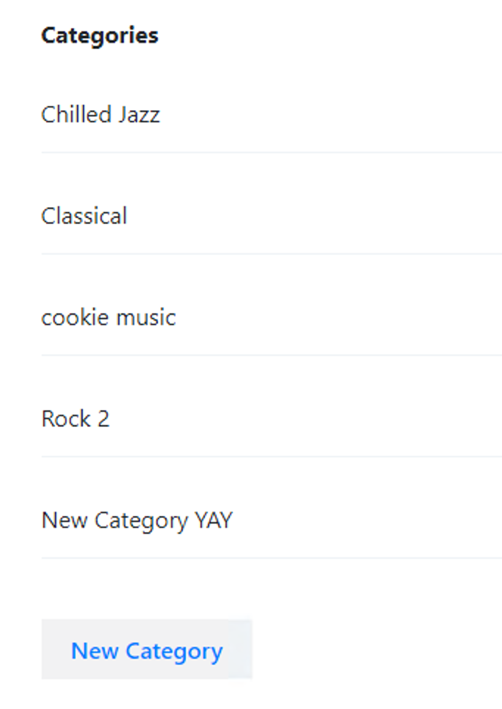
-CMS solution to add files/categories/playlists

-Database that stores the information

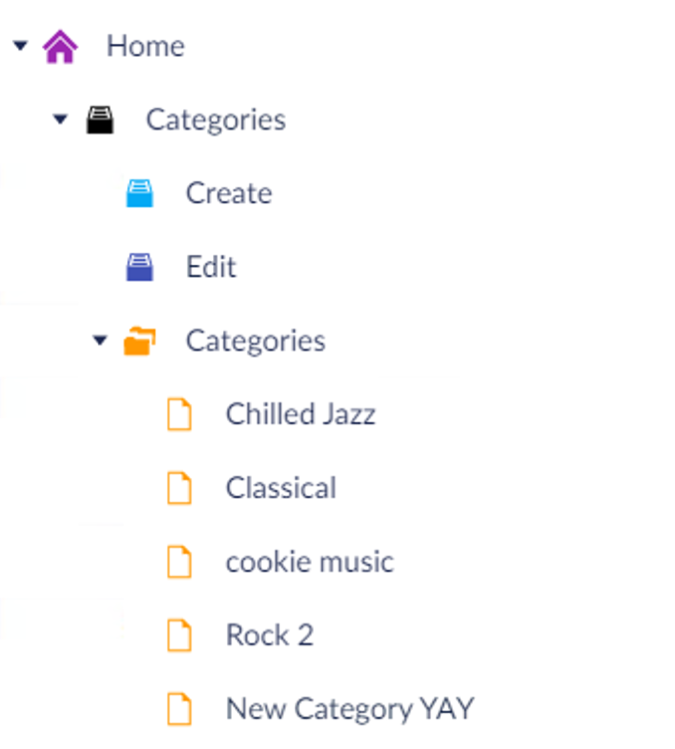
-Easier to work with

**Category:**

A user creates a category, names it and saves it. On the Umbraco CMS, it has created a new Category Node.



**User Screen Created Category**



**Umbraco CMS**

If a user wants to rename the category, they can by clicking ‘edit’ and changing the name. Once this form is submitted, it changes the Umbraco Node name. The new data is pulled through to the front-end.

Graphical user interface, application

Description automatically generated

**Media View page with Category assigned**

Graphical user interface, text, application, chat or text message

Description automatically generated

**Category dropdown for editing media**

Deleting the category will delete the Umbraco node in all areas.

**Media:**

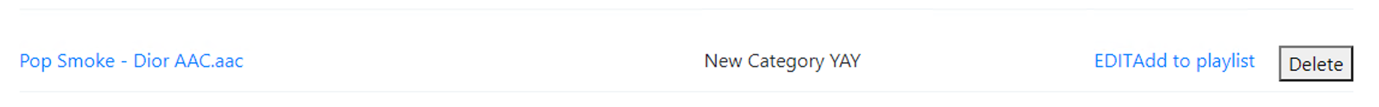
A user uploads a file through the file upload button. This saves on the Umbraco database as a MediaType file. On the backend this is ‘MediaService’.

Graphical user interface, text, application, chat or text message

Description automatically generated

**Create Media view uploading files**

The media screen pulls all uploaded data and loops around. This is done by the ‘mediaAtRoot()’ method. For each media, it has the name, category and the link to the actual file, alongside the file extension.



**Uploaded media data**

Editing the media file, brings the user to a screen where they can see these properties: name, file path, upload date. Users can change the category it is placed in.

Graphical user interface, application

Description automatically generated

Deleting will delete the media from the Umbraco CMS.

Users can add selected media to a playlist by clicking that option. On the CMS, the user is adding a media file to the multimedia picker property on the Playlist. In the backend, a UDI identifier has to be created which consists of 3 parts: Scheme, Type, Identifier. This is created by the following code on the controller:

var udi = Udi.Create(UdiEntityType.Document, selectedMedia.Key);

which returns

Scheme- umb://

Type- document/

Identifier - 4fed18d8c5e34d5e88cfff3a5b457bf2 (node GUID)

Then convert to a string and add to the list.

Chart

Description automatically generated

**Media add to playlist button pressed**

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, application

Description automatically generated**Playlist Node CMS Media Picker Property**

**Playlist front-end user**

**Playlists:**

On the Umbraco side, a media picker property is on the node. When media gets added, it is added to the media picker. Users can do this by clicking ‘add to media’.

A user can create a ‘playlist’ which acts as a folder to group media files. This can be named whatever the user wants such as: ‘Documents’, ‘Contracts’, ‘Music’ etc..

Editing a playlist screen pulls the data from the Node’s media picker. This data is casted through a HTML partial, which loops around and counts the media. The user can re-edit the name, count the files and have immediate access to the source.

Graphical user interface, application

Description automatically generated

**Edit playlist screen**

Users can delete playlists which will lose the files in the media picker.

Technologies Used:

NPM - Node package manager. Useful for downloading JS node modules.

Webpack:

Very useful JavaScript modules builder which watches changes on the front end and bundles SCSS/JavaScript files. This was needed to successfully get SCSS working.

HTML/SCSS:

HTML is the language used alongside SCSS for the styling. SCSS is a good fit because it has CSS + Superpowers. Nesting classes, having variables and separating code becomes a lot easier.

Principle prototyping:

Very simple prototypes were made so I can understand the brief better.

Bootstrap:

Useful classes for fonts, CSS and most importantly responsiveness.

Design Screens

The screens below were a starting point of what I wanted to accomplish. This was a rough design as I did not take into consideration bootstrap formatting.

Graphical user interface, table

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, application, Teams

Description automatically generated

Graphical user interface, application, Teams

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

**Limitations and Improvements**

It must be noted, this is only a working prototype and improvements can be made over time.

Reorder/order media:

-Users cannot Order and reorder their media.

-In the future, users will be able to drag and drop/sort/reorder in their list.

No authentication:

-Users don’t have to log in/authenticate. In theory anyone can upload anything which is a security breach.

-When live, by default a user must have login credentials.

Customise media uploads:

-Users cannot upload thumbnail images of their media or write comments.

-Edit media page will have a lot more functionality, such as allowing thumbnail images, comments/ratings and much more.

Export files/playlists:

-Users cannot export these files, only organise and sort.

-Allow users to download/export media/playlists.

Playlists:

-Users cannot play music as a playlist. The playlist is only used as a grouping tool

-Users will be able to listen to music playlists

**Testing**

Create/Read

Test: User creates category.

Expected result: Category Node created on CMS with the name value.

Result: Category Node created with the Name equal to the input value and pulled through the front-end (category listing and media edit dropdown).

Test: User creates Media.

Expected result: New File created on CMS and pulled through on the Media Listing page.

Result: New File created on the CMS and data pulled through front-end.

Test: User can view Media details.

Expected result: User can see media file path, name and other properties on view.

Result: User sees media file path, name and other properties on view.

Test: User creates Playlist.

Expected result: Playlist Node created on CMS with the name value from the front-end.

Result: Playlist Node created with the Name equal to the input value and pulled through the front-end.

Test: User adds media to the Playlist.

Expected result: Selected media is added to the Playlist Node’s media picker and pulled through the front-end.

Result: Selected media is added to the Playlist Node’s media picker and pulled through the front-end (edit playlist screen).

Update

Test: User can re-edit category name.

Expected result: User updates the category name and it saves to the CMS. This data is pulled to the front-end in the listing and category dropdown.

Result: User updates the category name and it saves to the CMS. This data is pulled to the front-end in the listing and category dropdown.

Test: User can update media file category.

Expected result: User can change/set the category. It will reject ‘select category’ as a value.

Result: Media has an updated category and data is pulled to the front-end.

Test: User can re-edit playlist name.

Expected result: User updates the playlist name and it saves to the CMS. This data is pulled to the front-end listing page.

Result: User updates the playlist name and it saves to the CMS. This data is pulled to the front-end listing page.

Destroy

Test: User can delete selected category.

Expected result: User deletes category on the front-end and it’s also deleted on the CMS. Category will no longer be shown on the view and the dropdown.

Result: User deletes category on the front-end and it’s also deleted on the CMS. Category will no longer be shown on the view and the dropdown.

Test: User can delete selected media.

Expected result: User deletes media on the front-end and it’s also deleted on the CMS. It will no longer be available on the media picker loop and listing view.

Result: User deletes media on the front-end and it’s also deleted on the CMS. It will no longer be available on the media picker loop and listing view.

Test: User can delete selected playlist.

Expected result: User deletes playlist on the front-end and it’s also deleted on the CMS. Playlist will no longer be shown on the listing view.

Result: User deletes playlist on the front-end and it’s also deleted on the CMS. Playlist will no longer be shown on the listing view.