SimSpace Front-End Challenge

Setup

Use `create-react-app` to instantiate a new React + TypeScript project:

```
$ yarn create react-app <projectName> --template typescript
```

(where `<projectName>` = `<firstName>_<lastName>_simspace_challenge_submission`)

Overview

Create a single-page application that allows a user to:

- Search for a specific breed of dog.
  - If the search field is empty, display a list of the first 12 breeds available, sorted alphabetically.
  - If the search field does not match any breeds, display a message that no breeds were found.
  - While the list of breeds is loading, display a message that the breeds are loading.
- Select one of the matching results and view pictures of that breed.
  - When a result is selected, mark the selection visually as being active.
  - While the images of the selected breed are loading, display a message that the images are loading.

See the attached screenshots for the UI design to follow.

Use the Dog API as your data source.

Problem

Step 1: Implement the UI

Display these elements arranged as in the screenshot below:

- A title in the top left reading “Dogs!”
- A search input in the top right with the placeholder text “Search”

After the initial UI renders:

- Make a call to `https://dog.ceo/api/breeds/list/all`
- Display some sort of loading indicator or message to give feedback to the user

After the call to `https://dog.ceo/api/breeds/list/all` completes:

- Remove the loading indicator/message
- Display the returned breed names in a grid of 3 rows of 4 buttons

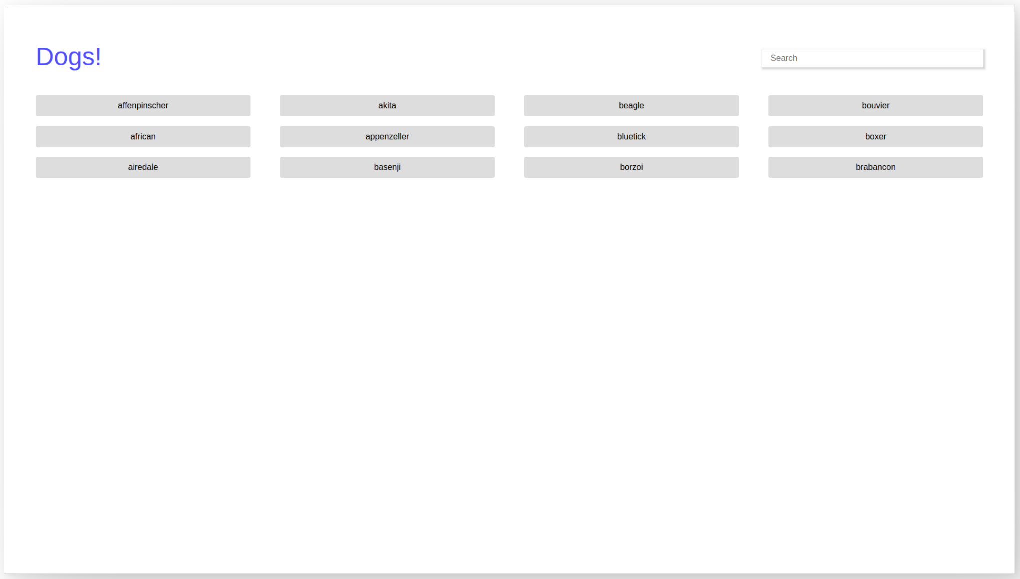

Figure 1: Display the returned breed names in a grid of 3 rows of 4 buttons

Step 2: Implement breed selection functionality

When a breed's button is clicked:

- Visually mark it as active
- Make a call to `https://dog.ceo/api/breed/{breed_name}/images`
- Display some sort of loading indicator or message to give feedback to the user

After the call to `https://dog.ceo/api/breed/{breed_name}/images` completes:

- Remove the loading indicator/message
- Display the returned images in a grid below the breed selection buttons

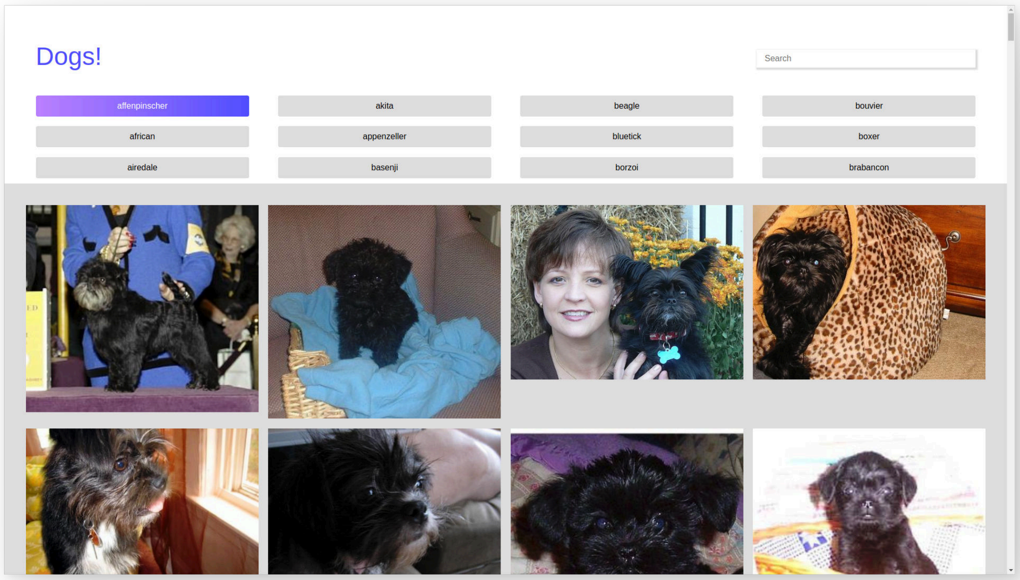

Figure 2: Display the returned images in a grid below the breed selection buttons

Step 3: Implement the search filter functionality

When a user inputs a search filter term:

If *12 or more breeds* match:

- Display the first 12 breeds (sorted alphabetically) whose name includes the search term

If *less than 12 breeds* match:

- Display all matching breeds (sorted alphabetically) whose name includes the search term

If *no breeds* match:

- Display a message reading “No breed matches found.”

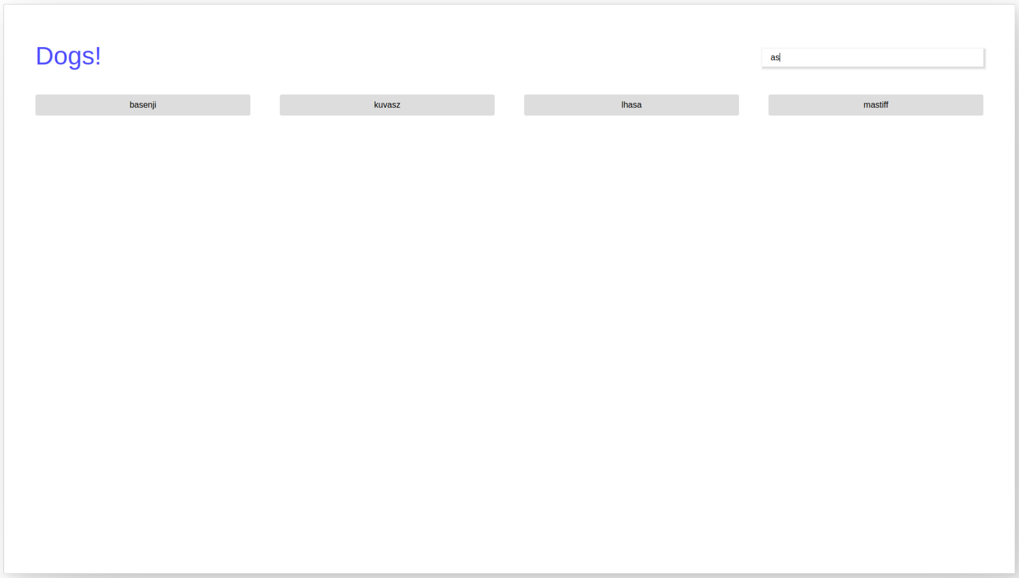

Figure 3: Display all matching breeds (sorted alphabetically) whose name includes the search term.

What we’re looking for

- **The ability to faithfully implement the provided design:** We have dedicated designers & UX engineers, but expect FE engineers to also be competent with CSS & semantic HTML
- **Good UX:** We strive to build polished, intuitive, and easy-to-use user interfaces
- **A solid understanding & effective usage of TypeScript:** We believe static types play an important role in understanding systems & writing safe, maintainable logic
- **A clean, thoughtful, & defensive approach to programming:** We believe employing a declarative & functional programming style leads to fewer bugs, easier refactoring, and, overall, more robust & predictable applications