Universidad Nacional de La Plata

Facultad de Informática

Interfaces Adaptadas a Dispositivos Móviles 2014

==========

Trabajo Final

CSS Grid

Una especificación nativa para el diseño de interfaces en la web

Coordinadora

Prof. Lic. Ivana Harari

Disertante

APU Lucio Flavio Di Giacomo Noack

Noviembre 2017

0 Resumen

El presente informe describe la incorporación y utilización de CSS Grid como nuevo estándar para diseñar la presentación de los elementos y contenidos en una plataforma web. En la primera parte del mismo se realizará un breve recorrido histórico sobre las técnicas y metodologías utilizadas para diseñar en la web, remarcando los aportes innovadores y los problemas aparejados con cada uno de ellos. Una vez inmersos en el contexto del diseño de interfaces web, en la segunda parte de este documento vamos a estudiar la propuesta y el alcance de CSS Grid, recorriendo con detalle todos sus componentes y la terminología necesaria y válida para trabajarlos. En la tercera parte vamos a involucrarnos en un caso de estudio, comparando las implementaciones de una plataforma web típica, una de ellas utilizando el framework Bootstrap 3 y la otra utilizando CSS Grid, analizando los cambios propuestos por la nueva tecnología y con particular énfasis sobre su repercusión en el desarrollo *mobile*. Finalmente, en la cuarta parte de este trabajo se compartirán las conclusiones producto de la elaboración del mismo.

1 Introducción

El primer sitio web fue publicado por Tim Berners-Lee en Agosto de 1991, y contenía básicamente texto en una única columna, con enlaces a otras páginas también basadas en texto. La colección de etiquetas disponibles para estructurar el contenido de estas primeras páginas estaba limitada a encabezados, párrafos, y enlaces, y debido a esto los sitios eran sumamente pobres en lo visual pero ricos desde un punto de vista semántico. Cada etiqueta aportaba a la presentación pero también al significado de su contenido dentro del documento.

En la segunda versión de HTML se agregaron nuevas funcionalidades y etiquetas, entre ellas para incluir imágenes y presentar tablas. Las tablas, originalmente pensadas para mostrar de manera tabular información estructurada y relacionada, permitieron que los diseñadores se animaran a proyectar sitios más atractivos, ignorando la importancia del valor semántico de la etiqueta, prefiriendo la estética por sobre la estructura del documento, y aprovechándola para crear diseños de múltiples columnas, con imágenes de fondo, secciones de menú, encabezados personalizados, etc. Los problemas subyacentes a esta forma de maquetar los sitios se volvieron cada vez más evidentes a medida que el valor de su contenido crecía frente al de su presentación, y además, esta última debía adaptarse a

los nuevos dispositivos desde los que los usuarios accedían a ellos. El avance de la Web 2.0, la optimización para motores de búsqueda (SEO) con el objetivo de lograr un mejor posicionamiento, los requerimientos de accesibilidad, y el uso de dispositivos móviles, fueron alguno de los motivos por los que se volvió necesario el uso de otras tecnologías y metodologías para el maquetado.

A partir del año 2000 los sitios comenzaron a utilizar diseños basados en hojas de estilo en cascada (CSS), lo que permitía separar los elementos de presentación de los contenidos, dando como resultado documentos de menor tamaño, con valor semántico, y más fáciles de mantener. Para simplificar el trabajo de los maquetadores se implementaron frameworks en CSS, como por ejemplo Bootstrap o Foundation, que unifican y resuelven los requerimientos visuales comunes a la mayoría de los sitios web, impusieron el diseño en rejilla, y pusieron a consideración la adopción de nuevos estándares, como CSS Flexbox y CSS Grid, siendo este último el eje de estudio en el presente trabajo.

2 Evolución de las interfaces web

Desde sus orígenes hace más de 25 años, los sitios web han crecido en cantidad, calidad, y potencialidad, evolucionando desde documentos simples enlazados consultados en las computadoras de las universidades hasta plataformas complejas utilizadas concurrentemente por miles de usuarios desde múltiples dispositivos. La búsqueda de alcanzar un máximo atractivo visual conjugada con la creatividad para brindar funcionalidad dieron origen al espiral de iteraciones de implementación y revisión que marca el rumbo del diseño de los sitios web. A continuación describiremos brevemente las principales técnicas utilizadas desde los primeros sitios web hasta la actualidad, con el objetivo de describir el contexto que da origen a CSS Grid y poder apreciar sus posibilidades.

2.1 HTML Tables

La capacidad de incorporar imágenes y tablas permitió dar un verdadero salto de calidad en la presentación de los sitios web. Las imágenes habilitar a los diseñadores a incorporar fondos, enlaces con formas de botones con tipografías y colores atractivos. Las tablas, pensadas para disponer datos relacionados, trajeron con ellas una posibilidad implícita en principio insospechada: organizar los elementos de un sitio web de forma tal que no sea sólo un documento sino una plataforma. Los enlaces, entonces, pasaron a integrar barras de navegación y paneles de búsqueda; las imágenes se dispusieron para integrar logos,

encabezados y pies de página. Incluso la combinación de ambas novedades permitieron crear sitios en base a imágenes que anteriormente hubieran resultado inimaginables.

Las tablas, expandidas a lo ancho del documento, solían contener una fila para el encabezado, otra para la barra de navegación, y luego presentar contenidos específicos del sitio, en presentaciones de dos o tres columnas. En esta época, además, comenzaba a expandirse la utilización de tecnologías para servir sitios web generados dinámicamente, tales como CGI (1993) y PHP (1995), dando forma al esquema de *layout* y contenidos y resaltando la relevancia de la presentación de la informació, lo que daría origen a CSS (1996); años más tarde esto sería la base de la Web 2.0.

Como contrapartida, el uso de imágenes como parte fundamental de los sitios complicó aún más la tarea de las tecnologías de asistencia, que no podían interpretar el texto presentado como mapa de bits si el creador del documento no lo hacía explícito en la etiqueta de la imagen. En el caso de las tablas, las consecuencias fueron más graves: el uso de tablas para posicionar los elementos del sitio era completamente incompatible con el valor semántico de la etiqueta dentro del documento, por lo que la presentación se combinaba con el contenido y daba como resultado documentos prácticamente inutilizables fuera del contexto en el que el desarrollador los había pensado, es decir, determinados tipos y tamaño de pantallas, tecnologías para ingresar datos, entre otros.

En la imagen 1 podemos apreciar un ejemplo de un sitio web que utiliza un *layout* de tres columnas creado con HTML Tables, un diseño típico de los primeros sitios web complejos.

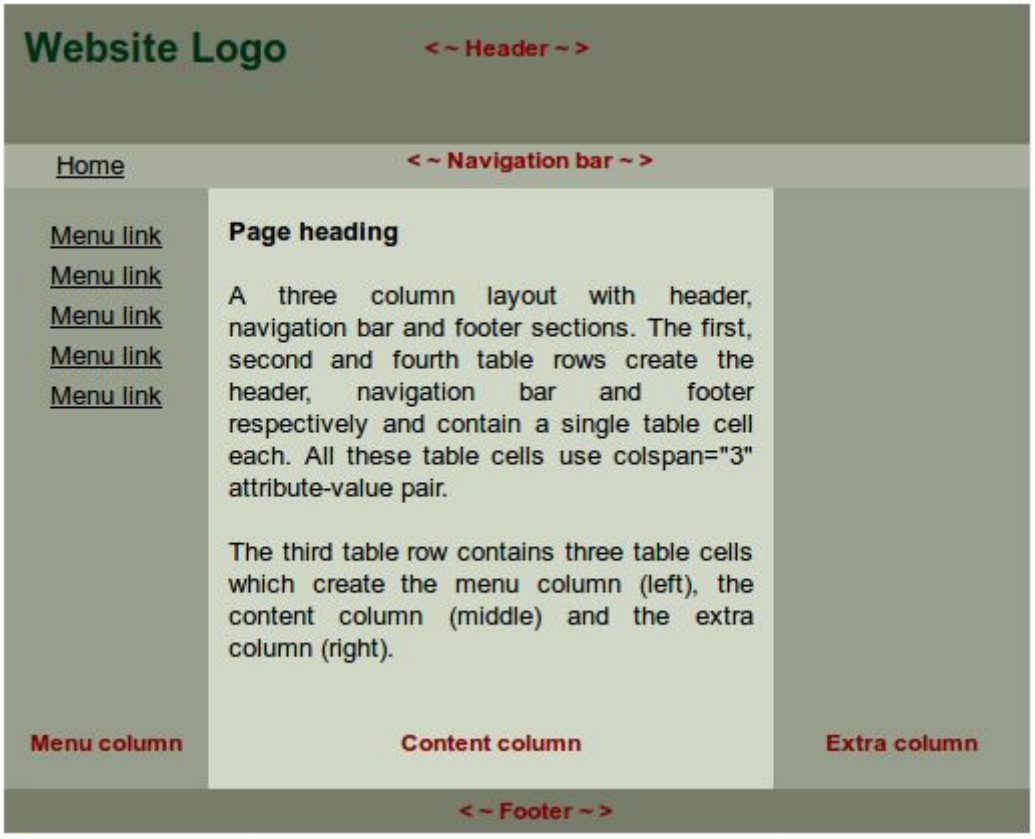

Imagen 1. Sitio web con *layout* de tres columnas implementado con HTML Tables

2.2 CSS Floats

La introducción de CSS (Cascading Style Sheets) como tecnología para definir la presentación de los elementos en un sitio web permitió a los diseñadores separar sus diseños de los contenidos. Con esta novedad ya no era necesario utilizar etiquetas exclusivamente para determinar cómo se presentaba la información sino que se podían determinar estilos en secciones o archivos independientes, asociados mediante el nombre de las etiquetas, sus identificadores o clases. La posibilidad de crear documentos web semánticamente correctos y visualmente atractivos volvió a ser real, con un sustento tecnológico adecuado.

Dos de las principales propiedades de CSS para disponer espacialmente de los elementos en un documento web y de esta forma diseñar layouts son *display* y *float*. Utilizando display, los elementos de bloque podían mostrarse como elementos de línea, y viceversa; adicionalmente, los elementos de línea pueden flotar a la izquierda o a la derecha de sus pares. Esto permite, por ejemplo, crear layouts de dos o tres columnas visualmente similares a los que anteriormente se creaban con tablas, pero semánticamente correctos,

utilizando elementos *div*. Adicionalmente se puede utilizar la propiedad *clear*, que configurada en un elemento de bloque con el valor *both* extiende a los *div* flotados hasta el final del bloque.

En la imagen 2 podemos ver tres elementos *div* configurados con dimensiones y bordes de colores para mostrarse como cajas, siendo una verde, otra roja, y la última azul. Las cajas verde y roja se muestra como elementos de línea (*display: inline*), con la caja roja flotando a la derecha (*float: right*). La caja azul se muestra como elemento de bloque, con la propiedad para estirar las cajas precedentes hasta la posición de ésta (*clear: both*). Como podemos apreciar, el resultado final es similar a un layout de dos columnas.

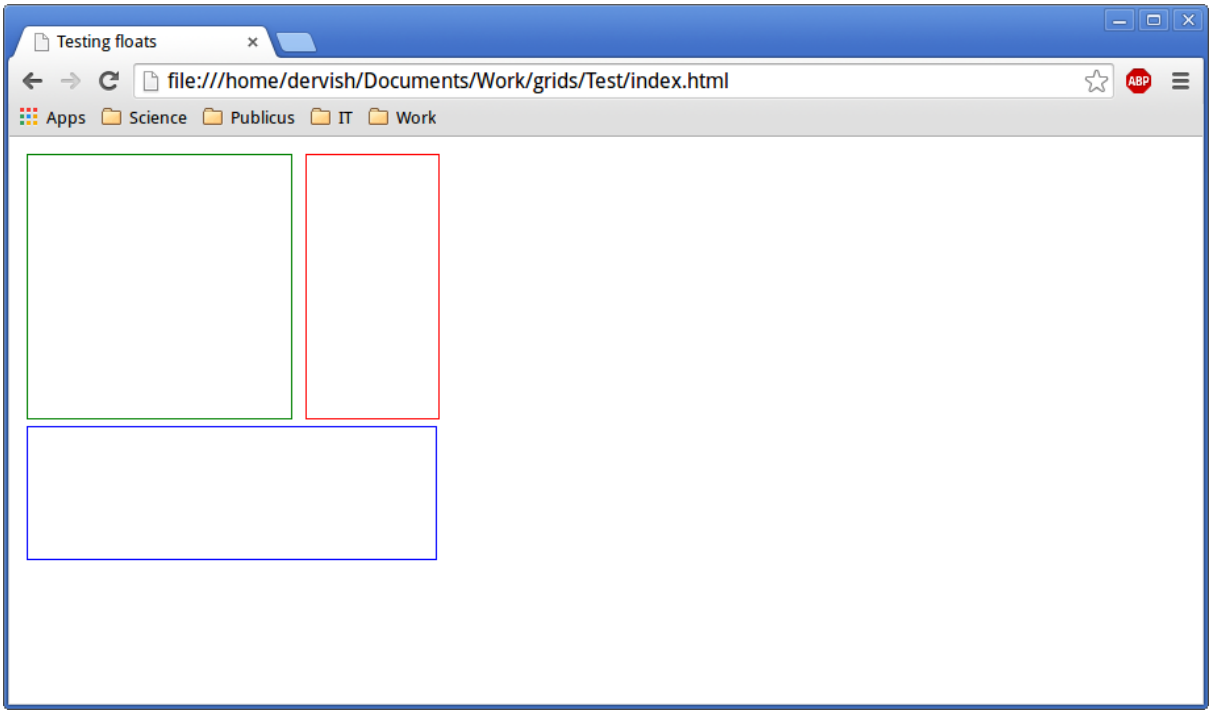

Imagen 2. Layout de dos columnas implementado con CSS Floats

2.3 CSS Frameworks

Con la proliferación del uso de CSS y de layouts de múltiples columnas comenzaron a hacerse comunes muchas prácticas con sus problemas aparejados. Un caso emblemático es el uso de *floats* con dimensiones fijas, lo que rompía los diseños de los sitios y hasta los volvía inutilizables si se los accedía desde dispositivos con determinadas resoluciones de pantalla, en especial desde los móviles, ya en poder de gran cantidad de usuarios en todo el mundo. Otro problema era la visualización desde distintos navegadores: una misma

implementación se presentaba diferente de acuerdo al motor de renderizado incluido en el navegador.

Para solucionar estos inconvenientes, comunes a miles de diseñadores, comenzaron a desarrollarse bibliotecas que unificaban soluciones. Además, dieron forma a los layouts *grid*, la verdadera abstracción detrás de los antiguos layouts con tablas. El proyecto Twitter Blueprint marcó un punto de inflexión en este aspecto cuando en el año 2011 se convirtió en Bootstrap, un framework para el diseño de sitios web que garantiza una presentación uniforme en distintos navegadores y además provee un sistema grid para configurar layouts donde disponer los elementos del sitio.

La propuesta de Bootstrap, luego adoptada por otros frameworks web como Foundation o Tuk Tuk, consiste en definir un elemento contenedor (*container* o *container-fluid*) que se divide en filas (*rows*) y estas a su vez en columnas (*col-*-**). Cada row cuenta con 12 franjas en las cuales pueden acomodarse las columnas. A su vez, la configuración de las columnas puede realizarse de acuerdo a la resolución de la pantalla en la que se presenta, gracias a *media queries*, incorporada en CSS3. Entonces, por ejemplo, una columna puede ocupar 9 franjas en dispositivos con una resolución horizontal mayor a 768px (*col-md-9*), y ocupar la totalidad del ancho de pantalla cuando deba mostrarse en resoluciones inferiores (*col-sm-12*). Las columnas pueden contener rows, y así indefinidamente, permitiendo implementar diseños realmente complejos.

Como contrapartida, los frameworks demandan la inclusión de elementos exclusivamente para presentación, principalmente divs contenedores, filas, y columnas, lo que en la práctica contamina la riqueza semántica del contenido del documento web en favor de la presentación.

En la Imagen 3 podemos apreciar un sitio web de ejemplo implementado con el sistema *grid* de Bootstrap, con un layout de dos columnas, una de ellas destinada a los contenidos y subdividida en tres columnas.

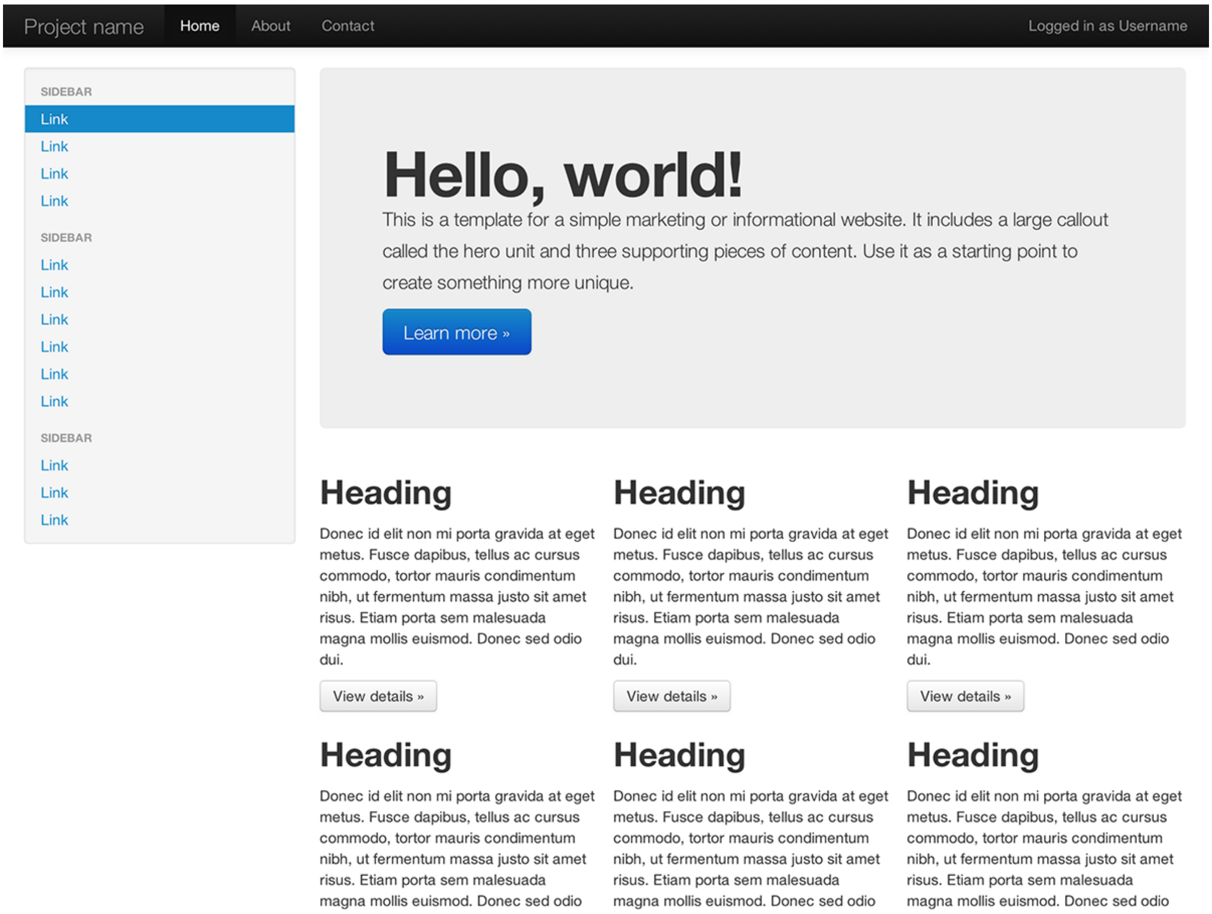

Imagen 3. Layout de dos columnas implementado con Bootstrap

2.4 CSS Flexbox

CSS Flexbox surge como una respuesta nativa en CSS a la necesidad de posicionar los elementos que componen un sitio web. Mediante la definición de un elemento contenedor, configurado con la propiedad *display: flex*, los elementos contenidos pueden organizarse tomando como eje una dirección, horizontal o vertical. Gracias a Flexbox podemos ordenar cómo se muestran los elementos independientemente de su posición dentro del documento, lo que nos permite mantener un documento semánticamente correcto y presentarlo de acuerdo al diseño que deseamos. También nos permite configurar el espacio que ocupa cada elemento sobre este eje, y alinear los elementos sobre los límites del contenedor.

Conceptualmente, Flexbox es la base sobre la que posteriormente se desarrolló CSS Grid, poniendo en discusión nuevamente la separación del contenido y de su presentación. La opción superadora de CSS Grid incluye la disposición bidimensional de los elementos, simplificando el diseño de layouts.

3 CSS Grid

3.1 Propuesta y alcance

La propuesta de CSS Grid consiste en disponer los contenidos de un elemento contenedor (*grid container*) en una rejilla (*grid*). El *grid* se define por la intersección de líneas verticales y horizontales (*grid lines*) que se numeran a partir del uno (1) comenzando por el extremo superior izquierdo del contenedor, aunque también pueden especificarse nombres para ellas. Estas *grid lines* definen regiones (*grid areas*) en las que pueden ubicarse cada uno de los elementos a mostrar (*grid items*). Las filas y columnas descritas por cada par de *grid lines* adyacentes se denominan pistas (*grid tracks*), y los espacios entre estos *grid tracks* se conocen como *grid gaps*.

En la Imagen 4 se pueden visualizar en forma sintética estos elementos, acompañada de una descripción detallada de cada uno de ellos.

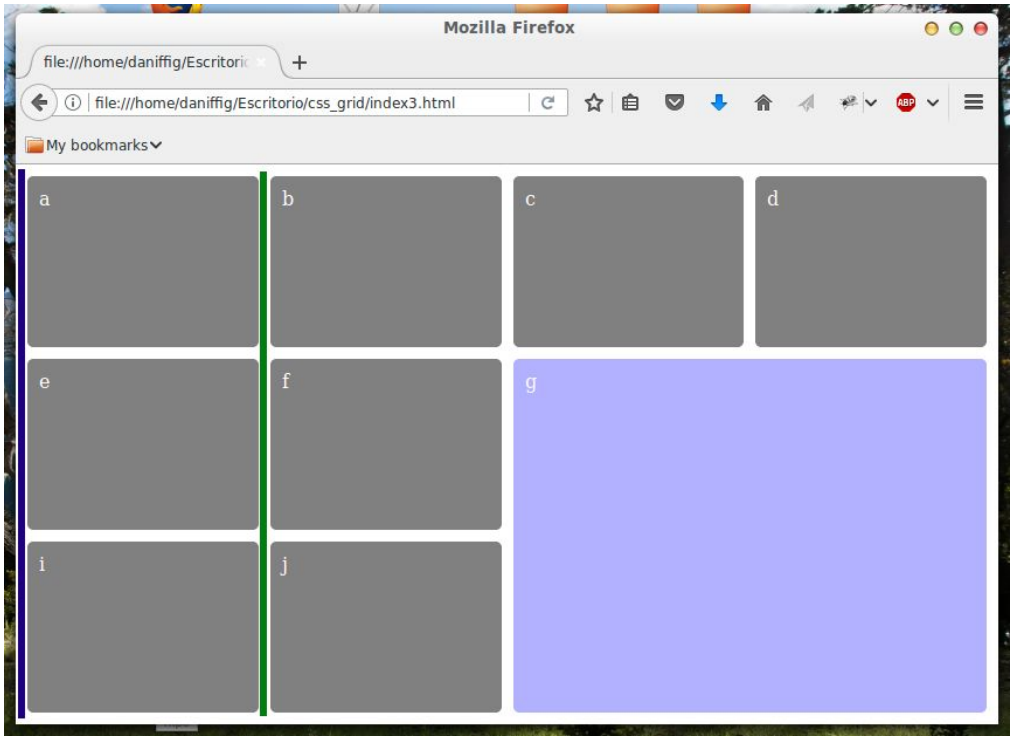

Imagen 4. Ejemplo de layout implementado con CSS Grid

El navegador presenta un sitio web con diseño *grid* definido por cuatro *grid lines* horizontales y cinco *grid lines* verticales. Esto define tres *grid rows* y cuatro *grid columns*. El *grid gap*, definido tanto entre filas como columnas, es de 10 pixeles. La línea vertical violeta

señala una *grid line* pura, mientras que la línea vertical verde señala una *grid line* que además es *grid gap*. Entre las líneas violeta y verde se define un *grid track* que por su verticalidad se denomina *grid column*. La intersección entre una *grid column* y una *grid row* se denomina *grid cell*. La caja de color lila identificada con la letra 'g' es un *grid item* presentado en un *grid area* que se define entre las *grid lines* horizontales 2 y 4, y las *grid lines* verticales 3 y 5, ocupando cuatro *grid cells*.

La implementación de los diseños *grid*, íntegramente desarrollada en CSS, se realiza desde dos elementos del documento; el *grid container*, y los *grid items* contenidos por el primero.

CSS Grid es actualmente soportado por los siguientes navegadores web.

Desktop	Chrome	Opera	Firefox	IE	Edge	Safari
	57	44	52	11	16	10.1
Mobile	iOS Safari	Opera Mobile	Opera Mini	Android	Android Chrome	Android Firefox
	10.3	No	No	56	61	56

3.2 Grid Container

Las propiedades disponibles para el *grid container* afectan dos aspectos principales del *grid*; por un lado, la definición de la rejilla en sí, con las dimensiones del contenedor, sus columnas, filas, y espaciados, y por el otro podemos definir la posición, las alineaciones horizontales y verticales de sus *grid items*. A continuación se detallan cada una de estas propiedades, acompañadas de sus posibles valores y sus efectos.

Propiedad	Descripción	*Valor*	Descripción
display	Define al elemento como un *grid container*	*grid*	Genera un *grid container* como elemento en bloque
		inline-grid	Genera un *grid container* como elemento en línea
		subgrid	Genera un *grid container* que es a su vez *grid item* de

			otro contenedor, y hereda las propiedades de este último
grid-template-columns **grid-template-rows**	Definen las dimensiones de columnas y filas, respectivamente. Adicionalmente se pueden definir los nombres de las líneas entre pistas	<dimension>	Determina la dimensión de una columna o una fila
		<nombre>	Define un nombre para una *grid line*, de manera tal que pueda ser invocada en la configuración de otras propiedades
grid-template-areas	Define la posición de los *grid items* en función de una plantilla que los referencia de acuerdo a identificación como *grid-area*	<nombre>	Posiciona al *grid item* de acuerdo a su identificación como *grid-area*
		.	Posiciona un *grid item* vacío
		none	Desactiva el uso de *grid-area*
grid-template	Define los mismos valores de *grid-template-columns , grid-template-rows,* y *grid-template-areas* en una única propiedad		
grid-column-gap **grid-row-gap**	Define las dimensiones de las *grid lines* internas entre columnas y filas, respectivamente	<dimension>	Determina la dimensión de la *grid line*
grid-gap	Define las dimensiones de todas las *grid lines*	<dimension>	Determina la dimensión de todas las *grid lines*
justify-items	Define la alineación horizontal de los *grid items* dentro de cada *grid cell*	*start*	Posiciona los *grid items* contra el lateral izquierdo de cada *grid cell*

		end	Posiciona los *grid items* contra el lateral derecho de cada *grid cell*
		center	Centra horizontalmente los *grid items* dentro de cada *grid cell*
		stretch	Utiliza la totalidad del ancho del *grid cell*
align-items	Define la alineación vertical de los *grid items* dentro de cada *grid cell*	*start*	Posiciona los *grid items* contra el lateral superior de cada *grid cell*
		end	Posiciona los *grid items* contra el lateral inferior de cada *grid cell*
		center	Centra verticalmente los *grid items* dentro de cada *grid cell*
		stretch	Utiliza la totalidad del alto del *grid cell*
justify-content	Posiciona horizontalmente al *grid* dentro del *grid container*	*start*	Posiciona al *grid* contra el lateral izquierdo de su *grid container*
		end	Posiciona al *grid* contra el lateral derecho de su *grid container*
		center	Centra horizontalmente al *grid* dentro de cada *grid container*

		stretch	Redimensiona a los *grid items* para que el *grid* ocupe toda la altura del *grid container*
		space-around	Dispone *grid gaps* verticales para ocupar la totalidad del ancho del *grid container*. El primer y el último *grid gap* tienen la mitad del ancho
		space-between	Dispone *grid gaps* verticales para ocupar la totalidad del ancho del *grid container*. El primer y el último *grid gap* no se muestran
		space-evenly	Dispone *grid gaps* verticales para ocupar la totalidad del ancho del *grid container*. El primer y el último *grid gap* tienen el mismo ancho que los *grid gaps* internos
align-content	Posiciona verticalmente al *grid* dentro del *grid container*	*start*	Posiciona al *grid* contra el lateral superior de su *grid container*
		end	Posiciona al *grid* contra el lateral inferior de su *grid container*
		center	Centra horizontalmente al *grid* dentro de cada *grid container*

		stretch	Redimensiona a los *grid items* para que el *grid* ocupe todo el alto del *grid container*
		space-around	Dispone *grid gaps* horizontales para ocupar la totalidad del ancho del *grid container*. El primer y el último *grid gap* tienen la mitad del alto
		space-between	Dispone *grid gaps* horizontales para ocupar la totalidad del alto del *grid container*. El primer y el último *grid gap* no se muestran
		space-evenly	Dispone *grid gaps* horizontales para ocupar la totalidad del alto del *grid container*. El primer y el último *grid gap* tienen el mismo alto que los *grid gaps* internos
grid-auto-columns **grid-auto-rows**	Define el ancho o el alto de los *grid tracks* generados automáticamente, es decir, cuando se definen en un *grid item* pero no están explícitamente definidos	<dimension>	Determina la dimensión de las columnas o las filaas generadas automáticamente
grid-auto-flow	Define la política a adoptar para ubicar a los *grid items* que no están explícitamente posicionados	*row*	Los *grid items* se posicionan en nuevas filas
		column	Los *grid items* se

			posicionan en nuevas columnas
		dense	Los *grid items* se posicionan en *grid cells* vacías
grid	Condensa todas las propiedades vistas anteriormente, con excepción de *justifiy-items, align-items, justify-content,* y *align-content*		

3.3 Grid Item

En el caso de los *grid items*, las propiedades disponibles nos permiten determinar la presentación de un item particular dentro de su contenedor, superponiéndose a la configuración general de este último, tanto en aspectos de posicionamiento como de alineación. Además, contamos con la propiedad *grid-area*, que permite asignar un nombre a un elemento para representarlo en la definición del *grid-template-areas* de su contenedor. A continuación se detallan cada una de estas propiedades, acompañadas de sus posibles valores y sus efectos.

Propiedad	Descripción	*Valor*	Descripción
grid-column-start **grid-column-end** **grid-row-start** **grid-row-end**	Determina dentro del *grid* la posición del *grid area* que contiene al *grid item*	<linea>	Toma el número o el nombre de la *grid line*
		span <numero>	El *grid item* se expande por el número de *grid tracks*
		span <nombre>	El *grid item* se expande hasta encontrar el *grid line* con el nombre indicado
		auto	El *grid item* se

			define automáticamente
grid-column **grid-row**	Determina las columnas y las filas que definen un *grid area*	<inicio> / <fin>	Define las *grid lines* verticales u horizontales que definen un *grid area*
grid-area	Define un identificador para un *grid item*, o la posición del *grid area* que ocupa	<nombre>	El *grid item* se asocia al nombre de *grid area* definido
		<inicio-fila> / <inicio-columna> / <fin-fila> / <fin-columna>	Define las *grid lines* que describen un *grid area* donde se ubica el *grid item*
justify-self	Define la alineación horizontal del contenido dentro del *grid item*	*start*	Posiciona al contenido contra el lateral izquierdo del *grid area*
		end	Posiciona al contenido contra el lateral derecho del *grid area*
		center	Centra horizontalmente el contenido dentro del *grid area*
		stretch	Utiliza la totalidad del ancho del *grid area*
align-self	Define la alineación vertical del contenido dentro del *grid item*	*start*	Posiciona al contenido contra el lateral superior del *grid area*
		end	Posiciona al contenido contra el lateral inferior del *grid area*
		center	Centra verticalmente el

			contenido dentro del *grid area*
		stretch	Utiliza la totalidad del alto del *grid area*

En la Imagen 5 podemos ver un ejemplo del uso de las propiedades tanto en el elemento contenedor como en los items contenidos, demostrando la complejidad que podemos alcanzar en los diseños utilizando CSS Grid. Este ejemplo puede descargarse desde el repositorio GitHub del documento, citado en la sección de Referencias.

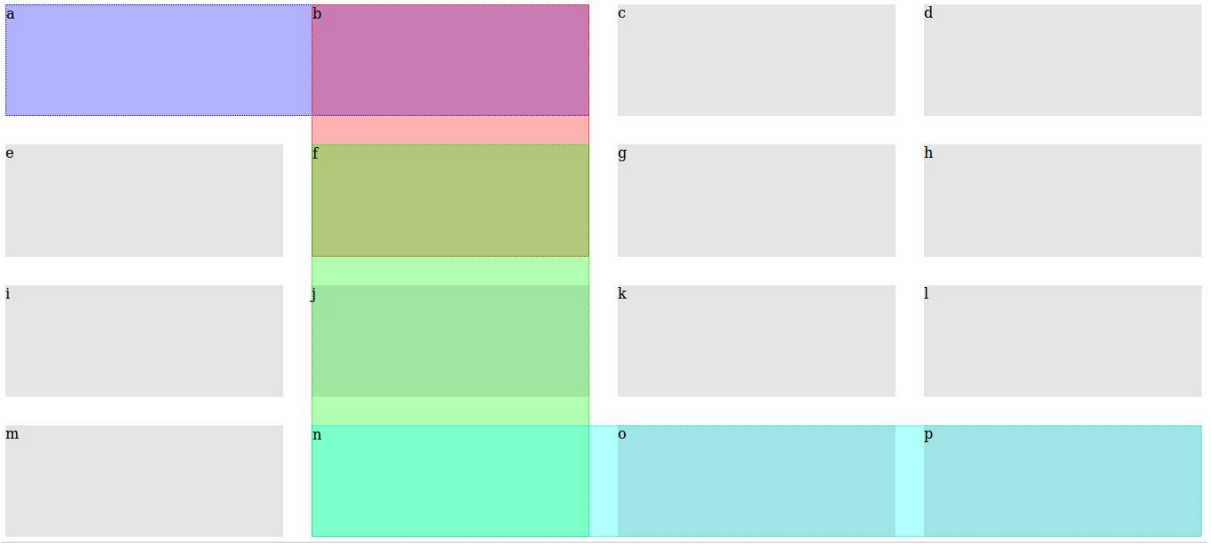

Imagen 5. Layout implementado con CSS Grid, utilizando posicionamiento y superposiciones

4 Caso de estudio

En esta sección vamos a realizar un estudio comparativo de las tecnologías utilizadas actualmente para diseñar interfaces de sitios y plataformas web frente a la propuesta de CSS Grid. El objetivo del mismo es identificar los puntos fuertes y desventajas de ésta última, con particular énfasis en el diseño de interfaces *mobile*.

La plataforma que se toma como referencia cuenta con un encabezado, un listado de resultados de búsqueda, dos paneles de acciones, y un pie de página. En dispositivos de baja resolución la plataforma se presenta como un listado, con los resultados de búsqueda primero y los paneles de acción después. En pantallas de mayor resolución (tablet,

computadoras) los resultados de búsqueda se muestran en una columna que ocupa el 75% de la pantalla, y los paneles de acción en otra, sobre el margen derecho, que ocupa el 25% de la pantalla. Cada resultado de búsqueda cuenta con un menú de acciones que se presenta sobre el vértice superior derecho de cada caja contenedora.

En ambos caso se utilizó Bootstrap 3 para dar estilos a títulos, botones, y campos de los formularios. Además, se utilizaron los íconos provistos por Font Awesome. Se priorizó compartir estilos comunes, tales como la paleta de colores y las reimplementaciones de Bootstrap. Los ejemplos pueden descargarse desde el repositorio GitHub del documento, citado en la sección de Referencias. Dentro de la carpeta *only-bootstrap* encontramos la versión implementada sólo con Bootstrap 3, mientras que dentro de *css-grid* encontramos la versión implementada con CSS Grid y Bootstrap 3 para los estilos.

En la Imagen 6 podemos apreciar la implementación realizada con Bootstrap 3. En la Imagen 7 vemos la versión *mobile* de la misma implementación.

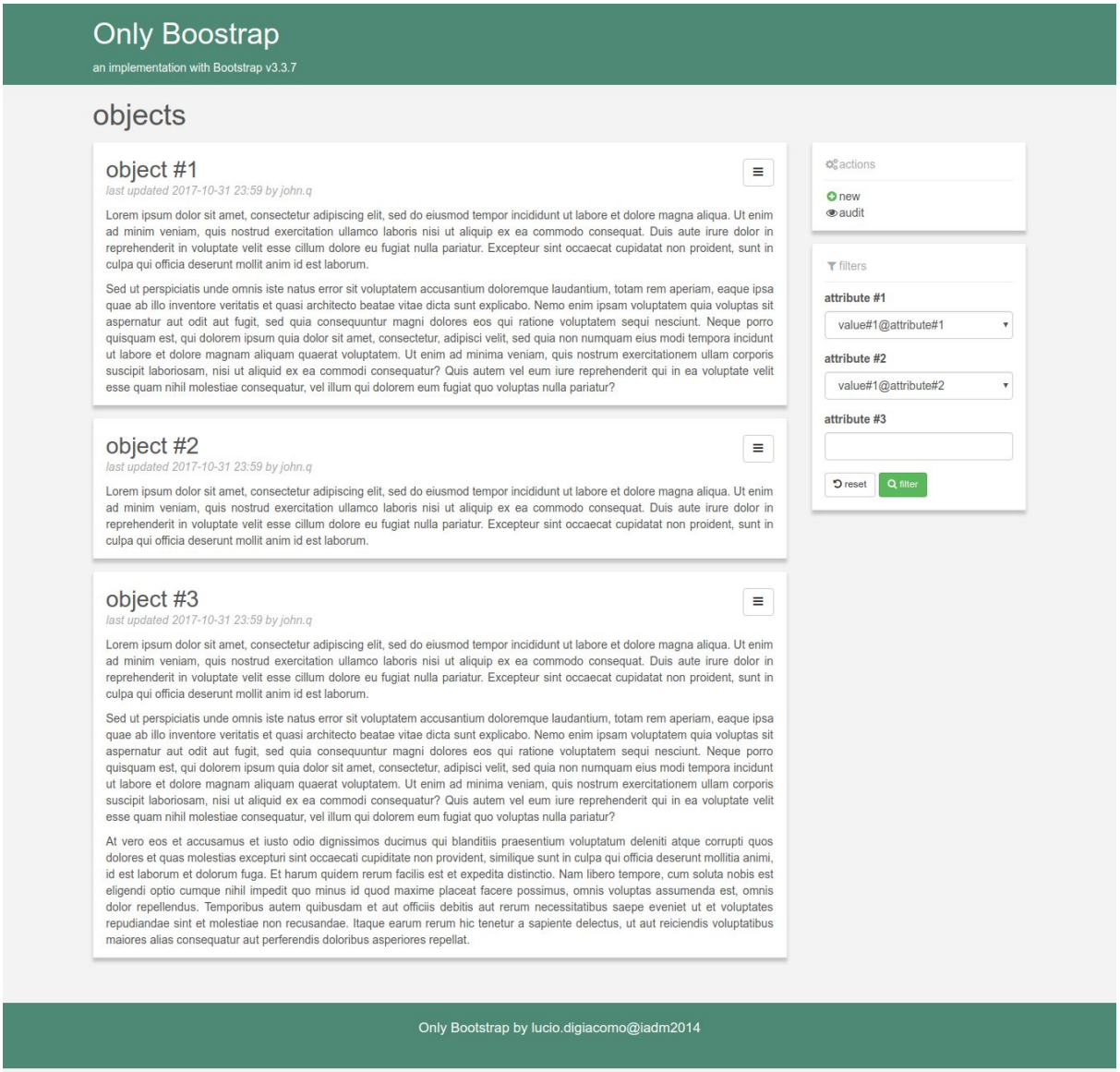

Imagen 6. Layout implementado con Bootstrap 3, versión escritorio

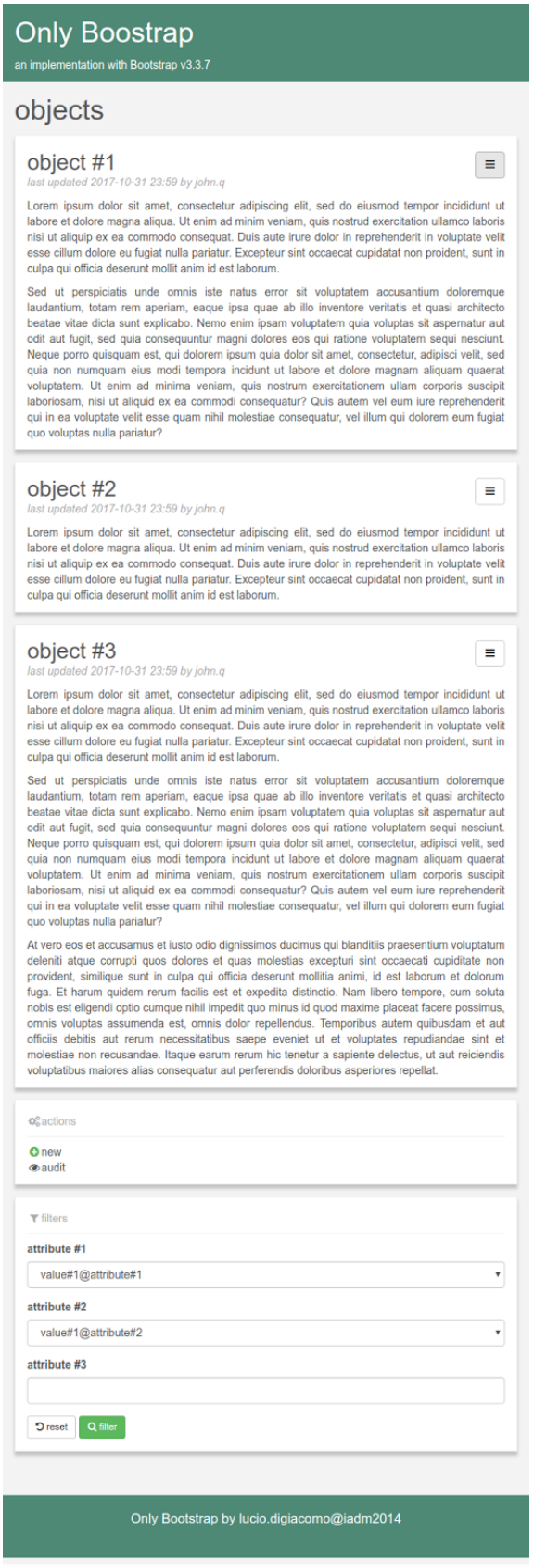

Imagen 7. Layout implementado con Bootstrap 3, versión *mobile*

En la Imagen 8 podemos apreciar la implementación realizada con CSS Grid. En la Imagen 8 vemos la versión *mobile* de la misma implementación.

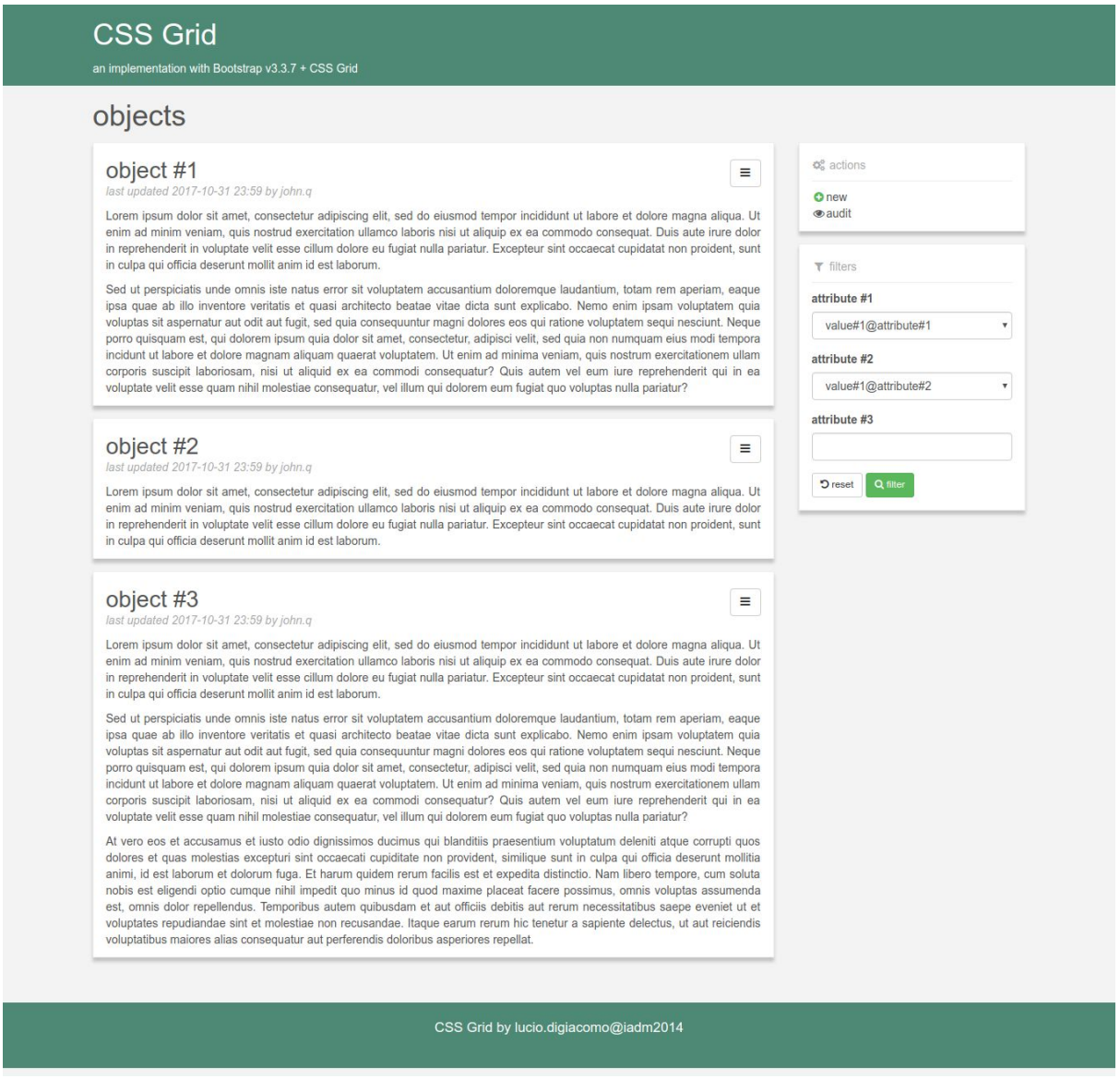

Imagen 8. Layout implementado con CSS Grid, versión escritorio

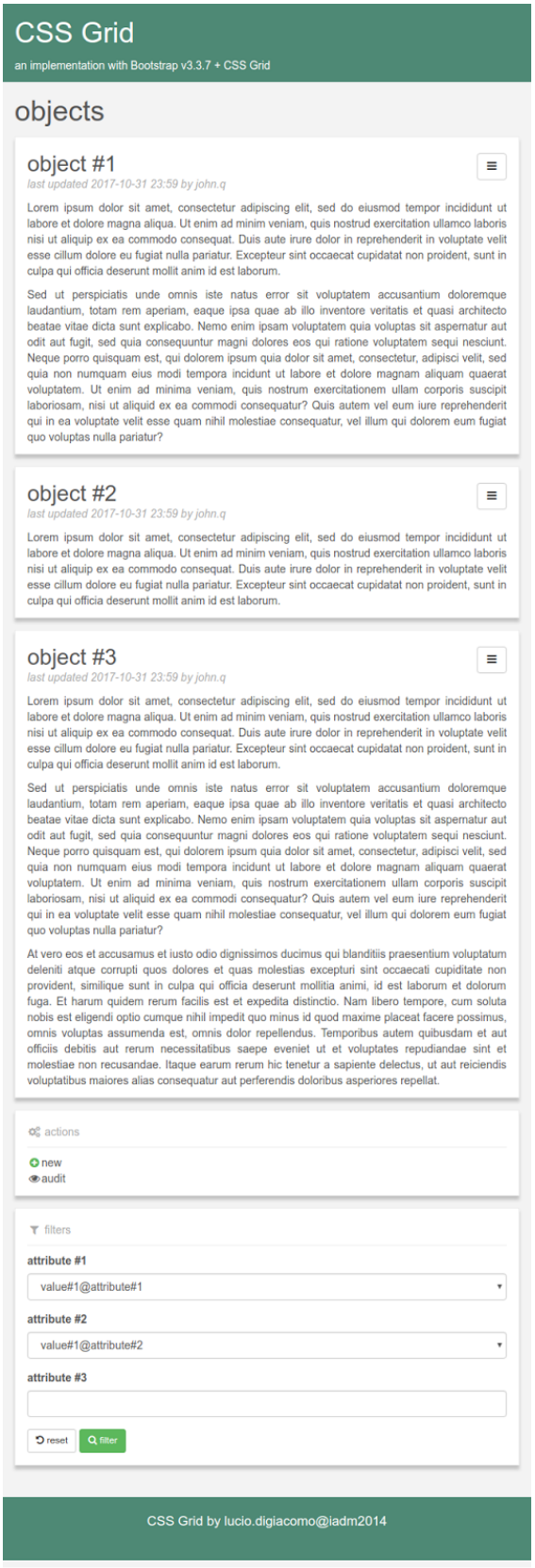

Imagen 9. Layout implementado con CSS Grid, versión *mobile*

Como podemos apreciar, existe una ligera diferencia entre ambas implementaciones. Esto se debe a que Bootstrap 3 y CSS Grid toman distintos elementos como referencia para calcular los porcentajes de pantalla a ocupar.

Con respecto a las implementaciones, analizamos las siguientes métricas.

	Only Bootstrap	CSS Grid
# líneas HTML base (sin objetos)	87	82
# líneas CSS adicionales*	0	65
# líneas base totales**	87	147
# líneas HTML en objeto #1	42	32
# líneas HTML en panel de filtros	29	29

* Sin considerar aquellas propias de Bootstrap y ni las hojas de estilo compartidas.
** Sin considerar aquellas propias de las bibliotecas compartidas.

Observamos que la cantidad de líneas totales de la implementación con CSS Grid es superior a la de Only Bootstrap debido a que esta última hace uso de las especificaciones disponibles en la biblioteca Bootstrap 3, mientras que la primera debe especificar en una hoja de estilos propia.

Si bien en primera instancia esto parecería ser anti-intuitivo, podemos observar que la cantidad de líneas necesarias para describir el objeto #1 del listado presentado en CSS Grid es un 25% menor que la de su equivalente en Only Bootstrap. Esto nos permite inferir que, para una búsqueda que arroje 10 objetos como resultados de búsqueda similares a objeto #1, CSS Grid los presentaría con 505 líneas de HTML totales, mientras que Only Bootstrap lo haría con 550, y en el caso de mostrar 20 objetos, la cantidad de líneas serían 825 y 970 respectivamente. Esto significa que a mayor cantidad de elementos iguales deba mostrar CSS Grid, más eficiente será en tamaño de los documentos, algo sumamente importante para la navegación desde conexiones móviles.

En la Imagen 10 podemos ver un gráfico de la evolución de la cantidad de líneas HTML a medida que crece la cantidad de objetos a mostrar.

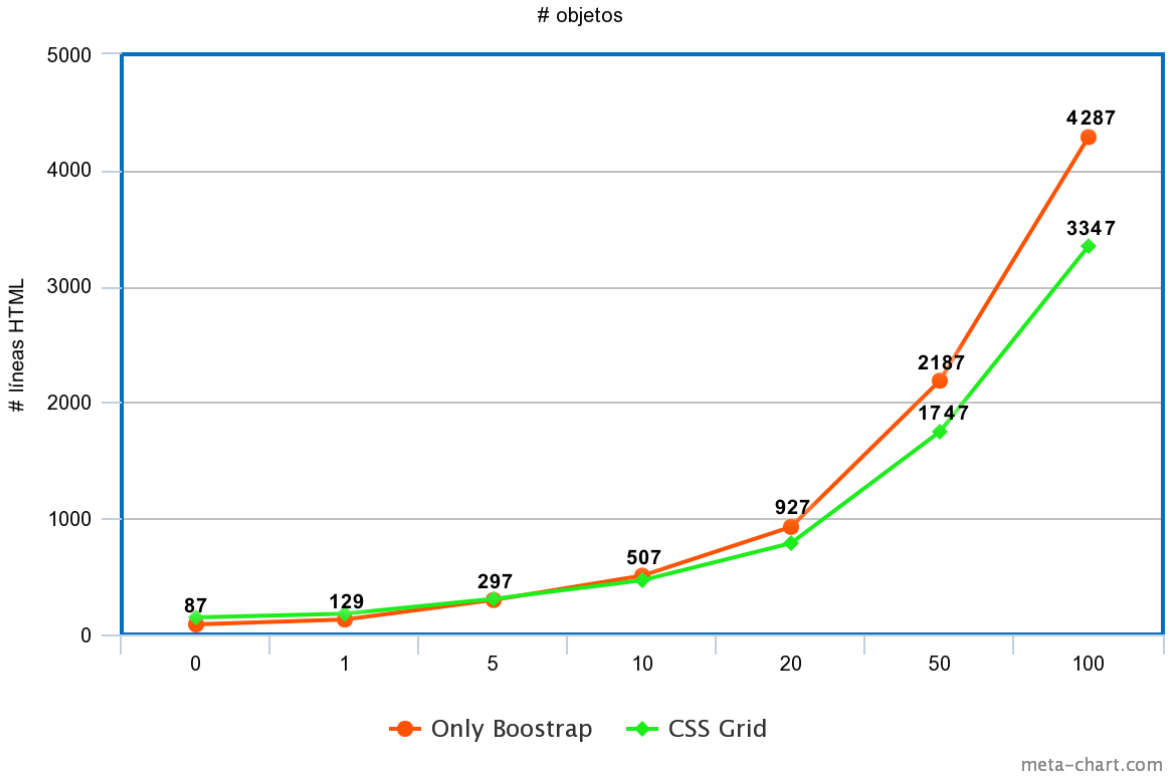

Imagen 10. Evolución de la cantidad de líneas HTML en función de los objetos presentados

Además, en la implementación de CSS Grid no existen elementos destinados exclusivamente a la presentación, como sucede en el caso de Only Bootstrap con los div con las clases *row* y *col*. Mientras que las hojas de estilo se guardan en caché, las líneas de los documentos destinadas a la presentación no, lo que concluye que una implementación con CSS Grid es más eficiente en términos del uso de transferencias de datos. Esto constituye buenas prácticas de acuerdo a las recomendaciones *Mobile Web Best Practices 1.0*, en cuanto al tamaño de los documentos como las hojas de estilo, y el correcto uso del marcado (con su respectivo valor semántico) y los elementos de presentación.

5 Conclusiones

CSS Grid es una propuesta superadora para el diseño de interfaces adaptables, otorgando a los diseñadores una herramienta fácil de aprender y utilizar, con reglas claras, e incorporando de manera intuitiva buenas prácticas como el diseño *mobile first*.

La separación de la presentación y el contenido es completa, por lo que documentos semánticamente correctos pueden mostrarse con diseños complejos y atractivos sin perder la posibilidad de ser utilizados por otras herramientas, como clientes automatizados, bases

de datos semánticas, o mismo con tecnologías de asistencia para la accesibilidad por parte de personas con distintas dificultades, además de facilitar el acceso desde dispositivos con conexiones que pueden no ser las óptimas, conservando su atractivo visual.

Si bien en la actualidad no existe un soporte extensivo para esta tecnología, la recomendación de la W3C existe apenas desde el año 2016, por lo que podemos esperar que en los próximos años la actualización de dispositivos, sistemas operativos y navegadores, en paralelo con la difusión entre los desarrolladores, la vuelvan un nuevo estándar para el desarrollo de interfaces adaptables de la misma manera que está sucediendo con su antecesor CSS Flexbox, incorporado en el desarrollo de Bootstrap 4.

La imposición de una nueva base como la que propone CSS Grid nos plantea una grata inquietud sobre todo lo que se vendrá en cuanto al desarrollo de interfaces web, y la responsabilidad de ser los autores de estas innovaciones.

6 Referencias

1. HTML Layouts. https://www.w3schools.com/html/html_layout.asp
2. HTML Tags: Past, Present, Proposed. http://www.martinrinehart.com/frontend-engineering/engineers/html/html-tag-history.html
3. The Evolution of Web Design. https://www.webpagefx.com/blog/web-design/the-evolution-of-web-design/
4. The Evolution of the Web. http://www.evolutionoftheweb.com/
5. 4 Different HTML/CSS Layout Techniques to Create a Site. https://www.codementor.io/codementorteam/4-different-html-css-layout-techniques-to-create-a-site-85i9t1x34
6. HTML Table Layouts. http://www.ironspider.ca/webdesign102/tables4layout2.htm
7. CSS Layout - float and clear. https://www.w3schools.com/css/css_float.asp
8. A Complete Guide to Flexbox. https://css-tricks.com/snippets/css/a-guide-to-flexbox/
9. CSS Grid Layout Module Level 1. https://www.w3.org/TR/css-grid-1/
10. A Complete Guide to Grid. https://css-tricks.com/snippets/css/complete-guide-grid/
11. CSS Grid Layout. https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout
12. CSS Grid vs Flexbox: A Practical Comparison. https://tutorialzine.com/2017/03/css-grid-vs-flexbox

13. Mobile Web Best Practices 1.0. https://www.w3.org/TR/mobile-bp/
14. IADM CSS Grid. https://github.com/daniffig/iadm-css-grid/