O'REILLY®

# Building Web Apps for Everyone


Adam D. Scott

# Building Web Apps for Everyone

*Adam D. Scott*

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

**Building Web Apps for Everyone**
by Adam D. Scott



Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://safaribooksonline.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

**Editor:** Meg Foley
**Production Editor:** Shiny Kalapurakkel
**Copyeditor:** Molly Ives Brower
**Proofreader:** Jasmine Kwityn

**Interior Designer:** David Futato
**Cover Designer:** Karen Montgomery
**Illustrator:** Rebecca Demarest

May 2016: First Edition

**Revision History for the First Edition**
2016-05-04: First Release



978-1-491-95552-9

[LSI]

# Table of Contents

# Preface

As web developers, we are responsible for shaping the experiences of users' online lives. By making ethical, user-centered choices, we create a better Web for everyone. The *Ethical Web Development* series aims to take a look at the ethical issues of web development.

With this in mind, I've attempted to divide the ethical issues of web development into four core principles.

- Web applications should work for everyone.
- Web applications should work everywhere.
- Web applications should respect a user's privacy and security.
- Web developers should be considerate of their peers.

The first three are all about making ethical decisions for the users of our sites and applications. When we build web applications, we are making decisions for others, often unknowingly to those users.

The fourth principle concerns how we interact with others in our industry. Though the media often presents the image of a lone hacker toiling away in a dim and dusty basement, the work we do is quite social and relies on a vast web of connected dependencies on the work of others.

## What Are Ethics?

If we're going to discuss the ethics of web development, we first need to establish a common understanding of how we apply the term *ethics*. The study of ethics falls into four categories:

*Meta-ethics*
: An attempt to understand the underlying questions of ethics and morality

*Descriptive ethics*
: The study and research of people's beliefs

*Normative ethics*
: The study of ethical action and creation of standards of right and wrong

*Applied ethics*
: The analysis of ethical issues, such as business ethics, environmental ethics, and social morality

For our purposes, we will do our best to determine a normative set of ethical standards as applied to web development, and then take an applied approach.

Within normative ethical theory, there is the idea of *consequentialism*, which argues that the ethical value of an action is based on the result of the action. In short, the consequences of doing something become the standard of right or wrong. One form of consequentialism, *utilitarianism*, states that an action is right if it leads to the most happiness, or well-being, for the greatest number of people. This utilitarian approach is the framework I've chosen to use as we explore the ethics of web development.

Whew! We fell down a deep dark hole of philosophical terminology, but I think it all boils down to this:

> *Make choices that have the most positive effect for the largest number of people.*

## Professional Ethics

Many professions have a standard expectation of behavior. These may be legally mandated or a social norm, but often take the form of a code of ethics that details conventions, standards, and expectations of those who practice the profession. The idea of a professional code of ethics can be traced back to the Hippocratic Oath, an oath taken by medical professionals that was written during the fifth century BC (see Figure P-1. Today, medical schools continue to administer the Hippocratic or a similar professional oath.

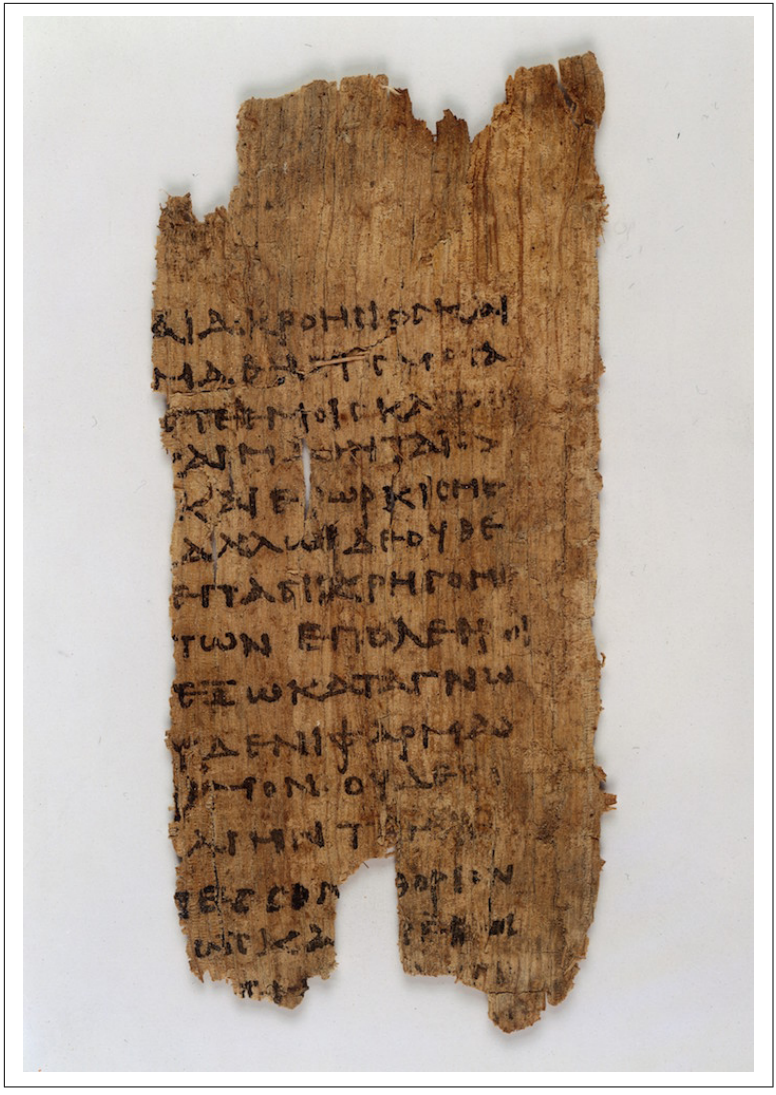

*Figure P-1. A fragment of the Hippocratic Oath from the third century (image courtesy of Wikimedia Commons)*

In the book *Thinking Like an Engineer*, Michael Davis says a code of conduct for professionals:

> prescribes how professionals are to pursue their common ideal so that each may do the best she can at a minimal cost to herself and those she cares about… The code is to protect each professional from certain pressures (for example, the pressure to cut corners to save money) by making it reasonably likely (and more likely then otherwise) that most other members of the profession will not take advantage of her good conduct. A code is a solution to a coordination problem.

My hope is that this report will help inspire a code of ethics for web developers, guiding our work in a way that is professional and inclusive.

The approaches I've laid out are merely my take on how web development can provide the greatest happiness for the greatest number of people. These approaches are likely to evolve as technology changes and may be unique for many development situations. I invite you to read my practical application of these ideas and hope that you apply them in some fashion to your own work.

This series is a work in progress, and I invite you to contribute. To learn more, visit the Ethical Web Development website.

## Intended Audience

This title, and others in the *Ethical Web Development* series, is intended for web developers and web development team decision makers who are interested in exploring the ethical boundaries of web development. I assume a basic understanding of fundamental web development topics such as HTML, JavaScript, and HTTP. Despite this assumption, I've done my best to describe these topics in a way that is approachable and understandable.

# CHAPTER 1
# Introduction

In 1998, Tim Berners-Lee, the creator of the Web, published ""The World Wide Web: A Very Short Personal History". In this brief essay, he states:

> The dream behind the Web is of a common information space in which we communicate by sharing information. Its universality is essential: the fact that a hypertext link can point to anything, be it personal, local or global, be it draft or highly polished.

To Berners-Lee, the universality of the Web is exactly what allowed it to grow from a single server at his desk to a global communication network with over three billion users worldwide.[1] While built upon the technologies established in those early days, today's Web has grown beyond the concept of hyperlinked documents. Today, we build rich graphical interfaces that encompass anything from a text document to a real-time video application, while also providing the primary means for personal interaction, information, and fundamental social services for many users.

With the rise of information and immersive applications on the Web, we have created a global network that society relies upon. Pausing to think about this, it is a beautiful thing and, true to Berners-Lee's vision, there remains little barrier to entry to publishing a site or application. However, as web developers, it is a profes-

1 Number of Internet Users (2016) - Internet Live Stats, *http://www.internetlivestats.com/internet-users/*.

sional and social responsibility to ensure that our sites and applications work for as many people as possible.

I have often been tempted to regard browser or device testing casually in favor of using the latest and greatest tools and browser features. Learning to use these new tools is one of the things that make web development so enjoyable, but we must temper this desire with the ability to build sites that work for as many users as possible. We should avoid shutting out users or denying them our services due to technical constraints. When we do this, we are taking an elite position, potentially shutting out the poor, disabled, and elderly. Imagine a storefront that didn't allow customers to enter if their shoes and clothes were too old. As a society we would find that offensive, and the shopkeeper would likely be publicly disgraced on the evening news. However, we often put banners on our site that say, "This site only supports *X* browser or newer," when a visitor accesses it with an older browser. Or worse, the site will silently fail, akin to the shopkeeper completely ignoring a customer.

## It Just Works

My wife and I began dating in 2003, and within a year or so I became her family's default "computer expert." In 2005, I helped my father-in-law, Marty, pick out a new computer for himself. To him, this was akin to an appliance purchase and we picked out a sturdy desktop computer, which has been in continuous use since then. We've made some upgrades to the RAM and I've done my best to point him to using an evergreen browser that automatically updates, but those no longer update on his aged XP system. When I asked him why he doesn't upgrade, he just shrugs and says "it still works." For him, the existence of the web browser is enough. He assumes that by typing in a URL, that the browser and machine connecting shouldn't make a difference.

When my grandfather passed away, my grandmother, Kathy, wanted to learn to use a computer and connect to the Web. Her primary device is an inexpensive and outdated Android tablet that connects to the Web through a wireless connection from the rural area where she lives. She uses it to check Facebook, read the news, read books, and play solitaire.

As developers, we want to assume that people like Marty and Kathy are edge cases. Looking at the top browsers currently in use, when

grouped together, device-specific browsers, outdated "evergreen" browser versions, and uncommon open source browsers occupy the second largest percentage of market share (see Figure 1-1).[2] Though each of these browsers and versions may only show up in our analytics as a fraction of a percent, when grouped together they become a powerful representation of the market.

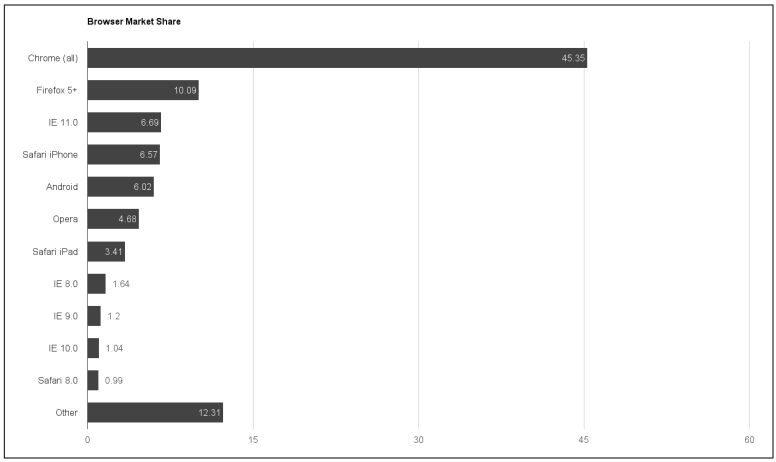


*Figure 1-1. Top browsers currently in use (nonstandard and outdated browsers comprise over 12% of the international browser market)*

Though the users of these browsers may not be the target demographic for our applications and services, by not accommodating for them we are denying them the opportunity to participate.

# A Responsibility

As web developers, we are gatekeepers to the vast troves of information and interaction across the Web. With this comes a responsibility to ensure that the Web is an open and inclusive space for all. The following chapters attempt to lay a groundwork for inclusive web development through:

2 The site StatCounter provides these metrics on its website. I've made the full list available as a CSV at *https://gist.github.com/ascott1/1f9b8fdc7529e4dd7823*.

*Progressive enhancement*
: By building progressively we can ensure that all users have access to a base experience, regardless of technology or network conditions.

*Accessibility*
: By building accessible user interfaces, we ensure that everyone has equal access to our applications regardless of disability or age.

*Inclusive forms*
: Forms allow users to interact directly with the Web, making it a two way form of communication. By creating web forms that are inclusive and usable, we demonstrate our dedication to inclusion.

# CHAPTER 2
# Progressive Enhancement

*Progressive enhancement* is a term that often incites intense debate. For many, progressive enhancement can be summed up as "make your site work without JavaScript." While developing a site that works without JavaScript often does fall under the umbrella of progressive enhancement, it can define a much more nuanced experience.

In Aaron Gustafson's seminal *A List Apart* article "Understanding Progressive Enhancement", he describes progressive enhancement as a peanut M&M: the peanut is the core experience, which is essential to the user. The chocolate is the features and design that take us beyond the naked peanut experience and add some much-loved flavor. Finally, the candy shell, though not necessarily needed, provides added features, such as not melting in your hand. Often this example uses HTML as the peanut, CSS as the chocolate, and JavaScript as the candy shell.

In today's web application landscape it may be an oversimplification to consider progressive enhancement as simply "works without JavaScript." In fact, many of the rich interactions and immersive experiences that have come to define the modern Web certainly require JavaScript. For progressive enhancement to be considered an ethical issue in web development, we must tie it back to user needs. Progressive enhancement is about defining what users need to get from your website and ensuring that it is always delivered to them, in a way that will work regardless of network conditions, device, or browser.

I prefer Jeremy Keith's view of progressive enhancement as a "process" rather than a specific technique or set of technologies. By Keith's definition, this process looks like:

1. Identify the core functionality
2. Make that functionality available using the simplest technology
3. Enhance!

As developers, it is our job to define the core functionality of our applications and establish what enhancements entail. This allows us to develop a baseline to build from—but the baseline for any given project may be different.

In his 2012 article "Stumbling on the Escalator", Christian Heilmann appropriated a Mitch Hedberg comedy bit about escalators for progressive enhancement:

> An escalator can never break—it can only become stairs. You would never see an "Escalator Temporarily Out Of Order" sign, just "Escalator Temporarily Stairs. Sorry for the convenience. We apologize for the fact that you can still get up there."

As a person who has spent a lot of time in Washington DC's Metro system, I can really appreciate this analogy. Fortunately, when an escalator is out I am not trapped underground, but instead can huff up the now-stairs to the street.

Often, when beginning a project, I am presented with a set of business requirements or a beautiful design. From these, it can be easy to see the end goal, but skip the baseline experience. If, in the case of the escalator, my requirement was to "build a transportation system that will allow Metro riders to travel from the terminal to the street," my initial reaction may be to create only an elevator. You can imagine how this might become problematic.

Developing web apps works in much the same way. If we only consider the end goal, we run the risk of leaving our users stranded. By focusing on and providing a solid baseline for our users, we set ourselves up for success in many other aspects of ethical web development, such as accessibility and performance.

## Defining Core Functionality

If progressive enhancement is the process of defining a core functionality and building up from there, how do we define that initial baseline? The goal is to consider the bare minimum that a user requires to use our application. Once we have defined this, we can layer on additional style and functionality. For some applications, this may be a completely JavaScript-free version of the experience, while for others it may be a less fully featured version; for still others it may be providing some server-rendered content on the initial page load only.

The key is to think of progressive enhancement not as a binary option, but instead as a range, determining what is the best decision for users. In this way, progressive enhancement is a gradient rather than an either/or option (see Figure 2-1). Our responsibility is to decide where on this gradient our particular application falls.

*Figure 2-1. Progressive enhancement is a gradient of choices*

I'd encourage you to take a few minutes and consider what the core functionality might look like for a few different types of websites and applications, including the following:

- News website
- Social network (write text posts and read your newsfeed)
- Image sharing website
- Web chat application
- Video chat application

Identify the primary goal of each type of site and determine the minimum amount of technology needed to implement it. To take it a step further, write some markup or pseudocode explaining how you might implement those baselines and features.

When working on your own applications, try to perform the same exercise. First, determine the core functionality for your users and build the application from there. This programmatic approach also

pairs well with the Agile approach to software development, where the goal is to deliver working software at the end of each development sprint. If we first deliver a core experience, we can iteratively build upon that experience while continuing to deliver value.

## Progressive Enhancement Is Still Relevant

Some may question how relevant progressive enhancement is today, when a small percentage of users browse the Web with JavaScript disabled.[1] This places the focus too heavily on progressive enhancement as a JavaScript-free version of a site. In fact, some types of applications, such as video chat, absolutely require some form of JavaScript to work in the browser. The goal of progressive enhancement is to provide the absolute minimum for a working product and ensure that it is delivered to each user's browser.

Ideally, this working minimum product is simply HTML without any external resources such as images, video, CSS, or JavaScript. When users' browsers request our site, we can be certain that they will receive HTML (or nothing at all). By creating a working version of our application, even with a minimal experience, using the smallest number of assets, we can be sure that the user is able to access our content in some form.

The Government Digital Service team (GDS) at GOV.UK provides a number of scenarios where asset requests may fail:

- Temporary network errors
- DNS lookup failures
- The server that the resource is found on could be overloaded or down, and fail to respond in time or at all
- Large institutions (e.g., banks and financial institutions, some government departments) having corporate firewalls that block, remove, or alter content from the Internet

1 In 2010, Yahoo conducted what is considered the definitive study of JavaScript usage, finding that the percentage of users with JavaScript disabled ranged from 0.26% to 2.06%, depending on the country of origin. Sadly, these statistics are long out of date. In 2013, GOV.UK's GDS team did a similar study and found that 1.1% of its users were not receiving JavaScript. The German site darwe.de analyzes JavaScript enablement in real time and shows a much larger percentage of users with JavaScript disabled visiting its site.

- Mobile network providers resampling images and altering content to make load times faster and reduce bandwidth consumed
- Antivirus and personal firewall software that will alter and/or block content

Additionally, in the blog post "How Many People Are Missing Out on JavaScript Enhancement?", the GDS added the following:

- Existing JavaScript errors in the browser (i.e., from browser add-ons, toolbars, etc.)
- Page being left between requesting the base image and the script/noscript image
- Browsers that preload pages they incorrectly predict you will visit

While those of us developing for the Web often have nice hardware and speedy web connections, that may not be true for many of our potential users. Those in developing or rural areas may have limited or outdated devices and slow connections. In 2015, the Facebook development team initiated a program called 2G Tuesdays, which allows them to experience their applications as though they are being served over these slower networks. I would encourage you to do the same.

Today's browser development tools allow us to mimic network conditions, experiencing what it is like for these users to access our sites (see Figure 2-2). We will explore the topic of web performance in greater detail in an upcoming title in this series.

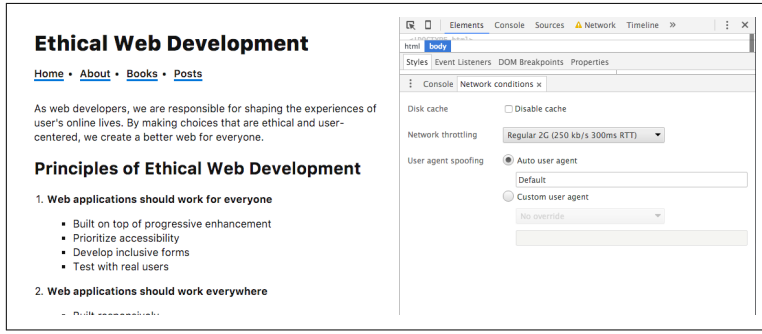


*Figure 2-2. The Google Chrome browser includes network connectivity simulation tools*

Though you may have never used, tested an application with, or even heard of it, the Opera Mini browser currently has over 300 mil-

lion users worldwide.[2] The browser is designed to greatly decrease mobile bandwidth usage by routing pages through Opera's servers and optimizing them. To do this, Opera Mini only supports a subset of modern web standards. Here are a few of the things that are unsupported by Opera Mini:

- Web fonts (which also means no icon fonts)
- HTML5 structural elements and form features
- Text-decoration styling
- Video and audio elements

The site Opera Mini Tips collects the full set of modern web standards that are not supported in the browser. As you can imagine, without a progressive enhancement strategy, our sites may be completely inaccessible for all 300+ million Opera Mini users.

When developing an application exclusively for users who are likely in an urban area with strong Internet speeds and nice hardware, we may feel as if we are exempt from concerning ourselves with connection issues. Recently, developer Jake Archibald coined the termed *Lie-Fi*. This is a connection where our mobile device seems to be connected to WiFi, but sites are slow to load as they feebly connect to our struggling signal.

In addition to the conditions just described, there may be external factors at play. In 2014, the UK's Sky broadband accidentally blocked the jQuery CDN for a brief amount of time, presumably leaving many users perplexed with broken websites. More recently, the ability to compose and publish a tweet became unavailable in the Twitter web client. This was caused by a regular expression that was being served by a CDN without the proper character encoding. Though the JavaScript likely worked in all local and quality assurance testing, once it was available on the Web it disabled one of the site's most critical features.

2 Owen Williams, "The Unknown Browser with 300 Million Users That's Breaking Your Site," TNW, *http://thenextweb.com/dd/2015/12/24/the-unknown-browser-with-300-million-users-thatsbreaking-your-site/*.

## Run Your Own Experiment

GDS was curious to see how many users were missing out on JavaScript resources when accessing its site. To test this, the team ran an experiment by adding three images to a page:

- An image that all browsers should request
- An image that would only be requested via JavaScript
- An image that only browsers with JavaScript disabled would request

The results of this experiment are really fascinating. Though only a fraction of a percentage of users requested the JavaScript-disabled image, those that failed to load the image requested via JavaScript were significantly higher.

If possible, I'd encourage you and your teams to conduct a similar experiment. This allows us to base the decision to support (or not support) Javascript-disabled users with data, rather than assumptions or world averages.

To run this experiment on your own site, first create three empty GIF files named *base-js.gif*, *with-js.gif*, and *without-js.gif*. Then you can use the following snippet (adapted from GOV.UK's experiment) in your HTML:

```
<img src="base-js.gif" alt="" role="presentation"
  style="position: absolute; left: -9999em; height: 0px;
         width: 0px;">
<script type="text/javascript">
  (function(){
    var a = document.createElement("img");
    a.src = "with-js.gif";
    a.alt = "";
    a.role = "presentation";
    a.style.position = "absolute";
    a.style.left = "-9999em";
    a.style.height = "0px";
    a.style.width = "0px";
    document.getElementById("wrapper").appendChild(a);
  })();
</script>
<noscript>
   <img src="without-js.gif" alt="" role="presentation"
    style="position: absolute; left: -9999em; height: 0px;
           width: 0px;">
</noscript>
```

**Web Beacons**

This approach, used by GOV.UK, is a recommendation to use *web beacons*, which may have some privacy implications. Web beacons are typically hidden pieces of content used to check if a user has accessed a piece of content, and are common in email marketing campaigns. Privacy issues such as this will be discussed in an upcoming title in the series.

## How Can We Approach Progressive Enhancement Today?

Recently, I was talking with my friend and colleague Scott Cranfill about a progressive enhancement strategy for a project he was working on. This project was mostly static content, but also included an auto loan calculator. When discussing how he might approach this from a progressive enhancement angle, he mentioned that he thought the default markup should simply include the formula that the calculator uses. Once the page's assets load, a fully functional dynamic calculator will display. This means that nearly every user will only see and interact with the calculator, but should something go wrong, a user will still be presented with something that is potentially useful. I loved this pragmatic approach. It wasn't about "making it work without JavaScript," but instead about making it work for everyone.

In the world of modern, JavaScript-driven web applications, there are still several practical approaches we can take to build progressively enhanced sites. These approaches can be simple or leverage exciting web technology buzzwords such as *isomorphic JavaScript* or *progressive web applications*. Because progressive enhancement is not a one-size-fits-all approach, you may want to evaluate these and choose the one that best works for your project.

Let's take a look at a few of these options and how they may be used to build the best possible experience for a user.

Perhaps the simplest and most progressive is to completely avoid a JavaScript-dependent first-page render. By rendering all of the *necessary* content on the server, we can ensure that users receive a usable page, even if only our HTML makes it to their browser. The key here

is to focus on what is necessary. There may be additional JavaScript-required functionality, but if it isn't necessary we can allow it to quietly fail in the background or present the user with different content.

If you choose to serve a library from a CDN, you should provide a local fallback as well, as recommended by HTML5 Boilerplate's. This allows us to leverage the benefits of the CDN while ensuring that the user has the opportunity to downloads the scripts should there be an issue with the CDN, such as unexpected down time or being blocked by an ISP or third-party browser add-on. Here's the code you'll need to use:

```
<script src="https://code.jquery.com/jquery-1.12.0.min.js">
</script>
<script>
  window.jQuery || document.write(
    '<script src="js/vendor/jquery.min.js"><\/script>'
  )
</script>
```

Another option, or one that may paired with the previous, is to sniff out outdated browsers and avoid serving JavaScript to those browsers. We can continue to serve our core content and functionality to those browsers (it was progressively enhanced, after all!), but offer a significantly simpler experience.

To sniff out older browsers, we can use a technique demonstrated by Jake Archibald from his 2015 Nordic.js talk. This checks for the availability of the Page Visibility JavaScript API, which is only available in modern browsers. If Page Visibility is unavailable, the code exits without attempting to execute. You can use the following code to check for the Page Visibility API:

```
(function() {
  if (!('visibilityState' in document)) {
    return false;
  }

  // rest of your code
}());
```

### IIFE

The preceding example is wrapped in an immediately invoked function expression (IIFE), which may look a bit odd if you're not a JavaScript developer. This ensures that the code executes immediately while avoiding the pollution of the global scope. If you are interested in learning more, see Ben Alman's detailed post about IIFEs.

For JavaScript-dependent applications, we could render the landing page as HTML on the server while *prefetching the JavaScript* for the rest of the application:

```
<link rel="prefetch" href="app.js">
```

This approach gives our users the opportunity to download and cache the application's JavaScript, without impacting the performance or requirement on a mostly static page. Soon browsers will begin implementing the Preload specification, which will be similar to `prefetch`, but enable additional browser features.

In action, preload looks similar to prefetch:

```
<link rel="preload" href="app.js" as="script">
```

Here are some prefetch and preload resources:

- Robin Rendle's "Prefetching, Preloading, Prebrowsing"
- Luis Vieira's "HTML5 Prefetch"
- Yoav Weiss's "Preload: What Is It Good For?"

You may be thinking, "But I want to build *modern* JavaScript web applications." Certainly these techniques feel out of sync with the approaches of some of the popular JavaScript frameworks, but recently we've seen the most popular web application approaches trend back toward a progressive enhancement model.

*Isomorphic* or *universal* JavaScript is a technique that allows a developer to pair server-side and client-side JavaScript into a "write once, run everywhere" approach. This technique means that the initial application will render on the server, using Node.js, and then run in the browser. When building a progressively enhanced isomorphic application we can start by building our server-rendered version of the applications and layer on the isomorphic approach.

A similar approach was taken by the team behind the recent Google+ redesign:

> With server-side rendering we make sure that the user can begin reading as soon as the HTML is loaded, and no JavaScript needs to run in order to update the contents of the page. Once the page is loaded and the user clicks on a link, we do not want to perform a full round-trip to render everything again. This is where client-side rendering becomes important—we just need to fetch the data and the templates, and render the new page on the client. This involves lots of tradeoffs; so we used a framework that makes server-side and client-side rendering easy without the downside of having to implement everything twice—on the server and on the client.

#### Isomorphic JavaScript

Though my description may be oversimplified, isomorphic JavaScript is an exciting approach for developers and teams who are using server-side JavaScript. To learn more about isomorphic JavaScript, I recommend taking a look at the following resources:

- *Building Isomorphic JavaScript Apps* by Jason Strimpel (O'Reilly, 2015)
- Spike Brehm's "Isomorphic JavaScript: The Future of Web Apps"
- Jack Franklin's "Universal React"

If a fully isomorphic JavaScript approach is overkill for an application, Henrik Joreteg has coined the term *lazymorphic applications*. A lazymorphic application is simply one where the developer pre-renders as much of the application as possible as static files at build time. Using this approach, we can choose what we render, making something useful for the user while withholding JavaScript-dependent features.

Lastly, the term *progressive web apps* has recently taken hold. Rather than specific technology, this term has come to encompass several interrelated techniques and approaches to web development. This is an approach that pairs nicely with all of those listed earlier.

In his article "Progressive Web Apps: Escaping Tabs Without Losing Our Soul", Alex Russell described progressive web applications in this way:

- Responsive
- Connectivity independent
- App-like interactions
- Fresh
- Safe
- Discoverable
- Re-engageable
- Installable
- Linkable

The progressive web application approach just described is well aligned to an ethical web application experience by focusing on delivering an application experience that works for every user.

#### Progressive Web Applications

Though rooted in several technologies, the overarching concept of progressive web applications is just starting to take hold. Here are a few of the resources that I've found most useful for getting started:

- Google Developers: Progressive Web Apps
- Progressive Web Apps HQ
- Getting Started with Progressive Web Apps

## In Summary

There are a variety of techniques and approaches that allow us to build progressively enhanced modern websites and applications. This chapter has outlined a few of these options. By beginning with the core functionality, we are able to ensure that our application works for the maximum number of people. This provides us with a baseline to provide working software for all users in a range of situations.

From an ethical standpoint, progressive enhancement provides several benefits to our users. By following a progressive enhancement process, we can be sure that we are building our applications in a way that allows them to be available for as many users as possible, regardless of device, connection, or browser.

## Additional Resources

- Aaron Gustafson's "Understanding Progressive Enhancement"
- GOV.UK's "Progressive Enhancement: How to Create Pages That Work Regardless of Browser Capability"
- The Responsive News blog's "Cutting the Mustard"
- Jake Archibald's "Progressive Enhancement Is Still Important"
- Ponyfoo.com's "Stop Breaking the Web"

# CHAPTER 3
# Web Accessibility

> The power of the Web is in its universality. Access by everyone regardless of disability is an essential aspect.
>
> —Tim Berners-Lee, Inventor of the World Wide Web

Building accessible websites and applications means making those sites available to users regardless of physical ability. This covers a broad range of disabilities, such as visual, physical, auditory, cognitive, and age-related disabilities. When we build with accessibility in mind, we make an ethical decision of inclusion. Alternatively, when we choose to ignore accessibility, it excludes people with disabilities from participating in the Web.

Today, as web users, we access government services, educational resources, bank transactions, social interactions, work tasks, healthcare, entertainment, and more through our web browsers. As the Web continues to play an increasingly large role in our daily lives, this causes inaccessible websites to be a hurdle to fully participating in society. With the importance the Web plays in our society, it has become our responsibility as developers to ensure its equal access to all.

The W3C summarizes the social issues around web accessibility in three principles:

- It is essential that the Web is accessible in order to provide equal access and equal opportunity to people with disabilities.
- The Web is an opportunity for unprecedented access to information for people with disabilities.

- The Web is an opportunity for unprecedented interaction for people with disabilities.

The W3C has also called out that the United Nations Convention on the Rights of Persons with Disabilities expresses that accessibility across the web has become a human right. Specifically it states:

> To enable persons with disabilities to live independently and participate fully in all aspects of life, States Parties shall take appropriate measures to ensure to persons with disabilities access, on an equal basis with others […] to information and communications, including information and communications technologies and systems[…].
>
> States Parties shall take all appropriate measures to ensure that persons with disabilities can exercise the right to freedom of expression and opinion, including the freedom to seek, receive and impart information and ideas on an equal basis with others and through all forms of communication of their choice[…].

When done correctly, an accessible Web not only provides equal access to services and information, but also empowers those with disabilities.

### Further Reading

- W3C's "Social Factors in Developing a Web Accessibility Business Case for Your Organization"
- W3C's "Designing for Inclusion"

## Broadening the Scope of Accessibility

The need for accessibility is not limited to those with permanent disabilities. Accessible web interfaces may benefit a range of users.

As of the 2010 census, there are 17 million people 75 years of age and older living in the United States. Worldwide, over 8% of the earth's total population falls into the 65 and above category. This age group has consistently grown as a percentage of the total population, creating an increasing market segment. With an aging population, there are several factors that can impact the use of the Web, most notably motor and vision impairments. By building with accessibility in mind, we are able to accommodate users of all ages.

In addition to those with permanent or age-related disabilities, many users may have temporary disabilities. Often these may be related to an injury where vision or motor abilities are impaired.

### Further Reading

- Melody Kramer's "News for Betty"
- W3C's "Meeting the Needs of Aging Web Users"
- The A11Y Project's "MYTH: Accessibility Is 'Blind People'"

## Web Content Accessibility Guidelines

In 2008, the W3C released an update to the Web Content Accessibility Guidelines, commonly referred to as WCAG 2.0. This document covers a range of success criteria for creating accessible web content. Rather than focusing on specific technologies, the document offers suggestions for making all websites and applications more accessible.

### POUR

The guidelines and success criteria of building WCAG 2.0 accessible web applications are organized around the POUR principle. *POUR* stands for *P*erceivable, *O*perable, *U*nderstandable, and *R*obust. Following these guidelines allows us to build websites and applications that are usable by all. Let's take a closer look at each of these guidelines:

*Perceivable*
: Information and user interface components must be presentable to users in ways they can perceive.

*Perceivable* means that a user should be provided the opportunity to perceive the content of our web applications. To do this, we must ensure that the information being presented is perceptible to their senses. When we limit content to a single sense, we run the risk of alienating users.

One common use case of perception is providing written transcripts of audio material or captioning video material. Perhaps a less obvious example is the style of links across the Web. A link that is only a different color from the text would be imperceptible to color-blind

users.[1] Instead, we should be sure to alter the color as well as provide an underline to the link, as illustrated in Figure 3-1. This provides a perceptible option to color-blind users.


*Figure 3-1. Retaining the default link behavior of including an underline allows links to be perceptible to color-blind users*

*Operable*
: User interface components and navigation must be operable.

When a web application is *operable*, all users are able to operate and navigate its interface. Perhaps a simple example of this is providing users the ability to easily "tab through" our sites, by navigating through the page using only the Tab keyboard key. A user who is unable to operate a mouse or trackpad may navigate sites using only the keyboard. Ensuring that our sites are keyboard-accessible is one way to ensure that they are operable by all users.

*Understandable*
: Information and the operation of user interface must be understandable.

When we build understandable interfaces, we follow common development patterns of hierarchy and user interaction. When working with a team, these may often fall in the domain of a designer, but as developers we make choices about how we develop these patterns.

One common anti-pattern is using form-label `placeholder` text in the place of a form label (see Figure 3-2.

Here is a form element, properly marked up with a label and placeholder text:

```
<label for="password">Password:</label>
<input type="text" name="password">
```

1 Color blindess affects a significant portion of the population, specifically those born male. Roughly 8% of male-born people are said to have some form of color blindness. For a brief overview, see *https://en.wikipedia.org/wiki/Color_blindness#Epidemiology*.

Here is one that uses the placeholder to replace the label:

```
<input type="text" name="password" placeholder="Password">
```

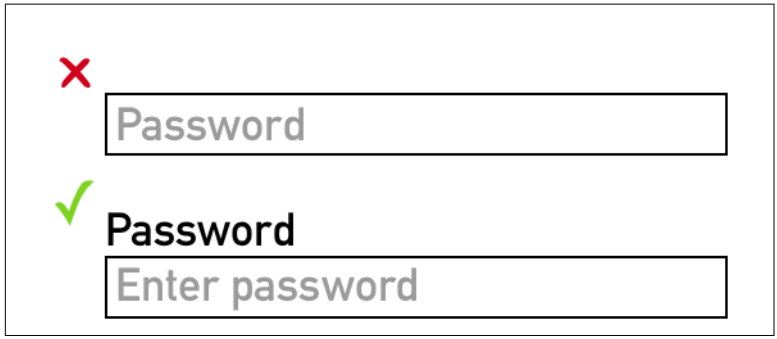


*Figure 3-2. Avoid using form placeholders in place of labels*

The use of placeholder text in the place of labels raises several potential usability and accessibility concerns due to low contrast, extra cognitive burden as users must recall the purpose of the field, and unreliable screen reader support. By making solid development decisions such as proper form markup, we can develop more understandable sites.

*Robust*
: Content must be robust enough that it can be interpreted reliably by a wide variety of user agents, including assistive technologies.

By building *robust* web applications, we build future-friendly sites that are device and browser agnostic. When we develop without a specific platform in mind and do not limit our browser support, we are able to build sites that are accessible to any user. This is a topic that we will cover in depth in a future title, *Building Web Applications that Work Everywhere*.

## WCAG Conformance

WCAG 2.0 is separated into three levels of conformance, based on success criteria that is defined in the WCAG 2.0 specification:

*Level A*
: Level A provides basic web accessibility support. This meets all Level A success criteria, or provides an alternate content version that does.

*Level AA*
: Level AA addresses the most common accessibility issues. This meets all Level A and Level AA success criteria or provides an alternate content version that does.

*Level AAA*
: Level AAA provides the highest level of web accessibility support for users. This meets all Level A, Level AA, and Level AAA success criteria, or provides an alternate version.

It is worth noting that the W3C does not recommend that Level AAA conformance be required for entire sites, as it is not possible to satisfy all Level AAA success criteria for some content. If you choose to put an accessibility policy in place for your organization, I recommend aiming for Level AA support.

## WCAG 2.0 Guidelines

The WCAG 2.0 guidelines are classified according to the POUR principle. Guideline 1 encompasses areas of perception, Guideline 2 covers areas of operation, Guideline 3 is directed at ensuring understanding, and Guideline 4 is targeted at the robustness of the application. The following are the specific guidelines from WCAG 2.0:

- Guideline 1.1 Text Alternatives: Provide text alternatives for any non-text content
- Guideline 1.2 Time-based Media: Provide alternatives for time-based media
- Guideline 1.3 Adaptable: Create content that can be presented in different ways (for example, simpler layout) without losing information or structure
- Guideline 1.4 Distinguishable: Make it easier for users to see and hear content including separating foreground from background.
- Guideline 2.1 Keyboard Accessible: Make all functionality available from a keyboard.
- Guideline 2.2 Enough Time: Provide users enough time to read and use content.
- Guideline 2.3 Seizures: Do not design content in a way that is known to cause seizures.
- Guideline 2.4 Navigable: Provide ways to help users navigate, find content, and determine where they are.

- Guideline 3.1 Readable: Make text content readable and understandable.
- Guideline 3.2 Predictable: Make Web pages appear and operate in predictable ways.
- Guideline 3.3 Input Assistance: Help users avoid and correct mistakes.
- Guideline 4.1 Compatible: Maximize compatibility with current and future user agents, including assistive technologies.

WebAIM provides a helpful checklist of recommendations for implementing HTML-related principles and techniques for WCAG 2.0 conformance, along with the associated support level.

### Further Reading

- W3C's "Techniques for WCAG 2.0"
- W3C's "WCAG 2.0"
- W3C's "Understanding WCAG 2.0"

## Using Your Keyboard to Navigate the Web

For many desktop users, navigating through websites with a keyboard is an essential function. These users may have motor disabilities, vision disabilities, limb loss, or injuries that make mouse usage impractical. By ensuring that our sites are keyboard navigable, we are increasing the availability of our applications to these users.

The basics of keyboard navigation are simple:

- Press the Tab key to navigate through the page
- Press Tab + Shift to go backward in your navigation

Using the Tab key, we are able to navigate to links, buttons, and form elements on the page. This allows us to quickly interact with elements on the page.

There are a few ways that as developers we can ensure the best possible keyboard navigation experience for our users. Primarily, we should aim to:

- Keep or apply custom `:focus` styles
- Be aware of navigation order and length
- Use default navigation elements

## Keep or Add :focus Style

When a user navigates to an element with the Tab key, it is visually indicated using a `:focus` style. By default, in most browsers this is either a thin dotted line or a blue highlighted border around the focused element. Unfortunately, it has become a fairly common trend to remove this focus style by applying an `outline:0` or `outline:none` in CSS. This can be avoided by either retaining that default behavior or adding a custom `focus` style with CSS:

```
a:focus {
  background-color: #01FF70;
}
```

### Further reading

- The A11Y Project's "Quick Tip: Never Remove CSS Outlines"

## Navigation Order and Length

When a user navigates through a site using the keyboard, the order and length of keyboard-navigable items become critically important. Imagine the frustration of a user pressing Tab through every link presented in a sidebar (such as Twitter or Facebook's "trends") before being able to access the main content of the application. By using a "skip to content" (or "skip navigation") link in our applications we can provide an extra guidepost that directs our users to the most valuable content.

*Skip navigation links*, or skip links, are hidden links which, when "tabbed" to or read by a screen reader, allow the user to skip to the main content of the page, rather than needing to navigate through navigation and other content that occurs earlier in the document order. These can be visually hidden, appearing only when a user presses the Tab key.

To begin, add a skip navigation link at the top of the document, linking directly to the main content of the page:

```
<body>
  <a href="#main" class="skip-link">Skip to main content</a>
  …
  <main id="main" role="main">
    <h1>Page Content</h1>
```

```
    …
  </main>
</body>
```

For styles, we can first hide the skip link visually while keeping it visible to screen readers:

```
.skip-link {
  position: absolute;
  width: 1px;
  height: 1px;
  margin: 0;
  overflow: hidden;
  clip: rect(1px, 1px, 1px, 1px);
}
```

Then make it appear visually when the link receives focus with the Tab key:

```
.skip-link:focus {
  top: 0;
  z-index: 1;
  width: auto;
  height: auto;
  clip: auto;
}
```

### Further reading

- The A11Y Project's "How–to: Use Skip Navigation Links"
- WebAIM's "'Skip Navigation' Links"
- NCZOnline's "Fixing 'Skip to content' Links"

## Default Navigation Elements

By default, links, buttons, and form items are navigable using the Tab key. However, there are times that we may want to include additional items that are tabbable by our users. To do this, we can set a "tab index" value on an HTML element:

```
<h3 tabindex="0">This heading is tabbable</h3>
```

The approach should generally be to set a `tabindex` value of zero. A number greater than 1 sets an explicit tab order, and should be avoided.

### Further reading

- WebAIM's "Tabindex"
- WebAIM's "Keyboard Accessibility"
- Nielsen Norman Group's "Keyboard-Only Navigation for Improved Accessibility"

## Using a Screen Reader to Navigate the Web

For the visually impaired, a screen reader may provide access to our sites and applications. They also provide a means to test the simplicity of navigation through the application. There are many screen readers available, but for quick testing I recommend ChromeVox for the Google Chrome Browser. The ChromeVox screenreader also adds additional focus styles to the active element, which make it possible for those with auditory issues to test screen reader support.

To get started with ChromeVox:

1. Install the Google Chrome Browser
2. Install the ChromeVox Extension
3. Follow the ChromeVox interactive tutorial

### Other Screen Readers

Though ChromeVox is great you may want to explore the use of more commonly used screen readers. Here are the most popular screen readers, according to the WebAIM screen reader survey:

- JAWS
- ZoomText
- Window-Eyes
- NVDA

Additionally, most operating systems ship with built-in screen reader support. This can be particularly useful for testing web applications on mobile devices.

- VoiceOver for Mac: CMD + F5 to enable
- Narrator for Windows
- VoiceOver for iOS
- Google Talkback

# Writing Accessible Markup

Perhaps the most important aspect of an accessible web application is providing users with semantic HTML, whether it is delivered server or client side. When writing semantic, well-structured HTML, we can ensure that our users will receive content that is perceptible to them in a variety of ways.

Here are a few tips for writing semantic and useful HTML:

- Make use of HTML5 content tags, including `<header>`, `<nav>`, `<main>`, `<footer>`, and `<aside>`.
- Mark up the HTML content in the order you would expect the user to read it and use CSS to change the order appearance when necessary.
- Use semantic heading tags that follow a clear outline of page content and avoid using styled paragraphs or other elements as headers.
- Use `<label>` elements to associate the label of form field with its input element.
- Always provide alternate text for images using the `alt` attribute. When an image is purely decorative, give the `alt` attribute an empty value (`alt=""`) so that screen readers know to ignore it.

## ARIA

In addition to writing clear and semantic markup, we can make use of ARIA to provide additional guideposts to screen reader users. Accessible Rich Internet Applications (ARIA) is a specification designed to make modern web application experiences more accessible. ARIA provides additional attributes that can be added to our markup to provide screen reader users with additional information to describe the role, state, and properties of elements on our page.

Perhaps one of the most immediately useful aspects of ARIA is the use of ARIA *roles*. ARIA roles allow us to provide specific information to screen readers about the context of HTML elements. Doing this allows screen reader users to quickly navigate to these subsections of our pages. I've found that watching videos of screen readers using ARIA roles can be a really helpful exercise in understanding the usefulness of ARIA.

To use an ARIA role, we add a `role` attribute to an HTML element and provide it with an ARIA value:

```
<div class="main-content" role="main">
```

There are two categories of ARIA roles that may commonly be used by web application developers:

*Landmark*
: Landmark roles define content areas that a screen reader user may want to quickly navigate to, such as navigation menus and the main content of the page.

*Widget*
: Widget roles are used for common interactive UI patterns such as tooltips, tabs, and progress bars.

The W3C provides a list of all of the possible ARIA role definitions. Here are some common ARIA role values that are useful to web application developers:

`application`
: Content that is declared as a web application rather than a web document

`banner`
: Typically the header of the page with the site name

`search`
: Indicates a search input

`main`
: The main content of the page

`navigation`
: Defines any navigation area of the page

`contentinfo`
: An area that contains information about the page such as author or organization title and copyright information; this is often the footer

`group`
: A collection of UI elements that are not intended to be included in a page summary or table of contents by assistive technologies

`menu`
: A list of user choices

`toolbar`
: A collection of commonly used functions "represented in a compact visual form"

ARIA is more than just roles, however. ARIA can be used to define values on progress bars, hidden states, dynamic content updates, and more that are outside of the scope of this introduction. I've provided additional links and tutorials in the "Further Reading" section that detail the usefulness of these additional values.

### Further Reading

- W3C's "Notes on Using ARIA in HTML"
- W3C's "WAI-ARIA 1.1 Authoring Practices"
- Mozilla Developer Network's documentation for ARIA
- The A11Y Project's "Getting Started with ARIA"
- The A11Y Project's "Quick Tip: Aria Landmark Roles and HTML5 Implicit Mapping"

## Accessibility Tools

There are a number of tools that can improve accessibility testing for developers. These allow us to quickly or automatically spot accessibility issues on our sites.

### Browser Extensions and Bookmarklets

Browser extensions and bookmarklets allow us to test the accessibility of a page as we are interacting with it. These are a great option for quickly doing spot checks for things like ARIA role usage, color contrast ratios, descriptive `alt` text, and form-label usage.

The following are some useful browser extensions and bookmarklets:

- tota11y
- WAVE Chrome Extension
- Chrome Accessibility Developer Tools
- aXe

## Command-Line Tools

In addition to browser extensions or bookmarklets, we can automate common accessibility checks from the command line. This allows us, as developers, to integrate accessibility checks into our workflow. Typically these tools run our sites in a headless browser—one that doesn't have a graphical user interface—and test for accessibility concerns such as color contrast, heading order, link content, and `alt` text.

The following are some useful command-line accessibility tools:

- pa11y
- a11y
- node-wcag

## Automating Accessibility Tests

We can also integrate those command-line tools into an automated build process. Here are a couple of simple examples of how we might do that.

As an npm script in *package.json*, we could automate the running of accessibility steps with a11y against our localhost server:

```
"scripts": {
  "accessibility": "a11y localhost:3000",
}
```

We could also run those tests as a Gulp task. This is a simple task that will log the accessibility audit checks and failures to the console:

```
var exec = require('child_process').exec;

gulp.task('accessibility', function (){
  exec('a11y localhost:3000', function(err, stdout, stderr) {
    console.log(stdout);
    console.log(stderr);
  });
});
```

When paired with a continuous integration system, such as Travis CI, we could run these accessibility checks against every build, failing if there are accessibility errors. This would ensure that our codebase remains accessibility-compliant when new code is added.

## Accessibility Checklists

While we may aim to build our applications with accessibility in mind throughout the development process, it is possible to overlook an accessibility issue. I've found it useful in to reference an accessibility checklist prior to launch of a new project. This allows me (and the team I'm working with) to add a final check for accessibility into the workflow of the project. The A11Y Project's Web Accessibility Checklist provides a concise checklist of accessibility best practices; for a more thorough check of WCAG compliance, there is WebAIM's WCAG 2.0 Checklist. Both of these lists allow you to quickly check for possible accessibility oversights.

# Creating an Accessibility Policy

Adopting an accessibility policy for your project or organization is a way of demonstrating both within and outside of the organization that you are dedicated to accessibility. For organizations that are unclear on the value of creating such a policy, W3C's "Developing a Web Accessibility Business Case for Your Organization" clearly outlines the positive outcomes of promoting accessibility.

The majority of national or federal government, state government, and public university websites offer some form of web accessibility policy. I recommend taking a look at your local and national government sites to see what they are. A smaller number of corporations and nonprofits do the same. I have begun cataloging these policies at *github.com/ascott1/accessibility-policies*.

Here are a few accessibility policies that you may want to explore when developing one for your organization:

- *The Economist*
- Travelocity
- Goodwill Industries
- Lloyds Bank
- Elsevier
- University of Wisconsin-Madison

The W3C's "Developing Organizational Policies on Web Accessibility" provides guidance on creating an accessibility policy, complete with a template. I have adapted that template to a more human-readable format, in Markdown, including a list of commitments:

```
[ORGANIZATION OR PROJECT NAME] is committed to ensuring the
accessibility of this site for people with disabilities.
We pledge to meet [W3C WAI's Web Content Accessibility
Guidelines 2.0](https://www.w3.org/TR/WCAG/), Level AA
conformance. Any issues should be reported to [EMAIL ADDRESS].

We are committed to ensuring accessibility through:

- Use of ARIA landmark roles.
- Proper use of HTML headings.
- Perceptible link text and focus states.
- Sufficient color contrast.
- The use of appropriate alt text for images.
- Keyboard navigable forms inputs with descriptive labels.
- Text transcripts for audio and closed captioning for video
  content.
- Navigation without the need for a mouse or track pad.
```

# In Summary

Accessibility is a core value for ethical web development. By ensuring that our applications are accessible, we are providing access to everyone.

## Additional Resources

- W3C's "Web Accessibility Initiative"
- W3C's "Web Accessibility Guidelines"
- W3C's "Web Accessibility Tutorials"
- The A11Y Project
- WebAIM
- Matt Long's "It's Tired In Here: Web Accessibility"

CHAPTER 4

# Developing Inclusive Forms

Forms allow users to interact directly with a site. They are often the thing that differentiates a *website* from a web *application*.

## What's in a Name?

In Dale Carnegie's influential 1936 self-help book, *How To Win Friends and Influence People*, he states "a person's name is, to that person, the sweetest and most important sound in any language." Names are a core part of our personal identities. We often identify with them, turn at the sound of them said across the room, and intuitively appreciate when a person we have just met remembers our names.

Unfortunately, as web developers, it is possible to make assumptions about names that lead to their incorrect handling. When working with names, we should be prepared for a variety of characters, spacing, and unique international formats.

In his article "Falsehoods Programmers Believe About Names", Patrick McKenzie lists out 40 common misconceptions, including these assumptions:

- People have exactly one canonical full name.
- People's names fit within a certain defined amount of space.
- People's names are written in any single character set.
- People have last names, family names, or anything else which is shared by folks recognized as their relatives.

- My system will never have to deal with names from China, Japan, Korea, Ireland, the United Kingdom, the United States, Spain, Mexico, Brazil, Peru, Russia, Sweden, Botswana, South Africa, Trinidad, Haiti, France, or the Klingon Empire, all of which have "weird" naming schemes in common use.

The full list is well worth a read, as it succinctly points out many potential missteps.

In her article ""Hello, My Name Is <error>", Aimee Gonzalez-Cameron shares her story of taking the GRE, an exam administered for admission to graduate school in the United States. One of the first instructions in registering for the exam was as follows:

> Important: The name you use when you register for a GRE test must exactly match (excluding accents, apostrophes and spaces) the name on the identification (ID) documents that you will present on the day of your GRE test. If it does not, you may be prohibited from taking the test or your test scores may be canceled after you take the test. For example, a last name of Fernandez de Córdova would be entered as FernandezdeCordova.

As she points out, "Students shouldn't stress about instructions or worry that their answers will be thrown out because they can't complete the first step correctly." The lack of a technical system that properly handles a common American surname format is both culturally insensitive and requires extra instruction for correct handling.

Perhaps relatable from the perspective of many developers is the case of Christopher Null. Without reading further, you may already be shaking your head at the heartache that a last name of "Null" may cause when dealing with web forms. In his article, ""Hello, I'm Mr. Null. My Name Makes Me Invisible to Computers", he details his experience using the Web with the last name of Null. Because "null" is used to represent an empty string in the majority of programming languages, it is sometimes used to check for blank form fields. Because of this, many form fields will assume the field is blank, report an error, or crash, forcing him to use a different last name.

As developers, we can take a more inclusive strategy to working with names, treating these not as edge cases, but instead by expecting a wide variety of potential inputs.

## International Names

Names come in many different formats around the world; however, it is easy to apply our own cultural biases when designing systems that deal with names. As an American, for instance, my bias is to consider names in the format of a first name followed by a surname. Based on that format, I make several potentially false assumptions about things such as familial relationship. However, there are many different ways that a name can be constructed even within a single country or culture. Let's look at a few of these structures to see how they may challenge our assumptions.

### Multiple names

Many names may be longer than the "given name, family name" format. In many Spanish- and Portuguese-speaking countries, it is common to compose a name of one or two given names and two or three family names consisting of the mother's surname followed by the father's surname. In some cases, the conjunction *de* ("of") may be added between the maternal and paternal surnames, or sometimes surnames may reflect geographic origin.

Arabic names are traditionally much longer than given and family names, often having specific meaning. This description from Wikipedia highlights the false assumptions that a non-Arabic speaking person may make about the traditional Arabic name Abdul Rahman bin Omar al-Ahmad:

> With "Abdul": Arabic names may be written "Abdul (something)," but "Abdul" means "servant of the" and is not, by itself, a name. Thus for example, to address Abdul Rahman bin Omar al-Ahmad by his given name, one says "Abdul Rahman," not merely "Abdul". If he introduces himself as "Abdul Rahman" (which means "the servant of the Merciful"), one does not say "Mr. Rahman" (as "Rahman" is not a family name but part of his (theophoric) personal name); instead it would be Mr. al-Ahmad, the latter being the family name.

### Name order

Names do not always appear in the format of a given name followed by a family name, meaning that a typical form field of "First name" followed by "Last name," may not produce the intended results. As an example, Chinese names place the surname before the personal name.

Rather than a family surname, Icelandic names follow a patronymic (and, occasionally, matronymic) naming format. For example, if an Icelandic man named Birgir has a son named Jón, Jón's full name would be Jón Birgisson ("Birgir's son"). If Jón then had a daughter named Sigrún, Sigrún would be named Sigrún Jónsdóttir ("Jón's daughter"). Because of this, a list of Icelandic names would be expected to be sorted by given name rather than family name.

### Characters

Names from many regions may consist of characters outside of the Latin alphabet. There are those that may not make use of the Latin alphabet in written form, such as Arabic, Cyrillic, or Japanese (though many of these languages also have Romanized versions, such as the Japanese name Yamada Tarō (山田太郎)). There are also accented characters such as ó, ü, and ñ. Names may also contain a mix of ß. Names may contain non-letter characters such as apostrophes (e.g., the Irish name Francis O'Neill), which forms may attempt to strip during validation as unacceptable characters.

## Further Reading

These are only a few examples of how names may differ around the world. Additionally, they assume that a person's name is derived from a single culture, ignoring the possibility that name attributes from multiple cultures may be applied to a person's name. W3C's "Personal Names Around the World" dives into greater detail and links to several additional Wikipedia articles discussing naming formats.

## Mojibake

*Mojibake* is a term used to describe the garbled set of characters that are produced through an improper use of character encoding. Mojibake is typically caused by text that lacks proper (or any) Unicode encoding. Users whose names contain special characters may often see mojibake versions of their name. A quick image search for mojibake reveals many encoding issues across the Web, though it is likely that the majority go undocumented or are documented without knowing the term.

In his talk, ""Hello, my name is _____."", developer Nova Patch found several examples of mojibake affecting users of web services.

Perhaps the best-documented and consistent mojibake mangling of a name belongs to Nóirín Plunkett, who shared several instances of her mojibaked name on Twitter (see Figure 4-1).[1]


*Figure 4-1. Nóirín's tweets displaying mojibake in action*

1 The Tweets referenced in Figure 4-1 can be found at: *https://twitter.com/noirinp/status/77745010547769344*; *https://twitter.com/noirinp/status/151004631818977281*; *https://twitter.com/noirinp/status/162264316203114498*; *https://twitter.com/noirinp/status/394893223145271296*, and *https://twitter.com/noirinp/status/600750084410642432*.

### Nóirín Plunkett

Through the research of this book, I discovered that Nóirín Plunkett passed away in July 2015. Nóirín was an invaluable part of the open source community and an advocate for good in the world of software development. Both the Apache Foundation and Ada Initiative have offered heartfelt tributes to Nóirín.

Perhaps one of the more impressive mojibake instances was of a Russian postal worker who hand-corrected a package's mojibake (see Figure 4-2). This illustrates how common encoding problems can be when working with Cyrillic languages. In fact, there is even a Russian specific term for mojibake: *krakozyabri*.


*Figure 4-2. Hand-decoded mojibake by a Russian postal worker (image source unknown)*

## What Are We to Do?

Now that we've taken a quick look at the importance and value of names, we can consider how we can best implement name-inclusive

fields in our forms. We can do this by considering the format of the field itself and the way we handle the character encoding of the field.

### Input format

If possible, create name fields that are a single text input. Allow the input field to take in long names as well as accepting special characters and spaces. If possible, avoid limiting the length of the field in your database as well, so that an individual's name is never truncated when it is returned. See Figure 4-3.

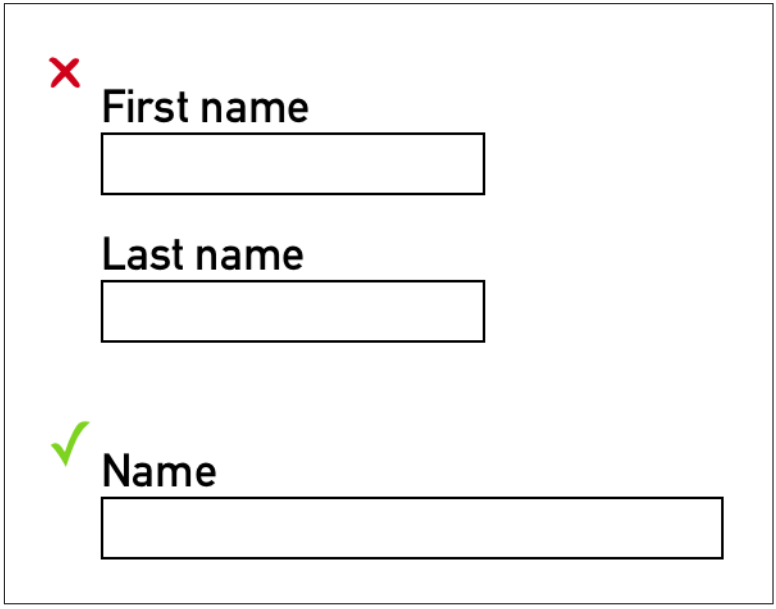


*Figure 4-3. If possible, use name fields that are a single text input*

If you plan to address the user through the web interface, email, or other means, it may be worth adding an additional field that asks "What should we call you?" (see Figure 4-4). This allows users to enter the name they most associate themselves with.

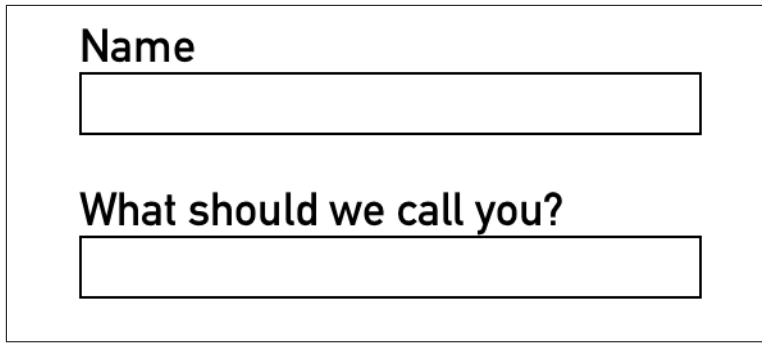


*Figure 4-4. If you will address the person, add a "What should we call you?" field*

### Character encoding

As we've seen with mojibake, character encoding can present its own unique set of challenges. To avoid the accidental mangling of names, we should permit punctuation (such as hyphens and apostrophes), allow spaces, and avoid changing character encoding formats between systems, such as form to database. A complete discussion of character encoding is beyond the scope of this book, but as a rule of thumb use UTF-8 encoding both on the front-end and the database.

In HTML, simply add the character set meta tag specifying UTF-8 to the `<head>` of the page:

```
<meta charset="utf-8">
```

## Further Reading

- Nova Patch's "Hello, my name Is _____."
- Patrick McKenzie's "Falsehoods Programmers Believe About Names"
- W3C's "Personal Names Around the World"
- Aimee Gonzalez-Cameron's "Hello, My Name Is <error>"
- Sara Wachter-Boettcher's "Personal Histories"
- Joel Spolsky's "The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!)"
- Dr. Florian Deißenböck's "No Such Thing as Plain Text"
- Pavel Radzivilovsky, Yakov Galka, and Slava Novgorodov's "UTF–8 Everywhere"

- W3C's "Multilingual Form Encoding"

## Inclusive Gender

For many, gender is not simply the binary sex of either male or female as determined at birth. The advocacy group GLAAD defines transgender as:

> An umbrella term (adj.) for people whose gender identity and/or gender expression differs from the sex they were assigned at birth. The term may include but is not limited to: transsexuals, cross-dressers and other gender-variant people.

The most cited study on transgender population numbers in the United States places the transgender population at 0.3%, or roughly 700,000 adults in the United States. According to Monica Chalabi, the author of the report and the FiveThirtyEight article "Why We Don't Know the Size of the Transgender Population", these numbers may be inaccurate, tending toward low, due to the lack of non-binary gender options on official forms such as the census as well as a reluctance to provide the information when asked.

To be as inclusive as possible, we can build systems that accept and respect non-binary gender options. When including gender in a form, my recommendation is to:

- Provide male and female options
- Provide a "custom" text input; if data collection is important, you may provide autocomplete suggestions, but still allow custom inputs
- Offer a "prefer not to say" option

Both Facebook and Google follow patterns similar to those.

Google offers the choices of "Male," "Female," "Decline to state," and "Custom" in a select menu (see Figure 4-5).

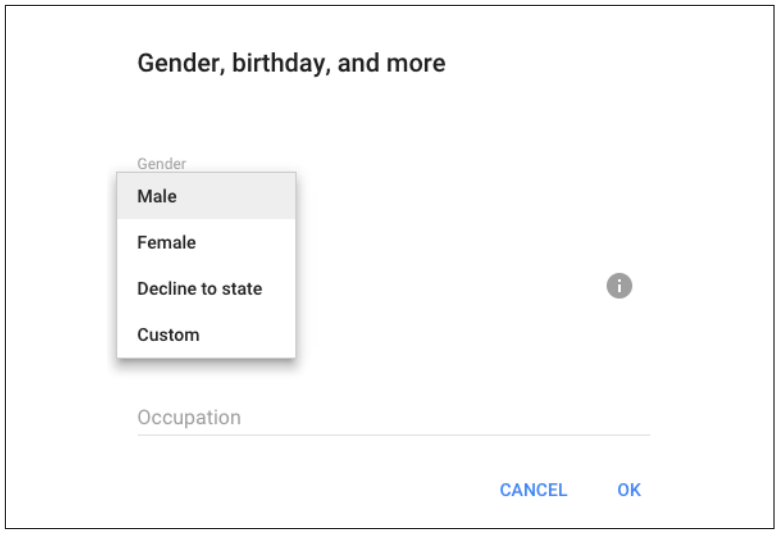


*Figure 4-5. Google offers four gender choices in a select menu*

If the "Custom" option is selected, the user is presented with a text input box and a choice of pronoun to be addressed by (see Figure 4-6).

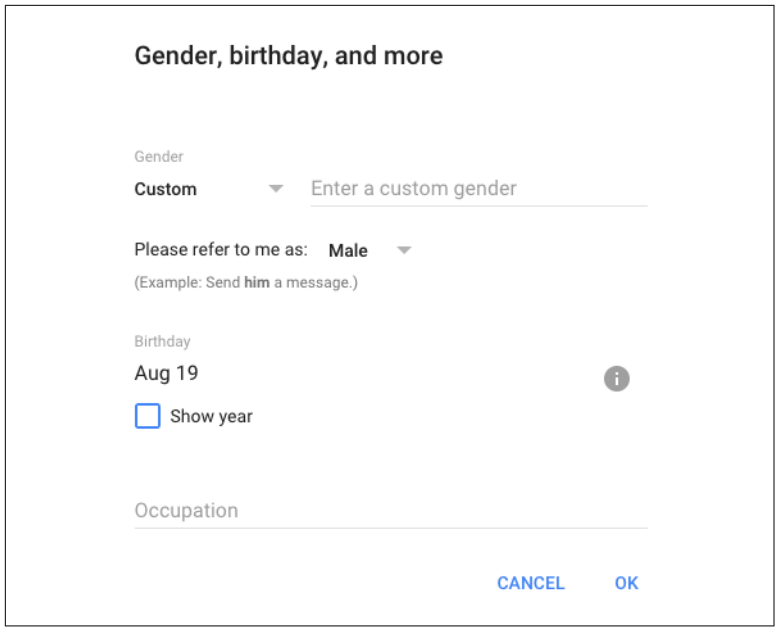


*Figure 4-6. A selection of "Custom" displays a text input and choice of pronoun*

By contrast, Facebook requires a binary gender choice during account creation (see Figure 4-7).

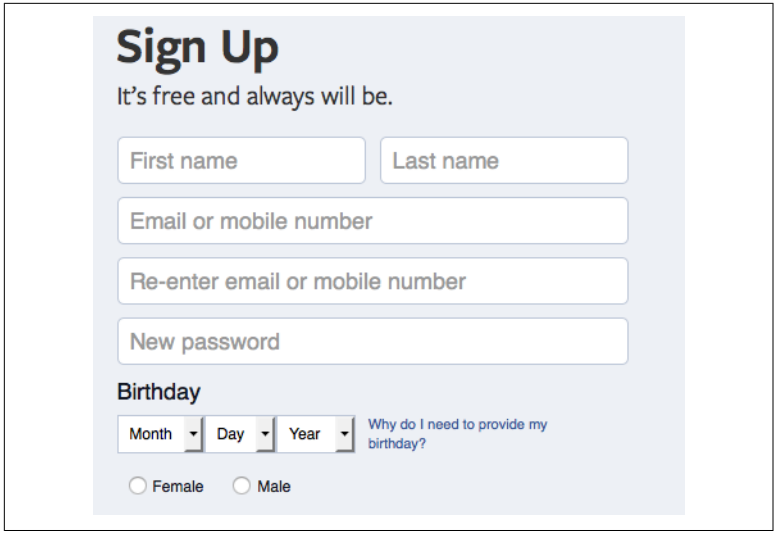


*Figure 4-7. Facebook's sign-up form presents users with a binary gender choice*

However, once a user has created a Facebook account, it's possible to select a more inclusive gender. Facebook's pattern offers three choices: "Male," "Female," and "Custom." When "Custom" is selected, users are given a text input box with autocomplete suggestions as well as a selection of pronouns to be addressed by (see Figure 4-8).


*Figure 4-8. Facebook allows a user to select a custom gender, offers autocomplete gender suggestions, and provides a choice of pronouns*

## What About Titles?

Forms may often include a title field, with gendered choices such as "Mr.," "Ms.," and "Mrs." Not requiring these fields or providing a text input option gives users the most control over this option. By doing so, we allow those who prefer not to use a title to do so as well as those with a non-binary gender to not be forced into using a gendered title.

## Further Reading

- Claire Gowler's "How to Ask About Gender"
- CUSU LGBT+'s "Think Outside the Box Recommendations for Forms"
- Formulate Information Design's "Sex and Gender"

# In Summary

When we ask users to complete a form with personal information, we are asking about their personal identity. By considering name formats, internationalization, and gender we provide online spaces that are welcoming and inclusive to all.

# CHAPTER 5
# Conclusion

Thank you for taking the time to read *Building Web Apps for Everyone*. I hope that through reading this report you see value in using progressive enhancement, considering accessibility, and building inclusive web forms. These encompass a small portion of the work we can do as web developers to ensure that the Web is an open and inclusive space for all users. My hope is that you now feel empowered and excited to build applications in this way.

If throughout your reading you have come across things that are missing or could be improved, I would encourage you to contribute back to the book. This title is available as open source and contributions can be made by:

- Contributing directly to the report's GitHub repository with a pull request.
- Creating an issue in the book's GitHub repository.
- Reaching out to me through *email* or via Twitter.

Twenty percent of the proceeds from each *Ethical Web Development* title will be donated to an organization whose work has a positive impact on the issues described. For this title, I will be donating to the World Wide Web Consortium (W3C). The W3C works to ensure "that the Web remains open, accessible and interoperable for everyone around the world" and authored much of the content that was used to research this title.

If you are interested in supporting the W3C's work, consider making a donation at *w3.org/support*.

This is the first in a series of digital reports I am authoring on the subject of ethical web development. Future titles in the series will cover building web applications that work everywhere, building web applications that respect a user's privacy and security, and working with development peers. You can learn more about the series at the Ethical Web Development website.

## About the Author

Adam D. Scott is a developer and educator based in Connecticut. He currently works as a Senior Technology Fellow at the Consumer Financial Protection Bureau, where he focuses on building open source tools. Additionally, he has worked in education for over a decade, teaching and writing curriculum on a range of technical topics. His first book, *WordPress for Education*, was published in 2012. His video course, *Introduction to Modern Front-End Development*, was published by O'Reilly in 2015.

## Technical Reviewer

Chris Contolini is an open source software developer. He is currently a Senior Technology Fellow at the Consumer Financial Protection Bureau, where he focuses on ChatOps and front-end web development. He lives and works from a 10-foot box truck retrofitted with solar panels, running water, and broadband Internet access. He works mostly from national forests and has been known to frequent the Bay Area and Portland, OR.

## Other Contributors

The following people have graciously contributed feedback and improvements:

- Meg Foley
- Andy Chosak
- Shashank Khandelwal

Contributions and suggestions have also been made to the Ethical Web Development website, as well as the detailed principles described there. Those contributions are stored at *ethicalweb.org/humans.txt*.