Making a Cloud Based Web Application with Amplify and Flutter

Carlos Puerta

December 11, 2023

Abstract

In this paper, I go over the design and set up of a cloud based web application using Flutter for development and Amazon Web Services to host and store user data. This approach is increadibly lightweight as the final application is static and all of the processing is done client-side. The final application allows users to view and share photos with tags or topics.

Contents

1 Introduction **3**

2 Design **3**

2.1 Features 3

2.2 Flutter 4

2.3 Amazon Web Services 4

2.3.1 Amplify 4

2.3.2 Amazon S3 4

2.3.3 DynamoDB 5

2.3.4 Amplify Hosting 6

2.3.5 Amazon Cognito 7

2.4 Server vs. Serverless 8

2.5 Permissions and Policies 8

2.6 Final Design 9

3 Implementation **10**

3.1 Pages 10

3.1.1 Home 10

3.1.2 Upload 11

3.1.3 Details 12

3.1.4 Full View 12

3.2 Demo and Testing 13

3.3 Code 13

4 Conclusion **15**

1 Introduction

Tasked with creating a web application to upload an share images, I will walk through the features that were specified, the design choices, and the services used to bring the entire application to life. All of the choices were made thinking not only about the present project specifications, but also about the potential growth and expansion in the future.

2 Design

Coming up with a design for this application involved making a lot of different decisions, from what framework to use, to how should images be stored. Below I will present the decisions I made alongside the reasoning for each one.

2.1 Features

The specifications given by the project included the following features:

1. Allow users to upload images.
    - User specifies their username (no authentication).
    - Images can have tags.
    - Users can select existing tags or add new ones.
    - Uploading saves the date of upload.
2. Display all uploaded images in a grid.
    - Display tags of picture.
    - Display username that uploaded the picture.
3. Clicking an image shows a bigger version of it alongside the image metadata.
4. Clicking the image while metadata is visible will show the full resolution image.
5. Users are able to search and filter based on image tags.
6. Users can delete uploaded images.

2.2 Flutter

Flutter was my choice for designing the entire front-end user interface. I chose Flutter because once the backend and frontend are done, the produced web app is served statically. This means flutter runs client-side which makes serving the application much simpler. Because of the static nature, scaling the application only requires making sure the host can handle the incoming traffic.

Flutter also leverages Google's Material 3 Theme, which makes it easy to design consistent and beautiful applications that can be rendered in a variety of screen sizes. The ability to render your application on varying screens, also known as responsive design, saves an immense amount of development time. Responsive design is critical in an age of accessing web applications from phones, tablets, laptops, and desktops.

More information on Flutter can be found at `https://flutter.dev/`.

2.3 Amazon Web Services

In order to apply many of the concepts learned in class, Amazon Web Services (AWS) was used for handling the storage and hosting solutions.

2.3.1 Amplify

Amplify is a great development tool that offers support for many development frameworks and easy connections to backend services like S3, DynamoDB, and continous deployment via Github. As such, I set up an Amplify environment which allowed me to easily add the necessary AWS services that I would be using, in this case, S3, DynamoDB, and Amplify hosting.

2.3.2 Amazon S3

For storing static images, I could have opted to store them directly into a database as a byte string. However, I opted for storing the files in an S3 bucket to lessen the load on the database. For the scale of this project it would likely be unnoticeable, but when hosting millions of images, setting up a content delivery network (CDN) would be more performant.

Although Amazon has specific solutions for CDN, such as CloudFront, this service is specialized in content delivery accross many regions. CloudFront is also more expensive than S3. If our user base is localized, there is very little benefit to using CloudFront, which is ultimately why I chose Amazon S3. If

the project had been designed to be worldwide, CloudFront would have been a better option.

Images were stored in S3 with their filename as their key for retrieval.

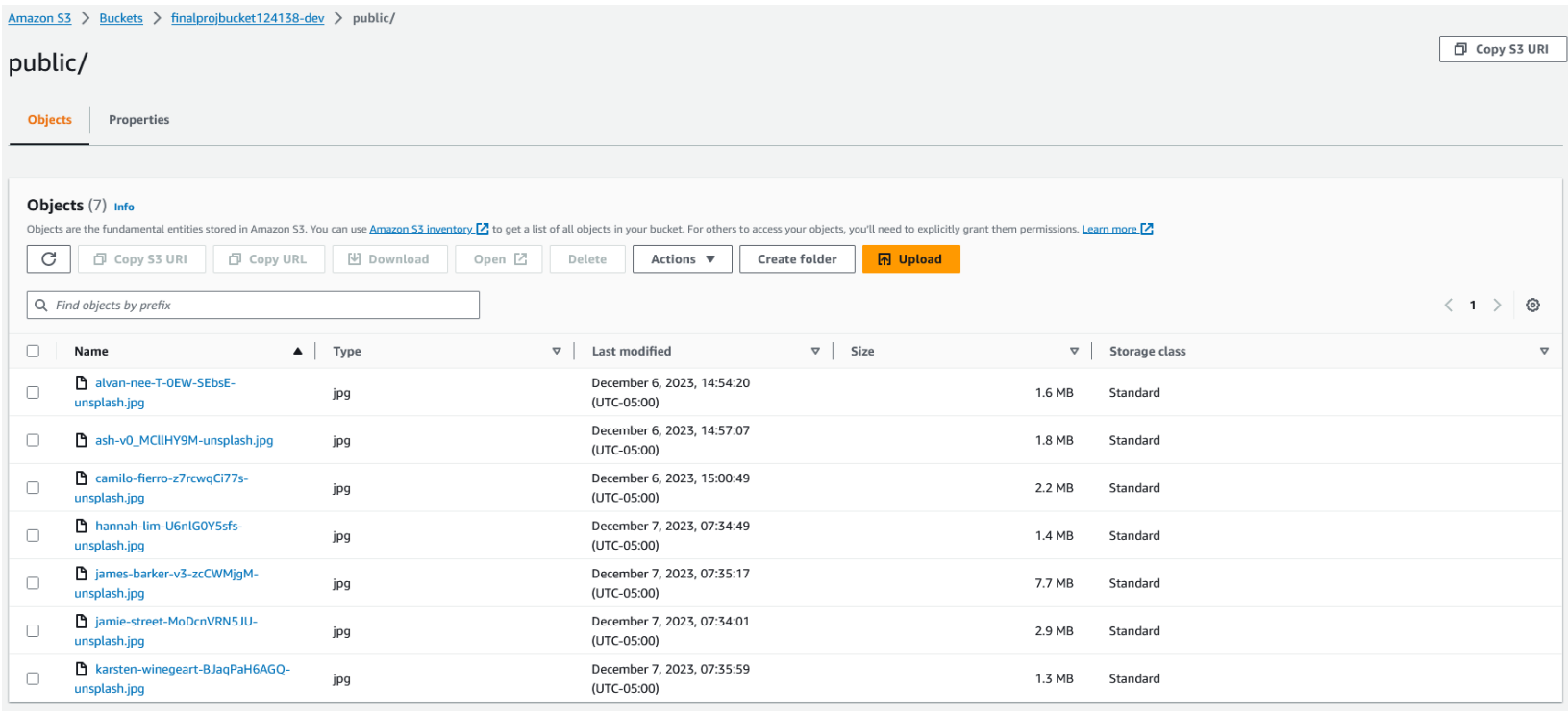

Figure 1: S3 bucket showing some of the files currently stored.

2.3.3 DynamoDB

Although image files carry their metadata in their byte string representation, I chose to extract the metadata and store it separately in a database. This was done for a few reasons.

The first reason was that the database response is faster than downloading images, especially large ones. By loading the metadata first, I can display the grid of images quickly, and simply load the images asynchroniously, giving a more seamless experience to the user.

The second, and biggest reason, was scalability and addition of features. As it was designed, the database is very simple with only a single table with the following fields:

- Key: String (Not Nullable)
- Resolution: String (Not Nullable)
- Owner: String (Not Nullable)
- Upload Date: Timestamp (Not Nullable)
- Colorspace: String (Not Nullable)
- Tags: List[String] (Nullable)

This table (or another table) could very easily be expanded to include additional metadata, such as location or device used, but it could also include metrics about the image. For example, if we wanted to track number of views, likes, favorites, etc, we can store that in their respetive fields. Having a connection to a database sets us up in a better position to grow the application in the future.

Although DynamoDB is a NoSQL database, when setting it up through Amplify, it is done using a GraphQL schema for both pushing and pulling information. This allows DynamoDB to be highly structured while remaining very flexible.

2.3.4 Amplify Hosting

Another feature that was stellar from Amplify was the hosting ability. I could simply connect to a codebase, point it to the entry point, and it would host the application for me.

In my case, I created a separate repo that hosted exclusively the web build of the application and connected that to Amplify hosting. This connection is terrific because it allows development to continue in parallel, but also loads changes every time the codebase is updated. As such, any new features, bugfixes, or changes, can be done quickly by pushing a commit to the repo.

By default Amplify hosting assigns a URL that can be used to access the application, but this can be changed to host at any domain owned. Hosting at an owned domain is great for branding, visibility, search engine optimization, and user access. Having this option is very helpful, although I did not use it in this project and simply used the assigned address.

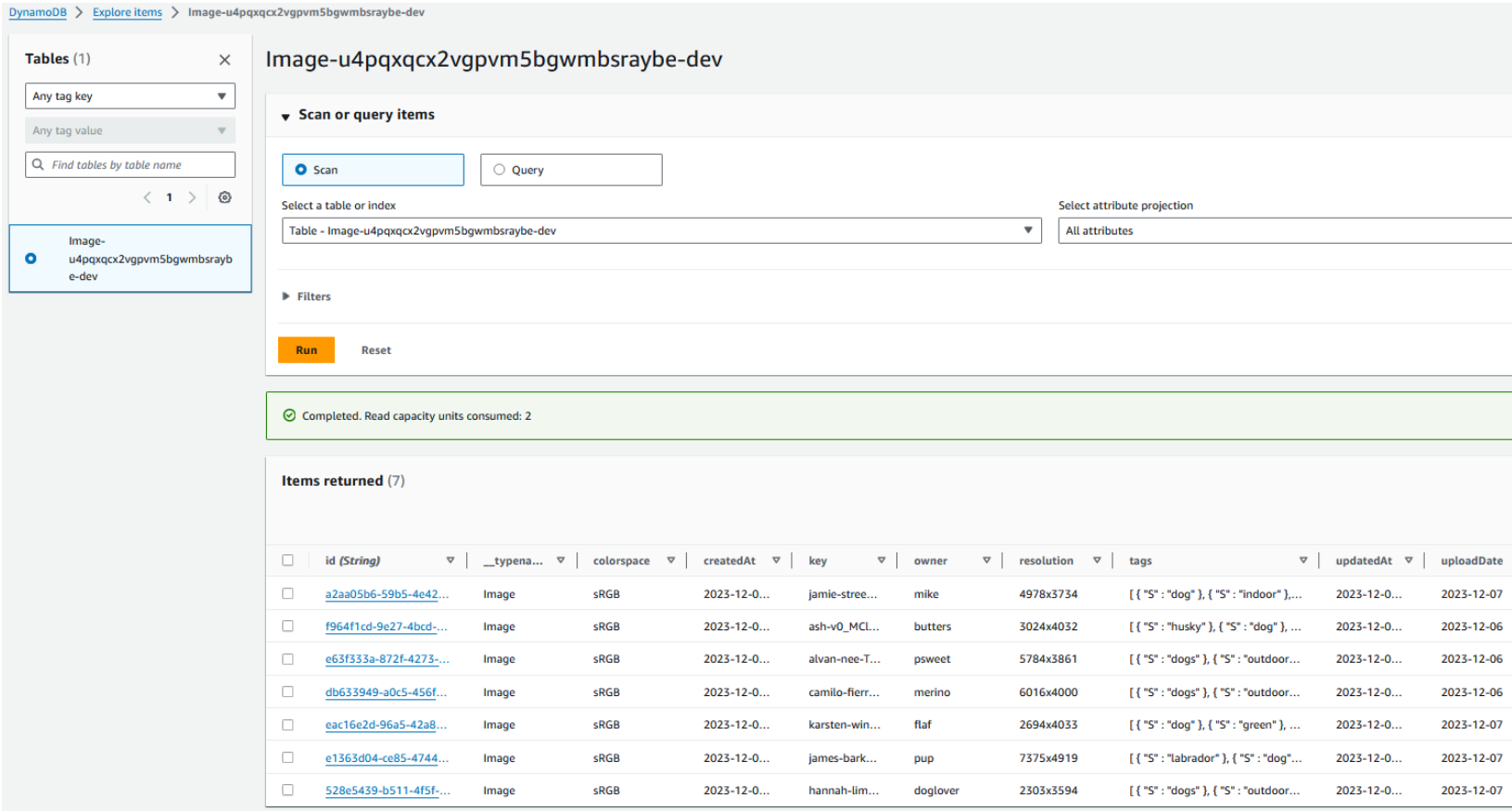

Figure 2: DynamoDB showing some of the information currently stored in the database.

2.3.5 Amazon Cognito

Amplify requires some level of authentication or guest tracking in order to work with certain services like S3. As such, Amazon Cognito was used to track the users. There was no user sign in, sign up, or authentication, and all users were treated as 'guest' users.

Although the setup of Cognito was very primitive, expanding it to have a fully functional authentication system is very straightforward. Cognito also supports more complex authentication settings such as two factor or code authentication to ensure very thorough protection of your application and user data.

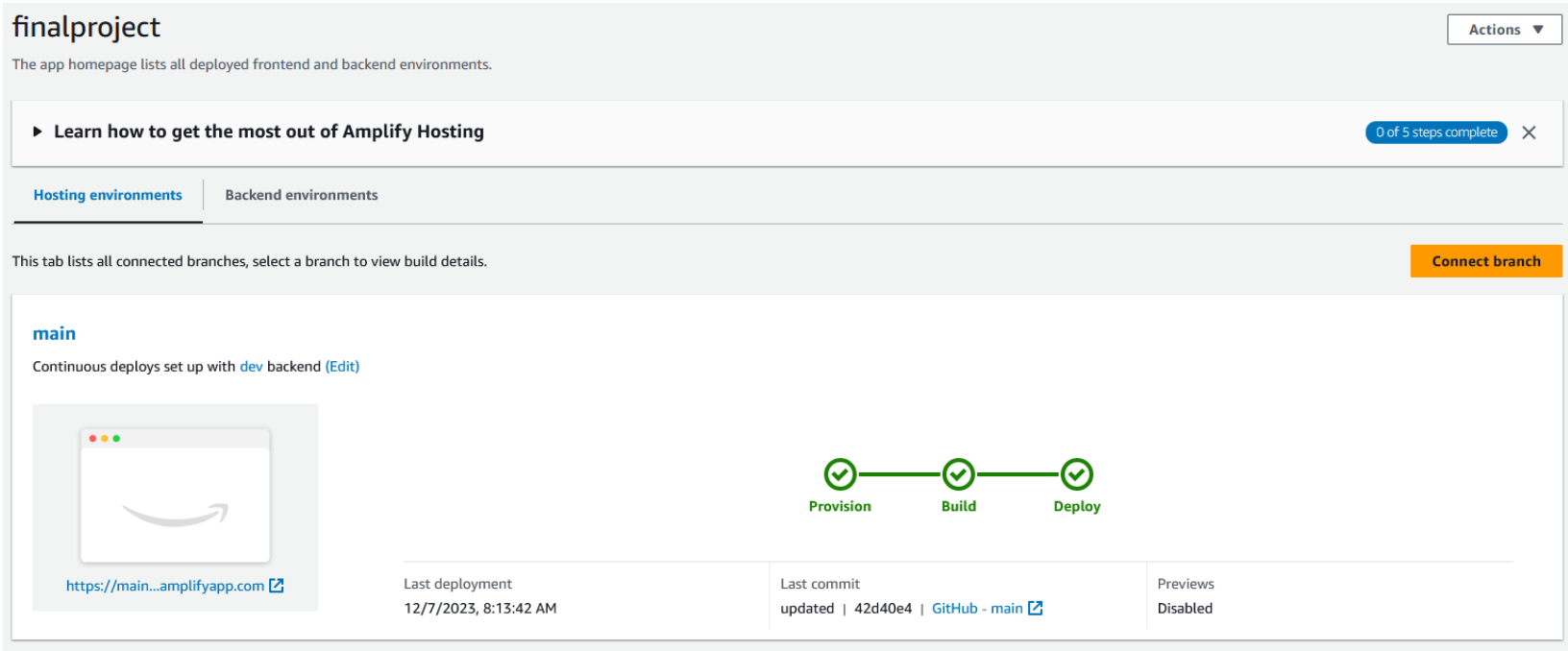

Figure 3: Amplify hosting showing the application up and running on their assigned domain.

2.4 Server vs. Serverless

As previously mentioned, Flutter offloads processing to the client. This makes it cheaper to run this application as opposed to running AWS lambdas. The client is doing the processing and thus we do not pay for it. Hosting the application on an EC2 instance is also unecessary since the final application is static, and it only has to serve these files. As such, using an S3 bucket, elastic beanstalk, or Amplify are easier, cheaper, more lightweight approaches to serve the files.

2.5 Permissions and Policies

As was briefly mentioned, Cognito was set up to treat all users accessing the application as 'guest' users. Since there was no authentication, the bucket and database were open compeletely. Users had read and write access to all data, regardless of who they were. Needless to say, this is no secure and should not remain the case in a true web application in production.

Implementing better authentication would allow us to write more strict poli-

cies to control who has access to what. For example, read access to everyone, but only write access to the user's entries.

2.6 Final Design

To summarize, the final application will be developed using Flutter. Flutter will connect to AWS (S3 and DynamoDB) via Amplify. After the frontend and backend are completed, Flutter will compile down to a set of static files. These files will then be served by Amplify hosting. Users accessing the application will be served the static files and their instance of it will run locally on their machine. As such, the only resources that I must account for are the requests incoming and outgoing to the hosting, database, and bucket. Since I must only manage traffic load, and not application load, the scalability of the application is very good and cost efficient.

This set up also allows the creation of complex features or computations that are run on the client machine. For example, if I wanted to use a custom model to upscale images, I can offload those computations to the client and save the resource cost that I would otherwise be responsible for if I used AWS Lambdas, EC2 instances, or other similar solutions.

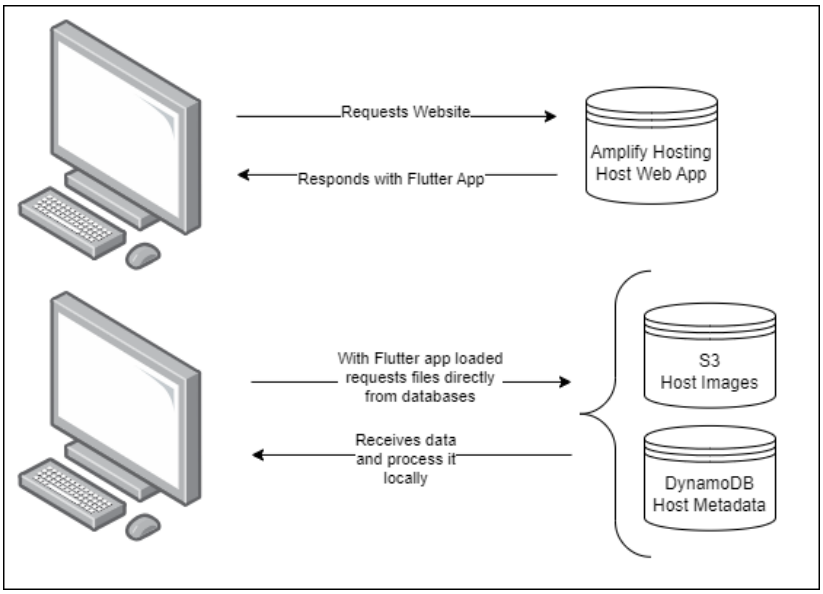

Figure 4: Complete architecture design.

3 Implementation

3.1 Pages

To accomodate all of the features previously mentioned, I designed 4 pages. These 4 pages had very simple design with ease of use as their primary focus.

3.1.1 Home

This page served as the grid that showcased all of the images that were currently uploaded. We can see each image has the username of the person who uploaded it, as well as the tags of the image.

At the bottom of the page we see a search bar that allows us to filter for specific tags. On the right side of the search bar, we see the upload button which brings up the upload page.

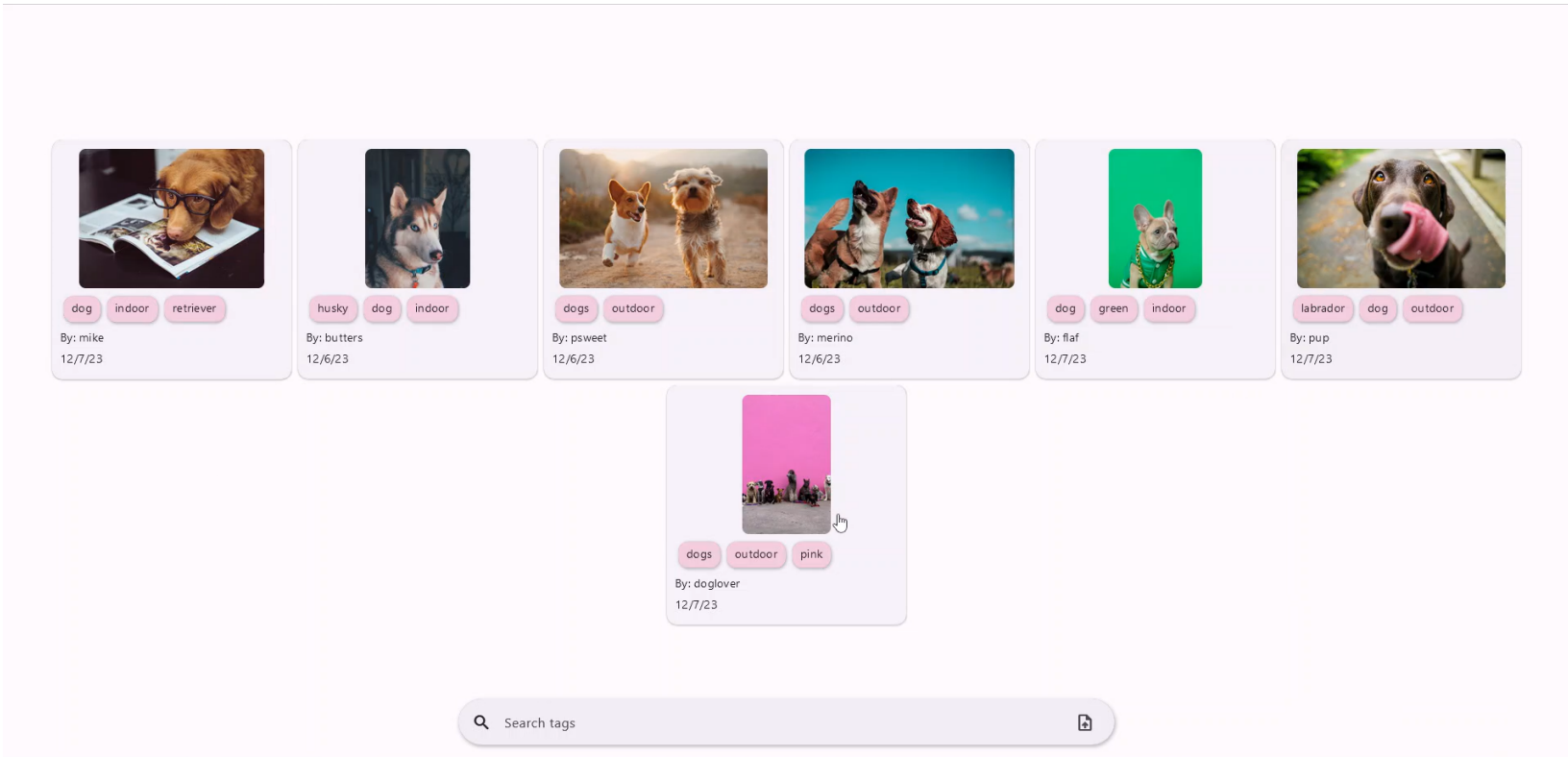

Figure 5: Home page showing image grid and search bar.

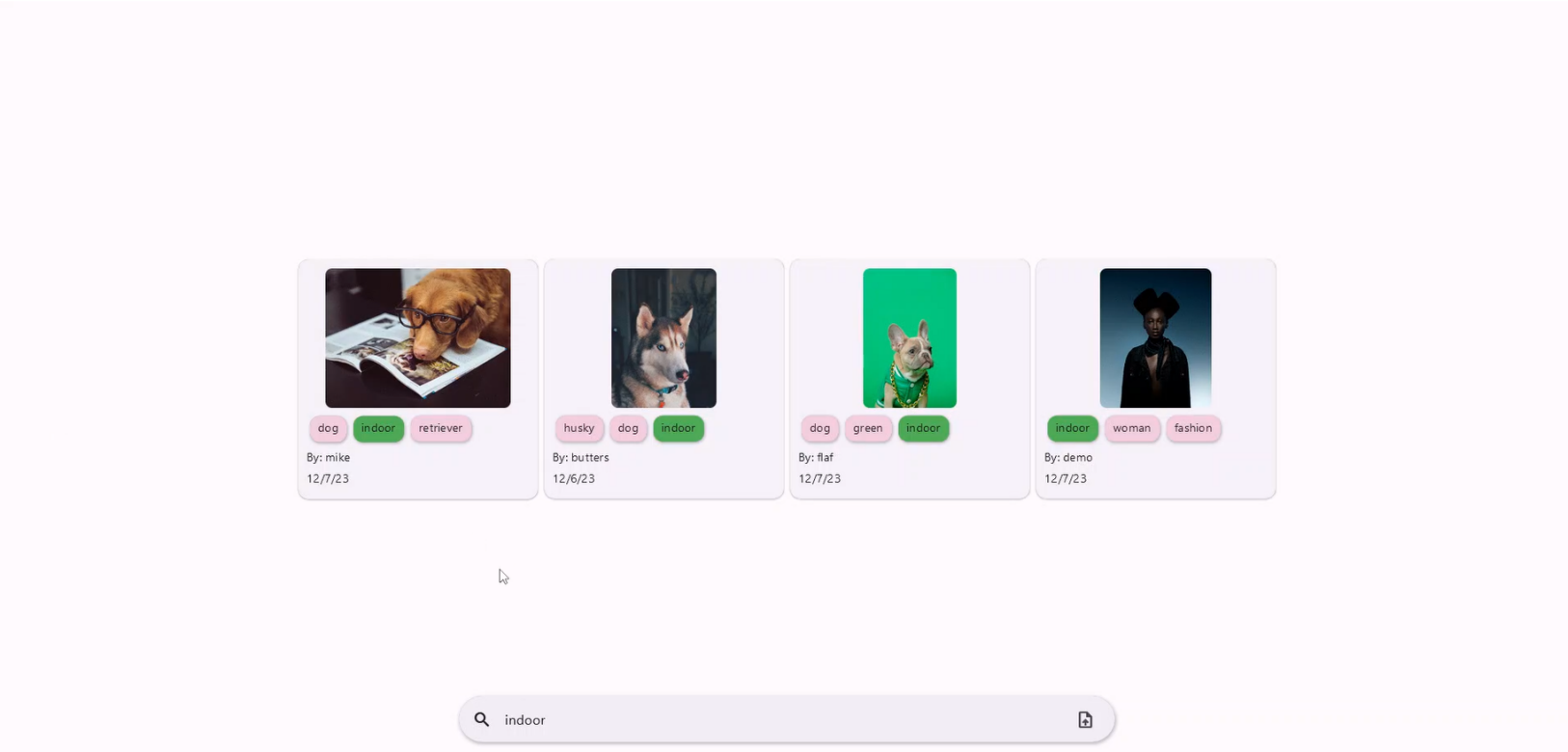

Figure 6: Home page showing images filtered by searching for 'indoor' tag.

3.1.2 Upload

This page allows you to select a file, input the user name, and add (or select) any number of tags for your image. At the bottom there are cancel and upload buttons.

- The upload button is not active unless there is both an image and a username given. This is to prevent users from uploading incomplete information.
- Tags are optional. An image may have zero or infininite tags. In the future, a tag limit would be helpful.
- Upload date is captured automatically when the user clicks submit, this is the reason it is not a field.

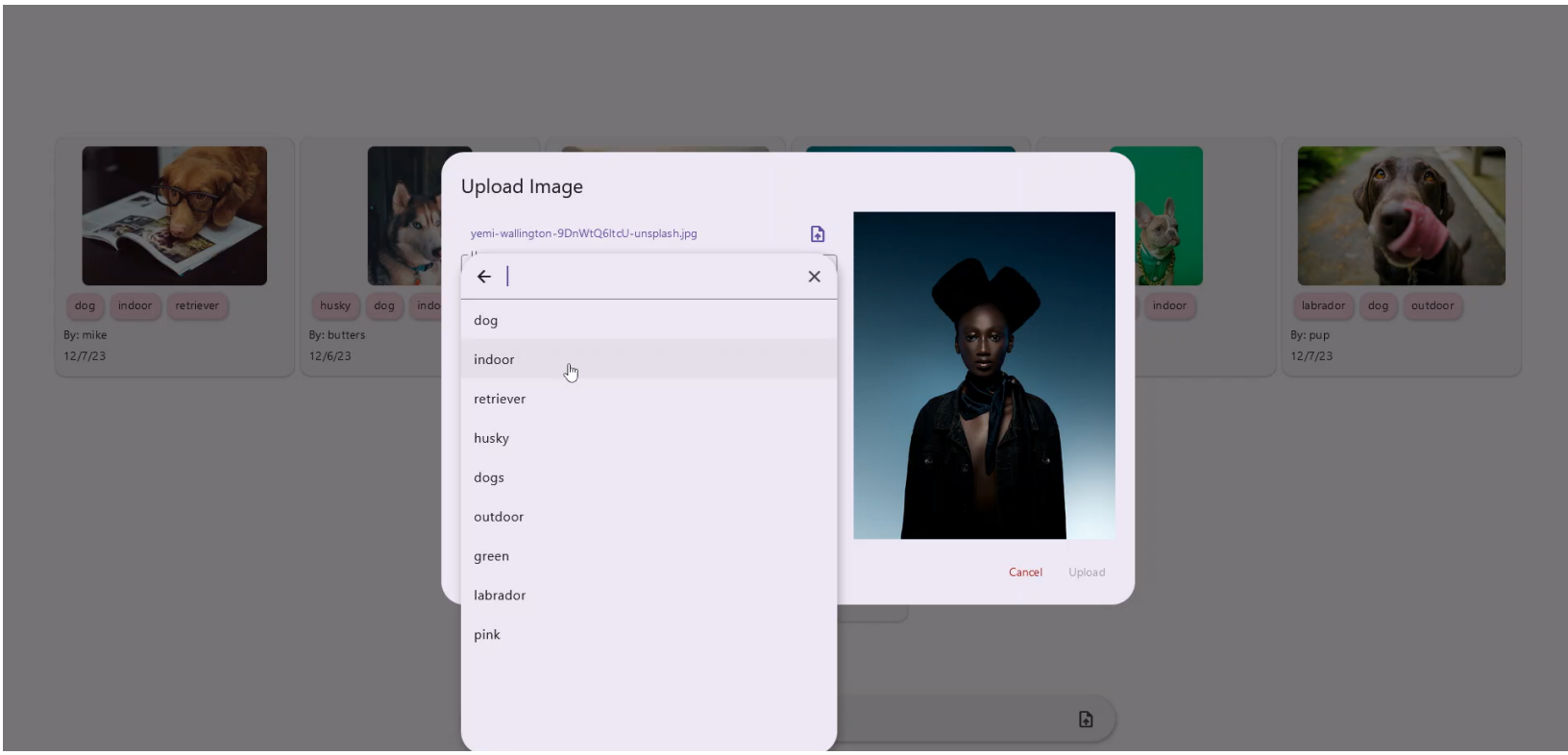

Figure 7: Upload page showing selection of existing tags.

3.1.3 Details

When clicking an image from the home page, the details view is shown. This displays the image in a much larger size, alongside the metadata and the option to delete the image.

In the current implementation, deleting images does not require confirmation. This is something that should be updated to avoid simple mistakes from having destructive consequences.

3.1.4 Full View

Clicking the image while looking at the details brings up the full resolution version of the image. On smaller monitors, this often means that the image can only be seen partially and users must scroll to be able to see the entire image. In the example image, the monitor had a resolution of 1920×1080 versus the

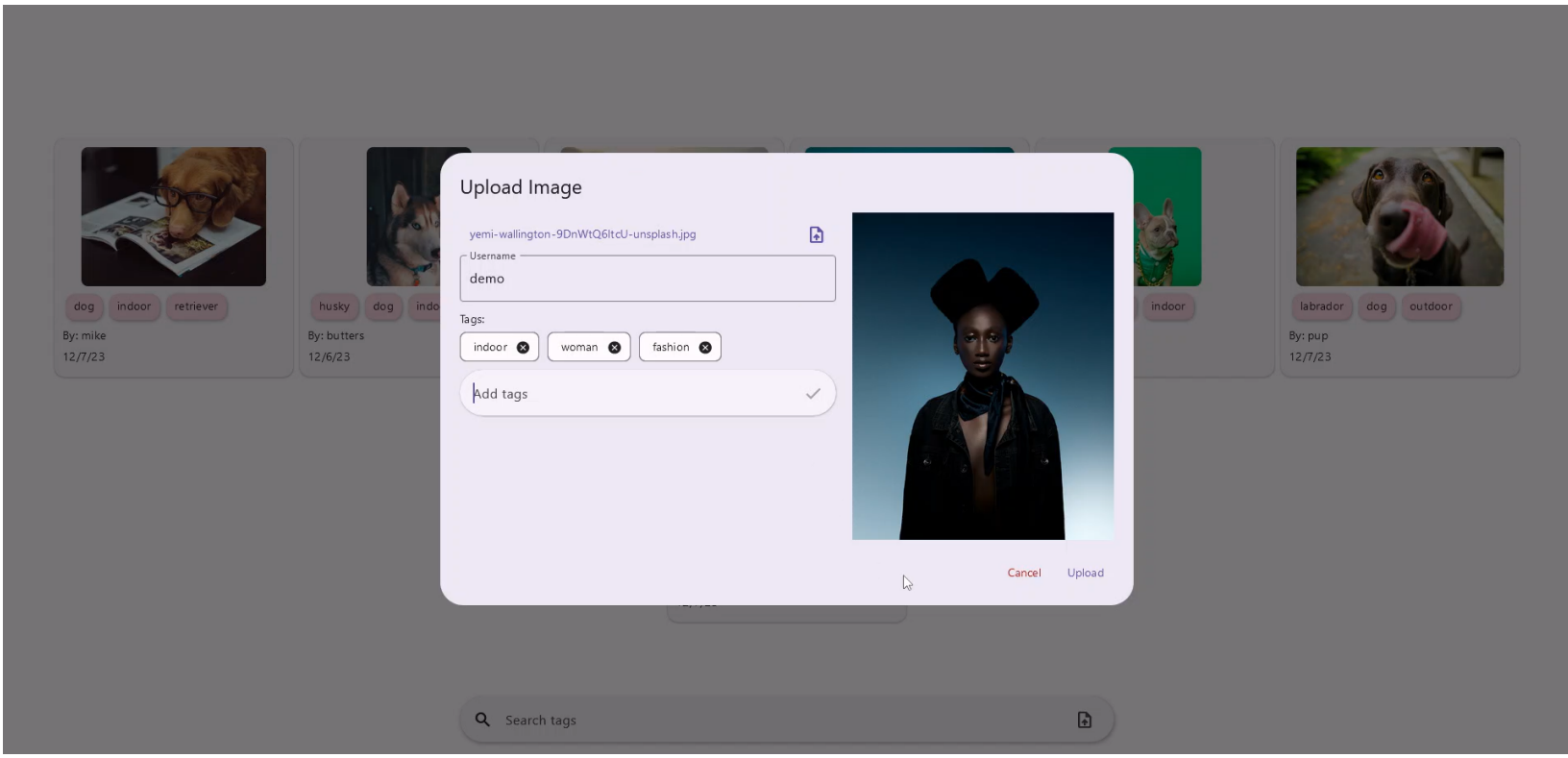

Figure 8: Upload page showing the everything filled out and ready to upload.

image resolution of 2303×3594 which is why the image is partially seen.

3.2 Demo and Testing

All of the features previously mentioned were successfully implemented and tested both locally and when hosted in the cloud. A short video recording was made showcasing everything in working order, from the user interface, to showing the live updates to the database and bucket, while it was all being hosted in the cloud.

The video can be accessed here `https://youtu.be/Rt02GCPLRdY`

3.3 Code

The codebase for this project was large, complex, and the development code is very different than the fully built web application. As such, I opted for not in-

Figure 9: Details page displaying image metadata and delete button.

cluding the code in this paper and instead have made the code available publicly on Github. The code includes the Flutter project with all of the dependencies needed, but does not include any of the relevant files from Amplify or the final build; this is to protect my AWS account. Rebuilding the application is fairly simple, and instructions are provided in the Github repo.

The repo can be found here: `https://github.com/psweet/650-final-project`.

Figure 10: Full view page displaying image in native resolution. If the image is too big the user must scroll to see everything.

4 Conclusion

Through this project I learned a lot about practical application of concepts we had discussed in class. This particular project had a lot of features which required many considerations. I am certain other people would approach it differently, but from I believe there were different solutions to this problem, each with pros and cons. Nevertheless, I think my solution is great and I created a very clear, scalable, and intuitive application, which allowed me to improve my cloud computing skills.

References

[1] Google. *Flutter*. Last accessed 11 December 2023. URL: https://flutter.dev/.

[2] Carlos Puerta. *DATA650 Final Project Demo*. Last accessed 11 December 2023. 2023. URL: https://youtu.be/Rt02GCPLRdY.

[3] Carlos Puerta. *DATA650 Final Project Github*. Last accessed 11 December 2023. 2023. URL: https://github.com/psweet/650-final-project.

[4] Amazon Web Services. *Amazon Cognito Documentation*. Last accessed 11 December 2023. URL: https://docs.aws.amazon.com/cognito/.

[5] Amazon Web Services. *AWS Amplify Documentation*. Last accessed 11 December 2023. URL: https://docs.aws.amazon.com/amplify/.

[6] Amazon Web Services. *Set up Amplify GraphQL API*. Last accessed 11 December 2023. URL: https://docs.amplify.aws/flutter/build-a-backend/graphqlapi/set-up-graphql-api/.

[7] Amazon Web Services. *Set up Amplify Storage*. Last accessed 11 December 2023. URL: https://docs.amplify.aws/flutter/build-a-backend/storage/set-up-storage/.

[8] Unsplash. *Unsplash*. Last accessed 11 December 2023. URL: https://unsplash.com/.