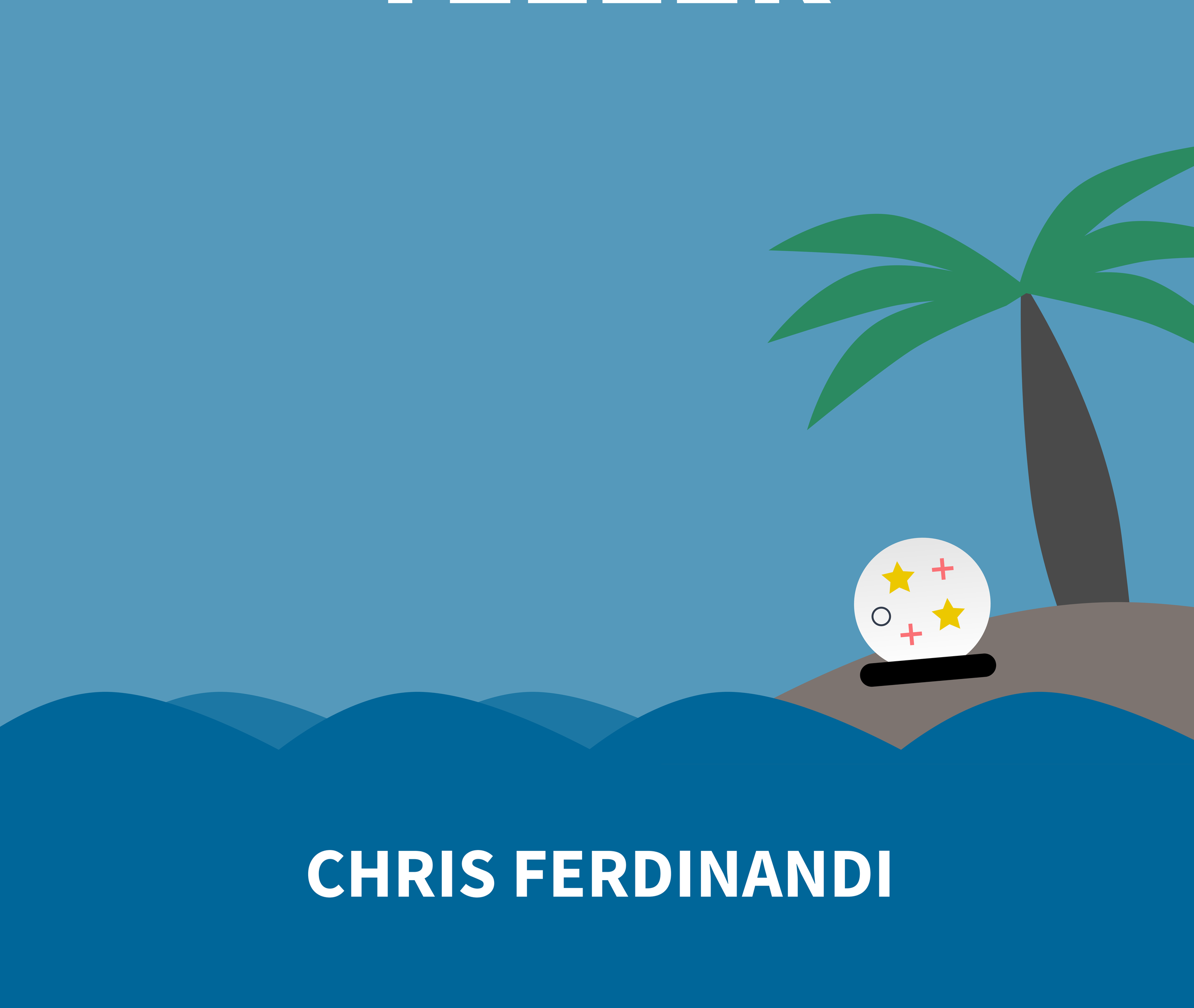
Vanilla JS Projects ⚓ Go Make Things

FORTUNE TELLER



CHRIS FERDINANDI

Fortune Teller

By Chris Ferdinandi
Go Make Things, LLC
v1.0.0

Copyright 2019 Chris Ferdinandi and Go Make Things, LLC. All Rights Reserved.

Table of Contents

How it works

There are two ways you can approach this project:

1. **Try to build it on your own.** When you're done, check out my completed project to see how you did. I provide you with a template to get started, and some recommended articles and resources to help you along the way.
2. **Build it along with me.** I'll walk you through how I would approach this project, step-by-step. I'll code in real-time, and explain what I'm doing and why.

Neither approach is better than the other, and you might find yourself jumping back and forth between the two as you go along.

Everyone's learning style is different. Pick the approach you're most comfortable with and that fits how you learn best.

Using the code from this project

Unless otherwise noted, all of the code from this project is free to use under the MIT license. You can view of copy of the license at https://gomakethings.com/mit.

Let's get started!

Getting Started

A screenshot of the completed project

When this project is done, you should be able to type a question into a field in your app, click *Ask the Fortune Teller*, and get back one a handful of yes/no/maybe responses.

The template

The template includes some HTML to get you started.

There's a form with an ID of `#app` that contains a text `input` field, its `label`, and a button to submit the question. There's also an empty `div` with an ID of `#answer` where the response from the fortune teller will go.

```
<form id="app">
    <label for="question">What's your questi
on?</label>
    <input type="text" id="question">
    <button>Ask the 8-Ball</button>
</form>

<div id="answer"></div>
```

Some things to consider

As you build this project, here are some questions to keep in mind.

- How will you know when the user has asked a question?
- How will you select an answer?
- How will people using screen readers know what the answer is?

Resources

Here are some articles, tips, and resources that will be helpful for working on this project.

If you're building it on your own, these may help you get started or get unstuck. If you're following along with me, they provide more information on some of the concepts and approaches I use.

- Getting elements in the DOM - https://vanillajstoolkit.com/reference/selectors/document-queryselector/
- Listening for browser events - https://vanillajstoolkit.com/reference/event-listeners/addeventlistener/
- How to shuffle an array with vanilla JS - https://gomakethings.com/how-to-shuffle-an-array-with-vanilla-js/
- Sanitizing user submitted content - https://gomakethings.com/preventing-cross-site-scripting-attacks-when-using-innerhtml-in-vanilla-javascript/
- Announcing dynamic content to screen readers - https://gomakethings.com/how-to-use-aria-live-with-dynamic-content/

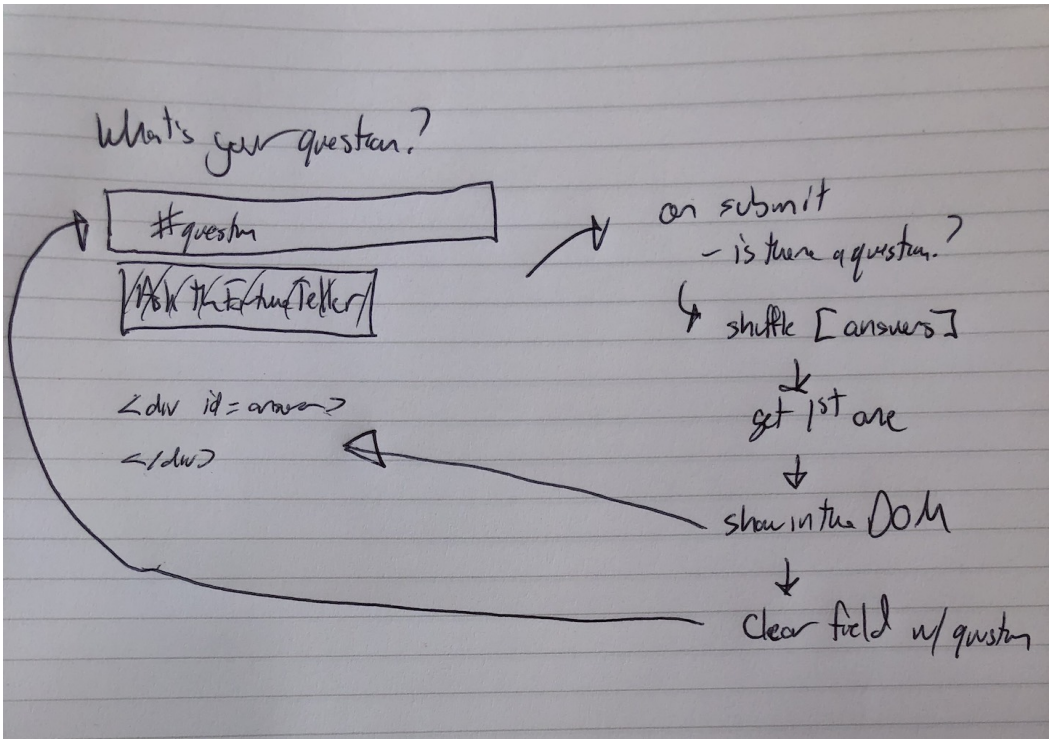
Planning

I like to plan out my projects before I start writing code.

Before I ever open a text editor or a browser, I plan my script out on paper. It helps me think big picture, and think through the logic of my code before I get bogged down in the specific methods and tactics I need to implement it.

If you're building the project on your own but need help getting started, my thought below may help. You might also want to try this yourself. And if you're building along with me, this is how I plan on approaching the project.

My Plan



A screenshot of my project plan

1. Detect when the `#app` form is submitted
    - If there's no question, do nothing
2. Shuffle an array of answers and get the first one
3. Display the answer and question in the DOM
    - Clear out the question field

The Project

Now let's walk through how I would approach this project.

Cache variables

First, let's cache some DOM elements that we'll be using over-and-over again to variables. We'll use the `querySelector()` to find the `#app` form, the `#question` field, and the `#answer` element.

```
//
// Variables
//

var form = document.querySelector('#app');
var question = document.querySelector('#ques
tion');
var answer = document.querySelector('#answer
');
```

Listen for user questions

Next, we need to detect when the user asks a question.

We can use the `addEventListener()` method on our `form` element to detect when it's submitted. To keep our code more organized, I'm going to make my event handler a named function: `submitHandler`.

We'll use a `submit` event here. This event will first when users click the button *and* if they type and hit the `enter` or `return` keys.

```
//
// Event Listeners
//

form.addEventListener('submit', submitHandle
r, false);
```

We'll pass in the `event` as an argument into the `submitHandler()` function.

In the handler, we'll use `event.preventDefault()` to prevent the form from submitting to the server and causing the page to reload.

```
/**
 * Run when the user asks a question
 * @param  {Event} event The form submission
 event
 */
var submitHandler = function (event) {

    // Prevent the form from causing a page
reload
    event.preventDefault();

};
```

If there's not a question to answer, we shouldn't do anything else.

We'll get the `question` field's `value`, and use the `length` property to make sure it has text in it. If the `value` is less than `1`—that is, if the field is empty—we'll use the`return` operator to end our handler function.

```
/**
 * Run when the user asks a question
 * @param  {Event} event The form submission
 event
 */
var submitHandler = function (event) {

    // Prevent the form from causing a page
reload
    event.preventDefault();

    // If there's no question, do nothing
    if (question.value.length < 1) return;

};
```

Creating an answer

Now we need to answer the user's question.

You can make up any sort of yes/no/maybe answers that you want. Because I'm lazy, I'm going to use the answers from a Magic 8-Ball toy[1] as inspiration.

Let's add them to an array in our variables section.

```
//
```

```
// Variables
//

var form = document.querySelector('#app');
var question = document.querySelector('#ques
tion');
var answer = document.querySelector('#answer
');
var answers = [
    'It is certain.',
    'It is decidedly so.',
    'Without a doubt.',
    'Yes - definitely.',
    'You may rely on it.',
    'As I see it, yes.',
    'Most likely.',
    'Outlook good.',
    'Yes.',
    'Signs point to yes.',
    'Reply hazy, try again.',
    'Ask again later.',
    'Better not tell you now.',
    'Cannot predict now.',
    'Concentrate and ask again.',
    'Don\'t count on it.',
    'My reply is no.',
    'My sources say no.',
    'Outlook not so good.',
    'Very doubtful.',
];
```

We need to pick a random answer from our array.

While languages like PHP and Ruby have built in methods for shuffling arrays, JavaScript does not. The most commonly recommended solution for this is to use the Fisher-Yates (or Knuth) Shuffle algorithm.[2]

Let's use a `shuffle()` helper function[3] for this.

```
/**
 * Randomly shuffle an array
 * https://stackoverflow.com/a/2450976/1293256
 * @param  {Array} array The array to shuffle
 * @return {String}      The first item in the shuffled array
 */
var shuffle = function (array) {

    var currentIndex = array.length;
    var temporaryValue, randomIndex;

    // While there remain elements to shuffle...
    while (0 !== currentIndex) {
        // Pick a remaining element...
        randomIndex = Math.floor(Math.random() * currentIndex);
```

```
        currentIndex -= 1;

        // And swap it with the current elem
ent.
        temporaryValue = array[currentIndex]
;
        array[currentIndex] = array[randomIn
dex];
        array[randomIndex] = temporaryValue;
    }

    return array;

};
```

To pick our answer, we'll create a copy of the`answers` array using the `Array.slice()` method[4], pass it into the `shuffle()` method, and return the first item.

Let's create a little function, `getAnswer()`, to handle that for us.

```
/**
 * Get a random answer from the list
 * @return {String} The answer
 */
var getAnswer = function () {
    return shuffle(answers.slice())[0];
};
```

Displaying the answer

In our `submitHandler()` function, we can use `getAnswer()` to get a response to the user's question.

Then, we can use the `innerHTMl` method to add the answer to our `answer` element.

```
/**
 * Run when the user asks a question
 * @param  {Event} event The form submission
 event
 */
var submitHandler = function (event) {

    // Prevent the form from causing a page
reload
    event.preventDefault();

    // If there's no question, do nothing
    if (question.value.length < 1) return;

    // Display the answer
    answer.innerHTML = '<p>' + getAnswer() +
 '</p>';

};
```

It might be helpful to submit the user's question above the answer.

But using `innerHTML` with user-submitted content can expose you to cross-site scripting (XSS) attacks.[5] We'll use a `sanitizeHTML()` helper function[6] to remove any markup from their response, first.

```
/*!
 * Sanitize and encode all HTML in a user-su
bmitted string
 * (c) 2018 Chris Ferdinandi, MIT License, h
ttps://gomakethings.com
 * @param  {String} str  The user-submitted
string
 * @return {String} str  The sanitized strin
g
 */
var sanitizeHTML = function (str) {
    var temp = document.createElement('div')
;
    temp.textContent = str;
    return temp.innerHTML;
};
```

Now, we can pass their question into `sanitizeHTML()` before injecting it into the DOM with `innerHTML`.

```
/**
 * Run when the user asks a question
 * @param  {Event} event The form submission
 event
 */
var submitHandler = function (event) {

    // Prevent the form from causing a page
reload
    event.preventDefault();

    // If there's no question, do nothing
    if (question.value.length < 1) return;

    // Display the question and answer
    answer.innerHTML =
        '<p><strong>' + sanitizeHTML(questio
n.value) + '</strong></p>' +
        '<p>' + getAnswer() + '</p>';

};
```

Returning focus

After the user asks a question, they may want to ask another. Let's clear the `question` input and return focus to it.

To do that, we'll set the `value` property on our `question` element to an empty string (`''`). Then, we'll call the `focus()` method on it.

```
/**
 * Run when the user asks a question
 * @param  {Event} event The form submission
 event
 */
var submitHandler = function (event) {

    // Prevent the form from causing a page
reload
    event.preventDefault();

    // If there's no question, do nothing
    if (question.value.length < 1) return;

    // Display the question and answer
    answer.innerHTML =
        '<p><strong>' + sanitizeHTML(questio
n.value) + '</strong></p>' +
        '<p>' + getAnswer() + '</p>';

    // Clear the question field
    question.value = '';
    question.focus();

};
```

Announcing changes to screen readers

This is all great, but how will people who use screen readers know when the value of the `answer` element has changed?

The `aria-live` attribute let's screen readers know that content in a particular element is going to change dynamically, and that they should pay attention to it and announce those changes.

It's value can be set to `off` (the same as not using it at all), `assertive` (in which screen readers interrupt user actions to announce changes), and `polite` (which tells the screen reader to wait until the user is done to announce updates).

Generally speaking, we should always use `polite`.

```
<div id="answer" aria-live="polite"></div>
```

And with that, congratulations! You just built a fortune teller app with vanilla JS.

Browser Compatibility

All of the techniques, methods, and browser APIs used in this project have great browser support.

The project works in all modern browsers, and IE9 and above.

Keep Learning

Now that you've completed the guide, here are some resources to help you continue learning.

- **Daily Developer Tips.**[7] I send out a short email each weekday with code snippets, tools, techniques, and interesting stuff from around the web.
- **Guides & Courses.**[8] Short, focused ebooks and video courses made for beginners. Start building real JavaScript projects in under an hour.
- **Vanilla JS Academy.**[9] A project-based online JavaScript training program for beginners.

Want more? I also created a vanilla JS learning roadmap[10], maintain the Vanilla JS Toolkit[11], host the Vanilla JS Podcast[12], and give talks and appear on podcasts[13].

About the Author

Hi, I'm Chris Ferdinandi. I believe there's a simpler, more resilient way to make things for the web.

I'm the author of the Vanilla JS Pocket Guide series, creator of the Vanilla JS Academy training program, and host of the Vanilla JS Podcast. My developer tips newsletter is read by thousands of developers each weekday.

I love pirates, puppies, and Pixar movies, and live near horse farms in rural Massachusetts. I run Go Make Things with Bailey Puppy, a lab-mix from Tennessee.

You can find me:

- On my website at GoMakeThings.com.
- By email at chris@gomakethings.com.
- On Twitter at @ChrisFerdinandi.

1. https://en.wikipedia.org/wiki/Magic_8-Ball↩
2. https://gomakethings.com/how-to-shuffle-an-array-with-vanilla-js/↩
3. https://vanillajstoolkit.com/helpers/shuffle/↩
4. https://vanillajstoolkit.com/reference/arrays/array-slice/↩
5. https://gomakethings.com/preventing-cross-site-scripting-attacks-when-using-innerhtml-in-vanilla-javascript/↩
6. https://vanillajstoolkit.com/helpers/sanitizehtml/↩
7. https://gomakethings.com/articles/↩
8. https://vanillajsguides.com/↩
9. https://vanillajsacademy.com/↩
10. https://learnvanillajs.com/↩
11. https://vanillajstoolkit.com/↩
12. https://vanillajspodcast.com/↩
13. https://gomakethings.com/talks↩