Contacts Web App

Overview

The goal of this project is to build a contact viewer that displays a list of contact cards with minimal details (Name & Avatar). The viewer should be able to view more detailed information about each contact and edit or delete a contact. The project should be simple in terms of functionality and design.

Functionality

- Display a list of contact cards with the name and avatar of each contact.
- On clicking a contact card, display more detailed information about the contact, including createdAt, email, phone, and birthday, by expanding the contact component vertically. You do not have to animate this expansion.
- Provide an option to edit a contact by sending a PUT request to the API.
- Provide an option to delete a contact by sending a DELETE request to the API.
- Implement a client-side search feature that allows users to search for a contact by name.
    - No need to worry about searching across all existing contacts in the back end; what has been fetched so far on the front end is enough for the scope of this implementation.
    - Analyse the time complexity of the search operation using big O notation and explain the reasoning behind the complexity calculated.
    - Discuss the choice of data structure used to store the contacts and how it affects the time complexity of the search operation.

Layout

Demonstrate advanced knowledge of CSS and Flexbox by implementing a responsive masonry layout for the contact cards. While there are libraries that offer masonry layouts, the purpose here is to gauge the candidate's ability to implement it from scratch. Proper structuring of CSS and semantic naming will also be considered during the evaluation.

Requirements:

1. **Basic Layout**: On smaller screen sizes (up to 600px), the contact cards should be displayed in a single column.
2. **Tablet Layout**: On medium screen sizes (601px to 900px), the cards should display in two columns.
3. **Desktop Layout**: On larger screen sizes (901px and above), the cards should display in three columns.
4. **Masonry Effect**: Within each column, the contact cards should be displayed in such a way that they utilize the vertical space optimally, similar to a vertical masonry layout. This means that contact cards should always have the same amount of spacing between them despite their differing heights, and these cards should all have the same width.
5. **Gap Handling**: There should be a consistent gap (margin) between individual contact cards both horizontally and vertically. Ensure that there are no unnecessary gaps at the beginning or end of each column.
6. **Flexbox**: The layout should primarily be achieved using CSS flexbox. Avoid using CSS Grid or other layout methods for this challenge.

API

- The project will use the below API to retrieve, create, edit, and delete contacts where `baseUrl` is `https://61c32f169cfb8f0017a3e9f4.mockapi.io/`:
  - GET - `${baseUrl}/api/v1/contacts` - returns a list of contacts
  - POST - `${baseUrl}/api/v1/contacts` - creates a contact
  - PUT - `${baseUrl}/api/v1/contacts/:id` - edits a contact
  - DELETE - `${baseUrl}/api/v1/contacts/:id` - deletes a contact

Data

- The API returns an array of contact objects in JSON format, with each object having the following properties:
  - createdAt: date and time of creation (YYYY-MM-DDTHH:MM:SS.SSSZ)
  - name: name of the contact
  - avatar: URL of the contact's avatar
  - email: email of the contact
  - phone: phone number of the contact
  - birthday: birthday of the contact (YYYY-MM-DDTHH:MM:SS.SSSZ)
  - id: the unique identifier of the contact

Technology Stack

- React
- TypeScript
- The rest is up to you. Keep in mind we care about proper and smart usage of hooks.

Testing

- The project should have tests to verify the functionality and ensure quality.
- At the very least, you should have unit tests and end-to-end tests. At Equals, we use Jest and Cypress, but feel free to pick what you’re comfortable with.