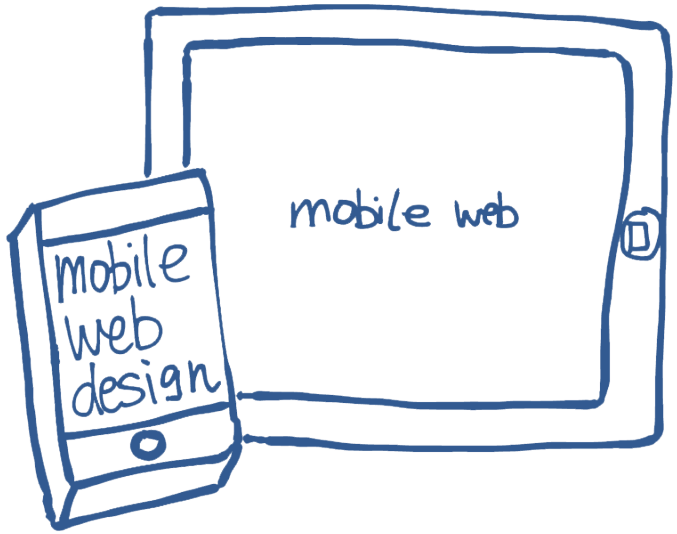
Mobile Web Design

Hi there, this is Thomas (aka. Makzan on the web). I am an author on web technology and game design. This is a course about making website that works in mobile devices.

Overview



In this course, we will explore different techniques to create our content in mobile web. We will focus on **content strategy** for mobile devices. We will try to fit our **layout** into small screen. We will make the **form inputs** fit the virtual keyboards in touch devices. We will fetch device data such as raw **touches events** and **device orientation**. Finally, we will pack and deploy our web.

Course Style

In my course, I focus on the concept rather that listing all the specs here. It's because you can find the official spec from Mozilla or W3C. My value here is to show how things are designed and why it is designed like that. It's about the concept that you apply in your practical work.

I used to use examples to during the learning journey. You may find the code example repository on this Github repo.

In case you have any questions, you may reach me via online social networks or send me an email.

- @makzan
- mak@makzan.net

Version history

This is a work-in-progress of upgrading the book content into 2nd edition.

- 2019-03-13: Re-import 1st edition content.
- 2019-03-16: Updating Chapter 3 Zurb Foundation.
- 2019-03-23: Moved the SCSS pre-processor into appendix.

Chapter 1 — Getting started

Let's get started!

Setting up the development environment

How can we get started?

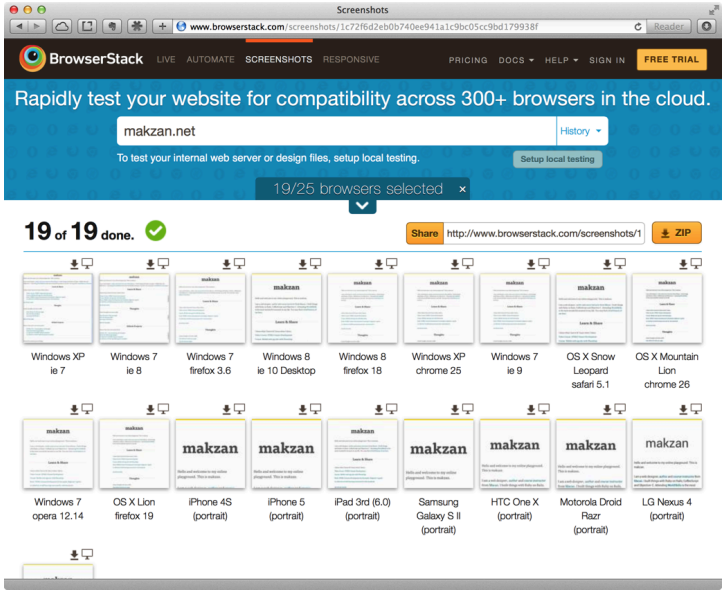
We need to install some softwares. We need **web browsers** to run the web. We'll need *Google Chrome*, *Mozilla Firefox* and *Apple Safari* to test our web pages.

Just the browsers? How about the editor?

Any **plain text code editor** would work. IDE such as Dreamweaver or WebStorm may be used but they are not required. For me, I would use Codepen to demonstrate the code in class.

Testing environment

Some services, such as Browser Stack, provide easy webpage testing across multiple browsers. For example, we can test how our web look on different sizes of devices.



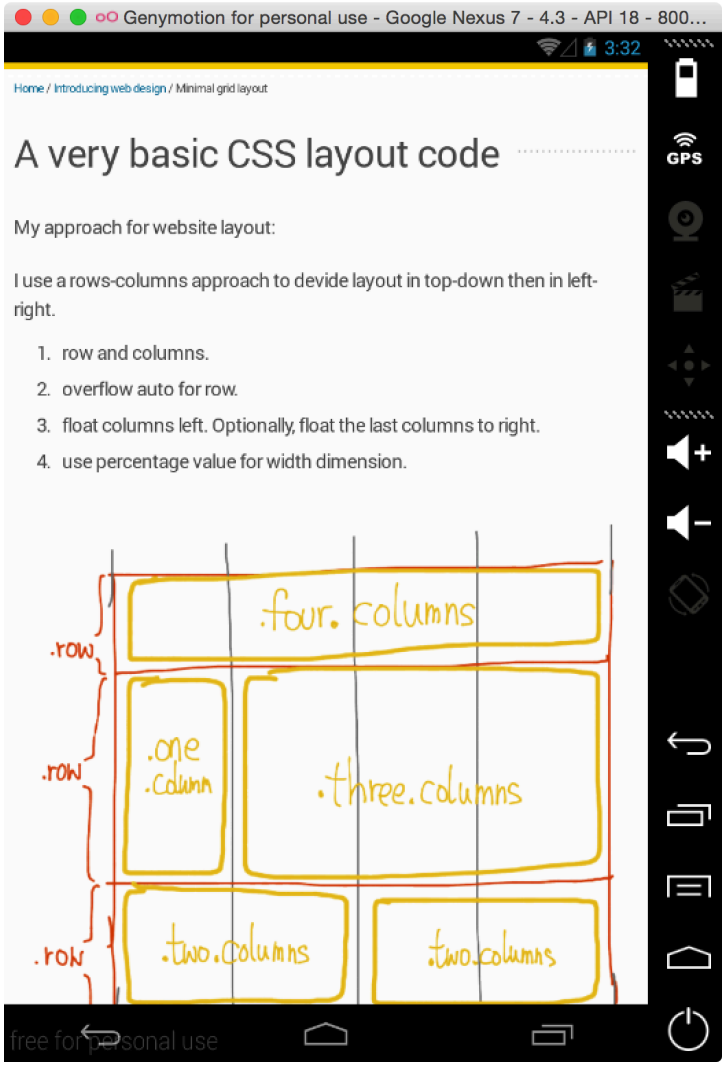
Testing in simulator

We will also need to install emulator for both iOS and android system.

The iOS simulator can be installed via the Xcode, the apple development toolkit.

The android emulator, on the other hand, can be installed via the Android SDK.

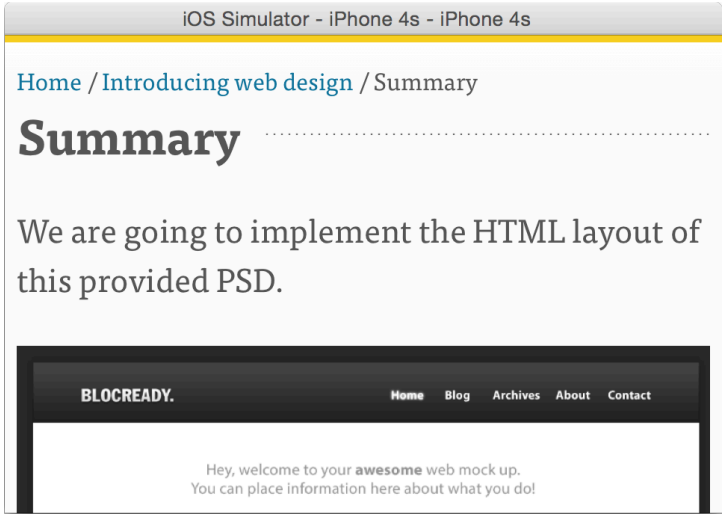
Alternatively, 3rd party emulator may be used, such as genymotion.



Genymotion simulation

iOS Simulator

We need macOS with Xcode (Free from Mac App Store) to run the iOS Simulator.



iOS simulator

The iOS for iPhone runs on ARM architecture CPU. But the iOS Simulator runs iOS that's designed to run directly on Intel CPU architecture. That's why the performance on iOS Simulator is smooth.

In order to open website in iOS simulator, we can simply drag an URL from the macOS into the simulator.

If you want to test the orientation of the simulator, you can press command+left or command+right shortcut to rotate the

simulator.

There are different device simulators that we can choose from. We can test the smallest screen and the largest screen.

Testing in real devices

Beside simulating our web in the mobile simulator, we need to have real devices to further test it.

Not only because simulator cannot simulate all the things, but also because holding the web on hand is important. We need the feeling of scrolling the screen, tapping on links and buttons, and inputting the form value with the virtual keyboard.

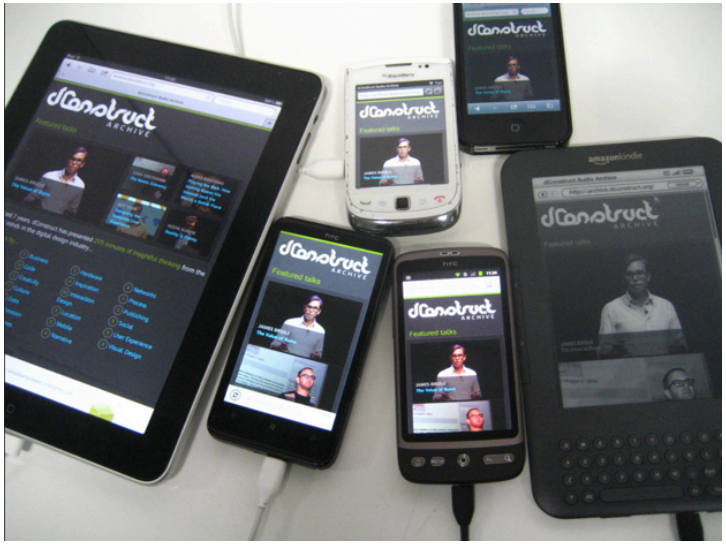


Photo Credit: Jeremy Keith

Where to reference documentation?

Googling the topic on the Internet may be fast to get some understands on specific topic. But if you are seeking documentation, here are some suggestions to high quality documentation.

- Mozilla developer network (Add “mdn” as keyword when searching on web)
- W3C

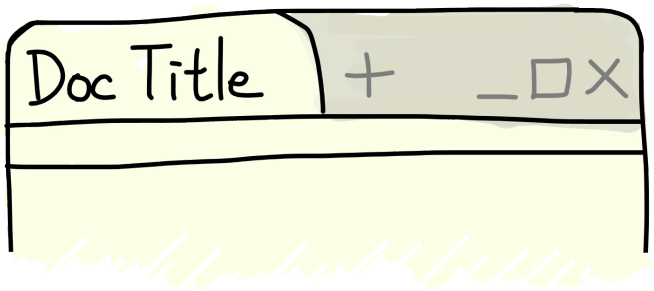
Some desktop applications may provide quick access to these documentations. For example, in my Mac, I use Dash to have quick access to the spec without opening the browser.

What’s next? We’re going to take a look at content strategy.

Chapter 2 – Content First

In this chapter, we focus on the fundamental of the web. It's so fundamental that these techniques apply to both desktop web and mobile web design. It's so fundamental that we often overlook and forget about it.

Document title



First, we take a look at the title of the document. Title of the document is used in the following places:

- Home screen
- Bookmark
- History log
- Tab
- Back button in browser
- Social network

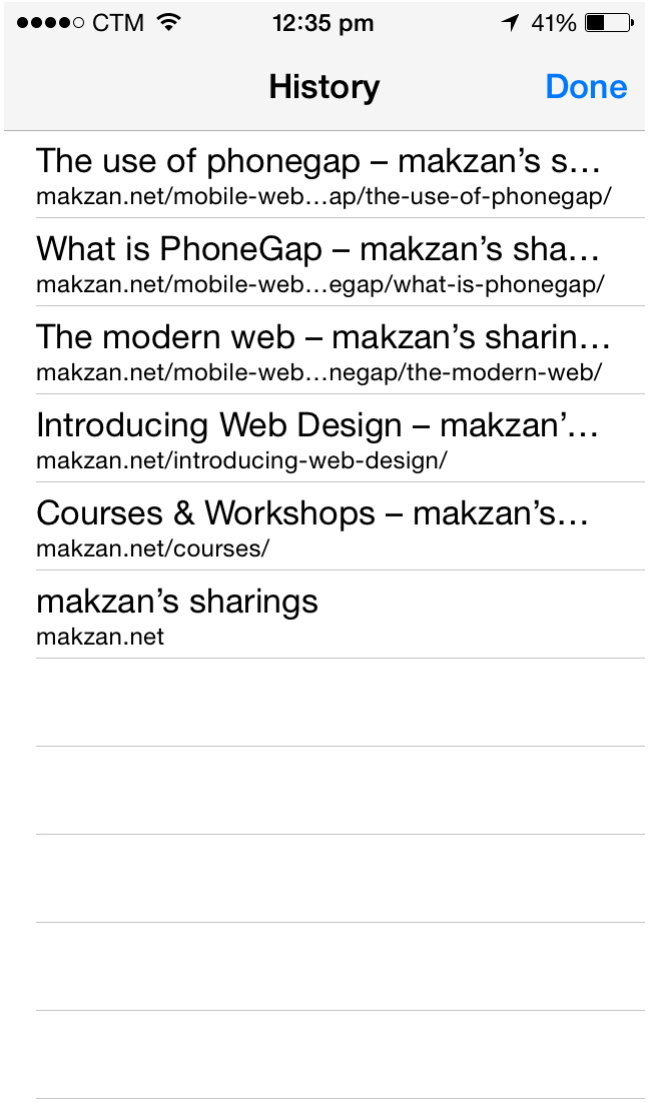
The following is the screen of sharing a webpage on Facebook. Facebook fetches the document title when it is shared.

Document title appears in the facebook sharing box

The following screenshot shows how confuse it is when every page shares the same document title.

Same document title on every page

The following screenshot, on the other hand, shows the website with corresponding title set for each page.



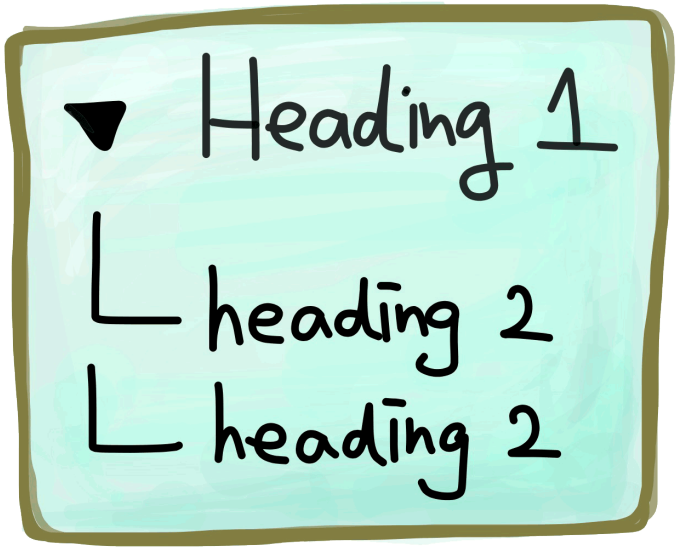
Different document title for each page

Content strategy

Why we should care about content first?

Because no matter how the web and app present our content, the content itself doesn't change. So the first thing we should focus is the content structure.

By structure, I meant the important order of our content. I meant the correct headings for our content. The informative HTML tags we applied to our content.



The next most important thing after document title is our headings.

Headings define the document outline. It shows the architecture of the essay.

Correct use of headings helps both readers and software process the content.

Our main heading `h1` may be used in many places. It could be the title of our essay that's displayed in:

- Email newsletter
- Read later service

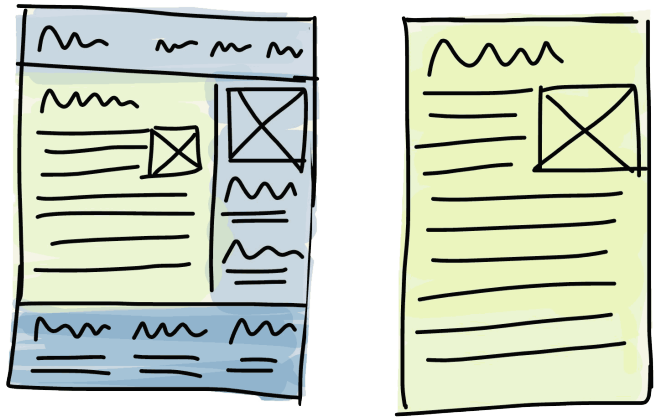
For detail about how we should use the headings, we may check the reference to the W3C spec about the headings.

Exercise

Now try to make use of the HTML5 tags to write a news web page. You may create or copy dummy content. We focus only on the structure.

Try to write the core content first. Then add the author information. Then the category of this essay. Then add a website header and footer. Then add a navigation list. Then add a side bar to display related content. After adding all these things, can the reading tool still extract your essay content?

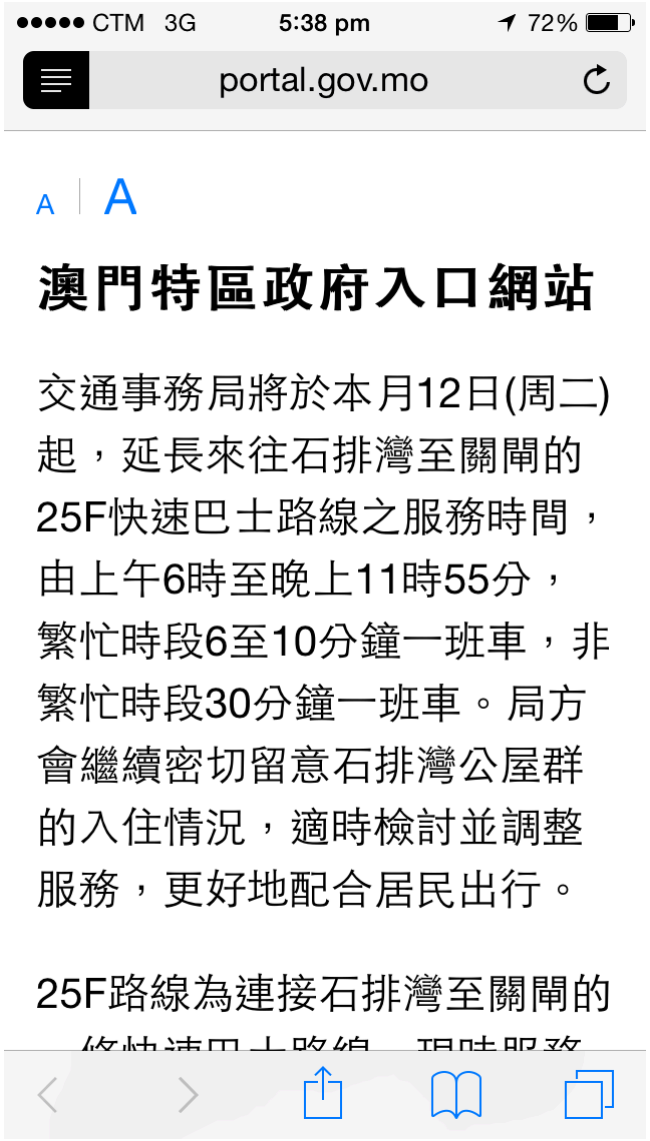
Content structure



The first step is not make your website fit in mobile. The first thing you should do is make your content fits in the mobile device.

Remember, people don't consume website. They consume content.

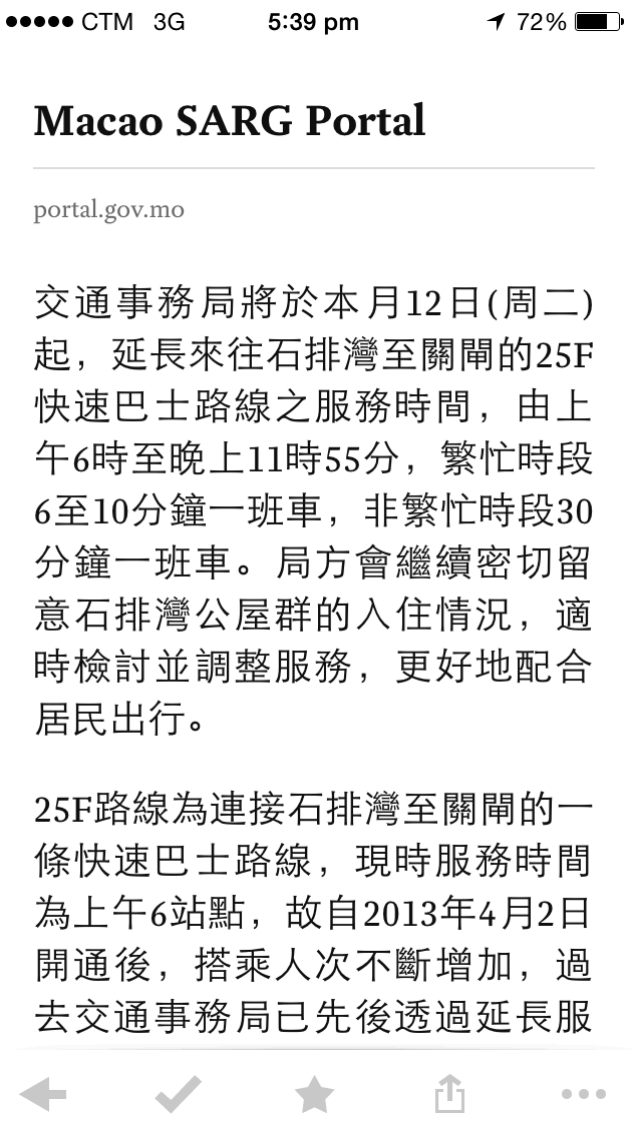
There is an article view in mobile Safari. The iOS parsed the web page and try to extract the content for a better reading experience.



The same article displayed by the browser's

article-only view.

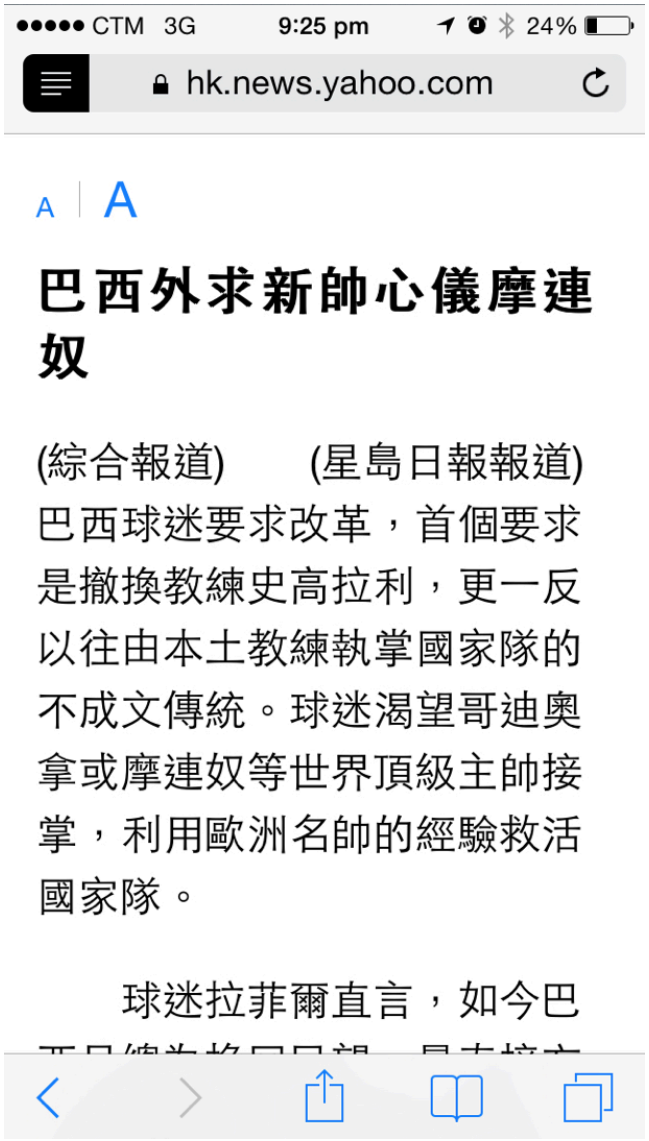
Moreover, most read-later service also supports content extraction. For example, the following screenshot shows how Pocket displays the content in its mobile app, with a good reading experience.



The same article being displayed in a read-later

service.

Another example:



The same article displayed by the browser's article-only view.

Similar article view is not only use in mobile but also desktop service, such as Evernote.

Article being extracted by Evernote.

So how we could make the system or service extract the content correctly?

When we plan our HTML structure, we can make use of the `<article>` tag. Here is the explanation from Mozilla.

> The HTML `<article>` Element represents a self-contained composition in a document, page, application, or site, which is intended to be independently distributable or reusable, e.g., in syndication.

It's often confusing when deciding when to use the section tag and when to use the article tag.

Good structure helps web service analysis your content.

Two quotes from the w3 spec helping us to choose between section and article.

> Authors are encouraged to use the article element instead of the section element when it would make sense to syndicate the contents of the element.
>
> When an element is needed for styling purposes or as a convenience for scripting, authors are encouraged to use the div element instead. A general rule is that the section element is appropriate only if the element's contents would be listed explicitly in the document's outline.

Mobile first approach

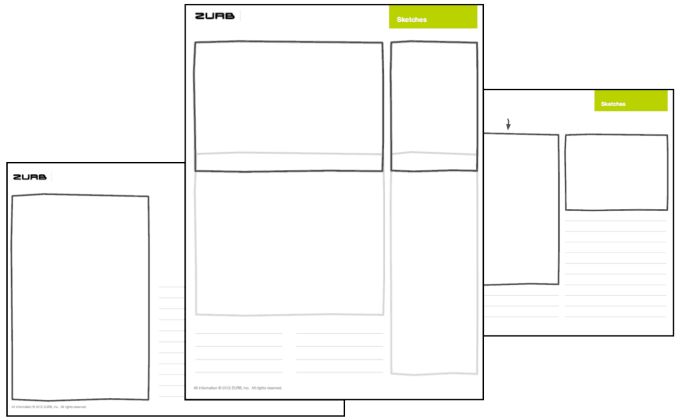
We talk about mobile first. Mobile first means that during our website planning, we plan the content and layout for the mobile first.

Planning for mobile first ensures us to consider the most important thing of our website.

Sketching for mobile web

For sketching the layout of the mobile web, we can use a draft paper with narrow boundary box. Zurb's sketchsheet is one of my favorites.

Zurb Sketch sheet is a collection of website sketching toolkit. It contains different kinds of sketch sheets that we can use.



For example, this one contains both wide and narrow boundary that we can draw our website layout in both wide and small screen.

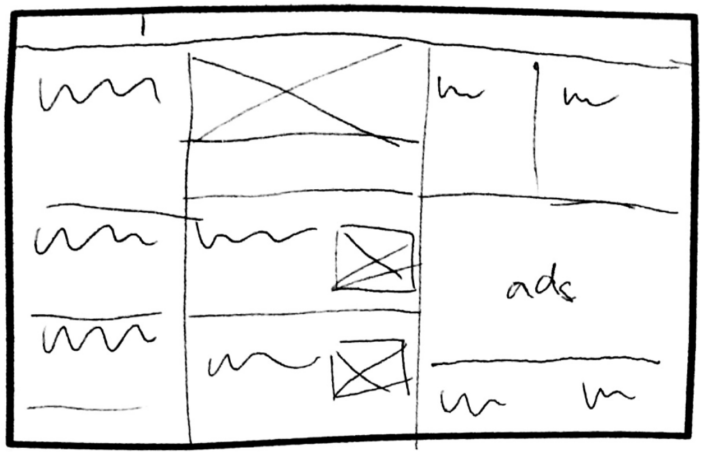
Exercise time

Now, browse to the the Macao Gov Portal. Follow the website layout and draw a rough layout sketch in the wide screen on the left of the sketch sheet.

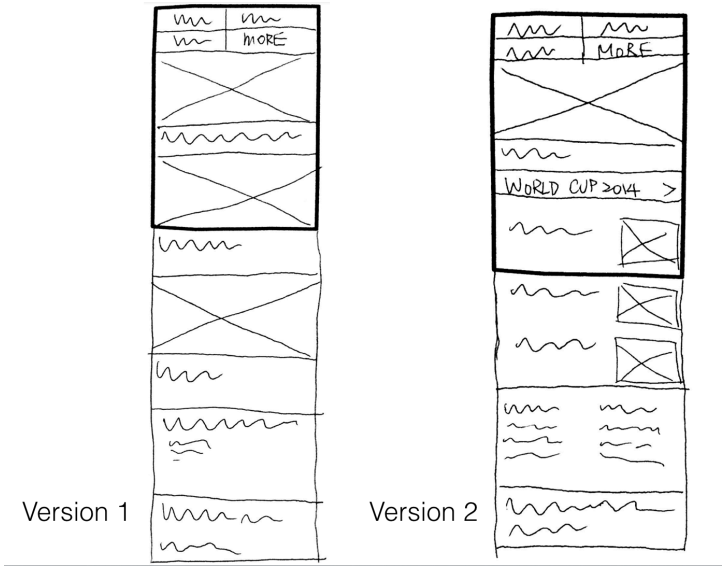
What I want you to do is then think how you would layout the content in the narrow screen. Try to draw your design on the right side.

Here is a sketching example on NYtimes.com.

NYtimes.com screenshot



NYtimes.com in wide-screen sketching.



Version 1 Version 2

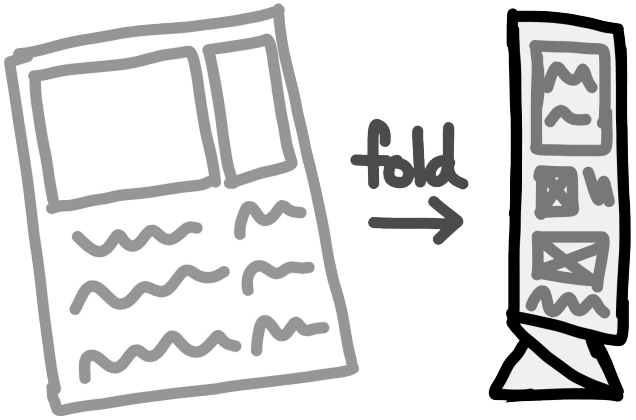
NYtimes.com in small-screen sketching.

Now it's your turn to work on the sketching of [Macao Gov Portal].

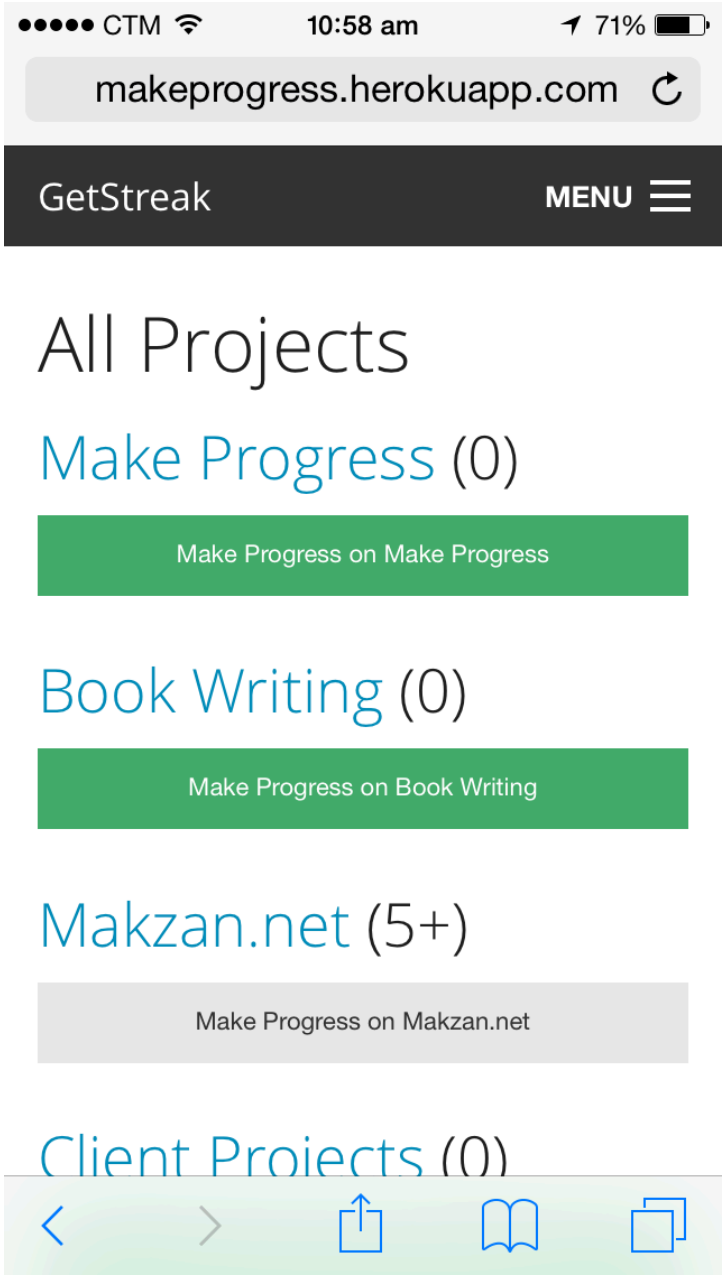
You may take a photo of your design and send me your design via email.

Remember the mobile first approach? To maximise the usage of this sketch sheet, you'll actually need to draw the right side first. Then you draw on the left side only after you've done your mobile first design.

Creating your own mobile first sketch sheet



What's more? You can fold the the sketch sheet into 1/3 of the original one. Then hold it on your hand and feel your sketch. Are your buttons easy to tap? Are content too intense to read clearly?



A screenshot showing how mobile-friendly web makes use of large buttons and list view.

Exercise time

It's time for some exercises. Now use your mobile device and explore some websites. Try to check if these websites provide a nice reading experience. Or can the system or service extract the content of the website.

Minimal mobile friendly website

The following code example shows how we can create a minimal website style that fits in small screen reading.

The HTML:

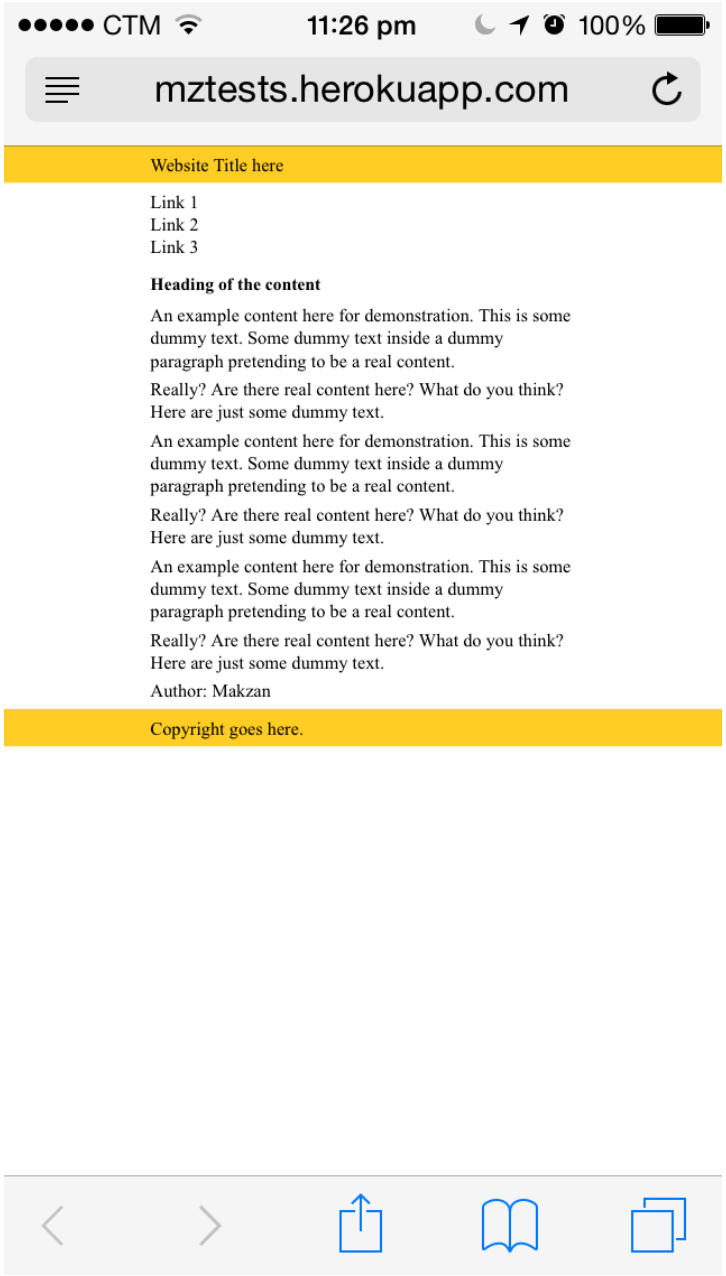
```
<header>
  <div class="row">
    Website Title here
  </div>
</header>

<nav>
  <div class="row">
    <ul>
      <li>Link 1</li>
      <li>Link 2</li>
      <li>Link 3</li>
    </ul>
  </div>
</nav>

<div class="row">
  <article>
    <header>
      <h1>Heading of the content</h1>
    </header>

    <p>Content paragraphs go here.</p>

    <footer>
      Author: Makzan
    </footer>

  </article>
</div>

<footer>
  <div class="row">
    Copyright goes here.
  </div>
</footer>
```

The CSS:

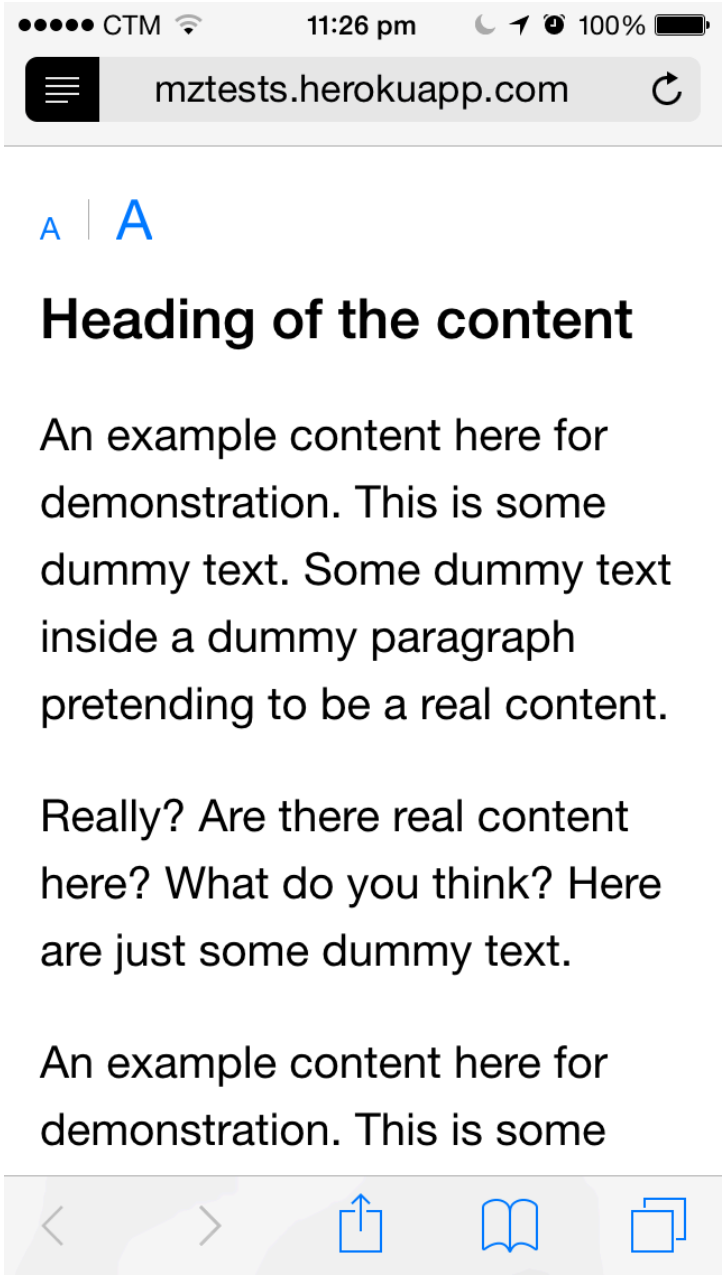
```
/* normalize */
body, nav, ul, li, p, h1, h2, h3 {
    padding: 0;
    margin: 0;
}

/* core styles */
* {
```

```
    -webkit-box-sizing: border-box;
    box-sizing: border-box;
}

body > header,
body > footer {
    background: #ffce25;
}

.row {
    width: 100%;
    max-width: 600px;
    margin: 0 auto;
    padding: 10px;
}

/* addition styles */
ul {
    list-style: none;
}

h1, p {
    margin-bottom: .5em;
}
```

Please find the following screenshot showing the minimal web site display in both normal view and article view in mobile Safari.



Website showing in iPhone screen.



Website showing in iPhone article view.

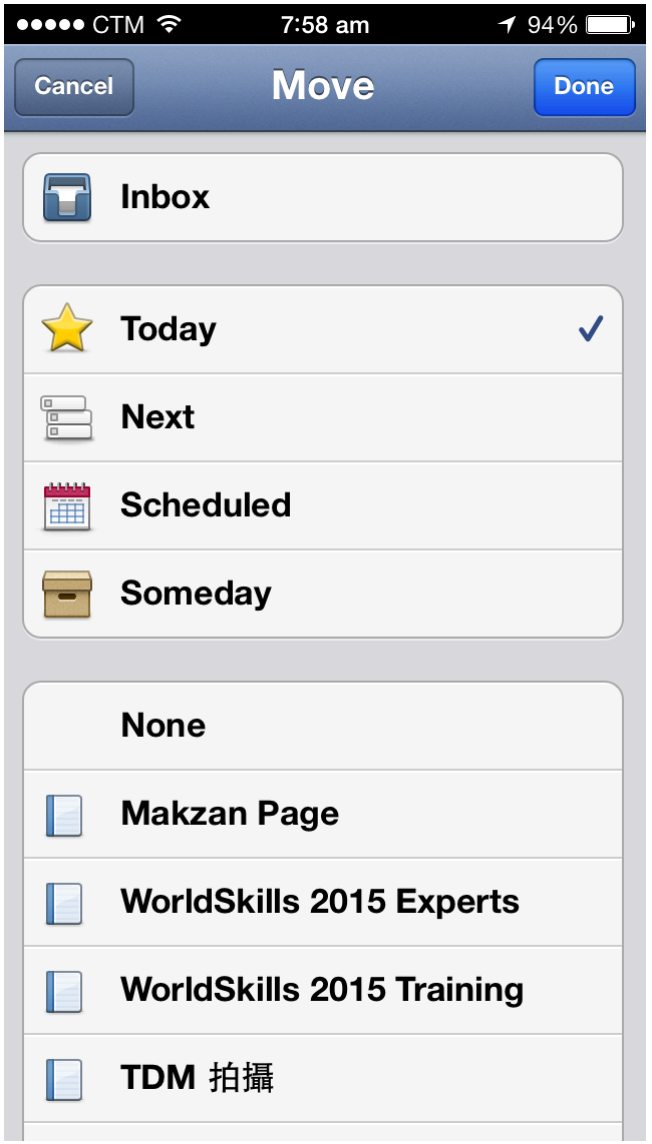
Note: We used `box-sizing` in our CSS. You may reference to the Mozilla documentation.

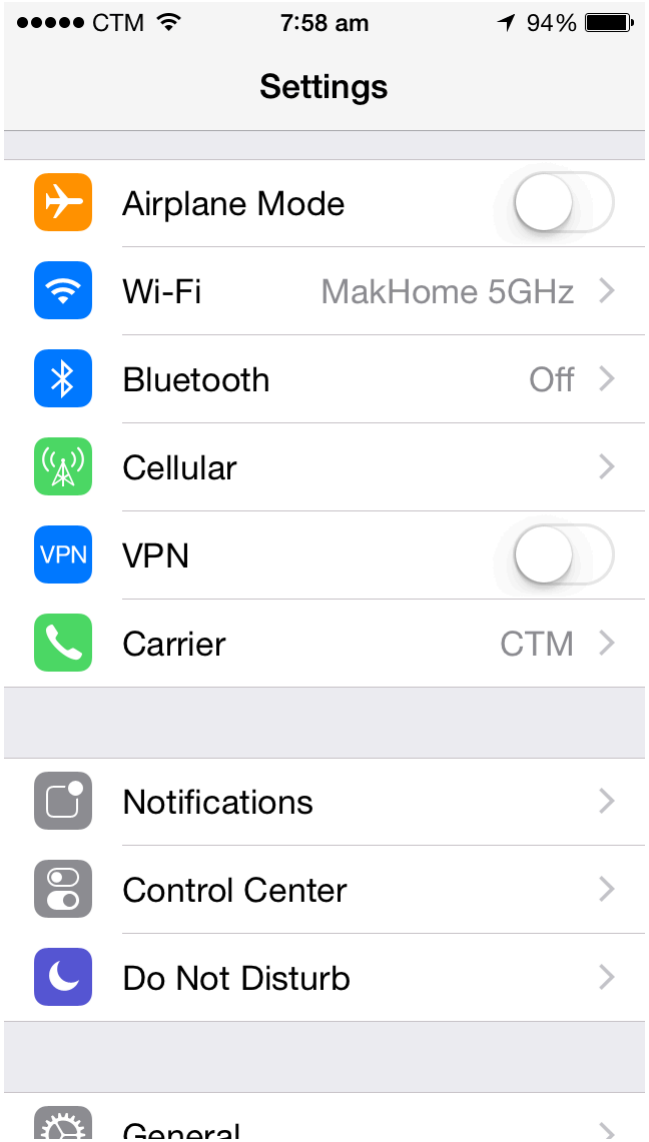
Listview

Mobile devices are often in narrow size and most of the users hold the device in portrait. So when we provides information or options for mobile device, we may use list view.

In iOS, it's called table view.

Here are some screenshots showing how iOS uses table view to display both information and configuration.

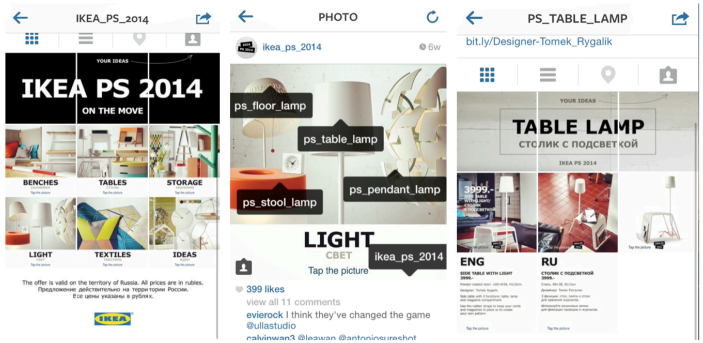



●●●●● CTM
7:58 am
94%
Settings
Airplane Mode
Wi-Fi
MakHome 5GHz
Bluetooth
Off
Cellular
VPN
VPN
Carrier
CTM
Notifications
Control Center
Do Not Disturb
General


●●●●○ CTM
7:59 am
93%
Back
Buy More Storage
Buy
CURRENT PLAN
20GB
HK$ 300.00 / year
25 GB total iCloud storage
CHOOSE AN UPGRADE...
You will be charged immediately and each year until you change or cancel your plan.
50GB
HK$ 750.00 / year
55 GB total iCloud storage
Downgrade Options
Contact Apple for a full refund within 15 days of an upgrade or within 45 days after a yearly payment. Partial refunds are available where required by law.

Do mobile web has to be a webpage?

Content is what matters here. Has the content be put in web page? Take a look at how Ikea turns instagram profile into a clickable “webpage” with description, images and movie clips.



Ikea puts a ‘website’ on instagram.

What’s next? We’re going to take a look at “Chapter 3 – Using Zurb foundation”.

Chapter 3 – Using Zurb foundation

In this chapter, we focus on using a CSS framework to build a full-functional mobile web site.

Assignment

Now try to make use of the Zurb Foundation framework and apply it to a web project. If you don't have any active web project, try to work on the government website.

Trying Zurb Foundation on Codepen

If you want to experiment Zurb Foundation in Codepen, you'll need to include the CSS and JavaScript. The fiddle should also be selected to use jQuery where Foundation depends on it.

```
<!-- Compressed CSS -->
<link rel="stylesheet" href="https://cdn.jsdelivr.net/
npm/foundation-sites@6.5.3/dist/css/foundation.min.css">

<!-- Compressed JavaScript -->
<script src="https://cdn.jsdelivr.net/npm/foundation-
sites@6.5.3/dist/js/foundation.min.js"></script>

<!-- foundation-float.min.css: Compressed CSS with
legacy Float Grid -->
<link rel="stylesheet" href="https://cdn.jsdelivr.net/
npm/foundation-sites@6.5.3/dist/css/foundation-
float.min.css">
```

(Code from https://foundation.zurb.com/sites/docs/installation.html)

```
<!doctype html>
<html class="no-js" lang="en">
  <head>
    <meta charset="utf-8" />
    <meta http-equiv="x-ua-compatible" content="ie=edge">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0" />
    <title>Foundation Starter Template</title>
    <link rel="stylesheet" href="css/foundation.css" />
  </head>
  <body>
    <h1>Hello, world!</h1>

    <script src="js/vendor/jquery.js"></script>
    <script src="js/vendor/what-input.js"></script>
    <script src="js/vendor/foundation.min.js"></script>
    <script>
      $(document).foundation();
    </script>

  </body>
</html>
```

Or you may start in the following codepen where I have included the Foundation resources.

https://codepen.io/makzan/pen/OqZyBm

Setting up Foundation

Foundation is a library with both css and javascript code.

If we don't use any components that required javascript, we can omit the javascript part and just include the css.

In this chapter, we will use the full library.

1. Go to the Foundation website.
2. Choose "Download Foundation"
3. Then choose "Download Everything"

If you need to customize the layout or the framework variable, you can customize the variables in their download page before downloading. The website will compile the custom build for us.

Later in the preprocessing chapter, we will learn the option to customize the framework locally in the development machine.

Here you can find the list of the files we get and their purposes from Foundation package.

This is the suggested HTML setup by Zurb.

```
<!DOCTYPE html>
<!--[if IE 9]><html class="lt-ie10" lang="en" >
<![endif]-->
<html class="no-js" lang="en">

<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>Foundation 5</title>

  <!-- You may add app.css to use for your overrides if
you like -->
  <link rel="stylesheet" href="css/normalize.css">
  <link rel="stylesheet" href="css/foundation.css">

  <script src="js/vendor/modernizr.js"></script>

</head>
<body>

  <!-- body content here -->

  <script src="js/vendor/jquery.js"></script>
  <script src="js/foundation.min.js"></script>
  <script>
    $(document).foundation();
  </script>
</body>
</html>
```

If we don't need the modernizr, we can use the following:

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>Foundation 5</title>

  <!-- You may add app.css to use for your overrides if
you like -->
  <link rel="stylesheet" href="css/normalize.css">
  <link rel="stylesheet" href="css/foundation.css">

</head>
<body>

  <!-- body content here -->

  <script src="js/vendor/jquery.js"></script>
  <script src="js/foundation.min.js"></script>
  <script>
    $(document).foundation();
  </script>
</body>
</html>
```

If we just need the CSS part and not using any of the JavaScript depended component, we can further trim the code into the following:

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>Foundation 5</title>

  <!-- You may add app.css to use for your overrides if
you like -->
  <link rel="stylesheet" href="css/normalize.css">
  <link rel="stylesheet" href="css/foundation.css">

</head>
<body>

  <!-- body content here -->

</body>
</html>
```

Foundation grid

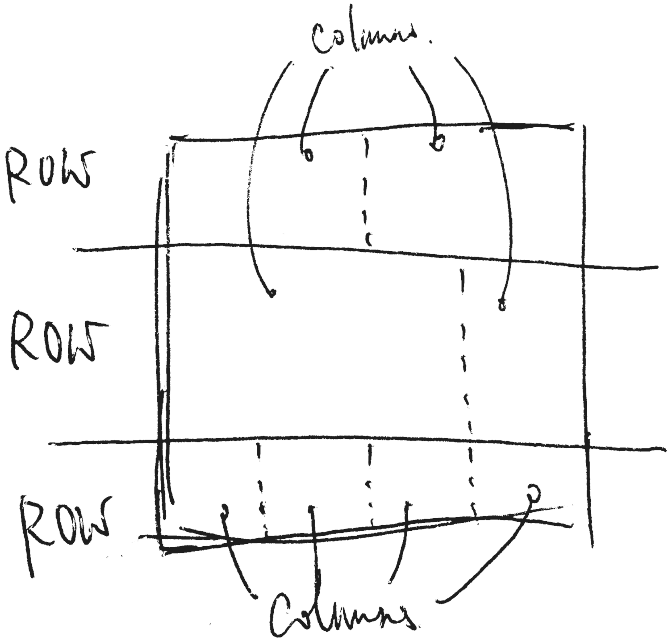
Foundation uses row and columns approach for the layout.

Rows are the space lay out from top to bottom.

Columns are the space inside each row, lay out fro left to right.

If the content are meant to be inside the grid system, make

sure we put the content inside column and columns inside row.



The columns order from left to right following the order of the divs by default. we may change the order by using push and pull classes where we will discuss very soon.

Here is a demo for the basic grid.

```
<div class='row'>
  <div class='small-3 columns'>Left Nav Bar</div>
  <div class='small-9 columns'>Content</div>
</div>
```

Left Nav Bar	Content

small grid expands to wide screen.

Left Nav Bar	Content

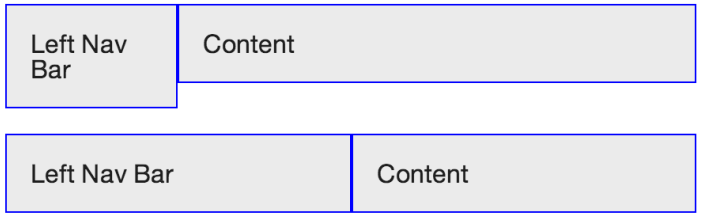
small grid for small screen.

After we add the medium columns setting, the proportional varies between small and medium screen.

```
<div class='row'>
  <div class='small-6 medium-3 columns'>Left Nav Bar</div>
  <div class='small-6 medium-9 columns'>Content</div>
</div>
```

Left Nav Bar	Content
Left Nav Bar	Content

small grid expands to wide screen.



small grid for small screen.

Foundation is mobile first. Zurb Foundation uses `small-N`, `medium-N` and `large-N` to define how many columns span for **small**, **medium** and **large** screen. But only the small one is required, medium and large one is *optional*.

When designing the grid in Zurb Foundation, we craft the grid for small screen first.

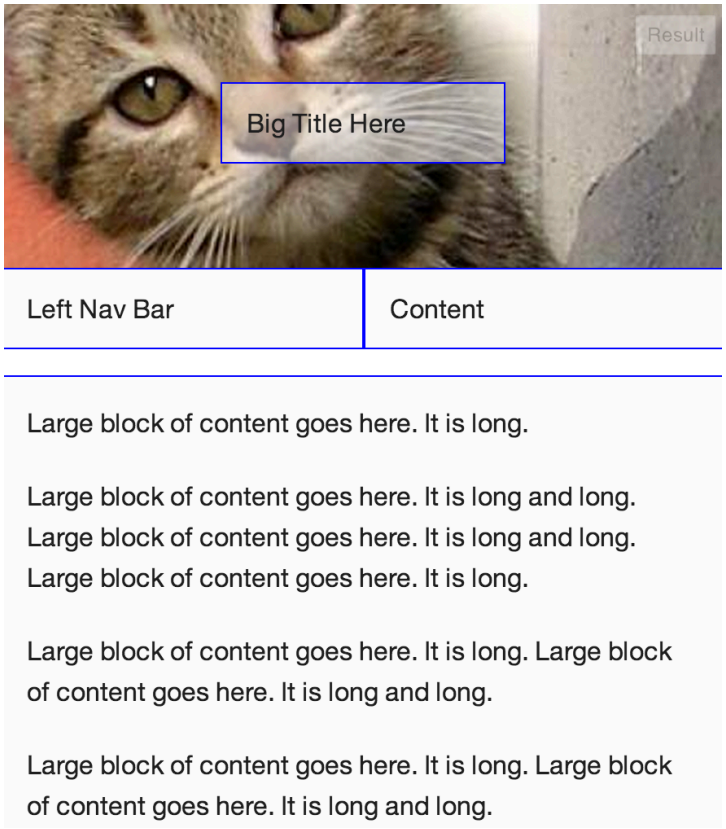
If the grid for small screen work in larger size (medium and large), we don't need to do anything and it will expand to the larger width until reaching the max width.

If we think we should divide the row into different columns when the screen is at medium or large size, we can further use the `medium-N` and `large-N` column class to define them.

We can even provide different proportion for the various screen sizes.

Centered column

We can center a column by using the small-centered class.



A demo on centered column.

As same as the grid, there is medium-centered and large-centered class. We don't need the medium and large because they will inherit the small-centered class if we hasn't

specify.

If we want to un-center a column in specific screen size, we can use the medium-uncentered and large-uncentered.

HTML

```
<div class='header-bg'>
  <div class='row'>
    <div class='small-6 small-centered columns'>Big
Title Here</div>
  </div>
</div>
```

CSS

```
.header-bg {
  background: url(http://placekitten.com/1000/500)
center center;
  background-size: cover;
  background-attachment: fixed;
  padding: 3em;
}

.columns {
  background: rgba(255, 255, 255, .5);
  border: 1px solid blue;
  padding-top: 1em;
  padding-bottom: 1em;
  margin-bottom: 1em;
}
```

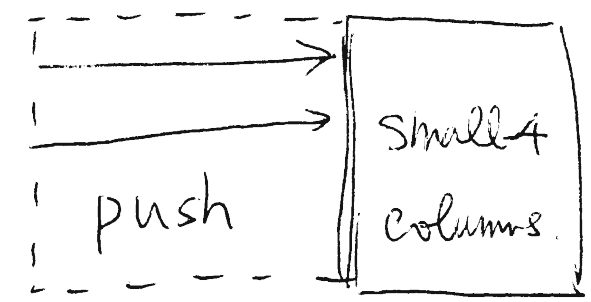
Push and pull column

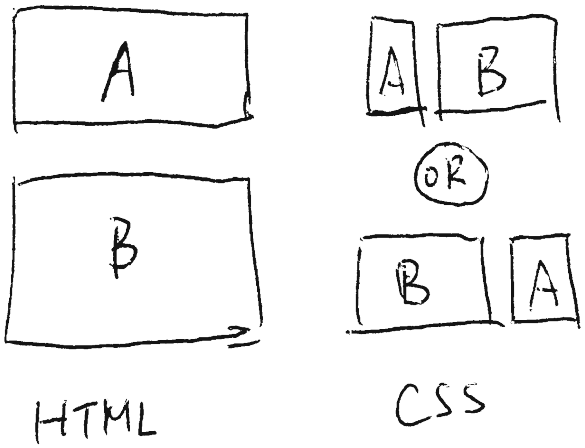
A special push and pull class allows us to change the order of the columns.


```
<div class='row'>
  <div class='small-9 columns'>Content</div>
  <div class='small-3 columns'>Sidebar</div>
</div>
```


```
<div class='row'>
  <div class='small-9 small-push-3 columns'>Content</div>
  <div class='small-3 small-pull-9 columns'>Sidebar</div>
</div>
```

HTML defines the order of the content. The order shows how important the content is. More important content are placed before other content.





When we float the columns from left to right. It follows the order of the column divs.

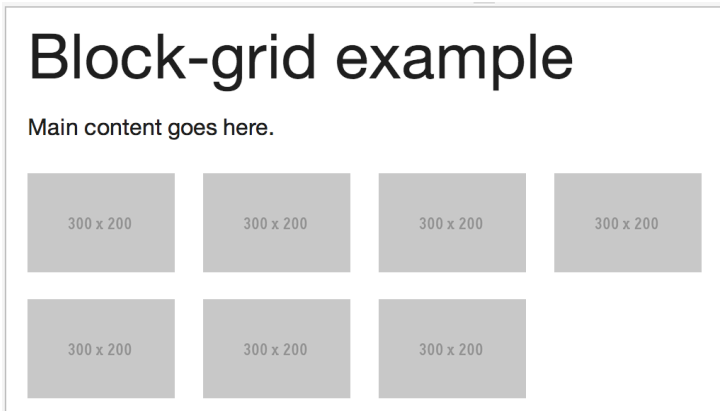
For instance, if we want to swap the side bar from right to left, we can either change the HTML or we can change the CSS class by using this foundation class. Both code result in the same output visually. So what is the difference?

The difference is that the foundation class solution doesn't change the order of the HTML element. We keep the same definition that content is more important than the side bar. But visually, we have our choice to define whether the side bar goes to the left or right side visually.

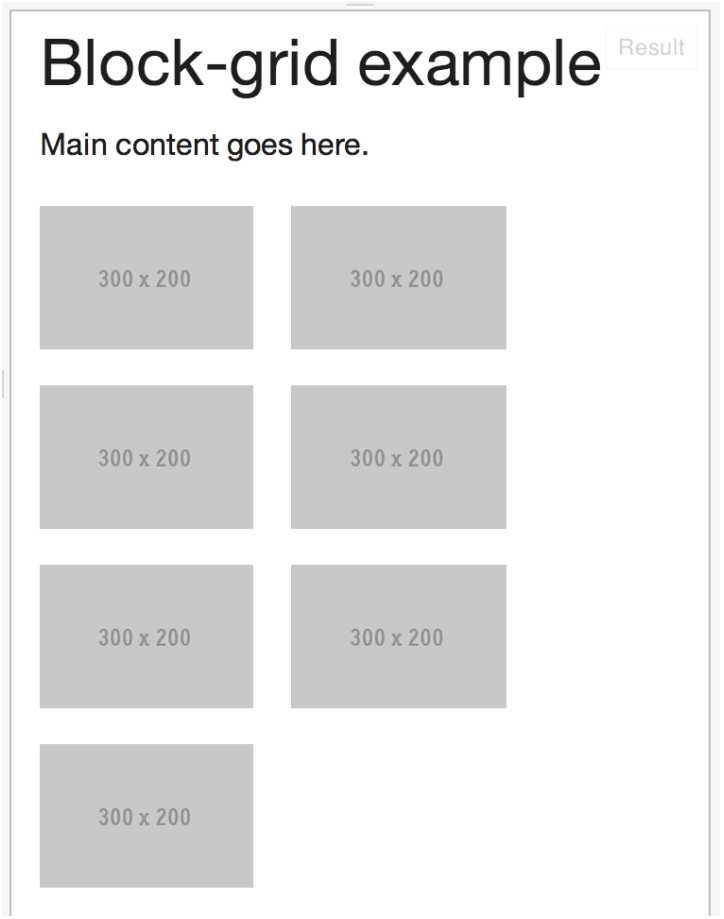
Block grid

Foundation provides `small-block-grid-N`, `medium-block-grid-N` and `large-block-grid-N`. It's very convenient when we need to spread the content evenly inside a space.

```
<ul class="small-block-grid-2 medium-block-grid-4">
  <li><img src="http://placehold.it/300x200"
alt="placeholder" /></li>
  ...
  <li><img src="http://placehold.it/300x200"
alt="placeholder" /></li>
</ul>
```



Block grid displayed in wide screen.



Block grid displayed in small screen.

Flex video

Here is a demo showing the usage of `flex-video` class from Zurb Foundation. It keeps the proportion of the embed video player, resulting in a much better looking in various

screen sizes.

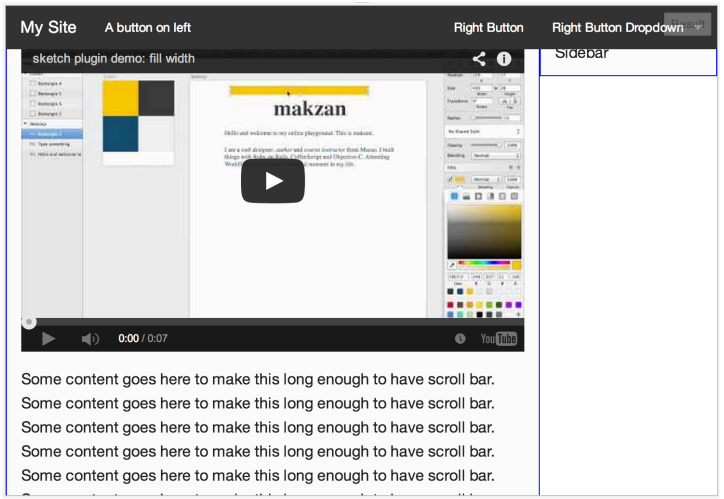
An extra `widescreen` class would make the hight calculation uses the widescreen ratio.

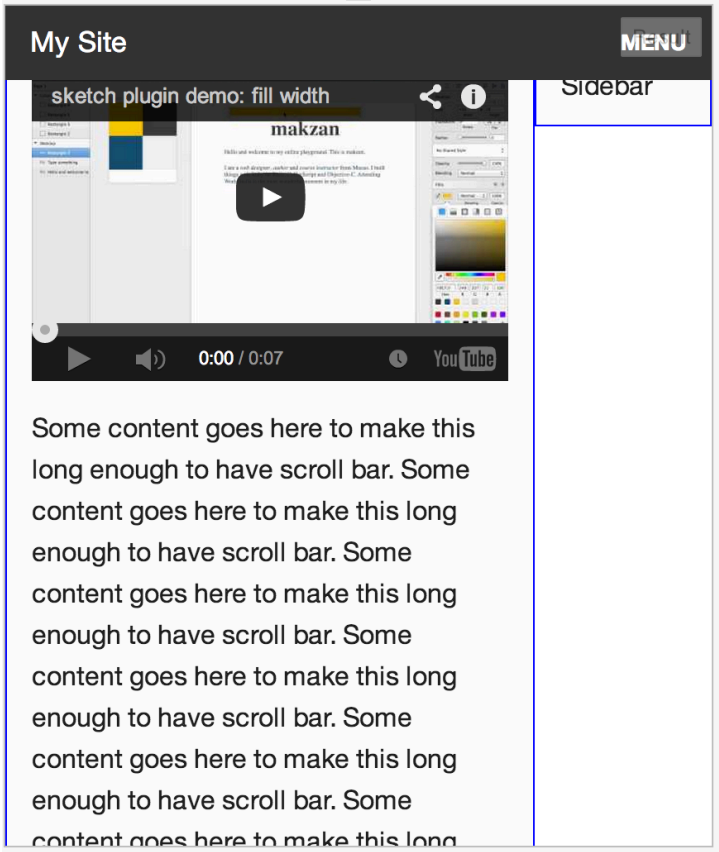
```
<div class='row'>
  <div class='small-9 columns'>
    <div class="flex-video widescreen">
      <iframe src="//www.youtube.com/embed/
KBA4IzyEUDU?rel=0" frameborder="0"
allowfullscreen></iframe>
    </div>
  </div>
  <div class='small-3 columns'>Sidebar</div>
</div>
```

Foundation topbar

Foundation comes with a `topbar` with navigation supports. It is quite useful for website with minimal header. The navigation items are displayed in wide screen and then hide into a menu button in small screen.

Here is the result of the same top bar displaying in both desktop and mobile.





A basic example:

```
<nav class="top-bar" data-topbar>
  <ul class="title-area">
    <li class="name">
       <h1><a href="#">My Site</a></h1>
    </li>
    <li class="toggle-topbar"><a
href="#"><span>Menu</span></a>
    </li>
  </ul>
  <section class="top-bar-section">
    <ul class="right">
      <li><a href="#">Right Button</a></li>
      <li><a href="#">Another button</a></li>
    </ul>
  </section>
</nav>
```

A full example:

```
<div class="sticky">
  <nav class="top-bar" data-topbar>
    <ul class="title-area">
      <li class="name">
        <h1><a href="#">My Site</a></h1>
      </li>
      <li class="toggle-topbar"><a
href="#"><span>Menu</span></a></li>
    </ul>
    <section class="top-bar-section">
      <ul class="left">
        <li><a href="#">A button on left</a></li>
      </ul>
      <ul class="right">
        <li><a href="#">Right Button</a></li>
        <li class="has-dropdown">
          <a href="#">Right Button Dropdown</a>
          <ul class="dropdown">
            <li><a href="#">First link in
```

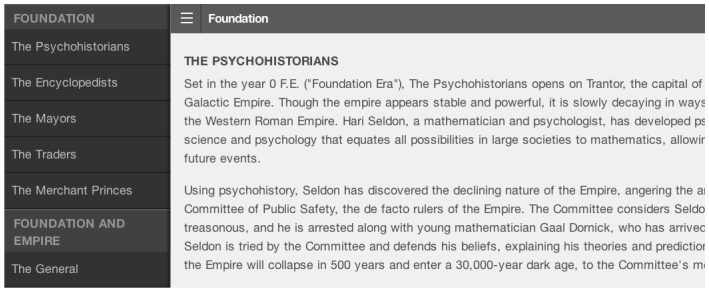
```
dropdown</a></li>
            <li><a href="#">Second link in
dropdown</a></li>
          </ul>
        </li>
      </ul>
    </section>
  </nav>
</div>
```

For more examples, please take a look at the Foundation documentation.

Note: Using the Foundation top-bar doesn't mean it has to be in black background. This tutorial by Mark Teekman shows how we can customize the top-bar.

Off canvas

Off Canvas is a slide-in navigation approach from Zurb. The top bar that we just discussed uses drop down menu. This Off-canvas widget use slide in menu from left or right side of the screen.



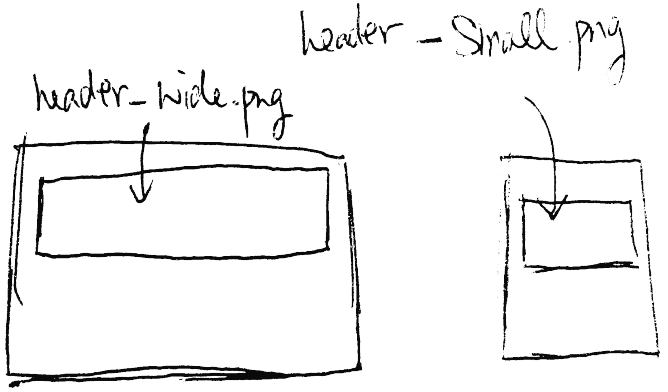

```
<div class="off-canvas-wrap" data-offcanvas>
  <div class="inner-wrap">

    <a class="left-off-canvas-toggle" href="#">Left
Menu</a>

    <a class="right-off-canvas-toggle" href="#">Right
Menu</a>

    <!-- Off Canvas Menu (Left) -->
    <aside class="left-off-canvas-menu">
      <!-- whatever you want goes here -->
      <ul>
        <li><a href="#">Item 1</a>
        </li>
        <li><a href="#">Item 1</a>
        </li>
        <li><a href="#">Item 1</a>
        </li>
      </ul>
    </aside>

    <!-- Off Canvas Menu (Right) -->
```

```
    <aside class="right-off-canvas-menu">
      <img src='http://placehold.it/300x100'
alt='placeholder' />
      <ul>
        <li><a href="#">Right Item 1</a>
        </li>
        <li><a href="#">Right Item 1</a>
        </li>
        <li><a href="#">Right Item 1</a>
        </li>
      </ul>
    </aside>

    <p>Your main content goes here.</p>

    <!-- closing the off-canvas -->
    <a class="exit-off-canvas"></a>

  </div>
</div>
```

As usual, the official documentation provides more usage examples.

Interchangable content

Responsive design comes with a downside: It wastes (mobile data) bandwidth if a lot of information is hidden in small screen. And it slows down the page loading.



An optimal way to archive the same effect is by using interchange. We use javascript to load the image at the size that fits the reader's screen. Smaller image for small screen. Higher resolution image for larger screen. The readers only need to load what they are going to read. They should never load something that they don't see and use.

The reason we need to interchange content is because of the loading bandwidth. In the visibility section, we show and hide content based on the screen size. But the reader have to download all the content. Assuming the reader is reading on mobile but we load a large image for large screen reader. Then the mobile reader needs to download a high resolution image with the mobile data but never see this image.

```
<img data-interchange="[/path/to/default.jpg,
(default)], [/path/to/bigger-image.jpg, (large)]">
<noscript><img src="/path/to/default.jpg"></noscript>
```

Foundation website provides a detailed tutorial on using the interchangable content in different scenarios. Ensure to check it out.

Exercise Time

Given the following HTML:

```
<img data-interchange="[/path/to/default.jpg,
(default)], [/path/to/bigger-image.jpg, (medium)]" />
<noscript><img src="/path/to/default.jpg" /></noscript>
<h1>Interchange Exercise</h1>
<p>What you need to do is display an image
<code>http://placehold.it/500x200&text=normal</code>.
While the screen is large enough, we change this image
to a higher resolution one: <code>http://placehold.it/
1000x400&text=large</code></p>
```

Table

https://foundation.zurb.com/sites/docs/table.html

TODO

We need table to display tabular data.

Form

https://foundation.zurb.com/sites/docs/forms.html

TODO

What's next? We're going to take a look at "Chapter 4 – Response with media query".

Chapter 4—Responsive With Media Query

In this part, we focus on building our own grid system by using the viewport and media query techniques.

1. Recap – Minimal mobile friendly website
2. Using viewport to define initial browser rendering
3. Using media query to define styles with condition
4. Navigation strategy for small screen
5. Putting navigation at bottom
6. Building our own grid system
7. Introducing Ungrid – Another minimal grid approach
8. Media queries in Zurb Foundation

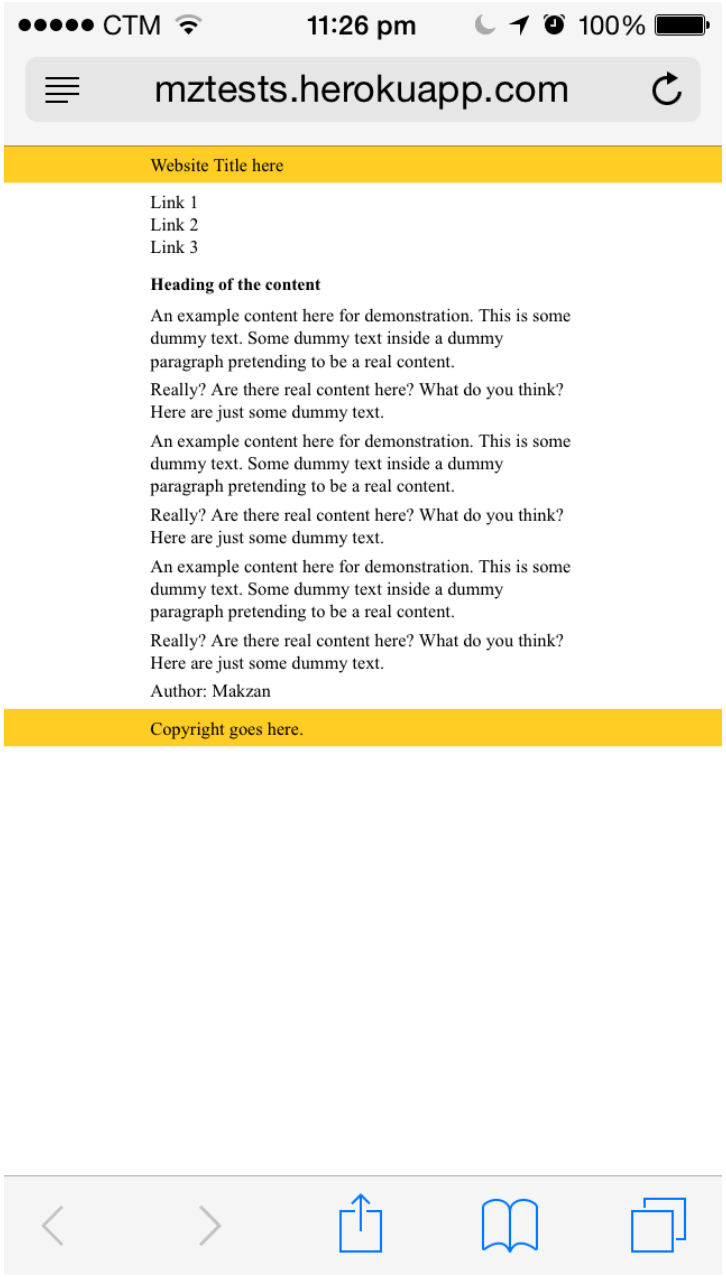
Media queries in Zurb Foundation

Now we've learnt media query. Let's take a look on how Foundation define the media queries into break points for small, medium and large screens.

```
// Small screens
@media only screen { } /* Define mobile styles */

@media only screen and (max-width: 40em) { } /* max-
width 640px, mobile-only styles, use when QAing mobile
issues */

// Medium screens
@media only screen and (min-width: 40.063em) { } /* min-
width 641px, medium screens */

@media only screen and (min-width: 40.063em) and (max-
width: 64em) { } /* min-width 641px and max-width
1024px, use when QAing tablet-only issues */

// Large screens
@media only screen and (min-width: 64.063em) { } /* min-
width 1025px, large screens */

@media only screen and (min-width: 64.063em) and (max-
width: 90em) { } /* min-width 1025px and max-width
1440px, use when QAing large screen-only issues */

// XLarge screens
@media only screen and (min-width: 90.063em) { } /* min-
width 1441px, xlarge screens */

@media only screen and (min-width: 90.063em) and (max-
width: 120em) { } /* min-width 1441px and max-width
1920px, use when QAing xlarge screen-only issues */

// XXLarge screens
@media only screen and (min-width: 120.063em) { } /* min-
width 1921px, xxlarge screens */
```

Re-visit the minimal website

In this chapter, we revisit the code and add more mobile specific configuration to it. As a recap, here is the original code.



The HTML:

```
<header>
  <div class="row">
    Website Title here
  </div>
</header>

<nav>
  <div class="row">
    <ul>
      <li>Link 1</li>
      <li>Link 2</li>
      <li>Link 3</li>
    </ul>
  </div>
</nav>

<div class="row">
  <article>
    <header>
      <h1>Heading of the content</h1>
    </header>

    <p>Content paragraphs go here.</p>

    <footer>
      Author: Makzan
    </footer>

  </article>
</div>
```

```
<footer>
  <div class="row">
    Copyright goes here.
  </div>
</footer>
```

The CSS:

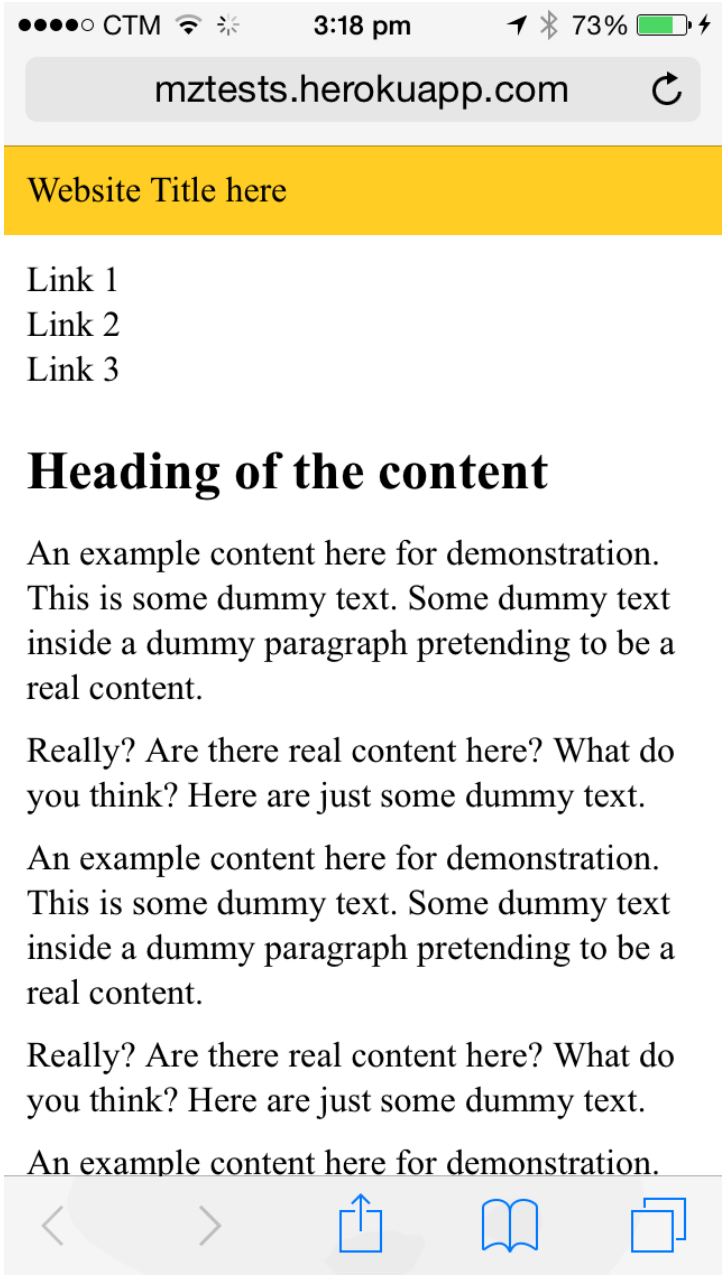
```
/* normalize */
body, nav, ul, li, p, h1, h2, h3 {
    padding: 0;
    margin: 0;
}

/* core styles */
* {
    -webkit-box-sizing: border-box;
    box-sizing: border-box;
}

body > header,
body > footer {
    background: #ffce25;
}

.row {
    width: 100%;
    max-width: 600px;
    margin: 0 auto;
    padding: 10px;
}
```

```
/* addition styles */
ul {
    list-style: none;
}

h1, p {
    margin-bottom: .5em;
}
```

Using viewport to define initial browser rendering

Remember that we created a minimal mobile friendly website in chapter 2? Now let's revise it with viewport and media queries configuration.

Add the following `<viewport>` tag inside tho `<head>` part of the HTML



The viewport tells the device, more specifically the mobile device, to use the real device with instead of pretending to be a desktop screen. Before we add the viewport configuration, mobile web browser usually pretends to be a desktop screen with 900 something pixels width. This ensures showing the whole website to the readers and let readers zoom in to the part where they are interested to read. After we set viewport to use the device width, mobile web browser would use the real device width, for instance, 320px in iPhone, to render and display the website.

QuirksMode provides a detail discussion on how the viewport options.

Using media query to define styles with condition

We will style our navigation on the minimal mobile-friendly website, and make it easy to click in both desktop and mobile.

Here is the result in both small and wide screen.

Here is the code to archive it.

The HTML:

```
<nav>
  <div class="row">
    <ul>
      <li><a href='#'>Link 1</a></li>
      <li><a href='#'>Link 2</a></li>
```

```
      <li><a href='#'>Link 3</a></li>
    </ul>
  </div>
</nav>
```

The CSS styles for normal wide screen styles.

```
nav li {
  width: 33%;
  float: left;
  padding: 5px;
}

nav li a {
  background: #efefef;
  display: block;
  padding: 15px 0px;
  text-align: center;
}
```

And the CSS style for small screen where the width of the viewport is smaller than 501px.

```
@media screen and (max-width: 500px) {
  nav li {
    width: 100%;
  }
}
```

The mobile first approach

Here we revert the styles definition to use styles that fits mobile screen by default. When the web is displayed in a wider screen, we use the media query to define extra styles.

It's a minor difference in code, but it's a huge mental shift in planning website.

```
nav li {
  width: 100%;
  padding: 5px;
}

@media screen and (min-width: 500px) {
  nav li {
    width: 33.33%;
    float: left;
  }
}
```

Navigation strategy for small screen

In small screen, we used to hide navigation to provide enough screen space for the main content.

There are several approaches:

- Dropdown select
- Hamburger menu
- Slide down
- Footer navigation

You may find [this essay about popular trends on navigation pattern] from Jake Rocheleau.

NYTimes' navigation example

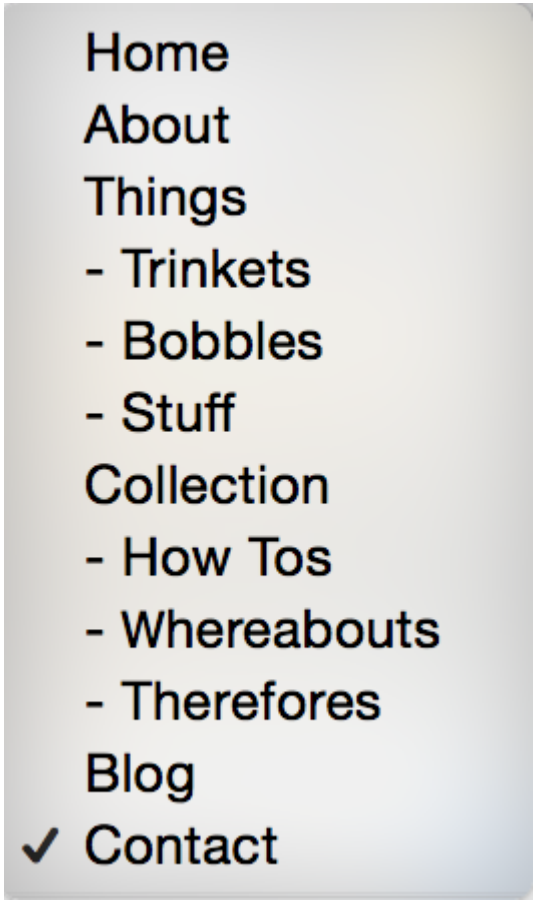
Also, there are a collection of patterns in this ResponsiveNavigation.com.

Examples of common responsive navigation

Block grid navigation

Navigation in block grid

Navigation based on <select>



Navigation based on select

Extra: Brad Frost has written two articles on the patterns with pros and cons on each navigation pattern:

- Responsive nav patterns
- Complex navigation patterns for responsive design

Putting navigation at bottom

There are several advantages by putting the navigation at the bottom.

1. **There is one copy of the navigation.** One copy mean no repeat on the HTML means a little bit faster. And easier to manage.
2. **No javascript, no even css, required.** This method is supported by HTML natively because it is just anchor hash link. We don't even need css and javascript make it works.
3. **The menu is revealed after clicking the menu button, anyway.** Whether we hide the navigation on the left slide-in pane, at the top or at the bottom. It's the same that we hide the navigation at the beginning and reveal the links after clicking the button.
4. **Ensuring we have some links at the footer.** Having links at the footer allows reader to have something to navigate after finish reading your page.

```
<header id='top'>
  <h1>Demo on footer navigation strategy</h1>
</header>
<nav>
  <ul class='small-block-grid-4 hide-for-small-only'>
    <li><a href='#'>Home</a></li>
    <li><a href='#'>About</a></li>
    <li><a href='#'>Works</a></li>
    <li><a href='#'>Contact</a></li>
```

```
  </ul>
  <p class='show-for-small-only'><a href='#footer-
nav'>Menu</a></p>
</nav>
<article>
  <p>A long text here.</p>
</article>
<footer>
  <nav id='footer-nav'>
    <ul>
      <li><a href='#top'>Top</a></li>
      <li><a href='#'>Home</a></li>
      <li><a href='#'>About</a></li>
      <li><a href='#'>Works</a></li>
      <li><a href='#'>Contact</a></li>
    </ul>
  </nav>
</footer>
```

Exercise: follow the code snippet above. Modify the navigation to make it work in small screen.

https://makclass.com/test*projects/2015*

Building our own grid system

The following CSS demonstrates a minimal grid system I created. It is inspired from the Zurb Foundation and I just keep the most essential part – the basic grid.

You will find that actually it takes less than 50 lines of code to build a responsive grid system with row/column approach.

By creating our own grid system, we learn the core of the row/column grid. One more benefit is that we can use a basic grid at the beginning of the project. And only include the Foundation framework into the project until we need their rich features.

https://codepen.io/makzan/pen/pPgerL?editors=1100

Code

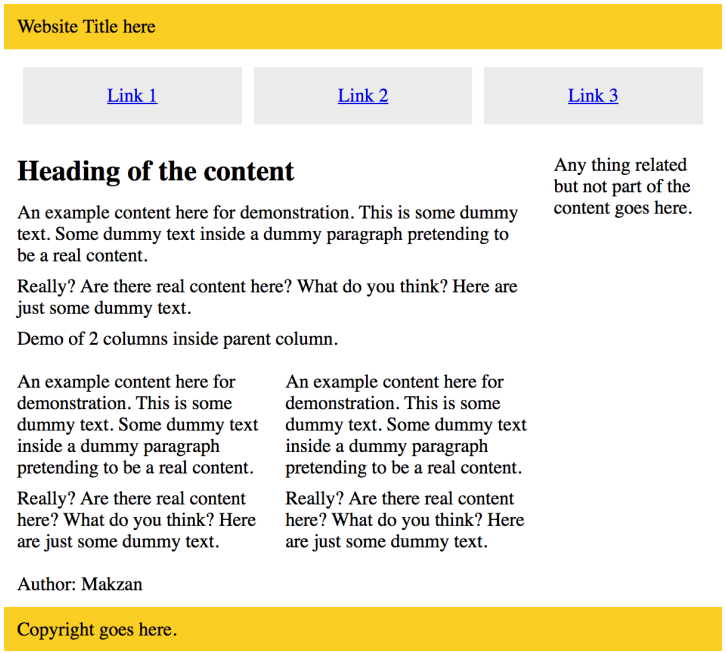
```
* {
    box-sizing: border-box;
}

img {
    max-width: 100%;
}

.row {
  width: 100%;
  max-width: 600px;
  margin: 0 auto;
  padding-top: 10px;
  padding-bottom: 10px;
}

.row .row {
  width: auto;
  max-width: none;
  margin: 0 -10px;
}
```

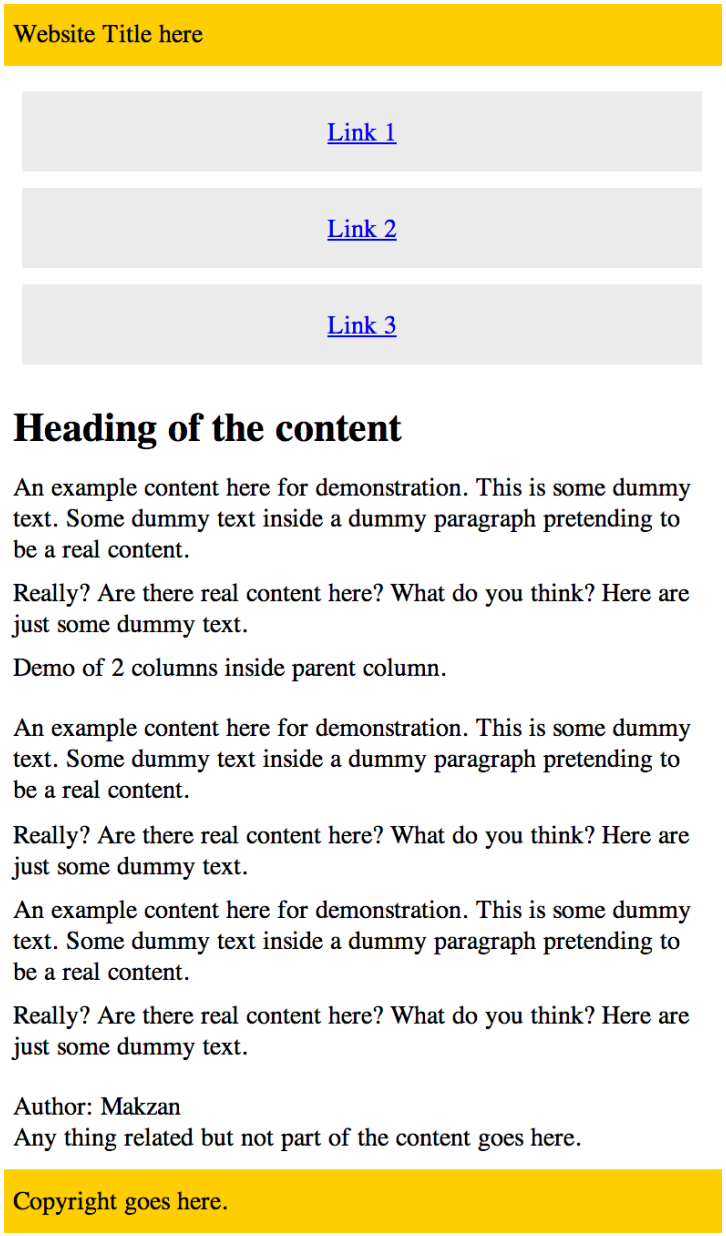
```
.col {
  padding: 0 10px;
  width: 100%;
}

@media screen and (min-width: 500px) {
  .row {
    overflow: auto;
  }
  .col {
    float:left;
  }
  .one {
    width: 25%;
  }
  .two {
    width: 50%;
  }
  .three {
    width: 75%;
  }
  .four {
  }
}
```

And the sample HTML that uses our CSS grid.

```
<header>
  <div class="row">
    <div class="four col">
      Website Title here
    </div>
  </div>
</header>

<nav>
  <div class="row">
    <div class="four col">
      <ul>
        <li><a href='#'>Link 1</a></li>
        <li><a href='#'>Link 2</a></li>
        <li><a href='#'>Link 3</a></li>
      </ul>
    </div>
  </div>
</nav>

<div class="row">
  <div class="three col">
    <article>
      <header>
        <h1>Heading of the content</h1>
      </header>

      <p>Content.</p>

      <p>Demo of 2 columns inside parent column.</p>
      <div class="row">
        <div class="two col">
          <p>Content.</p>
        </div>
        <div class="two col">
          <p>Content.</p>
        </div>
```

```
      </div>


      <footer>
        Content
      </footer>

    </article>
  </div>
  <div class="one col">
    <aside>
      Any thing related but not part of the content goes
here.
    </aside>
  </div>
</div>

<footer>
  <div class="row">
    <div class="four col">
      Copyright goes here.
    </div>
  </div>
</footer>
```





Examples

An example of the grid system with placeholder image.

https://codepen.io/makzan/pen/vmLNOV

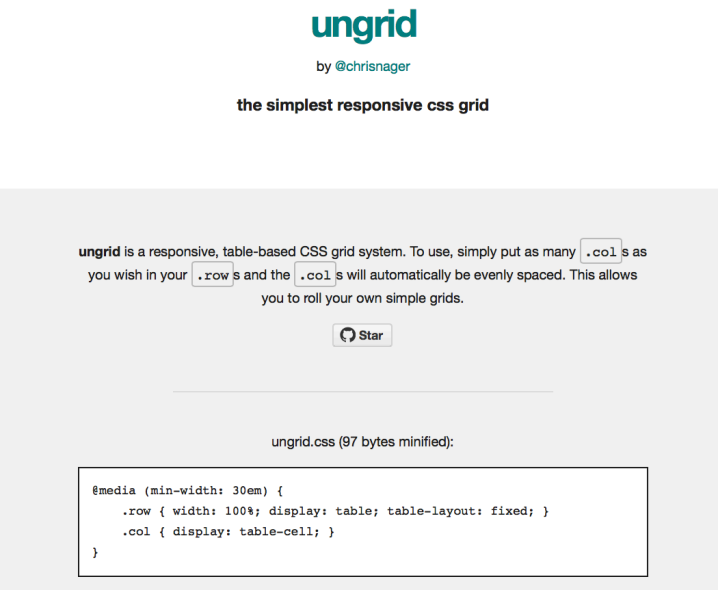
An example of using our grid system with the block-grid navigation approach.

https://codepen.io/makzan/pen/EmPVVv

An example of using our grid system with the footer navigation approach.

https://codepen.io/makzan/pen/KmVdVV

Introducing ungrid—another minimal grid approach



The ungrid is a very minimal grid system that makes use of the of the table cell display.

Because table cell is evenly distributed by default, so we just need to define columns and browser would handle the space spreading.

For any column that needs specific width, we can just set the width on that specific columns, using either fixed width or percentages.

```
@media (min-width: 30em) {
  .row { width: 100%; display: table; table-layout: fixed; }
  .col { display: table-cell; }
}
```

Chapter 5—Typography

In this chapter, we focus on the typography and how we can make it responsive.

In this chapter, we focus on building our own grid system by using the viewport and media query techniques.

1. Different font sizes
2. Reading distance and font size
3. Adaptive copywriting from Basecamp
4. Our own adaptive copywriting

Different font sizes

Dimension of a content box may be flexible and dynamic width based on the screen size. We also need different font sizes for different screen.

The following is the CSS I used in one of my website. I scale the font size for different width explictily.

```
$font-break-point-wide: 900px;
$font-break-point-medium: 800px;
$font-break-point-narrow: 700px;
$break-point: 560px;

body {
  font-size: 21px;
  font-family: Georgia, serif;
  line-height: 150%;

  @media (max-width: $font-break-point-wide) {
    font-size: 21px;
  }
  @media (max-width: $font-break-point-medium) {
    font-size: 19px;
  }
  @media (max-width: $font-break-point-narrow) {
    font-size: 18px;
  }
  @media (max-width: $break-point) {
    font-size: 16px;
  }

  /* iPhone Retina */
  @media only screen and (-webkit-min-device-pixel-ratio
: 1.5) and (max-width: $break-point), only screen and
(min-device-pixel-ratio : 1.5) and (max-width: $break-
point) {
    font-size: 16px;
  }
}
```

We call it adaptive design because we are not scaling the font automatically. Instead, we set the font size based on a certain criteria. The current criteria may mainly rely on the screen size. But actually we also need to consider the screen resolution such as the point-per-inch in the retina display. And the media such as the printing paper. And even the typical distance on how people use that device. For example, when

a website is displayed on TV, we are usually feet away from it and we need a larger font-size there.

For example, we used `min-device-pixel-ratio` and breakpont to define a specific font-size for mobile retina display.

Font size and distance

How we choose which font-size to use?

This depends on how far away we expect our readers read our web page.

The distance matters because font-size is relative to our vision and the distance.

iA posts an essay on the relation between reading distance and the font size.

Adaptive copywriting basecamp

It's not only about layout. We should also take care of our copywritings.

Here shows an example on Basecamp homepage. Please focus on the text in the top navigation.

Take a closer look on the top navigation. When the screen has more space, the copywriting of sign-up button is **“See prices & start a free trial”**, detail enough so I know what information to expect after the link, right? When the screen has less space, this link changes to a simply **“Sign up”** text. Short and precise enough.

Let’s take a look at another example. Here is the Signal vs. Noise from the same company. Take a look at the top sentence.

When wide enough, it shows:

> You’re reading Signal v. Noise, a publication about the web by Basecamp since 1999. Happy Thursday.

When it’s not wide enough, it shows:

> Signal v. Noise, a publication about the web by Basecamp since 1999.

In extreme case that the screen is too narrow, it displays:

> Signal v. Noise by Basecamp

Adaptive copywriting for different screens

Here is an example showing how we can create the same effect of adaptive copywriting.

We need the following HTML which defines explicitly what content to display for specific screen size.

```
<div class='leading-sentence'>
  <p class='large-only'>You’re reading a sample site,
```

```
written by Makzan since 2014. Enjoy!</p>
  <p class='medium-only'>A sample site, written by
Makzan since 2014.</p>
  <p class='small-only'>A sample site by Makzan.</p>
</div>
```

Then, all we need to do is to control the visibility of the class `large-only`, `medium-only` and `small-only`, via the following CSS.

```
$break-point-medium: 400px;
$break-point-large: 600px;

.leading-sentence {
  text-align: center;
}

@media screen {
  .small-only {
    display: block;
  }
  .medium-only,
  .large-only {
    display: none;
  }
}

@media screen and (min-width:$break-point-medium) {
  .medium-only {
    display: block;
  }
  .small-only,
  .large-only {
    display: none;
  }
}

@media screen and (min-width:$break-point-large) {
  .large-only {
    display: block;
  }
  .small-only,
  .medium-only {
    display: none;
  }
}
```

Chapter 6—Handling touches

In this chapter, we focus on how we can handle the touch event.

1. Handling 300ms click delay
2. Using Slick for mobile carousel
3. Recognizing gesture with HammerJS

Click delay

There is a click delay in mobile web page. You may try the #link_to 'demo here', 'http://mztests.herokuapp.com/jq201/', target: '_blank' in your mobile devices. Try click on the buttons and see if you find some delays.

This is due to an intension delay, around 300ms, on the system to distinguish `touchstart`, `touchmove`, `touchend` and `click` event. `touchstart`, `touchmove`, `touchend` is similar to `mousedown`, `mousemove` and `mouneup` event in the desktop browser environment.

For detail, Matteo Spinelli has created a testing page to compare the `touchstart`/`touchend` event and `click` event.

The following JavaScript overcomes the delay by manually triggering the click event based on the `touchstart` and `touchend`.

```
$("input[type=submit]").bind('touchstart', function(e) {
  e.preventDefault();
});

$("input[type=submit]").bind('touchend', function(e) {
  e.preventDefault();
  $(this).trigger('click');
});
```

Or we can use the FastClick library.

For gestures recognizing and handling, you may check the Hammer library.

Using slick

Besides listing things from top to bottom, we may sometimes put information into a left/right swiping carousel. Slick is a great one among those many carousel or slider library. The following is an example showing a trip planning with multiple days. Putting these day information into horizontal list makes the website more organized, and makes use of the common mobile swiping gesture.

```
<div class="cards">
  <div class="card">
    <h2>Day 1</h2>
    <p>Blah Blah Blah...</p>
    <p>Blah Blah Blah...</p>
    <p>Blah Blah Blah...</p>
    <p>Blah Blah Blah...</p>
  </div>
  <div class="card">
    <h2>Day 2</h2>
    <p>Blah Blah Blah...</p>
    <p>Blah Blah Blah...</p>
    <p>Blah Blah Blah...</p>
    <p>Blah Blah Blah...</p>
  </div>
  <div class="card">
    <h2>Day 3</h2>
    <p>Blah Blah Blah...</p>
    <p>Blah Blah Blah...</p>
    <p>Blah Blah Blah...</p>
    <p>Blah Blah Blah...</p>
  </div>
</div>
```

```
.card {
  background: STEELBLUE;
  color: #fefefe;
  padding: 1rem;
  margin: .8rem;
}
```

This JavaScript library requires jQuery. We initialize the slick on specific element after the jQuery ready event. *Center mode* is important because it shows part of the neighbor cards. This lets user knows there are something on the side, so that user will swipe.

```
$(function(){
  $('.cards').slick({
    centerMode: true,
  });
});
```

Using Swiper

A nice alternative to Slick is the Swiper. It is the from the same team behind Framework7.

You can have a glimpse on what Swiper can do via its demo page.

Hammerjs

HammerJS allows us to detect and handle most of the touch gestures. If in case your mobile web page or web app needs to detect gestures, I recommend checking out this tool.

Chapter 7—Form inputs

In this chapter, we learn how to optimize the form input experience by configuring the input tags.

1. Specify input types
2. Styling inputs for mobile
3. Styling radio button
4. Styling input range
5. Using file button

Specify input types

Specific input types allows web browser knows what data to expect the user input. For example, an `email` type input means the input should be an email. Some browsers, such as Google Chrome, would provide extra email format checking to validate if the email is valid.

Also, the layout of virtual keyboard in mobile device would change based on the type. The following shows few types.

Styling inputs for mobile

Touch device requires a very different styling approach for inputs. Inputs in touch devices are usually bigger and easier for tap and select by fingers. An easy step is to make the font size and padding larger, via the following CSS.

```
input {
  width: 100%;
  padding: 1rem .5rem;
  font-size: 1rem;
  border: 1px solid #efefef;
}
```

The default button would be too small for tapping, we would make the button larger. We also make it look like a button by adding hightlight and shadow on the normal, hover and active states.

```
input[type="submit"],
input[type="button"]{
  appearance: none;  /* for mobile safari */
  background-color: YELLOWGREEN;
  color: white;
  border: 1px solid rgba(0,0,0,.1);
  border-radius: 3px;
  box-shadow: inset 0 1px rgba(255,255,255,.3), inset 0
-1px rgba(0,0,0,.1);

  &:hover {
    background-color: darken(YELLOWGREEN, 10%);
    box-shadow: inset 0 1px rgba(255,255,255,.7), inset
0 -1px rgba(0,0,0,.1);
  }
  &:active {
    background-color: YELLOWGREEN;
```

```
    box-shadow: inset 0 -1px rgba(255,255,255,.1), inset
0 1px rgba(0,0,0,.1);
  }
}
```

You may try the demo in this CodePen.

Styling radio button

```
<input type='radio' id='desktop-product' name='product'
value='desktop'>
<label for='desktop-product'>Desktop</label>
<br>
<input type='radio' id='mobile-product' name='product'
value='mobile'>
<label for='mobile-product'>Mobile</label>

input[type='radio'] {
  display: none;

  &+label{
    display: block;
    font-size: 1rem;
    padding: 1rem .5rem;
    padding-left: 3rem;
    border: 1px solid transparent;
  }
  &:checked+label {
    border-color:YELLOWGREEN;
    background-color: lighten(YELLOWGREEN, 49%);
    background-image: url(tick.png)
    background-position: 16px 50%;
    background-repeat: no-repeat;
  }
}
```

Styling input range

```
<form method='post' action='data:text/plain, Form
submited.' oninput='result.value=distance.value'>
  <p>Nearby:<br><input type='text'
placeholder='restaurants, ATM'></p>
  <p>
    Within distance:
    <output name='result' for='distance'>100</output>
    m
    <br>
    <input name='distance' type='range' value='100'
max='2000' min='100' step='100'>
  </p>
  <p><input type='submit' value='Search'></p>
</form>

input[type="range"] {
  appearance: none;
  background: #efefef;

  &::-webkit-slider-thumb {
    @include greeny-ui;
    appearance: none;
    width: 30px;
    height: 30px;
    border-radius: 50%;
  }

  /* TODO: add moz and ms styles too */
```

```
}
```

input range
input range

The live demo:

Using file button

```
input[type="file"] {
  border: 1px solid #aaa;
  border-radius: 3px;
  background: lighten(YELLOWGREEN, 46%);
  padding-left: 70px;

  &::-webkit-file-upload-button {
    visibility: hidden;
  }

  &::before {
    content: "Select Photo:";
    margin-left: -70px;
    padding: 1.5rem;
    @include greeny-ui;
```

```
    }
}
```

Reference: CSS-Tricks

Chapter 8—Taking the web offline

In this chapter, we discuss several techniques to store data offline and even cache the entire site for offline usage.

1. LocalStorage
2. Application Cache
3. Service Worker

Storing data with local storage

LocalStorage

```
<form method='post' action='data:text/plain, Form
submited.'>
  <p>Email:<br><input type='email' name='email'
id='email' placeholder='test@example.com'></p>
  <p>Password:<br><input type='password'
placeholder='********'></p>
  <p><input type='submit' value='Login'></p>
</form>

$("#email").on('keyup', function() {
  return localStorage['email'] = $(this).val();
});

$(function() {
  return $("#email").val(localStorage['email']);
});
```

In browser inspector, reader can view and modify any stored data in Local Storage.

You may try the demo in this CodePen. Try to enter a value in email, then refresh the page and you should see what you have typed.

Offline access with app cache

Beware of the deprecation of Application Cache. We should use Service Worker for offline cache unless you need to support old mobile browsers.

With the help of cache manifest, mobile web app can work in offline.

An example is from Pie Guy.

Check the Pie Guy game in your mobile device.

1. Add it to home screen
2. Go to air plane mode
3. Or even restart the device
4. Open the game in offline mode and it works as normal.

Here is how it looks like in the HTML tag.

```
<html manifest="cache-manifest.manifest">
```

And here is the content of the manifest file, which contain all the assets file the Pie Guy game needs.

```
CACHE MANIFEST
# Version 1.0.3

index.html
main.js
images/about-text.gif
images/default.png
images/font1.gif
images/game-over.gif
images/guy1.gif
images/guy2.gif
images/guyFlip.gif
images/chef0-1.gif
images/chef0-2.gif
images/chef0-flip.gif
images/chef1-1.gif
images/chef1-2.gif
images/chef1-flip.gif
images/life.gif
images/map0.png
images/map1.png
images/menu-about-active.gif
images/menu-about-small-active.png
images/menu-about-small.png
images/menu-about.gif
images/menu-back.gif
images/menu-back-active.gif
images/menu-continue-active.gif
images/menu-continue.gif
images/menu-new-active.gif
images/menu-new.gif
images/menu-rotate.gif
images/menuButton.gif
images/notice-level.gif
images/notice-time.gif
images/notice-tokens.gif
images/notice-total.gif
images/pie-complete.gif
images/bonus-score.gif
images/scoreBar.gif
images/splash.png
images/token1.gif
images/token2.gif
images/token3.gif
images/token4.gif
```

Reference:

This StackOverflow thread discussed how we can remove the cached files and force the browser to download the newer version. The HTML5 Rocks posts a detailed tutorial to get started the application cache.

Provide offline access with Service Worker

Application cache is deprecated. It is quite headache to maintain. Now we use Service Worker to cache offline file.

> This feature is in the process of being removed from the Web platform. (This is a long process that takes many years.) Using any of the offline Web application features at this time is highly discouraged. Use service workers instead. — WHATWG spec

Service Worker is a proxy between browser and server. It can do much more than caching files. For example, it can run in background process and push notifications to your device.

The Google Chrome's Github page provides an example code on the offline capability.

https://googlechrome.github.io/samples/service-worker/custom-offline-page/

Mozilla also created a Service Workers cookbook to showcase several essential examples.

For example, we can cache the images and have a notification when all the cache is downloaded and ready:

> Image Credit ServiceWorke.rs
> Image Credit ServiceWorke.rs

Summary

This is the end of the Mobile First Web Design introduction.

In conclusion, we have learnt:

- Importance of content.
- Importance of defining a good heading.
- Importance of content structure.
- Mobile-first grid system.
- Responsive grid via `@media` queries.
- Zurb Foundation.
- Responsive typography.
- Responsive copywriting.
- Touches and Clicks events.
- Form input improvements.
- Taking the data and web page offline.

At last, we learn some extra performance tips.

Performance Tips

Improving the website performance is important because the network speed is not usually stable. Here are several tips for fine tuning the performance.

1. Put all the `<script>` tag at the end of body.

Most script manipulate content and so there is no harm of putting the script after the content loading and rendering.

Putting the script before loading the content may cause the content loading delayed because of the characteristic of the `<script>` tag.

Note that some JavaScript libraries do require placing before content rendering. Mondernizer is one of them.

2. Keep the external font face loading minimal.

External font-face could be an issue in low-speed network. Texts with custom font-face is not rendered at all until the font file is loaded. In mobile device, user may need to wait seconds before they can read anything. So external font-face loading should be minimal.

3. Rely on minimal 3rd party libraries.

Libraries is a handy tool. But think twice before relying on any 3rd party libraries. Do you really need this library to archive the task? I'm not against any libraries, just make sure you are not including a large file just for a small part of features.

4. Use accelerated animation and transition

If you need animation or transition for your views, choose the accelerated method. For example, moving DOM elements or changing styling transition can be done via the CSS transition. Furthermore the 3D translate is hardware accelerated and faster than the 2D one.

5. Lazy load unnecessary content

Some content may not be necessary for the main content. Take comment as example. Not every readers read comment. But they all read your main content when they load your web page. You might defer the loading of the comment area only after reader finish the content reading or request for it.

One more thing—Mobile as first class citizen

We can treat mobile as first class citizen. It provides more capability than desktop.

In mobile, you can get current position by **GPS** sensor. You can get device rotation and motion via **accelerometer** and **gyroscope**. You can accept camera photo upload via **file input**. Your reader can pin your web to their **home screen**.

In near future, you mobile web can also access the **bluetooth**, **camera** raw data, and **push notification**.

Imagine a web page and web utility that makes use of these capability, which we never imagine them in desktop web. The mobile web is the first class client. That's the meaning of *Mobile First*.

Appendix—CSS Preprocessing

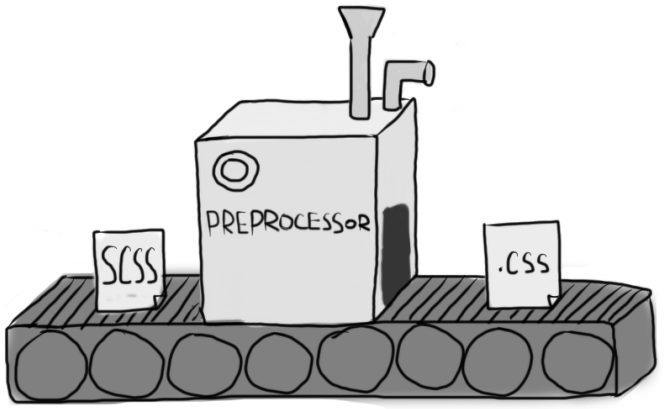
In this part, we learn how to add logic to the CSS via preprocessors.

What is Proprocessor?

1. Preprocessor options
2. Nested styles
3. SCSS variable
4. SCSS mixin
5. Using loop to define grid

Note: this chapter is optional but recommended.

What is CSS preprocessor?



Nowadays, we often craft our styles by using preprocessors syntax. They provides a CSS-like syntax with enhanced features to save our time when building the web.

Preprocessing

We may get help from CSS Preprocessors. They enhance the CSS with a couple of handy features. Here are common features that preprocessors provides:

- variables
- functions (mixin)
- nested selectors

Tools

For the tools to compile the code into CSS, we need tool. Someone may favour command-line tool and some may favour GUI ones. Here are the GUIs.

- CodeKit (mac only)
- LiveReload (win and mac)

And here is the command-line tools.

- GruntJS
- GulpJS

Codepen

Codepen also provides both compiled CSS and SCSS in their editor.

Preprocessor options

Here is three popular CSS preprocessors we can use:

- Less
- Sass / Scss
- Stylus

Zurb Foundation framework, which we covered in earlier chapters, uses scss as the preprocessor.

Nested styles

We can nest selector to put related style definition into groups.

An Scss example of nested selectors.

```
.page {
  p {
    margin: 1em 0;
  }
  .row {
    width: 100%;
  }
  &>div {
    padding: 10px;
  }
}
```

This scss will compiled into the following css.

```
.page p {
  margin: 1em 0;
}
.page .row {
  width: 100%;
}
.page>div {
  padding: 10px;
}
```

The Scss edition of our minimal grid system.

```
.row {
  width: 100%;
  max-width: 600px;
  margin: 0 auto;
  padding-top: 10px;
  padding-bottom: 10px;

  @media screen and (min-width: 500px) {
    overflow: auto;
  }

  .row {
    width: auto;
    max-width: none;
    margin: 0 -10px;
  }
}

.col {
  padding: 0 10px;
  width: 100%;

  @media screen and (min-width: 500px) {
    float:left;
  }
}


/* Navigation */
nav li {
  width: 100%;
  padding: 5px;

  @media screen and (min-width: 500px) {
    width: 33.33%;
    float: left;
  }

  a {
    background: #efefef;
    display: block;
    padding: 15px 0px;
    text-align: center;
  }
}

@media screen and (min-width: 500px) {
  .one {
    width: 25%;
  }
  .two {
    width: 50%;
  }
  .three {
    width: 75%;
  }
  .four {
  }
}
```

The compiled CSS from the Scss.

```
.row {
  width: 100%;
```

```
  max-width: 600px;
  margin: 0 auto;
  padding-top: 10px;
  padding-bottom: 10px; }
  @media screen and (min-width: 500px) {
    .row {
      overflow: auto; } }
  .row .row {
    width: auto;
    max-width: none;
    margin: 0 -10px; }

.col {
  padding: 0 10px;
  width: 100%; }
  @media screen and (min-width: 500px) {
    .col {
      float: left; } }

/* Navigation */
nav li {
  width: 100%;
  padding: 5px; }
  @media screen and (min-width: 500px) {
    nav li {
      width: 33.33%;
      float: left; } }
  nav li a {
    background: #efefef;
    display: block;
    padding: 15px 0px;
    text-align: center; }
```

```
@media screen and (min-width: 500px) {
  .one {
    width: 25%; }

  .two {
    width: 50%; }

  .three {
    width: 75%; }
}
```

SCSS variable

In last section, we put the `@media` inside a nested selectors in different groups. This leads to an issue that our breakport, 500px, is difined in more than one place.

Preprocessor allows us to define variable. Here is an example of using Scss that define 2 variables, breakpont and the major color.

```
$breakpoint: 500px;
$major-color: steelblue;

body > header,
body > footer {
    background: $major-color;
}

@media screen and (min-width: $breakpoint) {
  .one {
```

```
    width: 25%;
  }
  .two {
    width: 50%;
  }
  .three {
    width: 75%;
  }
  .four {
  }
}
```

SCSS mixin

Mixin lets us reuse common code in different places.

We can define styles like this:

```
@mixin bordered {
  border: 1px solid black;
}
```

Or we may provide argument:

```
@mixin bordered($color:black, $radius:0) {
  border: 1px solid $color;
  border-radius: $radius;
}
```

Then we can use our mixin in different places:

```
header {
  @include bordered(blue);

  width: 100%;
}
nav li {
  @include bordered(white, 5px);
}
```

Here is an example on how we can use the mixin to create a toggable debugging border.

```
$show-debug: true;
@mixin debug ($color:red) {
  @if $show-debug {
    border: 1px solid $color;
  }
}
```

In the `.col` and `nav li`:

```
.col {
  padding: 0 10px;
  width: 100%;

  @include debug;

  @media screen and (min-width: $breakpoint) {
    float:left;
  }
}
nav li {
  width: 100%;
```

```
  padding: 5px;

  @include debug(green);
}
```

Using loop to define grid

Let's get deeper on using Scss. We are going to define the grid system by calculation. In our minimal mobile web grid system, we defined the following CSS styles.

```
@media screen and (min-width: $breakpoint) {
  .one {
    width: 25%;
  }
  .two {
    width: 50%;
  }
  .three {
    width: 75%;
  }
  .four {
    width: 100%;
  }
}
```

Let's assume we use `col-N` for the column name, so `one` becomes `col-1`, and so on.

```
@media screen and (min-width: $breakpoint) {
  .col-1 {
    width: 25%;
  }
  .col-2 {
    width: 50%;
  }
  .col-3 {
    width: 75%;
  }
  .col-4 {
    width: 100%;
  }
}
```

Now we define how many columns count in total with a variable.

```
$columns-count: 12;
```

Now we can use the `@for $i from A through B` syntax to define the columns:

```
@media screen and (min-width: $breakpoint) {
  @for $i from 1 through $columns-count {
    .col-\#{$i} {
      width: 100%/$columns-count * $i;
    }
  }
}
```

And we would get this compiled CSS with all the percentages calculated.

```
@media screen and (min-width: 500px) {
```

```
.col-1 {
  width: 8.33333%; }

.col-2 {
  width: 16.66667%; }

.col-3 {
  width: 25%; }

.col-4 {
  width: 33.33333%; }

.col-5 {
  width: 41.66667%; }

.col-6 {
  width: 50%; }

.col-7 {
  width: 58.33333%; }

.col-8 {
  width: 66.66667%; }

.col-9 {
  width: 75%; }

.col-10 {
  width: 83.33333%; }

.col-11 {
  width: 91.66667%; }
  .col-12 {
    width: 100%; }
}
```

Bonus: Using English word for the column name

The loop works because we changed our columns name from `.one`, `.two` to `.col-1` and `.col-2`. What if we really want to use the English word to define the column name? Here we go.

A Scss variable can be treated as a list, similar to array. We can separte list via *space* or *comma*. Let's define a list of English number from 1 to 20. I guess it's enough unless we need more than 20 columns in our grid system:

```
$numbers-text: one two three four five six seven eight
nine ten eleven twelve thirteen fourteen fifteen sixteen
seventeen eighteen nineteen twenty;
```

Then we can access any one of the values via the `nth()` Scss function. Just make sure we know that the list index begins at 1 instead of the 0. For example, the first one would be:

```
nth($numbers-text, 1);
```

Now we can define our columns with the English word by using this technique:

```
@media screen and (min-width: $breakpoint) {
  @for $i from 1 through $columns-count {
    .col-\#{$i},
    .\#{nth($numbers-text, $i)} {
      width: 100%/$columns-count * $i;
    }
  }
}
```

And this is what we got, assume the `$columns-count` is 4.

```
@media screen and (min-width: 500px) {
  .col-1,
  .one {
    width: 25%; }

  .col-2,
  .two {
    width: 50%; }

  .col-3,
  .three {
    width: 75%; }

  .col-4,
  .four {
    width: 100%; }
}
```

This article from Hugo provides a detail usage exampels on the Scss list.