Frontend:

The frontend is a single page application built with the Angular Javascript framework that is written entirely in Typescript. The primary libraries used in the application are Dropzone.js for uploading images to the S3 backend storage, AWS Amplify for connecting user accounts and permissions to the AWS cognito service and directly interacting with the S3 backend storage, and Angular Material for creating and presenting the material design influenced UI components present in the frontend.

The frontend is divided into four main components composing the view, with three primary services acting as controllers that interact with the backend services. The four main components include the home component, which represents the main page where users can upload images, the gallery component, where users can view and update the tagging information of images they have already uploaded, the signup component, where users can register for a new account, and the signin component, where users who have already registered for an account can login to the application. The three primary services consist of the gallery service, which handles the getting and displaying of the images a user has already uploaded, the auth service which handles user authorization and authentication, and the store-image service, which handles the storing of images with the correctly associated tags when they are uploaded to the S3 bucket.

The functionality of note lies in the use of observables to display images within the gallery component. Through the use of Angular's observable pattern, when a user uploads an image it will automatically be displayed in the gallery, and when an image is deleted it will automatically be removed from the gallery view without any additional requests to the backend.

Backend:

Amazon Web Services (AWS) RDS was used to store all user and image data in a MySQL database instance. Each user entry in the database had the name of user, a collection of all existing images that belongs to the user associated with each category (or tag), a collection of all categories associated with each user's image, and users that are allowed to access a particular image.

The backend code (basically a collection of Lambda functions on AWS Lambda) retrieving and parsing data from the frontend was primarily hosted on AWS Lambda and the backend also communicates with AWS API Gateway and AWS RDS. Lambda functions connect to a MySQL database instance on AWS RDS and make use of data stored there for an operation requested by the frontend. AWS API Gateway was used to create and manage API endpoints for the frontend to be able to call specific functions hosted in the backend to either update or read certain user data. HTTP requests from the frontend (or anyone with access to the application's API endpoints) are parsed through AWS API Gateway before the information in the request is sent to AWS Lambda.The response returned to the frontend at the end of a Lambda function is in JSON format and contains a HTTP status code and a body which was usually a list, a dictionary or a string containing an error message.

Lambda functions were written in Python in conjunction with libraries that assisted with converting between JSON and Python data and communicating with the database. The json library was used to integrate the Python code with the JSON data from the MySQL database

instance while the PyMySQL library was used for executing SQL statements within AWS Lambda functions.

Tasks that the Lambda functions supported:

- Adding a new user to the database.
- Adding a new image and the categories associated with it for an existing user in the database.
- Updating the categories associated with a user's image.
- While adding/updating categories for an image, each tag was indexed so that all images associated with a particular category can be quickly looked up with the use of hashing.
    - Each image was also indexed so all tags associated with it can be easily looked up.
    - All images in the database are stored with their name and the true owner of the image attached to it so that multiple images with the same name, but owned by different users can be distinguished.
- Deleting an image for a user.
- Granting or removing authorization for a user to access an existing image.
- Giving the frontend all the images that a user owns and the respective categories associated with them.
- Giving the frontend all the images that a user has access to, but excluding user's own images.
- Giving the frontend all the images for a user, each image with a list of authorized users attached to it.