



# Code Foo X

## How it's made

### Summary

I built and deployed a [web application](#) using React and Firebase that consists of two main components:

- 1) A polling application allows visitors to create polls and cast votes. Only administrators are authorized to delete polls.
- 2) A quest calculator that takes a PDF URL to load in data and calculate the most lucrative quest sequence.

During this project I used functional [React](#), [React Router](#), [D3.js](#), [PDF.js](#), and [graphlib](#) to program and deploy a functional web application.

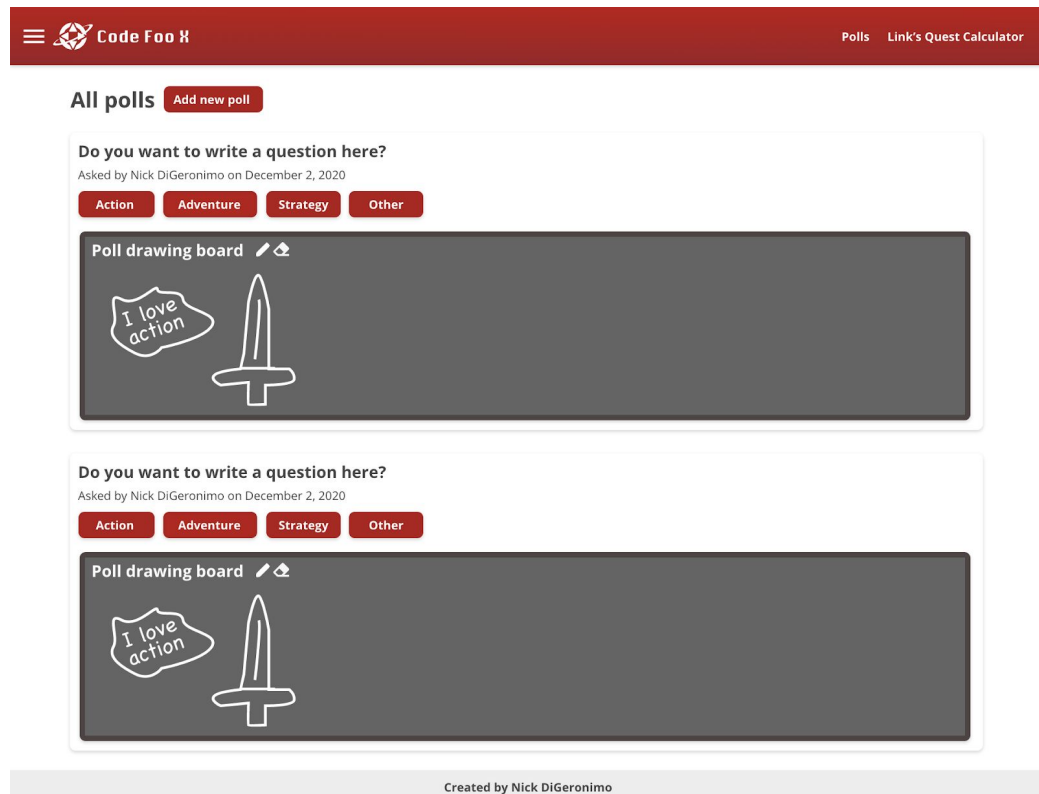
### Design

#### Visual Design

- 1.) I started by drawing a UX prototype using Adobe XD and updated the prototype over time as the design changed. The prototype can be viewed here:  
<https://xd.adobe.com/view/a7c1f5a7-9191-4579-5f82-27a8dc824fcc-d1f0/>.
- 2.) I made the logo using Adobe Illustrator.

For reference I've left older/incomplete designs I was experimenting with. Notably, I decided to leave out a drawing board function which would have allowed users to draw on polls.

Figure 1



*The drawing board functionality (see design above) was intended to allow users to be able to draw on polls. I decided to not implement it to save time, but it can definitely be done.*

## Programmatic Design

For the frontend I used React to manage the UI, React Router to split things up into pages, and Material UI to help style my React components. CSS (technically compiled CSS) was used when necessary to get things to look right. I invested an extensive amount of time to make the whole site look consistent and provide a user friendly experience.

**Figure 2**

**Create a poll** ×

Enter your poll's question

Type your question

What is your age?

Enter two or more answers to the question

Young Old Really old Ancient Other

Type your answer. Press Enter to add another.

Optionally, provide your name and email for public display:

Your name (email@example.com)

Add new poll

*I used IGN colors throughout my designs and when necessary built and themed my React components to make sure things looked nice.*

To speed up the development process I used Firebase Realtime Database specifically to provide near real-time data reading/writing and Firebase functions to host my API endpoints. Note that Firebase provides an SDK to allow read/write access directly from the (browser) client-side. When possible I use the SDK, but I wrote a couple of API endpoints because some functions require elevated privileges or it was cleaner/more reliable to use an endpoint.

## Polls page

The [polls page](#) displays existing polls to users allowing them to vote on existing polls or create new ones. Firebase handles the reading and writing data on this page, while React takes care of the UI rendering and management.

Figure 3

Code Foo X All polls

### Polls feed

Create a poll

**What is your favorite color?**  
Asked by Nick (perfectalgorithm@gmail.com)

Red Blue Yellow Purple

**What is your favorite fruit?**  
Asked by Nick (ndigeron@uci.edu)

Apple Orange Persimmon Grape

Other

*Example of the polls page with two added polls. Users can vote by tapping the option they want or create a poll on this page.*

## Link's Quest Calculator Page

The [quest calculator page](#) provides a solution to question #2 on the Code Foo X [application page](#). In addition to showing the quests Link should take to earn the maximal amount of rupees

It also provides a visualization of how the algorithm works. I've provided comments in [my Github repo](#) (in the `exports.calculateBestQuests` area) which explain how my code finds the most lucrative quest sequence. Basically it builds a directed acyclic graph using the quest data with the rupees earned as edges and the quests as vertices then finds the most expensive path in the graph. For more detail, please see my comments in the link provided above.

**Figure 4**



*I made the BestQuestGraph component using React and D3.js. It draws the same directed graph that is used to calculate which quest link should take.*

When the *Reload from source* button is pressed (by the client or by my code) the client side sends a GET request to the [API endpoint](#) where my Firebase function is running at. This initiates the process which downloads the PDF from the URL, reads the PDF using the PDF.js library, runs an algorithm to calculate the most lucrative quest sequence, and then writes it to Firebase. To prevent having to re-download, parse, and run the algorithm every time the page is loaded I use the last result saved to the database via the Firebase function running at the *calculateBestQuests* endpoint.

**Figure 5**

### Most lucrative quest sequence

Order	Quest	Earned Rupees
1	A New York state of mind	500
2	Giving donuts to strangers	1000
3	Cleaning the house	450
4	Rolling dice	300
Maximum rupees possible:		2250

*This table was made using Material UI and custom CSS I wrote.*

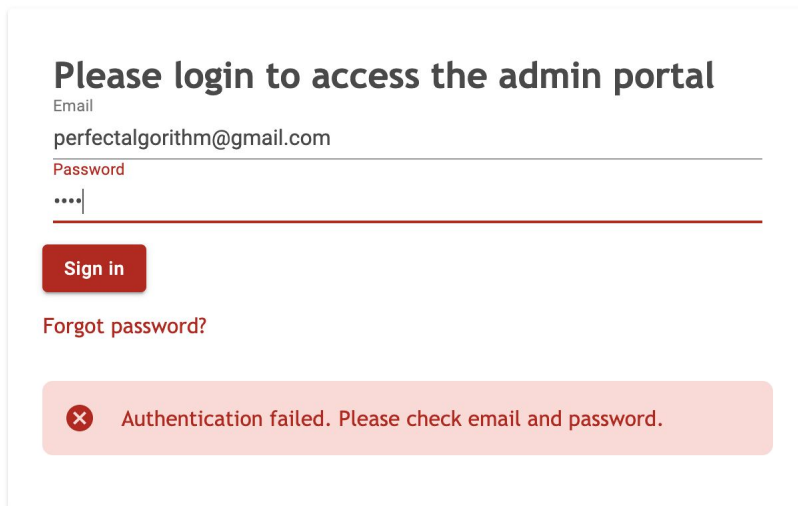
The *Modify data source* button allows the user to change which dataset is displayed on the quest calculator page. Simply change select the dataset you want to use from the dropdown in the dialog and press *Permanently change URL* and the table and graph will update to display the newly selected option. The change is permanent and the updated data will show for *all*

*future visitors* of the site. (Note after modifying the source that the *Reload from the source* button is triggered automatically, otherwise the visualizations will not be updated for the reason mentioned in the previous paragraph). I suggest switching to the sample PDF I provided if you're interested in learning how my algorithm works—it's a lot less complex than the one generated from the IGN quest PDF.

## Admin portal

The *Admin portal* page allows *owner* or *administrator* role users to add or delete users, delete existing polls, or add custom PDF files that can be read by my code. The *Add/delete users* makes use of the [second API endpoint](#) I wrote. That API endpoint accepts either POST or DELETE requests only, and will handle creating/deleting new users respectively. The reason why I use an API endpoint instead of the Firebase JavaScript SDK here is because the JS SDK doesn't support deleting users except the one currently signed in. To work around this I use the Firebase Admin Node.js SDK running on a cloud function from the endpoint above which has elevated privileges that allow creation/deletion of any user. I authenticate the user to make sure they have the appropriate permissions to prevent unauthenticated requests from adding/deleting users.

**Figure 6**



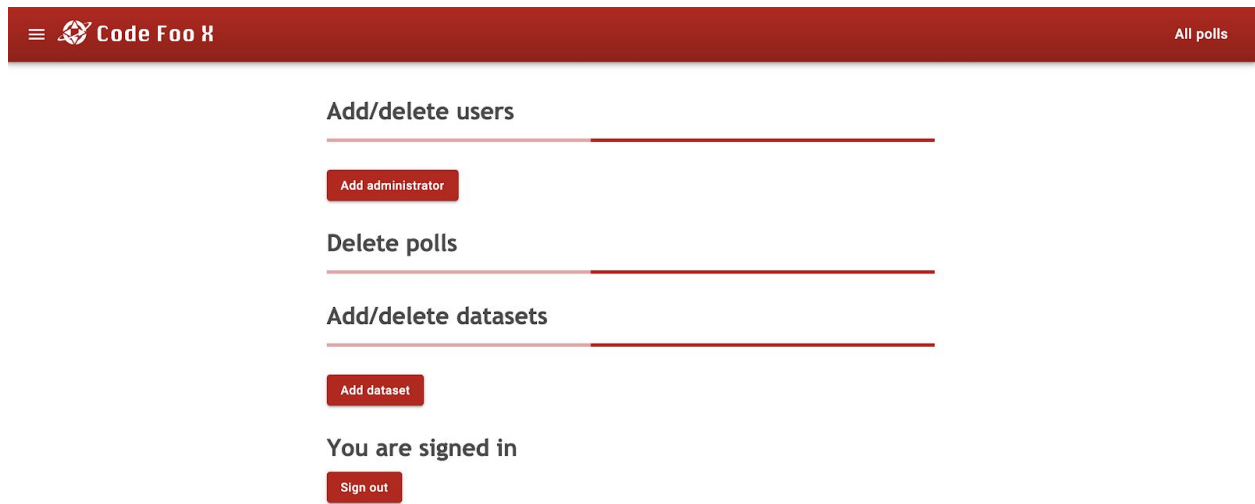
The screenshot shows a login form titled "Please login to access the admin portal". It has two input fields: "Email" with the value "perfectalgorithm@gmail.com" and "Password" with masked characters "....". Below the password field is a red "Sign in" button. A link "Forgot password?" is located below the button. At the bottom, a red error message box contains a red 'x' icon and the text "Authentication failed. Please check email and password."

*The login area for admin users features helpful error messages to increase usability and a password reset in case the user forgets their password.*

## Visibility of system status

Many screens feature loading indicators so users can be aware of what's going on in accordance with rule #1 of [Nielsen's usability heuristics](#).

**Figure 7**

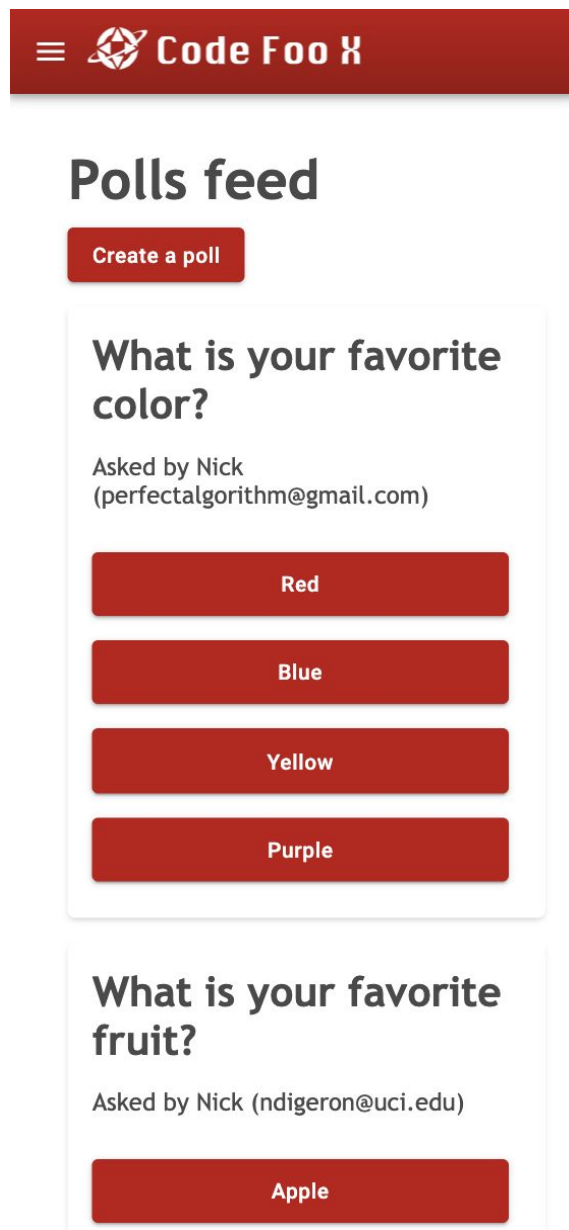


*The Admin Portal displays when content is loading so users know they need to wait.*

## Responsive design

The website is mobile-friendly and renders acceptably on different sized devices. This ensures that regardless of the devices users will still be able to use the site.

Figure 8



The screenshot shows a web application interface. At the top is a dark red header bar with a white hamburger menu icon, a white icon of a cube with an 'X' on it, and the text 'Code Foo X' in white. Below the header is the title 'Polls feed' in a large, bold, dark grey font. Under the title is a red button with the text 'Create a poll' in white. Below this is a white card with a light grey border. The card contains the title 'What is your favorite color?' in bold dark grey font, followed by the text 'Asked by Nick (perfectalgorithm@gmail.com)' in a smaller dark grey font. Below the text are four red buttons with white text: 'Red', 'Blue', 'Yellow', and 'Purple'. Below the first card is a second white card with a light grey border. It contains the title 'What is your favorite fruit?' in bold dark grey font, followed by the text 'Asked by Nick (ndigeron@uci.edu)' in a smaller dark grey font. Below the text is a single red button with white text: 'Apple'.

*The website automatically adapts to different screen sizes so users can use their preferred device.*

## Future improvements

Due to the unpaid nature of this project and the amount of time it was costing me I had to make some sacrifices and omit some features I would have liked to include. I'm writing this section to show you I was aware of them, and if I had the time would have made these improvements.



- Security
  - Quite simply this site was not designed to be secure. I employ basic server-side checking and require authentication to access data. However, because I primarily use Firebase rules to provide security there are several security flaws which I have not addressed. One is that incoming data from the client isn't strictly validated by the server. There is also a bug which would allow unauthenticated users to delete polls, but because it would require modifying client side code, the regular user would not be able to delete polls without authentication. The security flaws exist because Firebase's Realtime Database rules don't allow a lot of flexibility. There are workarounds, but it requires a lot of additional time I unfortunately need to spend that hunting for jobs. As an employee though, you can be assured I would pay careful attention to security.
- Global design
  - I used a lot of custom CSS to get things to look the way I wanted to, but I should have taken more care to set up my React components in such a way that I wouldn't have to use so many exceptions.
- Testing
  - I mainly tested the site using my family, myself, and Chrome DevTools. I would have preferred to write test cases to assure my code runs well, but to save time I avoided this.

## Lessons learned

- Avoid scope creep
  - It is good to build a sizable React web app even though the learning curve was intense. I kept adding on more and more features which made the development time even longer. I should have chosen a smaller scope—which would have made the project much more doable and as a result I could have made my code much more robust.
- Functional React problems
  - I used 100% functional React components to build my app. This means I made heavy use of React Hooks. A lot of confusion stemmed from the way React works—its asynchronous nature of state updates and how to allow my custom code to access the DOM correctly. Also, I had to find ways to get React to fetch things from my API efficiently which required heavy use of the `useEffect` hook. To accentuate my problems, when you pass a dependency to `useEffect` it retains a reference to that dependency this meant that sometimes things wouldn't work as expected because I was modifying my dependency directly and React wouldn't re-render things. To work around this I would create a copy and use the copy when setting things via `useState`.
- Asynchronous JavaScript
  - Another source of confusion was that my code was loaded with asynchronous Javascript. That meant that sometimes I had to wait for things to be ready before

I could do other things. I learned a lot about promises in JS and even did some work with async functions. Overall, I feel like I left with a better understanding of asynchronous JavaScript, but it was confusing nonetheless.

## Conclusion

If you haven't already seen my web app. Please check it out here:

<https://code-foo-x-firebase.web.app/>

To access the admin portal you will need credentials. You can use my account:

*Email:* [perfectalgorithm@gmail.com](mailto:perfectalgorithm@gmail.com)

*Password:* *nickreallywantsthisjob*

Or the account I've created for you:

*Email:* [alex\\_delapena@ziffdavis.com](mailto:alex_delapena@ziffdavis.com)

*Password:* *nickreallywantsthisjob*

If you'd like to test adding a new dataset via the admin portal I have an additional PDF at

<https://code-foo-x-firebase.web.app/assets/simp.pdf>.

The GitHub repository also publicly available if you want to look at my code:

<https://github.com/im-not-io/ign-code-foo-x-repo>

To wrap things up I'd like to thank my family and friends for helping to review and test my site. I am still actively looking for work. I spent weeks of time and probably hundreds of hours programming for the Code Foo X project. I completely understand that COVID-19 has been made hiring tough, but I do hope that when you're hiring again that you will consider me. If employed by IGN I promise to work hard to bring a positive impact to everyone that works there. Regardless of your decision I hope you have enjoyed what I made. Maybe try out a few polls with your co-workers—it could be fun!

Cordially,

Nick DiGeronimo