README.md

Bounteous Headless CMS Frontend Engineer Coding Challenge

Hello and thank you for your interest in applying for this position! This skill assessment will test various Headless CMS Frontend Engineer abilities and is intended to take 1-2 hours, but feel free to take the time you need or want to invest in impressing us with your solution. At the end of the assessment, you should have a simple SPA site that loads the allowed routes dynamically, renders some sample headless content, and then performs some basic styling updates on those elements. There are a wide variety of skills being tested in this assessment, so even if one or two elements are giving you trouble, you can take what steps are necessary to work around that in order to submit what you were able to accomplish.

Your end result should be a simple two-page site that performs API calls to determine the available routes and also loads the content. The content is rendered following rules laid out in the `Components` section below.

Using the framework of your choice (e.g. React, Vue, Angular) complete the following instructions. To submit, please provide a zip file containing the source code for your solution. If you used git to track your changes, you can optionally include the `.git` folder so we can review your development process if desired.

Instructions

1. Set up a one-way data flow state management library such as Redux or Flux.
2. Make an API request to https://raw.githubusercontent.com/Bounteous-Inc/headless-cms-assessment/main/routes.json to retrieve the valid routes your application should support, and store this value in the data store set up in step 1. If this API request fails, your application should default to directing all traffic to `/maintenance`. NOTE: when testing your submission, the URL used to request the routes will be changed to validate the provided solution is dynamic.
3. Any requests outside the routes provided in the API response should display the first route provided.
4. Create a header navigation which includes each of the URLs from the routes response using their value from the map as the name. This header should appear on every page.
5. When loading a particular route, the content for the route should be loaded from a JSON file located relative to the route. For example if the provided route is `/about-us` then the relevant JSON file will be `/about-us.json` or if the route is `/products/1bn23` the relevant JSON file is `/products/1bn23.json`. These paths are relative to the domain/path provided in step 2.
6. The JSON content files will utilize various components which will need to be rendered by your application code. Examples of these components and a description of various options for them are provided below.
7. Add some outline, drop shadow, and a hover effect to each of the "cards" on the bios page.
8. Add some styling around each of the books on the books page, including some alignment changes for the text and title of the books.
9. Below 600px of width, the cards on the bios page should be displayed 2 per row as opposed to their default 3 per row.

Components

Rich Text component

- type: "rich-text"
- textType: "plain" | "html"
- className: this value is sometimes provided to help with specific styling needs as needed, to target specific elements
- text: this field will contain either plain text or escaped HTML, according to the `textType` property, and will need to be rendered as

text or as HTML accordingly. If it is rendered as plain text, it should be rendered within a paragraph tag.

- style: "uppercase", etc. - If this is set, then the value should be passed through to the `text-transform` CSS property on the container.

Container component

This component should be rendered as a flex container (`display: flex`), and will default to the same default values as a flex.

- type: "container"
- items: [] - this will be an array of other components, which could include another container component.
- className: this value is sometimes provided to help with specific styling needs as needed, to target specific elements
- flexDirection: value provided should be passed to the `flex-direction` css property on the container
- flexWrap: value provided should be passed to the `flex-wrap` css property on the container
- justifyContent: value provided should be passed to the `justify-content` css property on the container
- alignItems: value provided should be passed to the `align-items` css property on the container
- width: relative width to parent container (e.g. 0.3 would be 30% the width of its parent)

Image component

If the image cannot be loaded, then a 100x100 placeholder from placeholder.com should be utilized. https://placeholder.com/#How_To_Use_Our_Placeholders Images should set `align-self: center;` to avoid scaling in flex containers.

- type: "image"
- className: this value is sometimes provided to help with specific styling needs as needed, to target specific elements
- alt: "string" - alt text
- height: if provided, it should be set as the `max-height` for the image element
- src: string value which provides the URL that should be displayed as an image