

```
/* Introducción a */  
.nomenclaturas--css {  
    MURCIA FRONTEND - 22/02/2017  
    Javier Sánchez Riquelme  
    @Xavis  
}
```



#Javier.SánchezRiquelme {

Trabajo: Web, Mobile and AR engineer

Empresa: ANSWARE TECH,

Edad: 24,

Twitter: @xavis,

Telegram: @xaviscrypt,

BattleNet: xavis#2298,

/* LeagueOfLegends: EternalTaxis */

}



Es importante tener
un estilo común y
“universal” al escribir
CSS



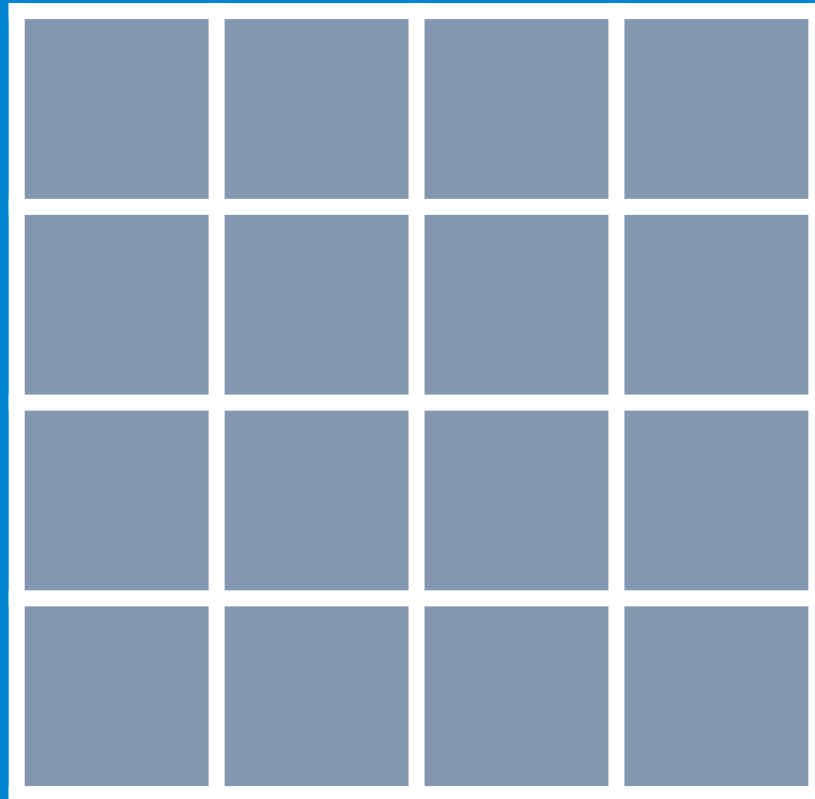
LOS 3 PROBLEMAS



Problema 1: Desarrollador IKEA



A MAQUETAR




```
<table class="container">
```

A MAQUETAR


```
<td class="cell">
```

IKEA MODE ON



```
<table class="kallax">
```


IKEA MODE ON



```
<td class="lekman">
```

```
.kallax {  
    border-color: white;  
}
```

```
.lekman {  
    border-color: red;  
    height: 35px;  
    width: 35px;  
}
```

```
/* ¿A QUÉ SE REFIERE ESTO? */
```

¡Y HAY QUE EVITAR LOS
CAMBIOS DE NOMBRE!

EXPEDIT



KALLAX



```
.container {  
    border-color: white;  
}
```

```
.cell {  
    border-color: red;  
    height: 35px;  
    width: 35px;  
}
```

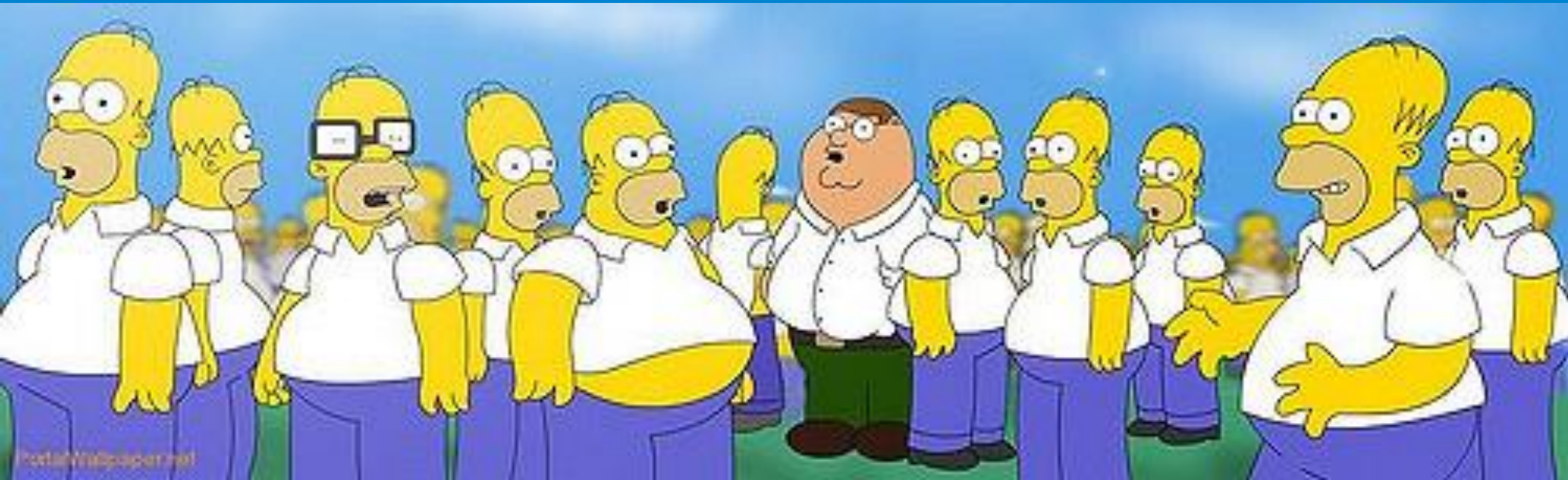
```
/* MEJOR, PERO PUEDE TRAER PROBLEMAS */
```

```
.container {  
    border-color: white;  
}
```

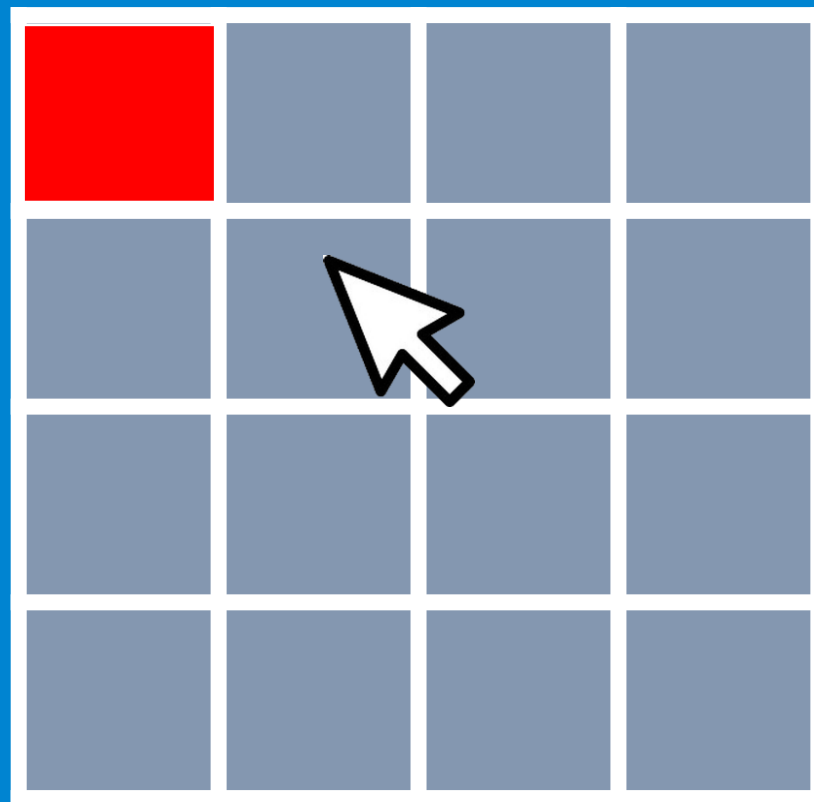
```
.container-cell {  
    border-color: red;  
    height: 35px;  
    width: 35px;  
}
```

```
/* MÁS INFO AL REVISAR DESDE HTML */
```

Problema 2: Repetición de código



NUEVO EFECTO



```
.container-cell {  
    [...]  
    transition: all 1s;  
}
```

```
.container-cell:hover {  
    background-color: red;  
}
```

```
/* MÁS INFO AL REVISAR DESDE HTML */
```


¿Y si quisiéramos
reusar el mismo
comportamiento?

```
._animation1s {  
    animation: all 1s;  
}  
  
._onHoverRed:hover {  
    background-color: red;  
}
```

```
<td class="container-cell _animation1s  
    _onHoverRed">
```

Problema 3: Unificar estilo de código



```
.my-header .link {  
    display: inline-block;  
    padding: 5px 10px;  
}
```

header.CSS

```
.main_content a {  
    text-decoration: underline;  
    color: red;  
}
```

main.CSS

```
.myFooter > a.link {  
    text-transform: uppercase;  
    font-size: 1.5em;  
}
```

footer.CSS

METODOLOGÍAS CSS



OOCSS – Object Oriented CSS

- Representa más una filosofía básica que una metodología, al no ofrecer una semántica del código.
- Intenta aplicar las técnicas de POO a CSS. Separando los contextos de estilo por “clases”.
- Entiende una clase como un elemento padre que queremos que se mantenga independiente del resto de la aplicación, haciéndolo más reusable.
- Sus hijos siempre serán referenciados usando el padre y evitar usar etiquetas puras como hijos.
- Separación de reglas de estructura y embellecimiento. Como Bootstrap.

OOCSS

```
.button { width: 200px; float: left; }
```

```
.button-primary { background: #33f; border-width: 0.5em; }
```

```
.button-small { width: 100px; }
```

```
.button .right-side { float: right; margin: 0 1em; }
```

```
.button .left-side { float: left; margin: 0 1em; }
```

```
.button span {}
```

```
.button .title { font-size: 2em; text-decoration: none; }
```

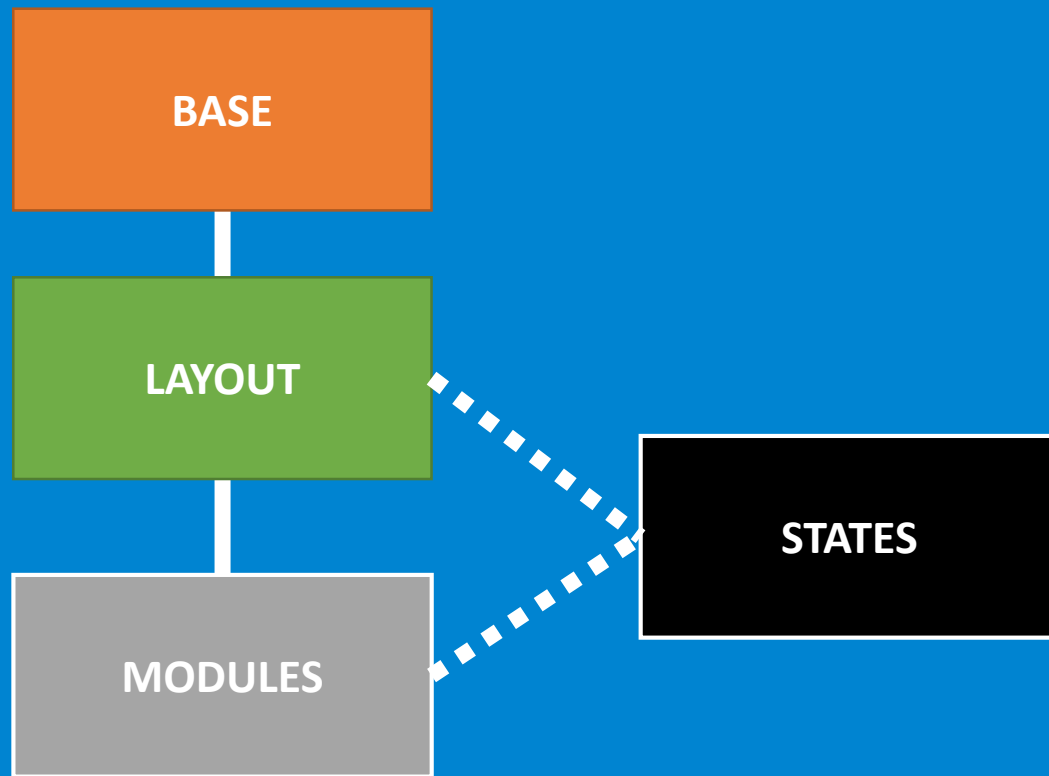
```
.button i {}
```

```
.button .icon { font-size: 2em; color: #0f0; }
```

SMACSS – Scalable and Modular Architecture for CSS

- Metodología creada por Jonathan Snook.
- Además de poder separar el estilado de la estructura como OOCSS, separa las reglas CSS en base a la funcionalidad.
- Pide evitar un gran nivel de profundidad al especificar estilos. Así se elimina la especialización.
- Favorece la reutilización y comprensión de uso.
- Distingue 4 secciones básicas de CSS en todo proyecto.

SMACSS – Estructura de secciones CSS



SMACSS – Sección CSS 1: BASE

- Estilos básicos de la aplicación web.
- Reseteará estilos por defecto del navegador y definirá las reglas básicas para nuestras etiquetas.
- No requiere de ninguna nomenclatura especial.

```
html { margin: 0; padding: 0;}
```

```
a { color: #aee; text-decoration:none; }
```

```
img { border: 0}
```

SMA CSS – Sección CSS 2: LAYOUT

- Estilos de la estructura de nuestro sitio.
- Aquí definiremos la cabecera, las secciones, nuestros sidebars y el pie de página.
- Aunque no es obligatorio se recomienda usar el prefijo `l-*` o `layout-*` para las clases de esta sección.

```
.l-header { height:2em; position: fixed; }  
  
.l-main { width:70%; display: inline-block; }  
  
.l-sidebar { width: 25%; float:right;}  
  
.l-grid td { display: flex; flex-wrap: wrap; }
```

SMACSS – Sección CSS 3: MODULES

- Estilos los módulos independientes reutilizables.
- Se pide no usar etiquetas genéricas (div, ul, span...) en nuestro CSS ya que puede generar conflictos.
- La nomenclatura de esta parte se basa en clases sin prefijo y subclases nombradas padre-hijo.

```
.box { width: 200px; }
```

```
.box-constrained { max-width: 100px; }
```

```
.box-title { font-size: 2em; color: #33a;}
```

```
.box-description { color: #333; padding: 3em; }
```

SMACSS – Sección CSS 4: STATES

- Los estados son alteraciones de un módulo o layout.
- Estos aparecen tras interacciones JavaScript o son añadidos mediante la creación dinámica de HTML.
- Su nomenclatura se basa en clases nombradas con guiones que suelen comenzar con `.is-*`

```
.is-active { border: 1em solid white; }
```

```
.is-hover { opacity: 0.6; }
```

```
.cal-today { background-color: #0f0; }
```

```
.date-soon { color: #333; padding: 3em; }
```

SMACSS – Sección CSS 5: ¡EXTRA! THEMES

- Esta sección no forma parte del núcleo de SMACSS.
- No suele ser usado en la mayoría de proyectos.
- Sirve para definir estilos propios sobre una aplicación ya desarrollada que quiere ser personalizada.
- Suele basarse en cambios de colores o tipografía.
- No requiere ninguna nomenclatura extra, pero se pide realizar a parte en lugar de sustituir el ya definido.

BEM – Block, Element, Modifier. Resumen.

- Metodología creada por **Yandex.com**
- Divide los estilos en las 3 partes que definen su nombre: **Bloques**, **elementos** y **modificadores**.
- Un **bloque** es el componente independiente capaz de ser reutilizado. Un **elemento** será una parte de un bloque que no podrá usarse sin pertenecer a él. Un **modificador** alterará la apariencia o estado de un bloque.
- Un **elemento** de un bloque puede **combinarse** con un **bloque** para obtener propiedades de ambos en contextos especiales. A esto se le conoce como **Mix**.

BEM – Nomenclatura

- Los **Bloques** se nombran sin ningún tipo de prefijo y siguen la nomenclatura con guiones simples. search-form, file-input, sidebar, header-menu, post.
- Los **Elementos** se escriben usando el nombre del bloque, dos guiones bajos y el nombre del elemento. .search-form__textbox, .post__title, .sidebar__
- Los modificadores se escribirán separándolos de su prefijo únicamente con un guión bajo. Si el nombre es compuesto usarán varios guiones bajos como separación.
- .search-form_active, .file-input__button_size_s

BEM – Estructura de directorios

- Todos los **elementos** relacionados con un **bloque** así como el mismo bloque serán contenidos en **un solo directorio**. El directorio tendrá **el mismo nombre** que el bloque.
- Otros elementos, modificadores y sub-bloques pueden estar en subdirectorios.
- La implementación de un bloque, elemento o modificador se dividirá en archivos por tecnología (css, js, html...).
- Los subdirectorios de elementos tendrán dos guiones bajos de prefijo (`__*`), los de modificadores uno (`_*`).

```
.menu {  
    width: 100%; background-color: black;  
}
```

menu/menu.css

```
.menu__item {  
    display: inline-block; padding: 1.5em;  
}
```

menu/__item/__item.css

```
.menu__item_active{  
    background-color: #666; color: white;  
}
```

menu/__item/_active/_active.css

SUIT CSS

- Metodología para desarrollo basado en componentes reutilizables.
- Ofrecen distintas herramientas como preprocesadores, herramientas de testing, además de un conjunto de estilos ya disponibles mediante NPM. De
- Su semántica es simple y no es tan exigente como BEM.
- Sus paquetes ya permiten la inserción de dependencias de manera automática agilizando el proceso de poner en marcha un proyecto. Esto se recomienda para proyectos pequeños.

SUIT CSS – Semántica (I)

- A diferencia del resto de nomenclaturas, en SUIT las clases se definen usando **CamelCase** para **componentes** y **camelCase** para **elementos** y **modificadores**.
- A los **componentes** se los nombra **sin ningún prefijo**, simplemente su **nombre**. Ej. **MyComponent**.
- A las partes del componente se las nombra usando como prefijo al padre más un guión simple.
Ej. HeaderMenu-itemList.
- Los modificadores se nombrarán usando dos guiones.
Ej. HeaderMenu-itemList--light

SUIT CSS – Semántica (II)

- En SUIT también se definen **estados** y **utilidades** en CSS,
- Los estados, al igual que en SMACSS se nombrarán como clases a parte que suelen estar nombradas con `.is-*`. Donde `*` sigue camelCase.
Ej: `.is-active`, `.is-today`.
- Las utilidades son modificaciones en estilo o estructura que pueden usarse para múltiples componentes o partes del proyecto. Se nombran con el prefijo `.u-*`. Usa camelCase
Ej: `.u-onLeft`, `u-fullWidth-`
- Los modificadores se nombrarán usando dos guiones.
Ej. `HeaderMenu-itemList--light`

SUIT CSS

```
.FileInput {  
    display: inline-block; margin: 1em;  
}  
  
.FileInput-button {  
    background-color #0f0; border-radius: 0.5em;  
}  
  
.FileInput-button--cancel {  
    background-color: #f00;  
}  
  
.is_disabled {  
    pointer-events: none;  
}  
  
._fullWidth {  
    width: 100%;  
}
```

Es importante tener
un estilo común y
“universal” al escribir
CSS



Todo es más fácil si usas
SASS



SUIT

```
.FileInput
  [...]
  &-button
    [...]
    &--cancel
      [...]
```

BEM

```
.file-input
  [...]
  &__button
    [...]
    &_cancel
      [...]
```

```
.button  
  width: 200px;  
  height: 60px;  
  background-color: #99ASE9;  
  border-radius: 10px;
```

```
.button-primary  
  @extend .button;  
  background-color: #33E9AS;
```

```
/* Introducción a */  
.nomenclaturas--css {  
    ¿PREGUNTAS?  
}
```

Error: Unexpected symbol "¿" on line 3