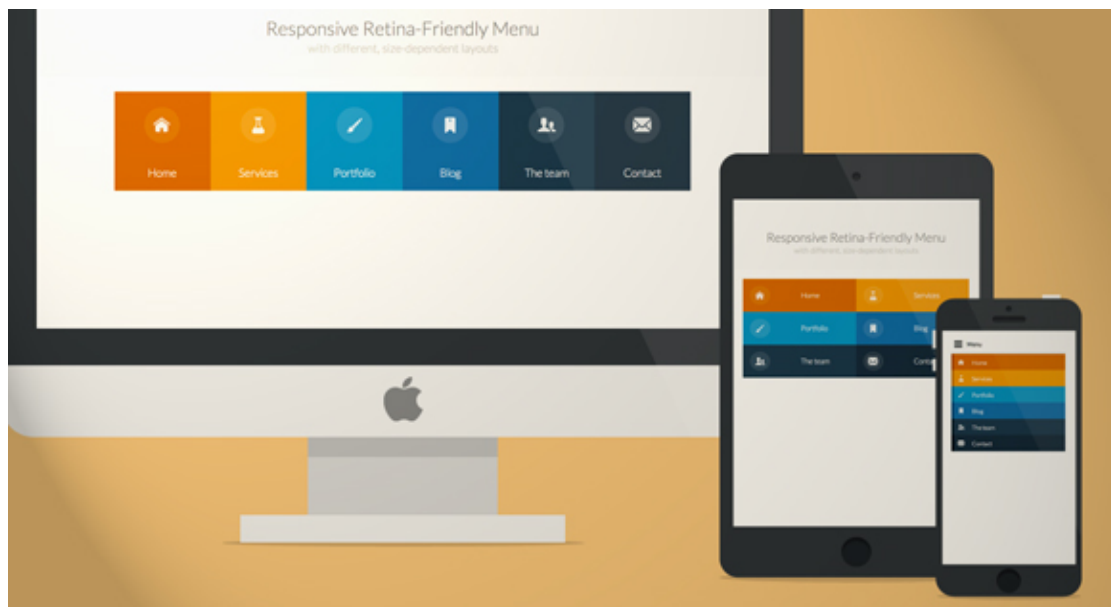




# Responsive Retina-Ready Menu

A responsive, touch-friendly and Retina-ready menu with three layouts for different browser sizes.


By [Stéphanie Walter](#) in [Tutorials](#) on May 8, 2013



Codrops uses cookies for its advertisement solutions and for analytics. We hope you're ok with this, but you can opt-out if you wish. Read our [Privacy Policy](#) and [Cookie Policy](#). [OK](#)

**From our sponsor:** Grow with Mailchimp's All-in-One Marketing Platform

Today we will create a colorful Retina-ready and responsive menu inspired by the colors of the Maliwan manufacturer of the Borderlands game. The menu automatically changes to one of three different layouts depending on the browser window size: a “desktop” inline version, a two columns tablet-optimized version and a mobile version with a menu link to display and hide the navigation for smaller screens. To make the menu fully retina-ready, we will use an icon font so that the icons of the menu won’t get pixelized on resize.

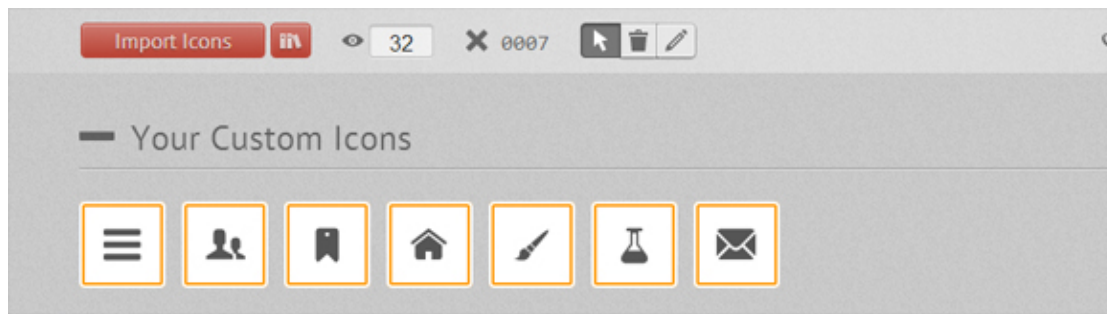
 Please note: some effects only work in browsers that support the respective CSS properties.

## Preparing the icon font

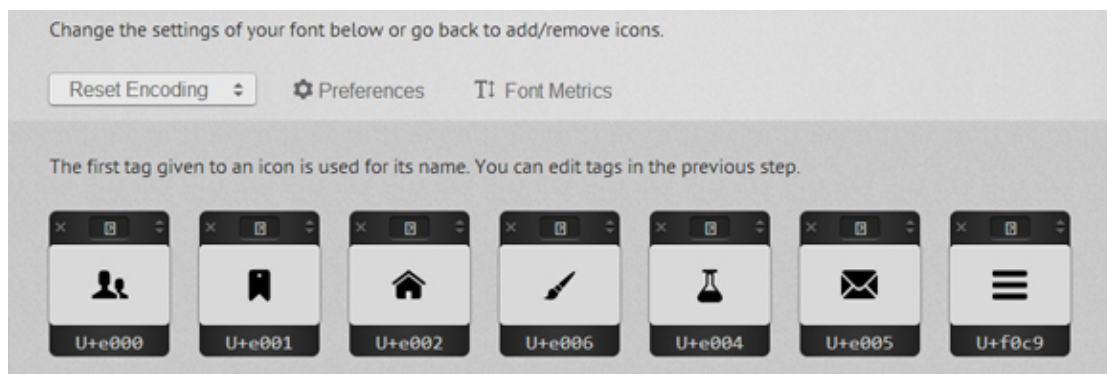
Creating a custom icon font might look a bit complicated, but with tools like [IcoMoon](#) it’s just a matter of creating the icons and importing them into the tool. Icon fonts behave like any font, so you can easily change the color, adapt the size and it won’t get pixelized. Perfect for retina devices without having to use multiple assets for different screen resolutions.

The first thing we need to do is to create the icons for the menu. I use Illustrator, but any vector graphics editor like, for example Inkscape, will do. We need to create each icon and export them as a SVG file. To make sure the icon will work properly in every browser, we have to convert all lines into full objects, and merge all the objects into one big shape for each icon. Once all have been exported into nice SVG files, we can import them all into the IcoMoon App tool:

Codrops uses cookies for its advertisement solutions and for analytics. We hope you're ok with this, but you can opt-out if you wish. Read our [Privacy Policy](#) and [Cookie Policy](#). [OK](#)



We can also enhance our font with icons from the big library that IcoMoon offers. Once we have all the icons we need, we click on the “Font” button at the bottom of the page to enter the detailed settings. On this page we can choose the encoding settings for the font, and also choose if we want to assign some letters for each icon, or prefer to use the Private Use Area of the font to make sure screen readers won’t be able to output them. I recommend using the default settings that work pretty well.



When we click on “Download” we get a ZIP file with 4 font formats (SVG, EOT, TTF and WOFF), the CSS styling and a demo page.

The first thing to do to be able to use the icons is to copy and paste the CSS IcoMoon provides to the top of our CSS file and make sure we also copy the font folder. There’s also [a little “hack” to make the fonts look nicer on Chrome Windows](#) you might want to check it out.

Here is what the HTML for our navigation looks like:

```
<nav id="menu" class="nav">
  <ul>
    <li>
      <a href="#" title="">
        <span class="icon"> <i aria-hidden="true" class="icon-home">
      </a>
    </li>
    <li>
      <a href="#" title=""><span class="icon"> <i aria-hidden="true" c
    </li>
    <li>
      <a href="#" title=""><span class="icon"><i aria-hidden="true"
    </li>
    <li>
      <a href="#" title=""><span class="icon"><i aria-hidden="true"
    </li>
    <li>
      <a href="#" title=""><span class="icon"><i aria-hidden="true"
    </li>
    <li>
      <a href="#" title=""><span class="icon"><i aria-hidden="true"
    </li>
  </ul>
</nav>
```

To use the icon font, we simply use the class “icon-iconname” inside an `i` element (a `span` will work as well). Also note that we added a `no-js` class to the body that will be changed to `js` with [Modernizr](#). The idea is to be able to leave the menu open if the user has JavaScript disabled. We’ll also use

## The CSS & JavaScript

Note that I won't prefix the CSS3 properties here but you will find the prefixed version in the files.

The global CSS that gets applied to all screen sizes looks as follows:

```
/* Global CSS that are applied for all screen sizes */

.nav ul {
  max-width: 1240px;
  margin: 0;
  padding: 0;
  list-style: none;
  font-size: 1.5em;
  font-weight: 300;
}

.nav li span {
  display: block;
}

.nav a {
  display: block;
  color: rgba(249, 249, 249, .9);
  text-decoration: none;
  transition: color .5s, background .5s, height .5s;
}

.nav i{
  /* Make the font smoother for Chrome */
  transform: translate3d(0, 0, 0);
}

/* Remove the blue Webkit background when element is tapped */
```

```
}
```

We want a first little transition on the whole navigation that lowers the opacity of all the items, except the one that is being hovered. This is the code for that:

```
/* Hover effect for the whole navigation to make the hovered item stand out */  
  
.no-touch .nav ul:hover a {  
  color: rgba(249, 249, 249, .5);  
}  
  
.no-touch .nav ul:hover a:hover {  
  color: rgba(249, 249, 249, 0.99);  
}
```

Then we want to add some nice background colors to all of the items. The code below uses a nth-child technique to select the list items. This way, you can add as many list items as you want, the color code will repeat itself.

```
.nav li:nth-child(6n+1) {  
  background: rgb(208, 101, 3);  
}  
  
.nav li:nth-child(6n+2) {  
  background: rgb(233, 147, 26);  
}
```

```
.nav li:nth-child(6n+4) {  
  background: rgb(22, 107, 162);  
}  
  
.nav li:nth-child(6n+5) {  
  background: rgb(27, 54, 71);  
}  
  
.nav li:nth-child(6n+6) {  
  background: rgb(21, 40, 54);  
}
```

Using a min-width media query, we can target screens that are bigger than 800px (50em, with a body font size of 15px) to transform our list into a nice horizontal navigation:

```
@media (min-width: 50em) {  
  
  /* Transforms the list into a horizontal navigation */  
  .nav li {  
    float: left;  
    width: 16.66666666666667%;  
    text-align: center;  
    transition: border .5s;  
  }  
  
  .nav a {  
    display: block;  
    width: auto;  
  }  
}
```

focus and active to make it work on touch devices and on keyboard access.

*/\* hover, focused and active effects that add a little colored border*

```
.no-touch .nav li:nth-child(6n+1) a:hover,  
.no-touch .nav li:nth-child(6n+1) a:active,  
.no-touch .nav li:nth-child(6n+1) a:focus {  
  border-bottom: 4px solid rgb(174, 78, 1);  
}  
  
.no-touch .nav li:nth-child(6n+2) a:hover,  
.no-touch .nav li:nth-child(6n+2) a:active,  
.no-touch .nav li:nth-child(6n+2) a:focus {  
  border-bottom: 4px solid rgb(191, 117, 20);  
}  
  
.no-touch .nav li:nth-child(6n+3) a:hover,  
.no-touch .nav li:nth-child(6n+3) a:active,  
.no-touch .nav li:nth-child(6n+3) a:focus {  
  border-bottom: 4px solid rgb(12, 110, 149);  
}  
  
.no-touch .nav li:nth-child(6n+4) a:hover,  
.no-touch .nav li:nth-child(6n+4) a:active,  
.no-touch .nav li:nth-child(6n+4) a:focus {  
  border-bottom: 4px solid rgb(10, 75, 117);  
}  
  
.no-touch .nav li:nth-child(6n+5) a:hover,  
.no-touch .nav li:nth-child(6n+5) a:active,  
.no-touch .nav li:nth-child(6n+5) a:focus {  
  border-bottom: 4px solid rgb(16, 34, 44);  
}  
  
.no-touch .nav li:nth-child(6n+6) a:hover,  
.no-touch .nav li:nth-child(6n+6) a:active,
```



Then we place the icons and the text:

```
/* Placing the icon */

.icon {
  padding-top: 1.4em;
}

.icon + span {
  margin-top: 2.1em;
  transition: margin .5s;
}
```

We animate the height of the elements when they are hovered:

```
/* Animating the height of the element*/

.nav a {
  height: 9em;
}

.no-touch .nav a:hover ,
.no-touch .nav a:active ,
.no-touch .nav a:focus {
  height: 10em;
}

/* Making the text follow the height animation */
.no-touch .nav a:hover .icon + span {
```

Then we position the icons and prepare them for the CSS transition:

```
/* Positioning the icons and preparing for the animation*/
.nav i {
  position: relative;
  display: inline-block;
  margin: 0 auto;
  padding: 0.4em;
  border-radius: 50%;
  font-size: 1.8em;
  box-shadow: 0 0 0 0.8em transparent;
  background: rgba(255,255,255,0.1);
  transform: translate3d(0, 0, 0);
  transition: box-shadow .6s ease-in-out;
}
```

To give the visual effect we want, we transition a box shadow and change its size from 0.8em to 0, and its color from transparent to some color with a high opacity. This is also where we close our first media-query.

```
/* Animate the box-shadow to create the effect */
.no-touch .nav a:hover i,
.no-touch .nav a:active i,
.no-touch .nav a:focus i {
  box-shadow: 0 0 0px 0px rgba(255,255,255,0.2);
  transition: box-shadow .4s ease-in-out;
}
```

We set a second media query to make some little adjustments for screens between 800 and 980px:

```
@media (min-width: 50em) and (max-width: 61.250em) {  
  
    /* Size and font adjustments to make it fit better */  
    .nav ul {  
        font-size: 1.2em;  
    }  
  
}
```

Now that we have finished the “desktop” version (with BIG quotation mark since more and more tablets now have 1024px and larger screens), we take care of the “global” CSS for screens that are smaller than 800px which equals to 49.938em here, using a max-width media query.

```
/* The "tablet" and "mobile" version */  
  
@media (max-width: 49.938em) {  
  
    /* Instead of adding a border, we transition the background color */  
    .no-touch .nav ul li:nth-child(6n+1) a:hover,  
    .no-touch .nav ul li:nth-child(6n+1) a:active,  
    .no-touch .nav ul li:nth-child(6n+1) a:focus {  
        background: rgb(227, 119, 20);  
    }  
  
    .no-touch .nav li:nth-child(6n+2) a:hover,  
    .no-touch .nav li:nth-child(6n+2) a:active,
```

```
.no-touch .nav li:nth-child(6n+3) a:hover,  
.no-touch .nav li:nth-child(6n+3) a:active,  
.no-touch .nav li:nth-child(6n+3) a:focus {  
  background: rgb(44, 168, 219);  
}  
  
.no-touch .nav li:nth-child(6n+4) a:hover,  
.no-touch .nav li:nth-child(6n+4) a:active,  
.no-touch .nav li:nth-child(6n+4) a:focus {  
  background: rgb(31, 120, 176);  
}  
  
.no-touch .nav li:nth-child(6n+5) a:hover,  
.no-touch .nav li:nth-child(6n+5) a:active,  
.no-touch .nav li:nth-child(6n+5) a:focus {  
  background: rgb(39, 70, 90);  
}  
  
.no-touch .nav li:nth-child(6n+6) a:hover,  
.no-touch .nav li:nth-child(6n+6) a:active,  
.no-touch .nav li:nth-child(6n+6) a:focus {  
  background: rgb(32, 54, 68);  
}  
  
.nav ul li {  
  transition: background 0.5s;  
}  
  
}
```

For screen size between 520px (32.5em) and 799px (49.938em), we want to display our menu into a 2 columns and 3 rows layout. We add some padding to make the elements easy to “tap”, and display the icons on the left and the text on the right.

```
/* CSS for a 2x3 columns version */

@media (min-width: 32.5em) and (max-width: 49.938em) {

/* Creating the 2 column layout using floating elements once again */
.nav li {
    display: block;
    float: left;
    width: 50%;
}

/* Adding some padding to make the elements look nicer*/
.nav a {
    padding: 0.8em;
}

/* Displaying the icons on the left, and the text on the right side */
.nav li span,
.nav li span.icon {
    display: inline-block;
}

.nav li span.icon {
    width: 50%;
}

.nav li .icon + span {
    font-size: 1em;
}

.icon + span {
    position: relative;
    top: -0.2em;
}
```

This is where we close our media query.

```
/* Adapting the icons to animate the size and border of the rounded li */
.nav li i {
  display: inline-block;
  padding: 8% 9%;
  border: 4px solid transparent;
  border-radius: 50%;
  font-size: 1.5em;
  background: rgba(255,255,255,0.1);
  transition: border .5s;
}

/* Transition effect on the border color */
.no-touch .nav li:hover i,
.no-touch .nav li:active i,
.no-touch .nav li:focus i {
  border: 4px solid rgba(255,255,255,0.1);
}
}
```

Again, for smaller screens we adapt the font-size and width.

```
/* Adapting the font size and width for smaller screens*/
@media (min-width: 32.5em) and (max-width: 38.688em) {

  .nav li span.icon {
    width: 50%;
  }
}
```

```
}  
}
```

For very small screens, we hide the navigation and display a “menu” button the user can click if he wants to display the navigation. To do this, we rely on some lines of JavaScript:

```
// The function to change the class  
var changeClass = function (r,className1,className2) {  
    var regex = new RegExp("(?:^|\\s+)" + className1 + "(?:\\s+|$)");  
    if( regex.test(r.className) ) {  
        r.className = r.className.replace(regex,' '+className2+' ');  
    }  
    else{  
        r.className = r.className.replace(new RegExp("(?:^|\\s+)" + className1 + "(?:\\s+|$)"), '');  
    }  
    return r.className;  
};  
  
// Creating our button for smaller screens  
var menuElements = document.getElementById('menu');  
menuElements.insertAdjacentHTML('afterBegin','<button type="button" id="menu-toggle">Menu</button>');  
  
// Toggle the class on click to show / hide the menu  
document.getElementById('menutoggle').onclick = function() {  
    changeClass(this, 'navtoogle active', 'navtoogle');  
}  
  
// document click to hide the menu  
// http://tympanus.net/codrops/2013/05/08/responsive-retina-ready-menu/  
document.onclick = function(e) {  
    var mobileButton = document.getElementById('menutoggle'),  
        menu = document.getElementById('menu');  
    if (mobileButton && menu && !mobileButton.contains(e.target) && !menu.contains(e.target)) {  
        changeClass(menu, 'navtoogle active', 'navtoogle');  
    }  
}
```

```
}  
}
```

In order to have a cleaner HTML, I chose to create the “menu” button and insert it in the DOM using JavaScript. The function `changeClass` helps us to toggle the class between `active` and `no class` when the users clicks on the button.

Now that we have all we need for the small screen version, we can style it with CSS. The following code styles the menu button:

```
/* Styling the toggle menu link and hiding it */  
.nav .navtoogle{  
  display: none;  
  width: 100%;  
  padding: 0.5em 0.5em 0.8em;  
  font-family: 'Lato', Calibri, Arial, sans-serif;  
  font-weight: normal;  
  text-align: left;  
  color: rgb(7, 16, 15);  
  font-size: 1.2em;  
  background: none;  
  border: none;  
  border-bottom: 4px solid rgb(221, 221, 221);  
  cursor: pointer;  
}  
  
.navtoogle i{  
  z-index: -1;  
}  
  
icon-menu f
```



```
font-size: 1.6em;
}
```

By default, the menu button is hidden. We want to display it for screens smaller than 519px (32.438em):

```
@media (max-width: 32.438em) {

  /* Unhiding the styled menu link */
  .nav .navtoogle{
    margin: 0;
    display: block;
  }
}
```

We animate the height of the navigation when the button is clicked. To close the navigation, we give it a 0em height, to open it, we give it a 30em max-height. If JavaScript is not enabled, we don't have any button, so we use the no-js class to always display the navigation.

```
/* Animating the height of the navigation when the button is clicked

/* If JavaScript is disabled, the menu stays open */
.no-js .nav ul {
  max-height: 30em;
  overflow: hidden;
}
```

When JavaScript is enabled we hide the menu by default, and display it when the users clicks on the button which then gets the active class:

```
/* When JavaScript is enabled, we hide the menu */
.js .nav ul {
  max-height: 0em;
  overflow: hidden;
}

/* Displaying the menu when the user has clicked on the button */
.js .nav .active + ul {
  max-height: 30em;
  overflow: hidden;
  transition: max-height .4s;
}
```

We adapt the layout for smaller screens, presenting the navigation in a list of items with the icon on the left and the text on the right side:

```
/* Adapting the layout of the menu for smaller screens: icon on the

.nav li span {
  display: inline-block;
  height: 100%;
}

.nav a {
  padding: 0.5em;
}
```

```
font-size: 0.8em;
}
```

We also add a 8px border on the left of each item with a nice color

```
/* Adding a left border of 8 px with a different color for each menu
.nav li:nth-child(6n+1) {
  border-left: 8px solid rgb(174, 78, 1);
}

.nav li:nth-child(6n+2) {
  border-left: 8px solid rgb(191, 117, 20);
}

.nav li:nth-child(6n+3) {
  border-left: 8px solid rgb(13, 111, 150);
}

.nav li:nth-child(6n+4) {
  border-left: 8px solid rgb(10, 75, 117);
}

.nav li:nth-child(6n+5) {
  border-left: 8px solid rgb(16, 34, 44);
}

.nav li:nth-child(6n+6) {
  border-left: 8px solid rgb(9, 18, 25);
}
```

the touch capability of the device. If the device has touch capabilities, a touch class is added to the body. We can use this class to enhance the experience on touch devices and make the navigation items a little bit bigger so that the user can tap them more easily. And here we close our last media query.

```
/* make the nav bigger on touch screens */  
.touch .nav a {  
  padding: 0.8em;  
}  
}
```

And that's it, we've build a nice, touch-friendly and retina-ready navigation that works fine on desktop, tablet and mobile devices. Hope you liked it!

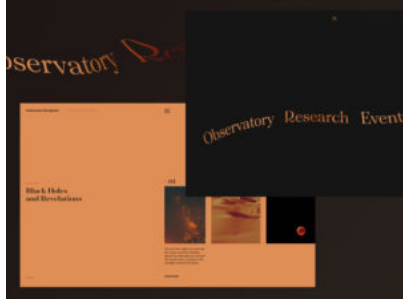
*Image Credits: Featured image created with [Minimal Apple Product Templates](#) from [WeGraphics.net](#)*



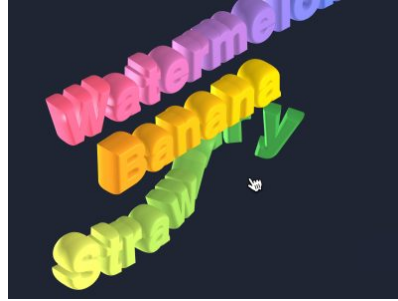
## Stéphanie Walter

I'm a Graphic & Web Designer, Pixel & CSS lover, WordPress & coffee addict. I also love UI-UX design for mobile and web apps. I like to experiment with new technologies, and share the knowledge.

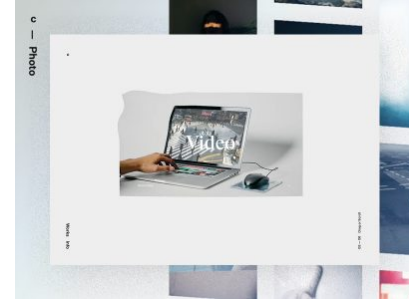
<http://www.inpixeltrust.fr>



## How to Build an Underwater-Style Navigation Using PixiJS



## Building a Physics-based 3D Menu with Cannon.js and Three.js



## Case Study: Chang Liu Portfolio V4