Lab 5

Due Apr 11 by 11:59pm **Points** 100 **Submitting** a file upload

CS-554 Lab 5

For this assignment, we're going to apply what we've learned with Redis, React, and now finally GraphQL to make an image curation website similar to Pinterest, called Binterest.

You are a developer working at a new startup called Binterest who wants to make a **garbage** copy of Pinterest, so here you are on your first day…

Data

Image posts are the primary pieces of data that we're going to concern ourselves with. A post will be defined in the following format:

```
type ImagePost {
    id: ID!
    url: String!
    posterName: String!
    description: String
    userPosted: Boolean!
    binned: Boolean!
}
```

In the above schema, `id`, `url`, `posterName` and `description` will be populated using the **Unsplash API (https://unsplash.com/documentation)** (it doesn't matter which URL you use for the image). `posterName` represents the author of the image post. `binned` will represent whether the user has added the image to their bin or not. If a user adds a post to their Bin, the post is then saved into Redis for later use. `userPosted` represents whether this post came from the Unsplash API (false) or if it was Binterest user-posted Image Post (true)

Your designer has provided a rough mock-up of the proposed app, Binterest. **Here's (https://www.figma.com/file/F4K8KoS8re5sv133DOP2Ld/Untitled?node-id=0%3A1)** a demo of what what the Frontend will look like. You do not need an account to view the document. Click on the top-left button to look at the other pages.

You do not need to implement CSS, the only thing that we require is that you follow the general route structure of this example. How you choose to lay out the elements on each page is up to you as long as it makes sense. (ie: use a `<ul>` or `<ol>` for lists instead of a bunch of `<p>` tags, etc.).

Do not just dump JSON payloads to the frontend, you will lose major points! (Or get fired from Binterest!)

Backend

This assignment's backend will be implemented using **Apollo Server (https://www.apollographql.com/docs/apollo-server/)**. In particular, you will be to support the following queries and mutations to make this webapp functional:

Queries

1. `unsplashImages(pageNum: Int) -> [ImagePost]`: You will map **this route in the Unsplash REST API (https://unsplash.com/documentation#list-photos)** in this query's resolver function to create ImagePost objects from the results of the REST API.
2. `binnedImages -> [ImagePost]`: You will go to redis to retrieve the ImagePost objects that the user has added to his/her bin
3. `userPostedImages -> [ImagePost]`: You will query all images that the user has posted.

Mutations

1. `uploadImage(url: String!, description: String, posterName: String) -> ImagePost`: This mutation will create an ImagePost and will be saved in Redis. Outside of the provided values from the "New Post" form, by default, the following values of `ImagePost` should be:
   - `binned`: false
   - `userPosted`: true
   - `id`: a uuid
2. `updateImage(id: ID!, url: String, posterName: String, description: String, userPosted: Boolean, binned: Boolean) -> ImagePost`: This mutation will take care of any updates that we want to perform on a particular image post
   1. If this image was not previously in the cache, and the user bins it, then add it to the cache using data from React state.
   2. If an image post that came from Unsplash and was unbinned (binned set to false), you should also remove it from the cache
3. `deleteImage(id: ID!) -> ImagePost`: Delete a user-posted Image Post from the cache.

Frontend Routes

Now that we have written our backend queries and mutation, we can now move onto the frontend!

Here, we need to use **Apollo Client (https://www.apollographql.com/docs/react/)** to interact with our backend to provide data to our frontend.

- `/`
  - Should display a list of image post results from Unsplash.

- A user should be able to click a “Get More” button in order to perform another query to get more images from Unsplash.
- A user should be able to click on a “Add to Bin” button found on a particular image post to “bin” a post.
    - Once a post has been added to the user’s bin, the “Add to bin” button should be toggled to “Remove from bin”
- Each image post should contain a description, image, and the original poster/author

- `/my-bin`
    - Should display a list of image posts that the user has previously added to their bin.
    - This page should have similar functionality as the `/` page.
- `/my-posts`
    - Should display a list of image posts that the user has posted.
    - This page should have similar functionality as the `/` page.
        - However, the user should be able to delete their posts from this page
        - Users can also only upload new posts from here as well.
- `/new-post`
    - Should render a form that has fields for:
        1. Image URL (we only allow image urls that are already on the internet)
        2. Description
        3. Poster name (this should in theory be the same all the time, but user identification is out of the scope of this lab)

The Senior Developer tells you that since the `/`, `my-bin`, and `self-posted` pages look very similar, he hints that maybe you can create one common React component instead of three to handle list logic and then use wrappers around this component to provide page-specific functionality.

Special Behavior

1. Binning Posts: when a user adds a post to their bin, if the post came from the Unsplash API, then you should save the post to the cache. However, if the post is user-posted, then even if the user doesn’t add the post to their bin, it should stay in the cache.

2. Any failures, like the failure to bin an image, or to delete should fail gracefully. (ie: if it fails, display a message)

Strategy for Completing this Assignment On Time

This assignment is not a trivial assignment, so please start early! Project/Time management is crucial in any company, and Binterest is no different!

The following are suggested milestones that you can complete in order to complete this assignment in a timely fashion:

Each section will be estimated in terms of how long each part will take.

A block of time is an arbitrary amount of time, but will serve as an estimate for how long each part will take to complete compared to the others.

1. Figure out how Apollo Server works: 1 block
    - Go to the docs and do their mini tutorial. It should take no more than 10 minutes
2. Finish Backend routes: 3-6 Blocks
3. Finish Frontend without any Apollo Client logic (skip or use dummy data): 6-8 blocks
4. Figure our how Apollo Client works: 1 block
    - Go to the docs and do their mini tutorial. It should take no more than 10 minutes
5. Implement Apollo Client into your React Frontend: 2-3 blocks

Extra Credit!!

So far, we've been using redis as we would perhaps use another other database, so why don't we use a cool feature that redis provides? Back in the Binterest offices…

Once you finished the MVP (do not attempt without finishing basic requirements!), your boss asks you if you can implement a new feature for him in return for a one-time bonus, so if you are eager, then read on…

Let's imagine a scene at a feature planning meeting at Binterest…

Your boss wants to grow a sense of community in the Binterest userbase, so he wants to add a feature that allows a user to see how popular an image that he/she binned is. He wants to allow users to see if their tastes are mainstream or not. Therefore, your boss wants you to add a new feature that arranges the content that the user has binned in order of popularity by `numBinned`.

Here are the changes that you need to make outlined:

Data

The GraphQL type, `ImagePost` will change slightly. We will add a new field called `numBinned`, which represents the number of people who like/bin the image. This field will be populated by the `likes` field from the response of the Unsplash API.

```
type ImagePost {
    id: ID!
    url: String!
    posteName: String!
    description: String
    userPosted: Boolean!
    binned: Boolean!
    numBinned: Int!
}
```

Backend

Now, you need to implement a new query function:

- `getTopTenBinnedPosts() -> [ImagePost]`: This is the query that will give us the new functionality that our boss wants. This query will look through our binned posts in Redis and return the top N posts in terms of popularity. Look at the **Z family of functions in Redis (https://redis.io/commands/zrevrange)** to see how to perform a query to get the top n most popular posts

Frontend

To make the life of your QA (Quality Assurance) engineers easier, instead of implementing this new feature as a “Sort by…” UI element, you are asked to implement this feature as a new page at the `/popularity` route.

- `/popularity`
  - Should display the user’s top 10 most popular image from his/her bin.
  - Should have the same features as in `/` and `/my-bin`
  - At the top of the page, it should say if the user is a different category if the total of the top 10 image posts is:
    - <200: Non-mainstream
    - >200: Mainstream