UDACITY Would You Rather?

REVIEW

Meets Specifications

Great job, you made it, congrats 🎉

I love the UI design, the brilliant idea of signing up, the navigation/routing experience, the folder structure setup, everything

Do you want to make it even better later after graduation?
Consider implementing a real backend API service so all the votes and new questions will be able to persist.
you can also add social sign on/ registration, manage user profile, browser/email notification for followers' new questions and so on.

I'm sure you can come up with better ones and implement them all 💪

Application Setup

The application requires only `npm install` and `npm start` to install and launch.

A README is included with the project. The README includes a description and clear instructions for installing and launching the project.

Login Flow

1. There should be a way for the user to impersonate/ log in as an existing user. (This could be as simple as having a login box that appears at the root of the application. The user could then select a name from the list of existing users.)
2. The application works correctly regardless of which user is selected.
3. The application allows the user to log out and log back in. The user should be logged in to submit new polling questions, vote, and view the leaderboard.
4. Once the user logs in, the home page is shown.

5. Whenever the user types something in the address bar, the user is asked to log in before the requested page is shown.

Application Functionality

1. The answered and unanswered polls are both available at the root.
2. The user can alternate between viewing answered and unanswered polls.
3. The unanswered questions are shown by default.
4. The name of the logged in user is visible on the page.
5. The user can navigate to the leaderboard.
6. The user can navigate to the form that allows the user to create a new poll.

Perfect 🙌🏼

Each polling question resides in the correct category. For example, if a question hasn't been answered by the current user, it should be in the "Unanswered" category.

A polling question links to details of that poll.

The polls in both categories are arranged from the most recently created (top) to the least recently created (bottom).

1. The details of the poll are available at `questions/:question_id`.
2. When a poll is clicked on the home page, the following is shown:
    - the text "Would You Rather";
    - the picture of the user who posted the polling question; and
    - the two options.
3. For answered polls, each of the two options contains the following:
    - the text of the option;
    - the number of people who voted for that option;
    - the percentage of people who voted for that option.
4. The option selected by the logged in user should be clearly marked.
5. When the user is logged in, the details of the poll are shown. If the user is logged out, he/she is asked to log in before before being able to access the poll.
6. The application asks the user to sign in and shows a 404 page if that poll does not exist. (In other words, if a user creates a poll and then the same or another user tries to access that poll by its url, the user should be asked to sign in and then be shown a 404 page. Please keep in mind that new polls will not be accessible at their url because of the way the backend is set up in this application.)

Minor, let's create a 404 page instead of redirecting to the Login page

1. Upon voting in a poll, all of the information of the answered poll is displayed.
2. The user's response is recorded and is clearly visible on the poll details page.
3. When the user comes back to the home page, the polling question appears in the "Answered" column.
4. The voting mechanism works correctly, and the data on the leaderboard changes appropriately.

1. The form is available at `/add`.
2. The application shows the text "Would You Rather" and has a form for creating two options.
3. Upon submitting the form, a new poll is created and the user is taken to the home page.
4. The new polling question appears in the correct category on the home page.

1. The Leaderboard is available at `/leaderboard`.
2. Each entry on the leaderboard contains the following:
    - the user's name;
    - the user's picture;
    - the number of questions the user asked; and
    - the number of questions the user answered.
3. Users are ordered in descending order based on the sum of the number of questions they've answered and the number of questions they've asked.

The app contains a navigation bar that is visible on all of the pages.

The user can navigate between the page for creating new polls, and the leaderboard page, and the home page without typing the address into the address bar.

The data that's initially displayed is populated correctly from the backend.

Each user's answer and each new poll is correctly recorded on the backend.

The code runs without errors. There are no warnings that resulted from not following the best practices listed in the documentation, such as using `key` for list items. All code is functional and formatted properly.

Architecture

The store is the application’s source of truth.

Components read the necessary state from the store; they do not have their own versions of the same state.

There are no direct API calls in the components' lifecycle methods.

Most application state is managed by the Redux store. State-based props are mapped from the store rather than stored as component state.

Form inputs and controlled components may have some state handled by the component.

Updates are triggered by dispatching action creators to reducers.

Reducers and actions are written properly and correctly return updated state to the store.

The code is structured and organized in a logical way.

Components are modular and reusable.

Rate this review