UDACITY

‹ Return to Classroom

DISCUSS ON STUDENT HUB

Would You Rather?

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

You've done a fantastic job completing the Would You Rather project 👏

You might want to check out the resources below for some inspiration and to further improve your model and solidify your understanding of the material:

- UI vs UX https://webflow.com/blog/ux-vs-ui-design
- CSS Tricks https://css-tricks.com/
- Smashing Magazine CSS guide https://www.smashingmagazine.com/guides/css-layout/
- CSS Zero To Hero https://dev.to/aspittel/css-from-zero-to-hero-3o16
- https://overreacted.io/
- https://reacttraining.com/blog/
- Understand react internals https://reactjs.org/docs/reconciliation.html

Keep up the great work 👍🏽

Congratulations once again 🎉 Happy Learning and have a great time with your further projects 😊

Leave a rating if the review was helpful 😇

Application Setup

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

The submission starts in a browser upon running `npm install` and `npm start` 👍🏼

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

Writing a `readme` file is the easiest way to explain our projects code structure and how to run the project. This is the most important file in any open source project 👏

Further Reading

- Awesome Readme https://github.com/matiassingers/awesome-readme

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.

Designing a good login screen takes a lot of effort. I see that you have provided a simple dropdown with users listed upon selection the user is logged in to the application

Excellent resources for inspiration

- Login Screen flows with code by Chris Coyier https://mediatemple.net/blog/web-development-tech/interesting-takes-log-sign-forms/

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.

You have done an amazing job implementing all the above required items from the rubric. I like the card philosophy you went with for designing this experience.

While this is good we can clearly see that for a new user the content inside cards is quite confusing, we can read some design articles here and can easily improve our layout for better clarity.

Further Reading

- Una Kravets posts have great depth to consume and get inspired from https://css-tricks.com/material-theming-making-material-your-own/

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.

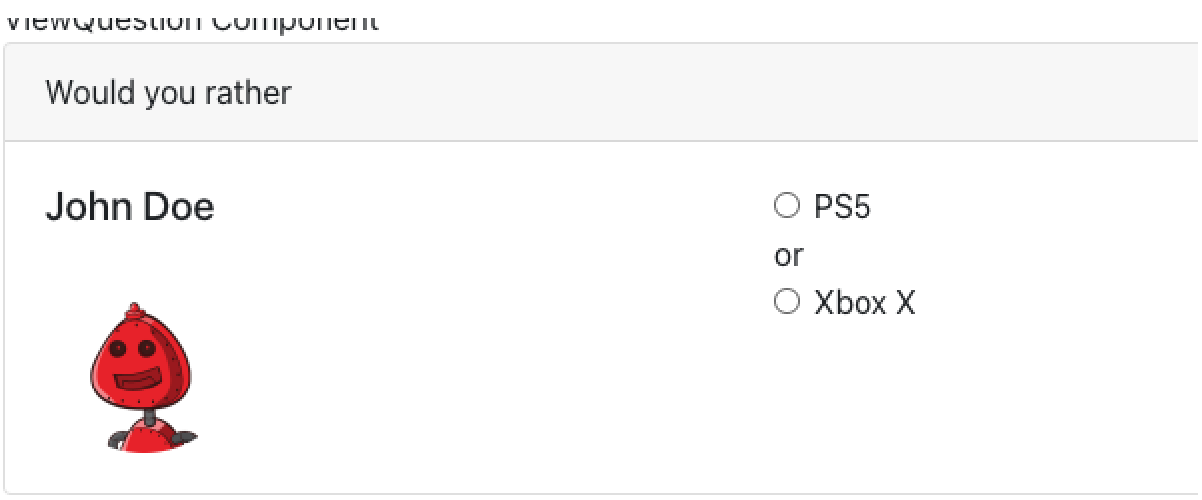
The polls in both categories are arranged from the most recently created (top) to the least recently created (bottom).

Separation of polls into their own categories i.e `answered/ Unanswered` works perfectly. Adding a new poll immediately shows up at the top of unanswered tab. Excellent work on implementing this functionality where we sort by timestamps.

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.)

The details of the poll is available at the route `question/:id` and it clearly updates the content based on answered/ unanswered poll

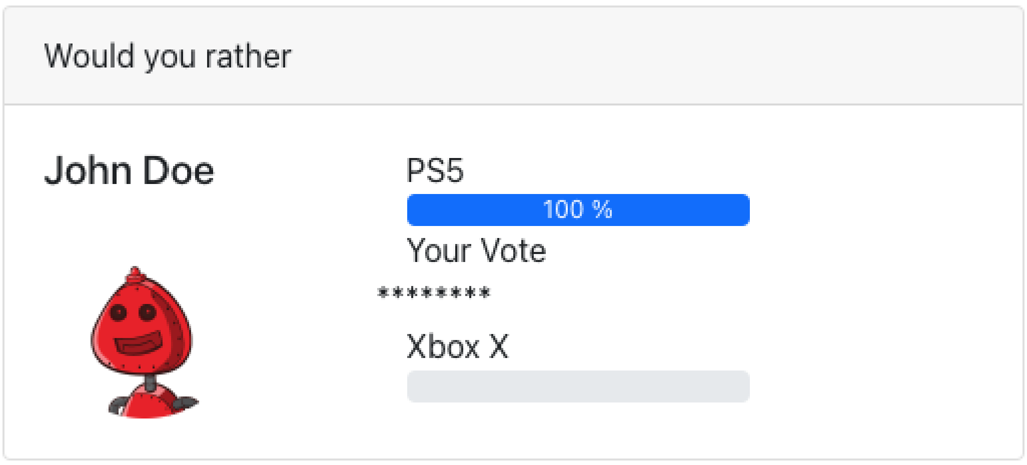
Unanswered poll route shows the options to select ✅


ViewQuestion Component

Would you rather

John Doe

Answered poll shows the results ✅


Would you rather

John Doe

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.

The voting mechanism works perfectly and redirection to home page works perfectly 👏

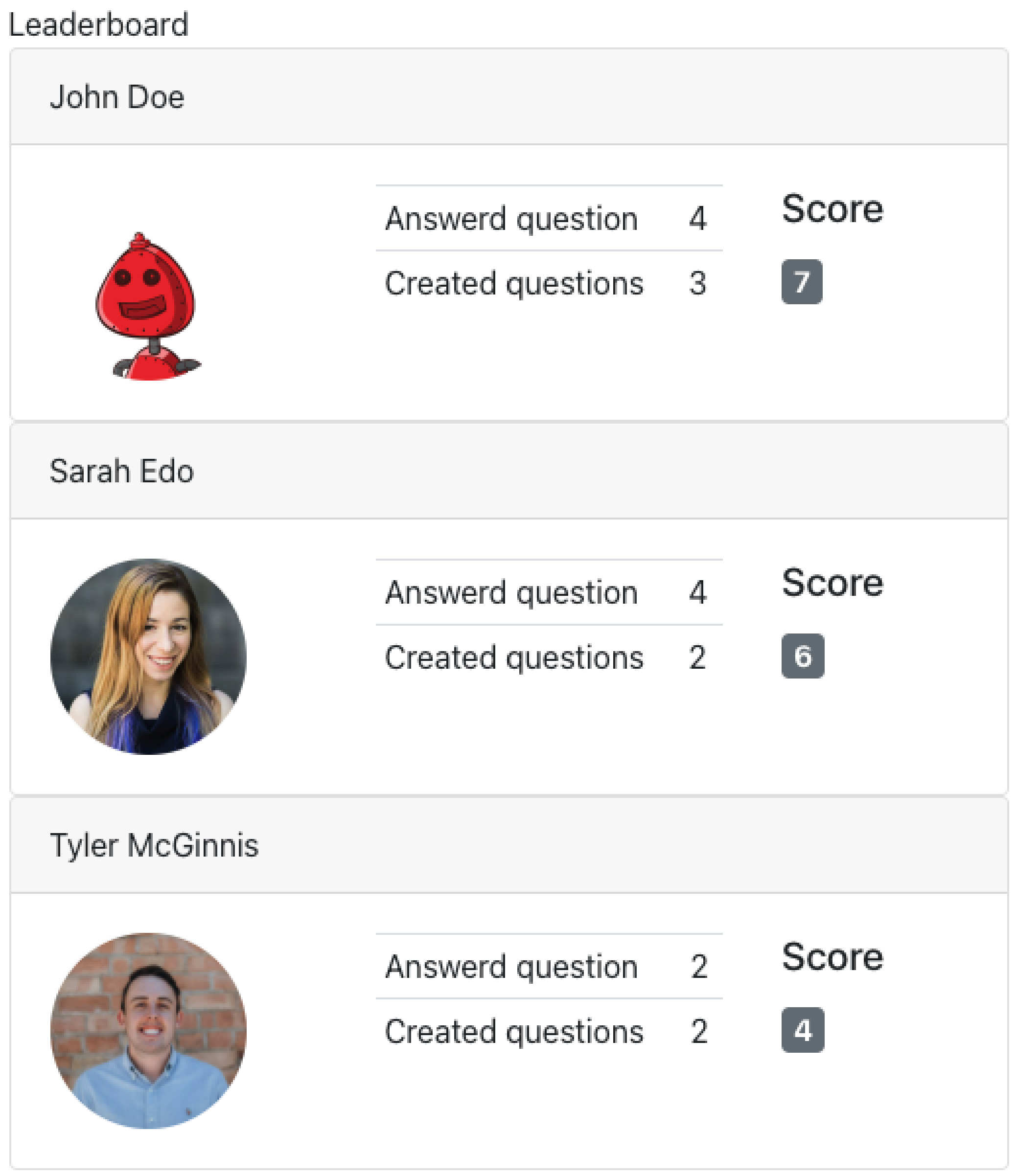
1. The form is available at `/add`.
2. The application shows the text "Would You Rather" and has a form for creating two options.
3. Upon submitting the form, a new poll is created and the user is taken to the home page.
4. The new polling question appears in the correct category on the home page.

The form at `/add` works as required.

- Able to enter 2 options ✅
- Options are saved along with the question ✅
- The poll is added to the top of unanswered polls tab ✅

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 leaderboard satisfies all the requirements for this rubric.



Leaderboard

John Doe

Answerd question	4	Score
Created questions	3	7

Sarah Edo

Answerd question	4	Score
Created questions	2	6

Tyler McGinnis

Answerd question	2	Score
Created questions	2	4

Tips

- Dribble has some amazing UI screens https://dribbble.com/tags/leaderboard

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 navigation bar is visible on all pages ✅

Inspiration for menu links styling

- https://tympanus.net/Development/DistortedLinkEffects/

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.

All the required data is correctly captured for every action on the site ✅

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.

You have done an amazing job making sure the errors are minimal 👏

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.

Yes you made sure the store is the source of truth in your submission.

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.

Yes, Redux handles most of your state

However a common question we might have when working with redux is when to use it vs local state. Understanding this is very important for a react developer

Tips

- https://redux.js.org/faq/organizing-state
- https://blog.logrocket.com/why-use-redux-reasons-with-clear-examples-d21bffd5835/

Updates are triggered by dispatching action creators to reducers.

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

The updates are happening via action creators ✅

The code is structured and organized in a logical way.

Components are modular and reusable.

Simple and Clearly organised code 👍🏽

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review

START