Project Overview

In the "Would You Rather?" Project, you'll build a web app that lets a user play the “Would You Rather?” game. The game goes like this: A user is asked a question in the form: “Would you rather [*option A*] or [*option B*] ?”. Answering "neither" or "both" is against the rules.

In your app, users will be able to answer questions, see which questions they haven’t answered, see how other people have voted, post questions, and see the ranking of users on the leaderboard.

Why this project?

This project will solidify your understanding of Angular and NgRx while giving you a chance to express your creativity. You’ll practice improving the predictability of your application’s state; establish strict rules for getting, listening, and updating the store; and identify what state should live inside of Store and what state should live inside of Angular components.

Starter Code

The starter code will consist of a `_DATA.js` file, which represents a fake database and contains methods that let you access the data. The `README` file outlines how the data is stored and details the methods you can use to access the database.
The only thing you need to edit in the `_DATA.js` file is the value of `avatarURL`. Each user should have an avatar, so you’ll need to add a path to each user’s avatar.
Using the provided starter code, you will build an Angular/NgRx front end for the application.

App Functionality

The person using your application should have a way of impersonating/logging in as an existing user. (This could be as simple as having a login box that appears at the root of the application that lets the user select a name from the list of existing users. Alternatively, you could create your own account creation process to allow a

user to sign up for an account.) Your application should work correctly regardless of which user is selected. Once the user logs in, the home page should be shown.

We always want to make sure we know who the logged in user is, so information about the logged in user should appear on the page. If someone tries to navigate anywhere by entering the address in the address bar, the user is asked to sign in and then the requested page is shown. The application allows the user to log out and log back in.

Once the user logs in, the user should be able to toggle between his/her answered and unanswered polls on the home page, which is located at the root. The polls in both categories are arranged from the most recently created (top) to the least recently created (bottom). The unanswered questions should be shown by default, and the name of the logged in user should be visible on the page.

What would be the point of seeing answered and unanswered polling questions if we couldn't actually vote or see the results? Each polling question should link to the details of that poll. The details of each poll should be available at `questions/:question_id`.
When a poll is clicked on the home page, the following is shown:

1. Text "Would You Rather";
2. Avatar of the user who posted the polling question; and
3. Two options.

For answered polls, each of the two options contains the following:

1. Text of the option;
2. Number of people who voted for that option; and
3. Percentage of people who voted for that option.

The option selected by the logged-in user should be clearly marked.

Since we want to make sure our application creates a good user experience, the application should show a 404 page if the user is trying to access a poll that does not exist. (Please keep in mind that newly created polls will not be accessible at their url because of the way the backend is set up in this application.) It should also display a navigation bar so that the user can easily navigate anywhere in the application.

So what happens when someone votes in a poll? Upon voting in a poll, all of the information of an answered poll should be displayed. The user's response should be recorded and clearly visible on the poll details page. Users can only vote once per poll; they shouldn't be allowed to change their answer after they've voted -- no

cheating allowed! When the user comes back to the home page, the polling question should appear in the “Answered” column.

It would be no fun to vote in polls if we couldn’t post our own questions! The form for posting new polling questions should be available at the `/add` route. The application should show the text “Would You Rather” and have a form for creating two options. Upon submitting the form, a new poll should be created, the user should be taken to the home page, and the new polling question should appear in the correct category on the home page.
But how can we know how many questions each user has asked and answered? Let’s get some healthy competition going here! The application should have a leaderboard that’s available at the `/leaderboard` route. Each entry on the leaderboard should contain the following:

1. User’s name;
2. User’s picture;
3. Number of questions the user asked; and
4. Number of questions the user answered

Users should be ordered in descending order based on the sum of the number of questions they’ve asked and the number of questions they’ve answered. The more questions you ask and answer, the higher up you move.

The user should be able to navigate to the leaderboard, to a specific question, and to the form that allows the user to create a new poll *both* from within the app and by typing in the address into the address bar. To make sure we’re showing the data that is relevant to *the user*, the application should require the user to be signed in order to access those pages.

Project Instructions

1) Use Angular to build your application’s UI. Remember that composition is key. It’s rarely a mistake to break a component into smaller pieces. Look for opportunities to reuse your components.

3) Remember that planning your project and its architecture before starting to code will save you a lot of debugging time later on!

4) Use NgRx to manage your application state. For this application, most of the application’s state should be managed by Store. You may use component state to handle form input fields and controlled components. Otherwise, the rest of the state for your application should be controlled by your reducers.

5) While the focus (and specification) of this project is based on functionality rather than styling, please ensure that your app is presentable and easy to navigate.

6) Please carefully test your app against the rubric to make sure *all* of the rubric requirements are met. Your project must meet all of the rubric criteria in order to pass.

Step 1

Before submitting, please verify that your project:

1. Meets specification according to the project rubric.
2. Includes all of the files necessary to install and launch your web application on a local web server. You can assume that your reviewer will have npm installed on their machine.

Step 2

1. Push your project to GitHub, making sure to push the master branch.