educación IT
CODE YOUR FUTURE

Bootcamp Java Developer

Fase 2 - Java Web Developer

Módulo 14

Viewport y Breakpoints

Viewport

El `viewport` es el área visible de nuestro navegador. Podemos manipular cómo se ve, súper importante dada la variedad de dispositivos disponibles actualmente.

El `viewport` nos permite configurarlo de tal manera que pueda **ajustarse dinámicamente al tamaño de cada dispositivo usando el atributo `'device-width'`** que es equivalente al **100% del ancho de la pantalla** de dicho dispositivo, independientemente de su tamaño, posición o resolución. La configuración básica del **`viewport`** es:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

Variantes de Viewport

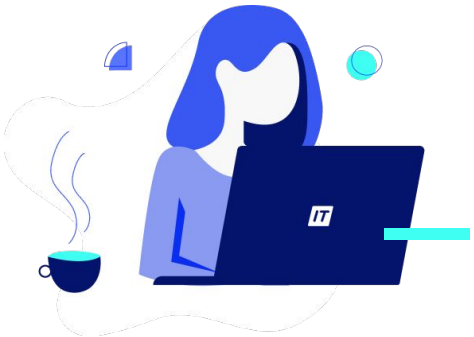
Existen medidas vinculadas al **`viewport`**.

`vw`	Relative to 1% of the width of the viewport
`vh`	Relative to 1% of the height of the viewport
`vmin`	Relative to 1% of viewport's smaller dimension
`vmax`	Relative to 1% of viewport's larger dimension

Para poder analizar con mayor detalle existen **herramientas de visualización**:

- Responsinator
- ResizeMe.
- Firesizer.

Este elemento es sumamente importante: aún realizando los estilos correctos, **si olvidamos la etiqueta `viewport`, nuestro navegador no tomará el cambio de tamaño de pantalla,** por lo que ningún estilo será tomado y la interfaz será siempre la misma para cualquier tamaño.



Notaremos que el **`viewport`** adquiere más o menos atributos, pero los más importantes son:

- **`name`**: **`viewport`**, o área visible.
- **`content`**: **`width`**, en este punto podemos trabajar con un ancho específico o jugar entre rangos, pero lo más usual es encontrar el valor **`device-width`**, que toma el tamaño del dispositivo a través del cual accede el usuario.
- **`Initial-scale`**: la escala inicial del documento.
- **`User-scalable`**: permite indicar si el usuario puede o no hacer zoom.

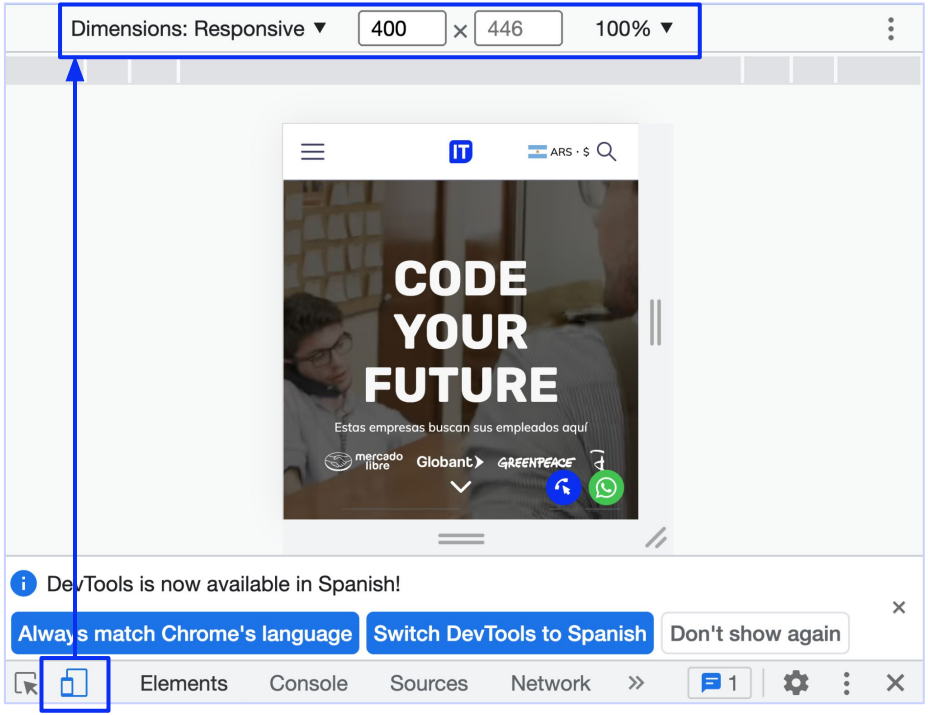
Escalas en Viewport

Se puede trabajar con **diferentes escalas en viewport**, estas propiedades **afectan a la escala inicial y al ancho**, además de l**imitar los cambios a nivel de zoom**.

```
<meta name=¨viewport¨ content=¨width=device-width; initial-scale=2¨>
<meta name=¨viewport¨ content=¨width=device-width; maximum-scale=2¨>
<meta name=¨viewport¨ content=¨width=device-width; user-scalable=no¨>
<meta name=¨viewport¨ content=¨width=device-width; initial-scale=1¨, maximum-scale=1¨>
```

Inspector de elementos

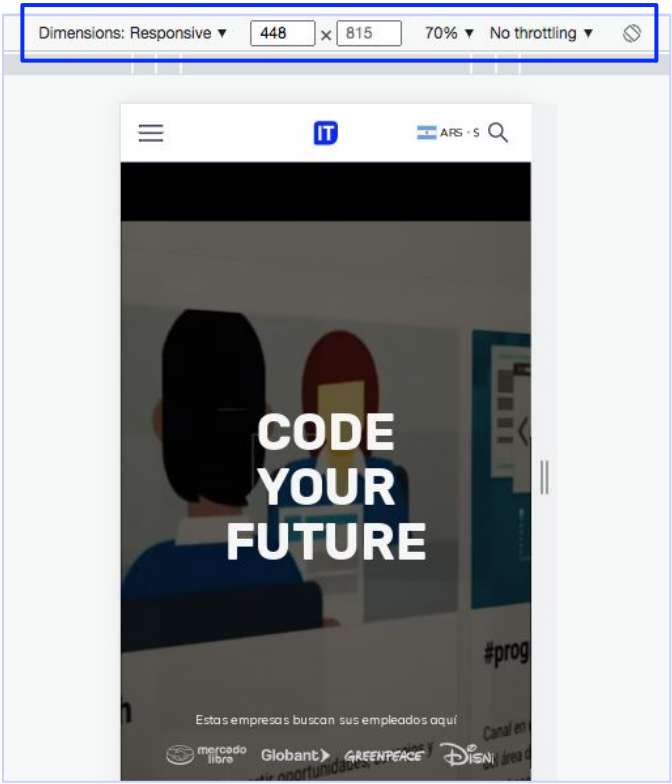
Otra opción, más simple, es usar el **inspector de elementos** de tu navegador (activar con **F12** o con botón derecho del mouse opción ***Inspector de elementos, Inspeccionar,*** o similares, dependiendo del navegador usado), que ofrece la posibilidad de **trabajar y visualizar nuestra interfaz en diferentes tamaños de pantalla** incluso con una lista de sugerencias con los posibles **`breakpoints`** generales a tener en cuenta.



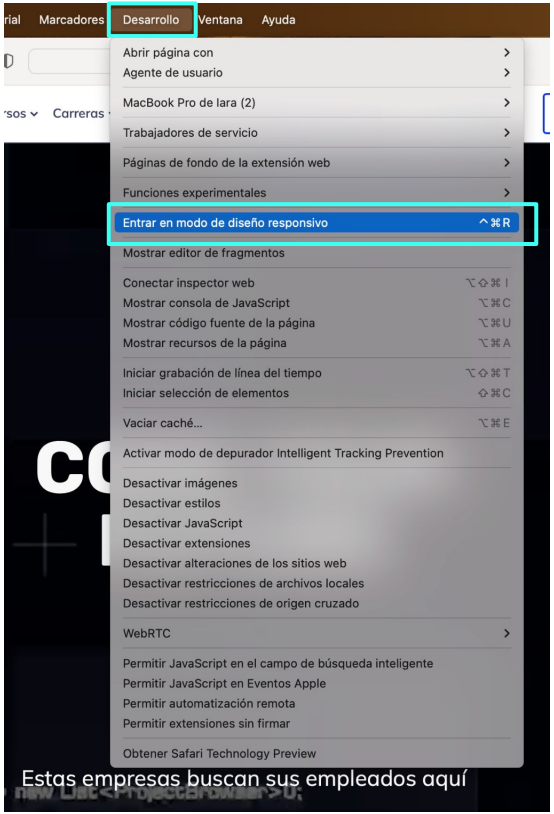
Ejemplo del Inspector de elementos de Chrome.

Así, nos permite trabajar con diferentes **`viewport`** y analizar si nuestros estilos están o no actuando correctamente de forma más segura.

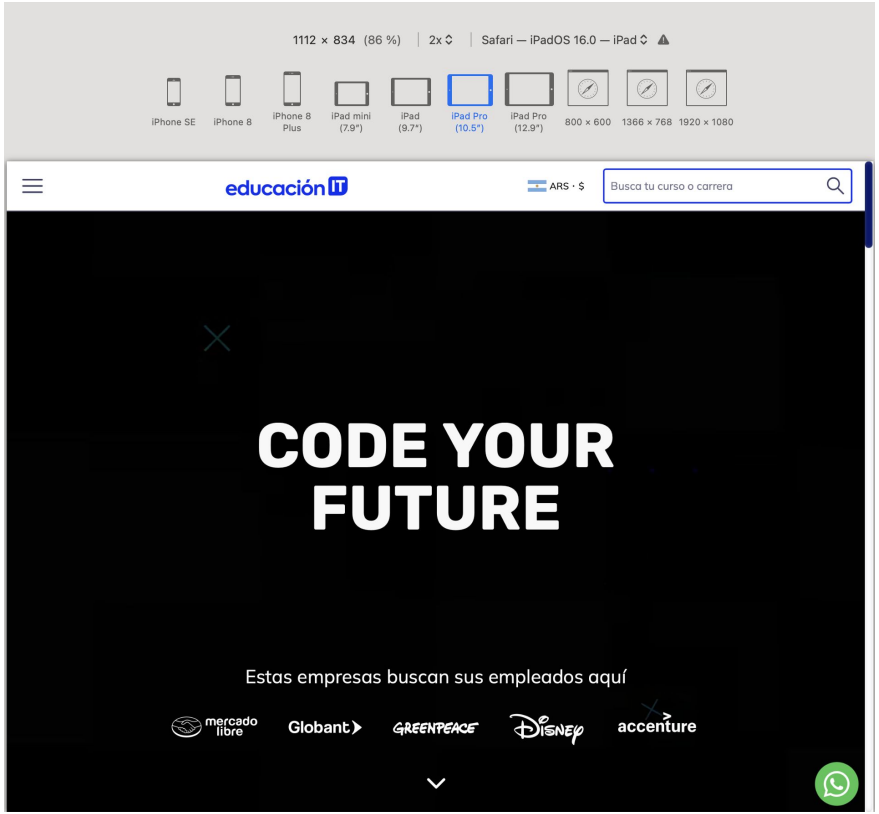
Ejemplo del Inspector de elementos de Chrome.



Si, en cambio, utilizas Safari, deberás presionar **Ctrl+Command+R** o ir a ***Desarrollo > Entrar en modo de diseño responsivo,*** y se mostrará en la pantalla opciones para modificar las dimensiones de forma manual y botones con posibles **`breakpoints`** generales para acceder a ellos rápidamente, como se muestra en la pantalla siguiente.



Ejemplo de las herramientas de Desarrollo de Safari.



Ejemplo del modo de diseño responsivo de Safari

Breakpoints

`Breakpoints` son bloques de diseño responsivo. Es importante su utilización para controlar o adaptar la interfaz a un tamaño de pantalla específico.

Existen diversos `breakpoints`, pero usualmente **se recomienda maquetar para 3 tamaños específicos:**

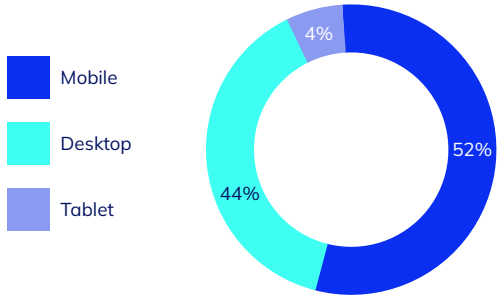
- 768px.
- 992px.
- 1200px.

Sin embargo, como mencionamos antes, existen diferentes tamaños de pantalla y dispositivos. Se hacen constantemente estudios para entender esa accesibilidad variada y el resultado está en el obvio proceso de trabajo responsivo puesto que es imposible generar una sola versión de nuestra interfaz para poder adaptarla a todos estos diferentes accesos.

Testeo constante

Nuestro testeo debe ser constante. Por ejemplo, no hace mucho en los navegadores se podía subir un video con audio y `autoplay`. Sin embargo, recientemente, primero Google Chrome y luego otros navegadores, **restringieron el `autoplay` a videos con audio**, por tanto para poder reproducirlos **debemos trabajar con `muted`** como característica ya preseteada para lograr que nuestro video se trabaje en reproducción automática.

El ejemplo anterior es solo uno entre tantos, del mismo modo que el testeo constante en el diseño responsivo, por lo tanto **siempre debemos estar al tanto de las novedades tecnológicas.**

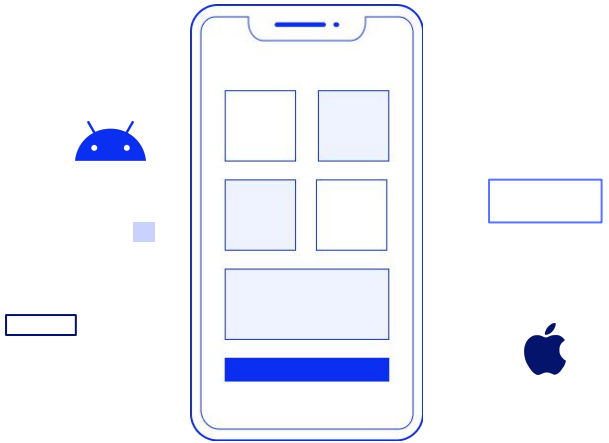


Traffic distribution by device type (Q3 2019)
Fuente: BrowserStack

Aplicaciones, fragmentación

Si bien no es parte de este curso, es importante entender que como maquetadores podemos trabajar en aplicaciones que también deben considerar la fragmentación para diferentes dispositivos.

Sin embargo, también en este trabajo se debe dar la característica colaborativa, de diseñador, maquetador, desarrollador tanto Back como Front, o en el caso de las aplicaciones el trabajo conjunto con un desarrollador Android o iOS, dependiendo el caso.



Regla @media

@media

Para poder trabajar con **breakpoints** desde CSS, debemos trabajar con la regla **@media**. Ésta nos permite **marcar los "cortes" en la interfaz** y avanzar sobre los estilos correctos al momento de adaptar nuestra interfaz.

Si bien puede trabajar con diversas variantes, generalmente se emplean las propiedades **min-width** y **max-width**. Estos rangos establecen los diferentes **breakpoints** hacia cuales irán orientados nuestros estilos.

```
@media (min-width: 993px){

    body { background-color: red;}
    section { width: 50%; display: flex;  }

}
```

min-width & max-width

Veamos algunos ejemplos típicos a establecer.

```
@media (max-width: 992px){
    body { background-color: blue;}

}

@media (max-width:768px){
    section { width: 100%; display: flex; flex-direction: column;}
    main { flex-direction: column;}
    body { background-color: yellow;}
}
```

En el segundo ejemplo del slide anterior, ordenamos al navegador que cuando la pantalla sea menor a 768px (inclusive) se modificará el **`flex`** de los elementos **`section`** pasándose a **`column`**. También modificamos la **`direction`** del **`main`** y el cambio de color del fondo a amarillo.

Así, conociendo los **`breakpoints`** y sus particularidades, vamos a destinar estilos específicos para cada uno.

En el ejemplo a la derecha, trabajando con **`min-width`** establecemos ciertos parámetros para cuando la pantalla sea mayor o igual a 993px, por ejemplo, cambiando el color de fondo, esta vez, a rojo.

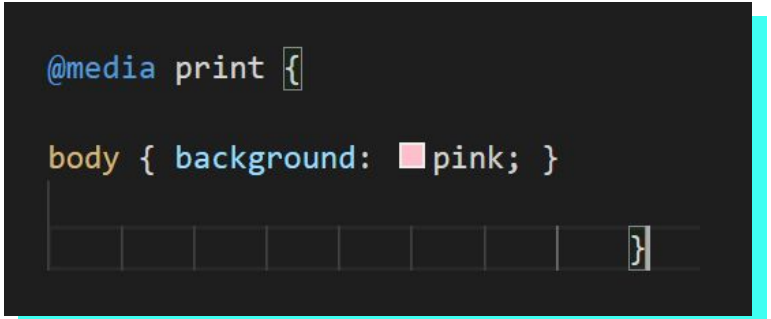
```
@media (min-width: 993px){

    body { background-color: red;}
    section { width: 50%; display: flex;  }

}
```

Otras variantes de @media

También podemos destinar nuestros estilos a medios específicos, por ejemplo, la impresión. Estos estilos podemos confirmarlos cuando vemos la vista previa de impresión donde podemos observar si se están llevando a cabo o no estás indicaciones.



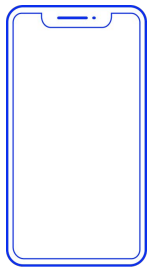
Variantes avanzadas de @media

Podemos realizar una regla más compleja, por ejemplo, incluyendo **varios medios o un rango de medida.**

Es importante recordar que **el último estilo es el que prevalece,** por ello debemos **asegurarnos de que no haya estilos que se sobreescriben** o pisen estilos anteriores por encontrarse dentro del mismo rango o medio.

```
@media print {
    body { font-size: 10cm }
}
@media screen {
    body { font-size: 15px }
}
@media screen, print {
    body { line-height: 1.2em }
}
@media only screen
    and (min-device-width: 576px)
    and (max-device-width: 768px)
{
    body { line-height: 1.4 }
}
```

Un uso muy común utilizado desde el uso masivo de dispositivos móviles es la **orientación de nuestra pantalla.** Para eso utilizamos **`landscape`** o **`portrait`**:



Portrait

El alto es mayor o igual al ancho.

Landscape

El ancho es mayor al alto.

```css
@media (orientation: landscape) {
  body {
    flex-direction: row;
  }
}

@media (orientation: portrait) {
  body {
    flex-direction: column;
  }
}
```

¡Sigamos trabajando!