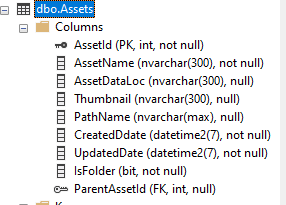
# Asset Management – Solution

## Major Tasks Pending

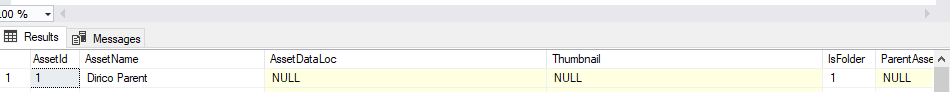
1. Updating the metadata information in parallel, while creating/uploading assets
   1. This can be achieved by using the AnalyzeImage functionality of computer vision
   2. A database table metadata has been created. If searching using tags are required, table of tags also can be created.
2. Supporting for uploading video
   1. From my understanding the videos can be manipulated using Media services provided by Azure
   2. Thumbnails can be generated using Media Encoder intelligently, by using the service
   3. Cropping a video can be done by the media encoder, by presetting the dimensions in the Network database table

## Folder Structure Solution

For the task, I see that only the immediate children are only needed to be fetched. The entire hierarchy need not be fetched here. Thus, I opted the Adjacency list approach to store the folders and assets (Else materialized path approach or nested set tree approach could be suggested). So, this would be using a self-referencing foreign key.



I have created a column Pathname but right now it is not used. But for an extensive search in future this column can be utilized. When running the solution for the first time, the migration of sql process is automatically done and the first row for Asset table is seeded. The first row is seeded as below.



## Rest API

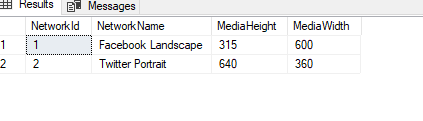
The rest API has been developed with .net core 3.1. Swagger has been configured to test the rest api.

All the major services are added with the lifecycle as scoped, whereas singleton is used for configurations. Initial records are seeded from the application as follows

* Facebook and twitter portrait, with their suggested image dimension are added into network table
* The parent record in Asset table is added as “Dirico Parent” with assetId “1”

While creating an asset, the thumbnail creation and upload to blob storage will be a sync call, other tasks like uploading the actual media, creation of variants, will be async calls and run in the background. Thus there might be slight delay, if an image is created and immediately accessed in asset details page.

More higher clarity pictures can be created for variants if required. Currently I have used the default GetThumbnail version for less complexity. Computer vision for thumbnail creation by default can only generate with a maximum height and width of 1040. Thus, in this solution, if the dimension of a social media exceeds, we will have to cut short it based on the aspect ratio in the Networks database table. Below is the seeded values



* This can be improved by selection of interesting area using computer vision automatically, then creating a variant based on the coordinates from the interest area.

## React Application

* The homepage has the functionality to add a folder or an image. A gridlist of assets uploaded is also shown. These shown images are thumbnails.
* The assetdetails page shows the actual image and the variants created in background
* The react application uses redux for the state management. The folderstruct component uses the common state. While creating a folder or asset from the homepage, it dispatches to update the redux state.

## Pending tasks yet to be done

1. Media needs to be loaded with lazy loading and order by latest asset created
   1. Only trigger the get request when more media is required for eg when scrolling
   2. From the front end, pass the starting index and the count.
   3. In the subsequent request pass the (starting index + count), and this would be repeated till the total count of images in the folder
2. Generic Exception handling
3. Showing responsive designs for gridlist with a mobile first approach
   1. I used the gridlist with material UI. I was not able to implement responsiveness directly. I think I need to use the default grid in ‘material ui’ itself
4. If assets size may grow like more that 20k, need to tweak the system for a container-based approach in azure blob storage. May be by month or so a new container shall be created.
5. Caching and Akame cache need to be configured for the images.
6. Move secrets to azure key vault
7. Creating env file for the react application to configure the API url. Right now it is in config.ts
8. Implement Logger

## Running the application

### Rest API

1. Update the connectionstring in \AssetBook\AssetManagement\appsettings.json.
2. Run the application. The migration and seeding will be done automatically.
3. If the react application is not running in <https://localhost:3000>, update the same in CorsOptions origin in AssetBook\AssetManagement\appsettings.json.

### React App

1. If at all the baseurl for the api needs a change, it needs to be updated in ‘src/config.ts’. This is because currently I have not created the env configuration.