

CSCC01 React Lab

University of Toronto
Summer 2025

Logistics:

- This lab is worth 2.5% of your grade.
- Attendance will be taken during the tutorial session and a 10% penalty will be applied for being absent with no valid excuse.
- The lab should be done **individually**.
- The due date is **Monday, May 26th @ 11:59PM**.

Learning Goals:

- Get familiar with the fundamental features of React
 - Components
 - Hooks
 - useState
 - useEffect
- Get experience reading through documentation for external libraries.

Introduction

For this lab, you will be creating a website that allows users to keep track of their favourite dog images! The site will interact with a public API which returns images of different dog breeds (the documentation can be found [here](#)).

Throughout the process of completing this lab, you will be asked short answer questions to help deepen your understanding of the material. Do not forget to submit those questions as part of the lab.

Prerequisites

The following must be installed prior to starting the lab:

1. [Node.js](#)
2. An IDE of your choice.

Features

We will support the following features in our website:

1. **Search**
 - a. Users will be able to search for images using a search bar with autocomplete capabilities.
 - b. On this page, users will also be able to favourite certain images to keep for later.
2. **Favourites (Optional)**
 - a. Once a user adds an image to their favourites, they will be able to view these images on a separate page and unfavourite them if they no longer like the image.

Getting Started

Firstly, clone the repository off of GitHub Classroom. Once you have accepted the assignment, your own personal repo will be generated for your submission.

In this repo you will find a basic React app built using Vite. For routing between pages, we will be using [react-router](#) (you can visit the link to learn more, but this is out of the scope for this lab).

You will find a collection of components in the **src/components/** folder. In it are some outlines for the basic components that we will use in our application.

- We will just be using the **DogCard** component, if you are interested, you can look into how the **Header** component works, but it is not required.

As it turns out, in React, we can also treat our pages themselves as components, and react-router will facilitate the navigation between each page so that the site is smooth and responsive to users. As such, our pages are located in **src/pages/**.

You'll notice that each component has its own folder with two files inside of it:

1. **component.jsx**
 - a. This is the file that exports the function of the component; in it contains the React component function and some logic.
2. **component.module.css**
 - a. This is the file that contains the styles specific to this component. Throughout the lab, you may edit this file as you wish to change the look of the components, but this is also optional.

The above two files allow for more modularity in our code, which is a plus!

To setup and start our project, perform the following after cloning the repo:

1. Run **npm install** from the root directory of the repository.
2. Run **npm run dev** from the root directory of the repository to start the development server.
 - a. Once this has started, any saves you make will be auto-populated to your website without needing to rebuild.
3. Visit the URL given in the terminal after running **npm run dev** to view the website.

Important Notes

- This lab uses [ES6 modules](#), which have a slightly different syntax for importing and exporting modules. Read the linked material to learn more.

Resources

Outside of this lab, it is a good idea to read up on the React learning guide located [here](#) for a more in-depth guide to React and its core features.

Additional Notes

- You may modify **any** part of **any** code in **any** file to complete the tasks.
- You can use **any** React features you want to complete the tasks (focus on **useState** and **useEffect**)
- Online documentation is really useful! Throughout the lab, there are relevant articles linked that contain information that will answer most commonly asked questions. Make sure to consult these resources before asking a TA.

Part 1 – Search Page

In this part, we will build the search page of our application.

To get started, open up `src/pages/Search/Search.jsx`. You should see something along the lines of:

```
export function Search() {  
  return (  
    <div>  
      This is the Search page.  
    </div>  
  );  
}
```

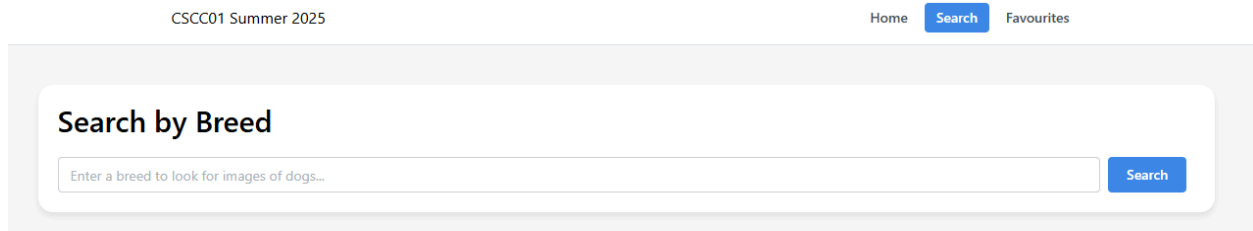
Part 1.1 – Search Bar

Let's first get started by adding a search bar to our site. We will be using a library called [Mantine](#) for some useful UI components. Specifically, let's use the [Autocomplete](#) module to do this. It is already imported for us at the top of the file, so we just need to use it like any other React component.

Replace the value inside of the parentheses of the return statement with the following:

```
<div className='page'>  
  <div className={`container container-background container-shadow`}>  
    <h1 id={styles.searchHeading}>Search by Breed</h1>  
    <div className={`row search-container`}>  
      <div className='col-11'>  
        <Autocomplete  
          placeholder="Enter a breed to look for images of dogs..."  
        />  
      </div>  
      <div className='col-1'>  
        <Button  
          variant="filled">  
          Search  
        </Button>  
      </div>  
    </div>  
  </div>  
</div>
```

After this, visit the Search page on your site. You should see the following:



The screenshot shows a web page with a header containing 'CSCC01 Summer 2025', 'Home', a 'Search' button, and 'Favourites'. Below the header is a large light gray box containing a white rounded rectangle. Inside this rectangle, the title 'Search by Breed' is at the top. Below the title is a search input field with the placeholder text 'Enter a breed to look for images of dogs...'. To the right of the input field is a blue 'Search' button.

Start typing in the search bar. You'll notice that there is no auto-complete! Well of course, we need to provide a list of values to the **Autocomplete** component to choose from.

Notice that we have abstracted our API logic into its own service located at **src/services/api.js**. Currently, this file exports 2 functions:

1. **getAllBreeds()**: Returns a list of all breeds and sub-breeds of dogs from the Dog API.
 - a. Inspect the [structure](#) of the data returned; how are the breeds represented?
2. **getDogsByBreed(breed)**: Returns a list of image links of dogs of the breed **breed** from the Dog API.

Both of these functions only return some mocked data currently; later we will implement actually calling the API.

Task: Implement the autocomplete functionality using the Mantine [Autocomplete](#) component and the **getAllBreeds()** and **getDogsByBreed()** functions in **src/services/api.js**.

- For now, you do not need to edit the API service's functions, we will do this in a later part of the lab.

Part 1.2 – Displaying Search Results

Now that our auto-complete is done, let's work on our search functionality. Let's first start off by building a component to display an individual search result. Open up `src/components/DogCard/DogCard.jsx`, and you should see the following:

```
export function DogCard() {
  return (
    <div>
      This is a dog card.
    </div>
  );
}
```

We will be using the Mantine [Card](#) component this time. Copy the following code to replace the return statement inside the DogCard component:

```
<Card shadow="sm" padding="lg" radius="md" w={250} h={300} withBorder>
  <Card.Section>
    <Image src={src} alt={`Dog image`} fit='cover' height={200}/>
  </Card.Section>
  <Button color="blue" fullWidth mt="md" radius="md">
    Favourite
  </Button>
</Card>
```

You'll notice that we are passing in a variable called **src** into the **Image** component, but where does this come from? As it turns out, we want whoever's using this component to pass this value into the **DogCard** component as a prop too! To facilitate this, change the function declaration for **DogCard** to the following:

```
export function DogCard({ src }) {
  ...
}
```

This tells React to expect a prop called **src** when the **DogCard** component is used.

You can pass all sorts of things as props into React components, even **functions**, which will be useful later. 😊

Task: When the user clicks the search button with their query, get the search results from the `getDogsByBreed()` function in the API service and display them on the **Search page**.

- You may update any part of the code in any file in order to achieve this.


Hint: Make use of the `useState` and `useEffect` React hooks! They will be helpful.

Task: When the user clicks the *Favourite* button, it should add that image to the user's favourites, and this list should persist between searches.


Here's how your Search page might look after completing the above task.

Search by Breed


Search




Unfavourite




Unfavourite




Unfavourite




Unfavourite




Favourite



Favourite



Favourite



Favourite

Part 2 – API Service

Now, we're going to work on implementing the two functions in our API service to actually call the API.

Open up **src/services/api.js**. This file makes use of a JavaScript technique called [closures](#), which are useful for encapsulation and modularization. In this case, we simulate a private class in JS using closures.

If you would like to learn more about closures, you may read the linked material, but for the sake of the lab, you only need to implement two functions:

1. **module.getAllBreeds()**: Return a list of all breeds given by the Dog API.
2. **module.getDogsByBreed(breed)**: Return a list of image links for a breed using the Dog API.

Currently, both functions return the mocked data.

Task: Using the [fetch](#) API, call the Dog API and return the results in the **same format** as the mocked data.

- You'll notice that at the top of the file, there are some useful variables that have been defined for your use.

Note: If the API requests are coming back with **500** errors, then try to call the endpoints using a manual tool (e.g., Postman).

- If this still doesn't work, you may implement this function to the best of your ability and leave a note in your submission letting us know that the API is down.

OPTIONAL Part 3 – Favourites Page

At this point in time, your website should be mostly functional. We can search for images using the Dog API and mark them as favourites (kind of). There are still some issues with our site:

1. We don't track our favourites list, so whenever we see the same image twice in our search, it doesn't know that we have marked it as a favourite previously.
2. Our favourites page still hasn't been implemented.

This section is completely optional and is for your own learning.

Task: Using your knowledge of React, implement features to fix the above two issues.

Although this section is optional, you will gain a lot of learning by going through these tasks and solving them, and it may help on your final project.

Part 4 – Short Answer Questions

Answer the following short answer questions about React (you may use the React documentation and other non-LLM resources to answer these questions):

1. In our **handleSearch** function, we pass in a **key** prop into **DogCard**. What does this do and why is it needed?
2. What does the second argument do in the **useEffect** React hook (what are the possible values, and how do they change the behaviour of the hook)?

Deliverables

Ensure that the last commit in your repository happens **before the due date** and that you have also committed a file titled **answers.txt** in the root directory of the repository with the answers for the questions in **Part 4**.

Grading

- Part 1 – Search Page (4 marks)
- Part 2 – API Service (3 marks)
- Part 4 – Short Answer (3 marks)

Total: 10 marks.