avionschool

Lesson 3.2 Responsiveness Across Devices - Media Queries

BATCH 4

NOVEMBER 21, 2020

Media Queries

- are useful when you want to modify your site or app depending on a device's general type (such as print vs. screen) or specific characteristics and parameters (such as screen resolution or browser viewport width)

Media queries are used for the following:

- To conditionally apply styles with the CSS @media and @import at-rules.
- To target specific media for the <style>, <link>, <source>, and other HTML elements with the media= attribute.
- To test and monitor media states using the Window.matchMedia() and MediaQueryList.addListener() JavaScript methods.

Syntax

Media Types

- Media types describe the general category of a device.
- Except when using the `not` or `only` logical operators, the media type is optional and the `all` type will be implied.

all

- Suitable for all devices.

print

- Intended for paged material and documents viewed on a screen in print preview mode.

screen

- Intended primarily for screens.

speech

- Intended for speech synthesizers.

Media Features

- Media features describe specific characteristics of the user agent, output device, or environment.
- Media feature expressions test for their presence or value, and are entirely optional.
- Each media feature expression must be surrounded by parentheses.

Height & Width Media Features

- The height and width may be found by using the `height` and `width` media features
- Each of these media features may then also be prefixed with the min or max qualifiers, building a feature such as `min-width` or `max-width`.
- The `height` and `width` features are based off the height and width of the viewport rendering area, the browser window for example.
- Values for these height and width media features may be any length unit, relative or absolute.

```
@media all and (min-width: 320px) and (max-width: 780px) {...}
```

Orientation Media Feature:

- The `landscape` mode is triggered when the display is wider than taller
- The `portrait` mode is triggered when the display is taller than wider.

```
@media all and (orientation: landscape) {...}
```

Aspect Ratio Media Features:

- The `aspect-ratio` and `device-aspect-ratio` features specifies the `width/height` pixel ratio of the targeted rendering area or output device.
- The `min` and `max` prefixes are available to use with the different aspect ratio features, identifying a ratio above or below that of which is stated.

```
@media all and (min-device-aspect-ratio: 16/9) {...}
```

Resolution Media Feature:

- The `resolution` media feature specifies the resolution of the output device in pixel density, also known as dots per inch or DPI
- The `resolution` media feature does accept the min and max prefixes.
- Additionally, the resolution media feature will accept dots per pixel `(1.3dppx)`, dots per centimeter `(118dpcm)`, and other length based resolution values.

```
@media print and (min-resolution: 300dpi) {...}
```

`any-hover` - Does any available input mechanism allow the user to hover over elements?
`any-pointer` - Is any available input mechanism a pointing device, and if so, how accurate is it?
`aspect-ratio` - Width-to-height aspect ratio of the viewport
`color` - Number of bits per color component of the output device, or zero if the device isn't color
`color-gamut` - Approximate range of colors that are supported by the user agent and output device
`color-index` - Number of entries in the output device's color lookup table, or zero if the device does not use such a table

Media Features

`display-mode` - The display mode of the application, as specified in the web app manifest's `display` member

`forced-colors` - Detect whether user agent restricts color palette

`grid` - Does the device use a grid or bitmap screen?

`height` - Height of the viewport

`hover` - Does the primary input mechanism allow the user to hover over elements?

`inverted-colors` - Is the user agent or underlying OS inverting colors?

`monochrome` - Bits per pixel in the output device's monochrome frame buffer, or zero if the device isn't monochrome

Media Features

`orientation` - Orientation of the viewport

`overflow-block` - How does the output device handle content that overflows the viewport along the block axis?

`overflow-inline` - Can content that overflows the viewport along the inline axis be scrolled?

`pointer` - Is the primary input mechanism a pointing device, and if so, how accurate is it?

`prefers-color-scheme` - Detect if the user prefers a light or dark color scheme

`prefers-contrast` - Detects if the user has requested the system increase or decrease the amount of contrast between adjacent colors

`prefers-reduced-motion` -The user prefers less motion on the page

`prefers-reduced-transparency` - The user prefers reduced transparency

`resolution` - Pixel density of the output device

`scan` - Scanning process of the output device

`scripting` - Detects whether scripting (i.e. JavaScript) is available

`update` - How frequently the output device can modify the appearance of content

`width` - Width of the viewport including width of scrollbar

Syntax

Logical Operators

- The logical operators `not`, `and`, and `only` can be used to compose a complex media query.
- You can also combine multiple media queries into a single rule by separating them with commas.

and

- The and operator is used for combining multiple media features together into a single media query, requiring each chained feature to return true in order for the query to be true.
- It is also used for joining media features with media types.

not

- The not operator is used to negate a media query, returning true if the query would otherwise return false.
- If present in a comma-separated list of queries, it will only negate the specific query to which it is applied.
- If you use the `not` operator, you must also specify a media type.

Logical Operators

only

- The only operator is used to apply a style only if an entire query matches, and is useful for preventing older browsers from applying selected styles.
- When not using `only`, older browsers would interpret the query `screen and (max-width: 500px)` simply as `screen`, ignoring the remainder of the query, and applying its styles on all screens.
- If you use the `only` operator, you must also specify a media type.

, (comma)

- Commas are used to combine multiple media queries into a single rule.
- Each query in a comma-separated list is treated separately from the others.
- Thus, if any of the queries in a list is true, the entire media statement returns true.
- In other words, lists behave like a logical `or` operator.

Targeting media types

CSS TARGETS PRINTERS

```
@media print { ... }
```

TARGET BOTH SCREEN AND PRINT DEVICES

```
@media screen, print { ... }
```

Targeting media features

THIS EXAMPLE APPLIES STYLES WHEN THE USER'S PRIMARY INPUT MECHANISM (SUCH AS A MOUSE) CAN HOVER OVER ELEMENTS:

```
@media (hover: hover) { ... }
```

FOR EXAMPLE, THIS CSS WILL APPLY TO ANY DEVICE WITH A COLOR SCREEN:

```
@media (color) { ... }
```

THE STYLES NESTED INSIDE THE FOLLOWING QUERY WILL NEVER BE USED, BECAUSE NO SPEECH-ONLY DEVICE HAS A SCREEN ASPECT RATIO:

```
@media speech and (aspect-ratio: 11/5) { ... }
```

Breakpoint

- a breakpoint is the “point” at which a website’s content and design will adapt in a certain way in order to provide the best possible user experience.
- For developers, a breakpoint is a media query.
- For designers, it is the juncture at which a change is made to the way the website content or design appears to the viewer.

BEST PRACTICES FOR ADDING RESPONSIVE BREAKPOINTS:

Develop for mobile-first - By developing and designing mobile-first content, the developer and designer receives multiple benefits.

1. Developers address what is most necessary, and can then make additions to match the preferences of desktop users.
2. Since mobile devices are more challenging to design for due to smaller screens, developers and designers will end up making the tough choices at the very beginning. This saves them time later.
3. The load time of a page is much higher when one starts with CSS and smaller assets.
4. Developers and designers are compelled to take into account functional differences occurring between different devices.
5. There is less coding involved since block-level elements such as a div, heading or section expands to fill 100% of it’s parent by default.

Breakpoint

Use the following points to reduce friction

1. Prioritize important menu options
2. Remove anything visually distracting
3. Remove minor form fields
4. Highlight the main CTA
5. Focus on a robust search and filter function.
6. Always keep major breakpoints in mind. The former usually matches common screen sizes (480px, 768px, 1024px, and 1280px).
7. Before choosing major breakpoints, use website analytics to discern the most commonly used devices from which your site is accessed. Add breakpoints for those screen sizes first.

Hide or display elements at certain breakpoints - If necessary, switch content or features at breakpoints. For example, consider implementing off-canvas navigation for smaller screens and a typical navigation bar for larger ones.

Don’t define standard breakpoints for responsive design on the basis of device size - The primary objective of responsive design breakpoints is to display content in the best possible way. So, let the content be the guide. Add a breakpoint when the content and design requires it.

Breakpoint

FOLLOWING SCREEN SIZES HAVE BEEN MOST COMMONLY USED IN 2019 ACROSS THE WORLD

360×640

360×720

375×667

768×1024

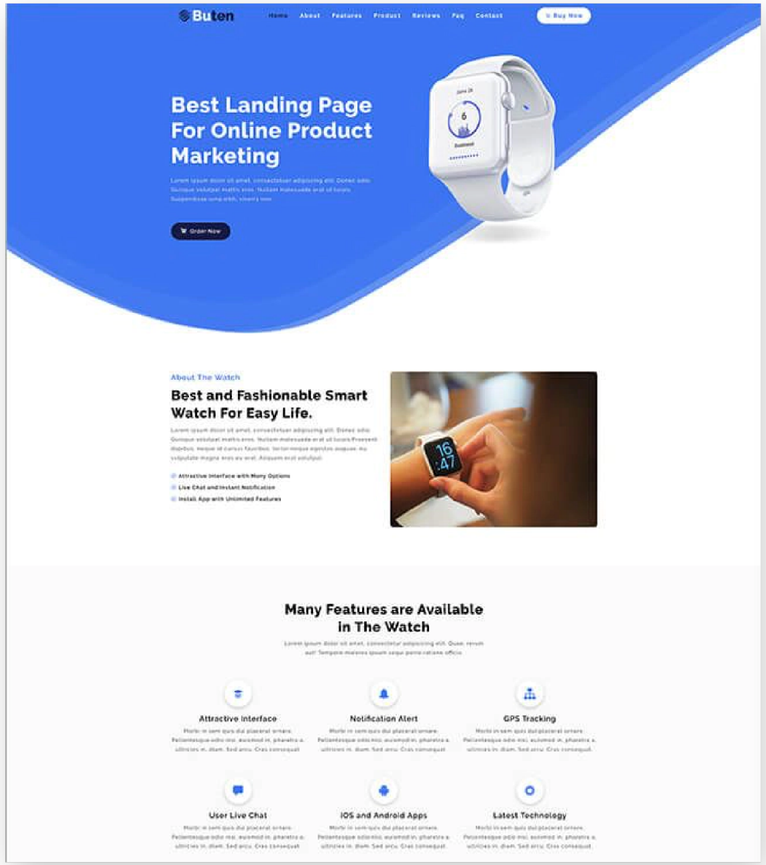
1366×768

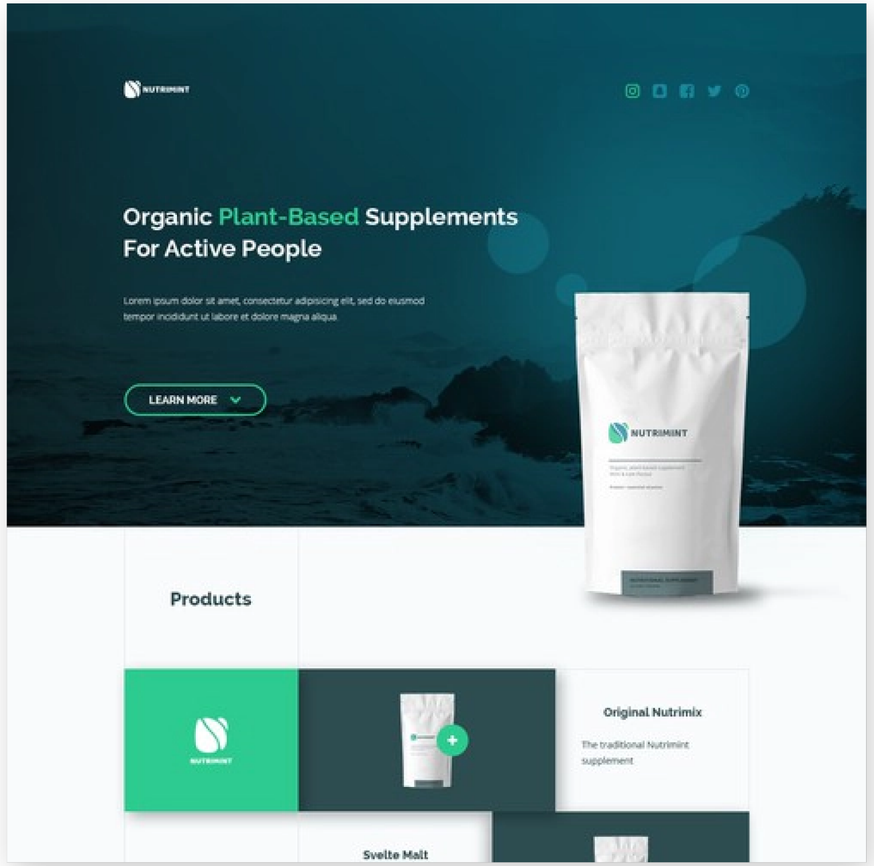
1920×1080

Project: Product Landing Page

DIRECTIONS

Choose your favorite product and follow the user stories in the following slides.





Project: Product Landing Page

USER STORIES

User Story #1: As a user, I want my product landing page to have a header element with a corresponding id="header".

User Story #2: As a user, I want to see an image within the header element with a corresponding id="header-img". A company logo would make a good image here.

User Story #3: Within the #header element I want to see a nav element with a corresponding id="nav-bar".

User Story #4: I want to see at least three clickable elements inside the nav element, each with the class nav-link.

User Story #5: As a user, when I click a .nav-link button in the nav element, I am taken to the corresponding section of the landing page.

User Story #6: As a user, I can watch an embedded product video with id="video".

User Story #7: As a user, my landing page should have a form element with a corresponding id="form".

Project: Product Landing Page

USER STORIES

User Story #8: Within the form, I want to see an input field with id="email" where I can enter an email address.

User Story #9: The #email input field should have placeholder text to let the user know what the field is for.

User Story #10: The #email input field uses HTML5 validation to confirm that the entered text is an email address.

User Story #11: Within the form, I want to see a submit input with a corresponding id="submit".

User Story #12: When I click the #submit element, the email is submitted to a static page.

User Story #13: The navbar should always be at the top of the viewport.

User Story #14: My product landing page should have at least one media query.

User Story #15: My product landing page should utilize CSS flexbox at least once.