

**Nombre de equipo**CVJ Solutions **Curso**Proyecto de Ingeniería del Software 1 **Trabajo**Informe de Diseño de Software

**Profesores**Álvaro Cordero  
Pablo Monestel  
José Romero **Fecha de entrega**Miércoles 28 de Junio del 2017 **Periodo lectivo**II Cuatrimestre

Contenido

[**Introducción**](#_5wsevn6ag01o) **3**

[**2 Especificación de diseño**](#_2et92p0) **4**

[2.1 Descripción de la arquitectura](#_tyjcwt) 4

[Arquitectura Física.](#_duabdmyxit6j) 5

[2.2 Diseño del sitio](#_frcw1oyiblxq) 5

[2.2.1 Wireframe.](#_1t3h5sf) 6

[2.2.2 Mapa de navegación](#_4d34og8) 6

[2.2.3 Estándares de programación](#_2s8eyo1) 8

[Interfaz gráfica](#_5uzz7fems16e) 8

[HTML](#_yuwxkjdqwyry) 14

[CSS](#_vnlkoql4p83h) 15

[JavaScript](#_a29sc5q9ssy) 15

[2.3 División de casos de uso](#_17dp8vu) 16

[**3 Escenarios de prueba por cada caso de uso.**](#_3rdcrjn) **16**

[**4 Anexo**](#_26in1rg) **18**

[**5 Referencias**](#_muwp6rvmts2l) **19**

**6** [**Glosario**](#_lnxbz9) **19**

# 

# Introducción

El siguiente documento tiene como finalidad el informe de diseño del software que CVJ Solutions va a desarrollar durante este curso.

En el mismo se detallarán aspectos esenciales que debe contener este informe, como la especificación de diseño, que contiene la descripción de la arquitectura y diseño del sitio a ser desarrollado. Dentro de este apartado también se abordará la descripción de la arquitectura, diseño del sitio, los wireframes, mapa de navegación, estándares de programación y finalmente la división de los casos de uso entre los integrantes del grupo.

El siguiente de los puntos a tratar en este documento es los escenarios de prueba por cada caso de uso, donde se dará énfasis a los siguientes puntos: pruebas de diseño de Mockups, pruebas de validación, pruebas de navegación y pruebas de funcionalidad.

Por otro, lado se contará con un anexo, el cual contendrá toda la documentación de apoyo necesaria para tener una mejor comprensión de toda la información contenida en este documento. Finalmente se contará con un glosario donde se incluirán todos los términos que se considere deben ser definidos para el buen entendimiento de los conceptos técnicos que se empleen en este trabajo.

# 2 Especificación de diseño

En este apartado se hará una descripción de la arquitectura utilizada para el desarrollo de la aplicación. Referente al diseño del sitio se abordarán los wireframes que son los modelos de cómo se verán finalmente las pantallas y los estándares de programación.

## 2.1 Descripción de la arquitectura

La programación por capa es una arquitectura cliente-servidor en que el objetivo primordial es la separación de la lógica de negocios de la lógica de diseños. Es una estrategia vital para el desarrollo de aplicaciones multiplataforma permitiendo escalabilidad así como un mantenimiento y soporte sencillo. En esta arquitectura podemos encontrar:

* **Capa de presentación:** es la capa que ve el usuario y es la encargada de presentar el sistema al usuario, presentándole la información solicitada y a su vez captura la información que suministre el usuario validando que no existan errores de formato en los datos. Esta capa se conoce como la interfaz gráfica y debe ser fácil de usar para el usuario final.
* **Capa de negocios:** es la capa en donde se alojan los programas que se ejecutan, reciben las peticiones del usuario y se encargan de enviar las respuestas luego de procesarlas. Se conoce como capa lógica del negocio porque es aquí en donde se definen las reglas a cumplir. La capa de negocios se comunica con la capa de presentación para recibir las solicitudes y presentar los resultados. La comunicación también se da con la capa de datos para gestionar el almacenamiento o recuperación de datos.
* **Capa de datos:** esta capa es la responsable de alojar los datos y de acceder a los mismos. Está constituida por uno o más gestores de bases de datos que se encargan de recibir las solicitudes de almacenamiento o recuperación de información desde la capa de negocio.

### Arquitectura Física

La arquitectura física que se usará en este caso será semi-distribuida. Este tipo de arquitectura se caracteriza por mantener separados el cliente (navegador) y el servidor en donde se aloja el sistema. El servidor estará conformado principalmente por el Servidor Web y el Servidor de Base de Datos.

## 2.2 Diseño del sitio

En esta sección se detalla el esquema de la aplicación a desarrollar, mediante los wireframes se pretende hacer una simulación de cómo lucirán las diferentes pantallas tomando en cuenta el tipo de usuario ya que no todas tendrán el mismo contenido, el mismo se basará en el rol que en ese momento está haciendo uso de esa pantalla.

También se mostrará el mapa de navegación que básicamente muestra las restricciones para los tipos de usuario, datos que viajarán de una página a otra y finalmente el flujo de navegación entre las mismas.

En la parte final de esta sección se detallarán los estándares que CVJ Solutions definió para una correcta estructuración interna del sitio donde se pueden mencionar tipografías, carpetas contenedoras de archivos, formato de las imágenes, resoluciones de pantallas lo cual ayudará a un mejor mantenimiento de la aplicación.

### 2.2.1 Wireframe.

Un wireframe es una representación visual de una interfaz de usuario, despojada de cualquier diseño visual o elementos de marca. Para la creación de los Wireframes de **Magki**, hemos utilizado la herramienta Axure RP 8, en caso de ser requerido, podemos proveer el archivo fuente.

Los wireframes que creamos son de **baja - media fidelidad,** incluye información real, la jerarquía del contenido y de las secciones, pero se excluye cualquier elemento relacionado con la marca.

En el punto *4. Anexos*, de este documento, encontrará los wireframes de la aplicación Magki.

### 2.2.2 Mapa de navegación

El mapa de navegación es un recurso valioso porque al crearlo, se pueden definir las restricciones en la navegación de un usuario y definir los datos que se transportarán de una página a otra, así como el flujo de navegación entre estas.

El proceso que seguimos para crear el mapa de navegación de **Magki** (Figura 1) inició identificando las posibles secciones, basados en los Casos de Uso de alto nivel, ya que, nos ayudan a generalizar qué acciones estarán disponibles para el usuario.

Con base en las secciones identificadas, se determinó la jerarquía de cada sección y sus subsecciones correspondientes, se descartaron secciones, por ejemplo, logramos identificar que no es necesaria una sección para realizar la funcionalidad de Deshabilitar/Habilitar entidades (usuario, evento, academia, entre otros), sino que, son acciones que se pueden realizar desde la sección de Editar; se determinó el flujo de navegación entre las diferentes secciones y subsecciones de la aplicación, incluyendo el proceso de identificación de usuario (login), y separando la aplicación en dos secciones generales, una pública, en la cual cualquier usuario puede navegar y consumir información relacionada con Eventos, y otra privada, sección en la que se tiene acceso si y sólo si, el usuario cuenta con una cuenta en la aplicación y completa exitosamente el proceso de ingreso.

La navegación es el elemento de interfaz más importante, ya que, el usuario la utilizará para llegar a las secciones y funcionalidades de la aplicación. Uno de los principales beneficios de realizar el mapa de navegación, ha sido que, nos ha permitido someter a discusión hipótesis que se tenían al inicio del proyecto y nos ha dado un panorama más claro para la toma de decisiones, tanto de diseño como de arquitectura de la información.

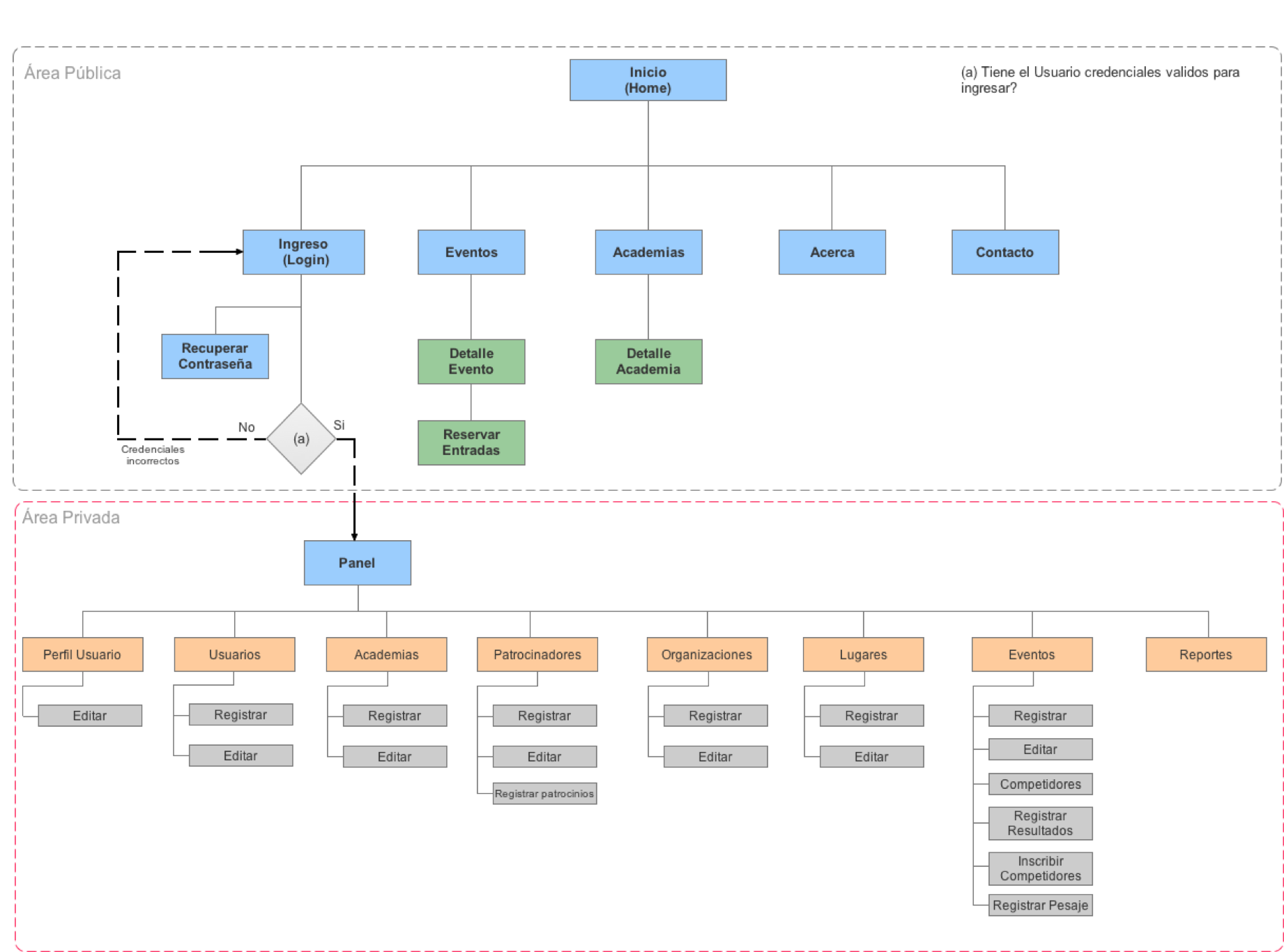


Figura 1.

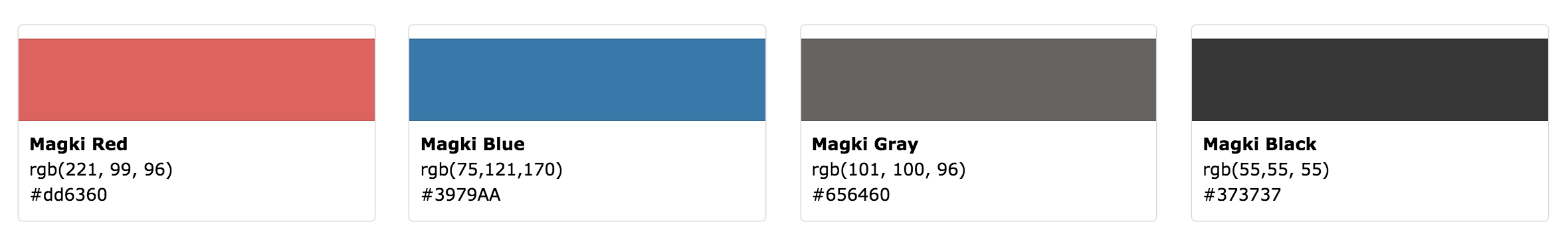
Mapa de Navegación - **Magki**

### 2.2.3 Estándares de programación

Esta sección incluye los estándares que se van a utilizar para el diseño y desarrollo de **Magki**, a manera general incluye: paleta de colores a utilizar, utilización del logo, tamaño y tipo de letras, el formato y casos de uso para las imágenes, iconos o ilustraciones, una explicación de la resolución de pantalla óptima, la estructura de los folders en el codebase y los estándares de sintaxis y formato para escribir HTML, CSS y JavaScript.

### Interfaz gráfica

1. **Esquema de colores.**



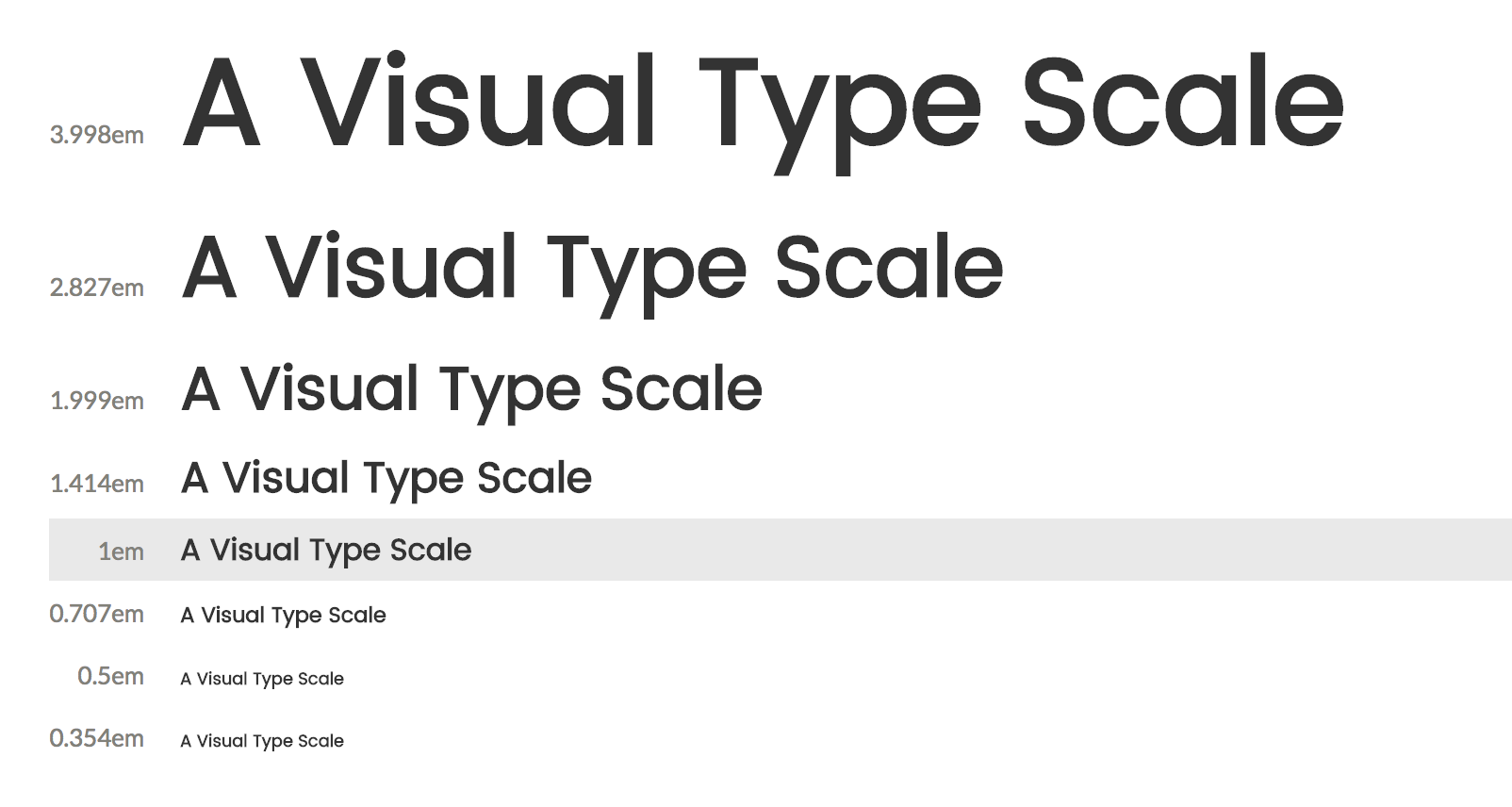




1. **Tamaño y tipo de letra.**

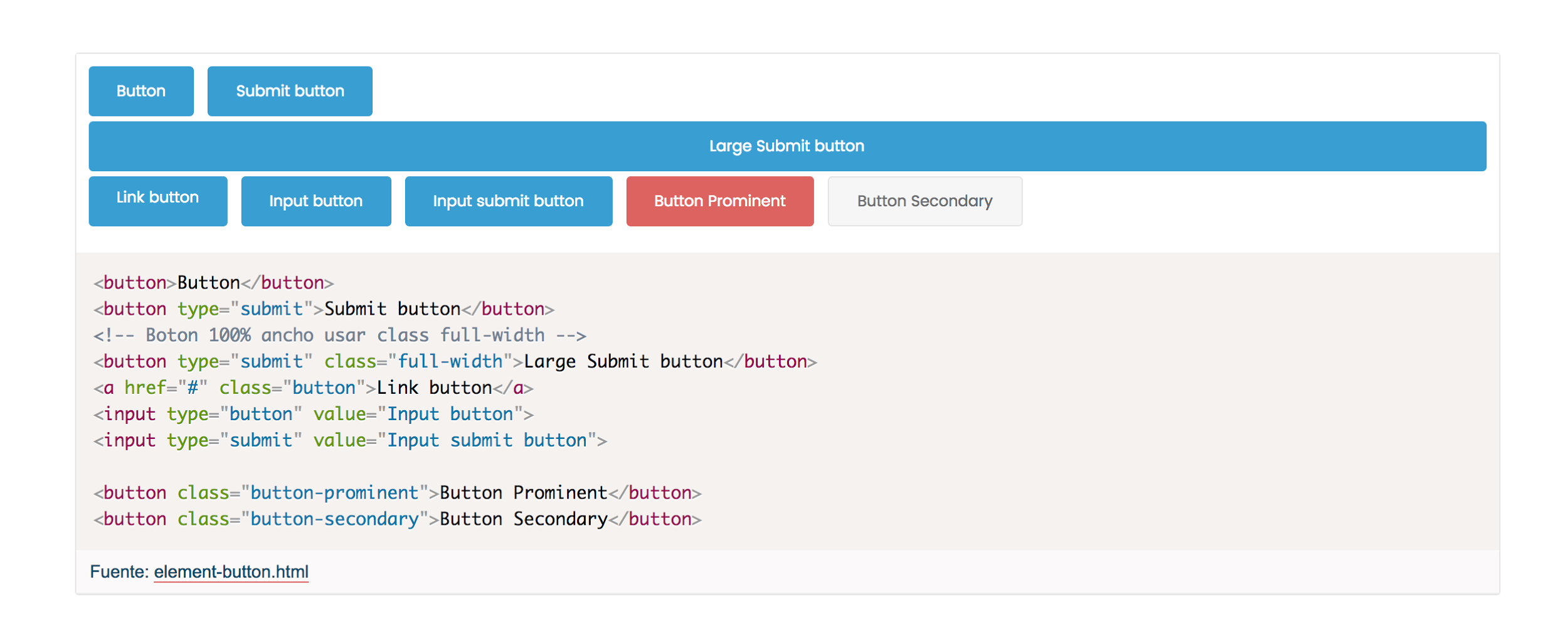
Se van a utilizar dos tipografías, Poppins y Arial. Poppins se sirve desde el servicio de Google Webfonts (https://fonts.google.com/specimen/Poppins), y se va a utilizar como fuente de contraste, específicamente se aplicará en los títulos (h1 a h6), las opciones de navegación (menú principal, menú en footer, tabs, etiquetas de filtros, entre otros), títulos de tablas y botones. Arial se utilizará en el resto de elementos, no mencionados anteriormente.

Con respecto a los valores para el tamaño de fuentes, estamos utilizando como base 16px, sin embargo, todos los valores para la propiedad de CSS font-size se declaran en la unidad em, esto con el fin de aprovechar que los em son escalables, a diferencia de los pixeles, que son unidades fijas; en la *Figura 2* incluimos un ejemplo del uso de Poppins y los valores asignados. Una guia completa de los elementos de HTML, y estilos de CSS asociados, la puede encontrar en [**https://jleiva.github.io/magki/index.html**](https://jleiva.github.io/magki/index.html)**;** la *Figura 3*. se muestra la sección de botones.



*Figura 2.*

*Poppin escala.*



*Figura 3.*

*Guia de estilos*

1. **Formato de las imágenes**

Por estándar, vamos a utilizar imágenes con extensión **.jpg** para toda imagen que sea una fotografía, hay que verificar que la imagen ha sido optimizada para Web, por ejemplo: fotografía de eventos.

El formato **.png** permite almacenar imágenes en blanco y negro (de una profundidad de color de 16 bits por píxel) y en color real (de una profundidad de color de 48 bits por píxel), así como también imágenes indexadas, utilizando una paleta de 256 colores. Además soporta un nivel muy sofisticado de transparencia de la imagen en canal alfa, es decir, la posibilidad de definir 256 niveles de transparencia, por estas razones, vamos a utilizar **.png** cuando se tengan imágenes que contienen textos, logos con transparencia y degradación, imágenes complejas y de mucho tamaño e imágenes en proceso de edición

El formato **.gif,** se utiliza en gráficos compuestos de pocos colores, patrones de colores sólidos o iconos pequeños.

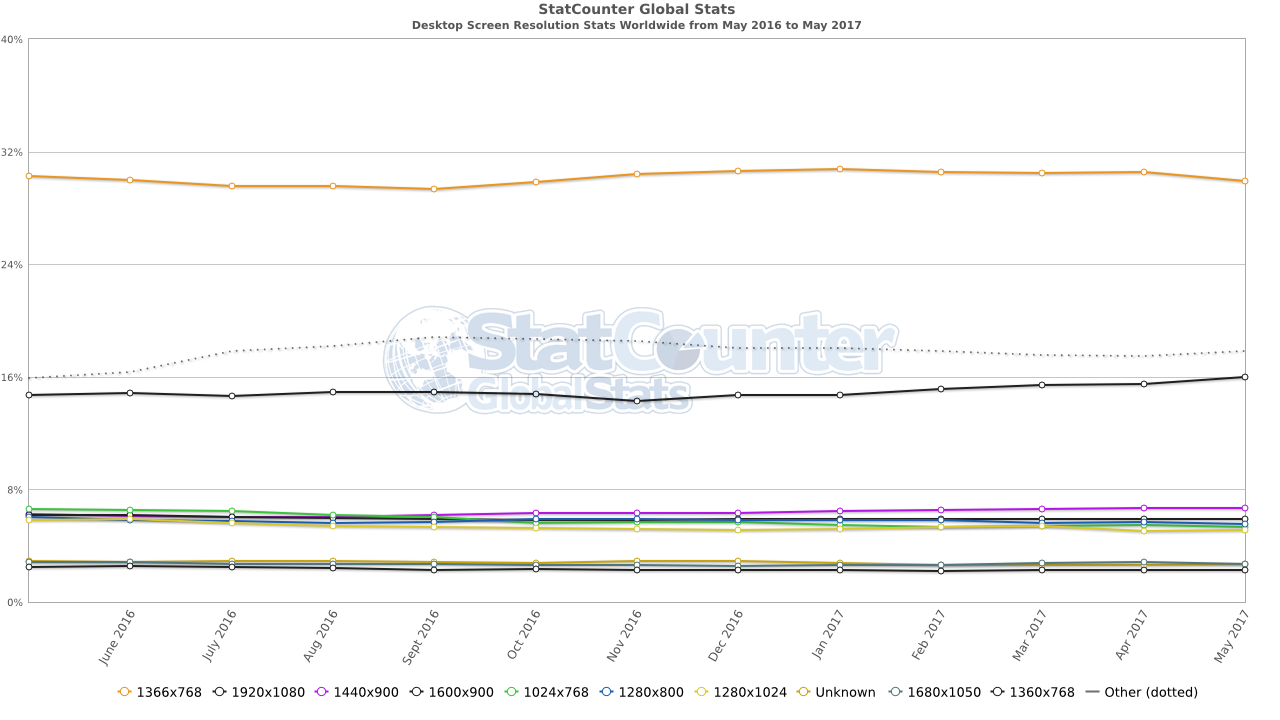
1. **Resolución utilizada o recomendada.**

La aplicación está optimizada para una resolución de pantalla máxima de 1200px y mínima de 900px.

Para decidir el rango máximo y mínimo de resolución a optimizar, utilizamos como base las estadísticas del sitio web [StatCounter](http://gs.statcounter.com/) (http://gs.statcounter.com/); en la *Figura 4* se presenta un gráfico de la resolución de pantallas (desktop) a nivel mundial, para el periodo Mayo 2016 a Mayo 2017.

Más del 70% de las pantallas tienen una resolución superior a 1200px, aunado a esto, aplicamos los siguientes criterios:

* Visibilidad inicial: ¿Está visible toda la información clave por encima del *fold* (mitad superior de la primera página) para que los usuarios puedan verla sin hacer scroll?
* Legibilidad: ¿Que tan fácil es leer el texto de las diferentes secciones? ¿Podemos mantener la longitud de una línea de texto entre 60 a 70 caracteres?
* Estética: ¿Que tan bien se ven todos los elementos de la aplicación? ¿Se alinean correctamente?



*Figura 3.*

*Fuente:* [*https://goo.gl/aN7JMo*](https://goo.gl/aN7JMo)

Cabe destacar que, cuando nos referimos a resolución a "optimizar", queremos decir que nuestra aplicación debe “verse” y funcionar sin problemas en el rango de 1200px a 900px; sin embargo, va a funcionar y verse sin problemas en cualquier resolución superior a 1200px, y en una resolución menor a 900px, la aplicación mantendrá un ancho fijo, lo que significa que puede existir un scroll horizontal, sin embargo, las funcionalidades se mantendrán sin problemas.

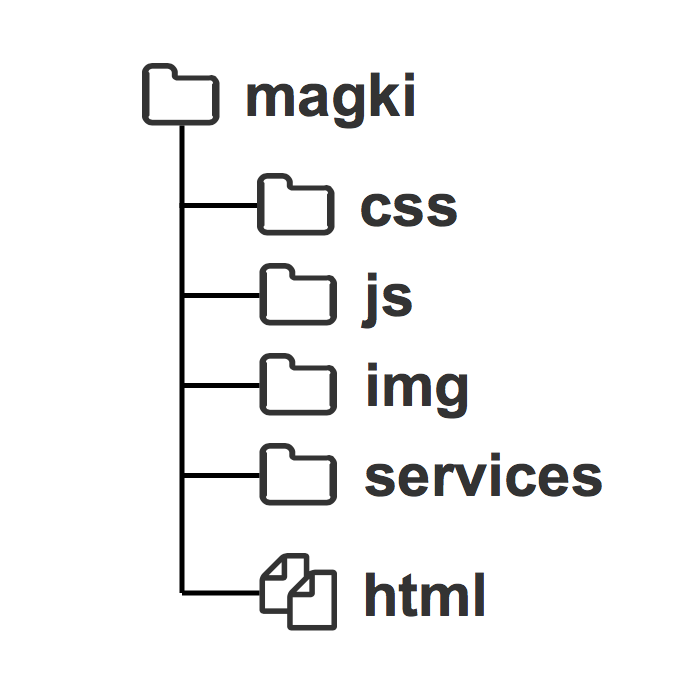
Para poder mantener el rango de optimización, vamos a utilizar un tipo de layout líquido, lo que significa que, los componentes estructurales de la aplicación tendrán medidas definidas en porcentajes, lo que permitirá que se ajusten a la resolución de pantalla del usuario.

1. **Estructura de directorios para el sitio.**

El code base de la aplicación se estructura de la siguiente manera:

* el folder raíz (root) tiene el nombre de **/magki**
* dentro del folder **/magki,** se encuentran los archivos de extensión .html, y los folders **/css, /js, /img** y **/services.**
* todos los archivos de extensión **.css** se deben incluir en el folder **/css**
* todos los archivos de extensión **.js** se deben incluir en el folder **/js**
* todas las imágenes (.jpg, .png, .gif, etc), ilustraciones ó iconosse deben incluir en el folder **/img**
* servicios para conexión a base de datos, archivos *partials* de PHP, se incluyen en el folder **/services**

Gráficamente, el code base se representa de la siguiente manera:



*Figura 4.*

*Codebase Magki*

1. **Estándares: HTML, CSS y JavaScript**

En el punto *4. Anexos* va a encontrar los siguientes documentos: estándar-HTML.pdf, estándar-CSS.pdf y estándar-JavaScript.pdf los cuales detallan y ejemplifican los estándares que seguimos al momento de desarrollar utilizando HTML, CSS y JavaScript.

La documentación oficial de los estándares la puede encontrar también en nuestro repositorio (<https://github.com/jleiva/magki/tree/master/docs>), un extracto de las convenciones que utilizamos son las siguientes:

### HTML

A manera general queremos:

* Indentar con dos (2) espacios, no tabs.
* Elementos anidados deben ser indentados una vez (2 espacios).
* En los atributos de elemento, usar comillas dobles (`" "`), nunca comillas simples.
* No incluir "trailing slash" (`/`) en elementos que se auto cierran, es opcional según la especificación de HTML5.
* No omitir etiquetas de cierre.
* Separar atributos por un espacio.
* Siempre usar igual y comillas dobles para valores de atributos.
* Siempre incluir etiquetas `html`, `head`, y `body`.
* Siempre usar minúscula para nombre de etiquetas y atributos.
* Siempre incluir el atributo `alt` para imágenes.
* Separar responsabilidades. Mantener markup (HTML), estilos (CSS) y comportamiento (JS) separado.
* Utilice HTML de acuerdo a su propósito. Utilice elementos para lo que se han creado. Por ejemplo, utilice elementos de encabezado (`h1 - h6`) para encabezados, elementos `p` para párrafos, elementos `a` para anclajes, etc.

### CSS

A manera general queremos:

* Indentar con dos (2) espacios, no tabs.
* Uso significativo de espacio en blanco.
* Uso de minúsculas y nombres en inglés para selectores por clase.
* Cambio de palabra separado por un guión (-), ejem. `. menu-item`
* Cuide el orden y especificidad, evite darle *estilo* a IDs y sobrecalificar selectores.
* Como regla general, evite anidación innecesaria, y nunca anidar más de tres niveles. Si no puede evitarlo, de un paso atrás y reconsidere su estrategia.
* Evite especificar unidades cuando el valor es 0 (cero).
* Siempre que aplique, utilice shorthand de los valores de una propiedad.
* Use shorthand para valores hexadecimales cuando sea posible.
* Use minúsculas para todos los valores hexadecimales.
* Todas las declaraciones deben terminar con un punto y coma (;).
* Todo archivo de extensión *.css* debe estar incluido en el folder **/css** del proyecto.

### JavaScript

A manera general queremos:

* Utilizamos ECMAScript 5.1 (ES5).
* Indentar con dos (2) espacios, no tabs.
* Dejar un espacio antes de la llave de apertura `{`.
* Usar llaves con todos los bloques de múltiples líneas (```if```, ```function```).
* Un espacio antes del paréntesis de apertura en las sentencias de control (```if```, ```while```, etc.).
* Bloques de muchas líneas con ```if``` y ```else```, poner el ```else``` en la misma línea que el ```if```.
* No espacio entre el keyword function o el nombre de la función y el primer paréntesis (`function() {}`)
* Separar a los operadores con espacios. `var x = y + 5;`
* Dejar una línea en blanco luego de los bloques y antes de la siguiente sentencia.
* Ser descriptivo con nombres de variables, métodos, funciones, etc.
* camelCase para nombrar objetos, funciones e instancias, ejem: `thisIsMyObject`, `thisIsMyFunction`
* PascalCase cuando nombre constructores o clases.
* Un guión bajo (\_) adelante de la variable cuando se nombre propiedades privadas, ejem: `var \_\_firstName`
* Cuando se guarde una referencia a `this` usa `\_this`
* No usar (http://es5.github.io/#x7.6.1) para nombres de propiedades, funciones o variables.
* Nombre sus funciones. Esto será de ayuda en caso de errores.

## 2.3 División de casos de uso

Los Casos de Uso han sido divididos entre los miembros del equipo, el detalle lo puede ver en el punto *4. Anexo*, documento con nombre **Casos de Uso por responsable.**

# 3 Escenarios de prueba por cada caso de uso.

El escenario de los casos de prueba del sistema **Magki** se llevarán a cabo por cada responsable del equipo el cual deberá seguir la plantilla establecida por todos los miembros, en la siguiente plantilla se tendrá que llenar los espacios de acuerdo al caso de uso que fue realizado.

La plantilla se puede usar varias veces para un mismo caso de uso hasta que las pruebas de dicho caso sean exitosas.

A continuación, se especificará cada uno de los campos de la plantilla.

**ID:** Indica el número de caso de prueba.

**Nombre:** el nombre del caso de prueba, este puede ser similar o igual al caso de uso.

**Dependencias:** Se indicará el número de caso de uso del cual se realiza la prueba.

**Tipo:** Se indica si es de navegación, validación o funcionalidad.

**Descripción:** Se debe detallar la descripción del caso de prueba.

**Resultado esperado:** Se describe el comportamiento que debe ocurrir en el sistema.

**Resultado Obtenido:** Se describe paso por paso lo que se realizó y el resultado que obtuvo ese proceso

**Responsable:** Se debe incluir el nombre del integrante que realizó la prueba

**Plantilla vacía:**

|  |  |
| --- | --- |
|  | **PLANTILLA DE CASOS DE PRUEBAS** |
| **ID** | **Nombre** |
| **Dependencias** |  |
| **Tipo** |  |
| **Descripción** |  |
| **Resultado Esperado** |  |
| **Resultado Obtenido** |  |
| **Responsable** |  |

**Ejemplo de plantilla:**

|  |  |
| --- | --- |
|  | **PLANTILLA DE CASOS DE PRUEBAS** |
| **ID** 001-test | **Nombre** = Login de Usuarios |
| **Dependencias** | UC-001 |
| **Tipo** | Navegación |
| **Descripción** | Verificar que los usuarios registrados tengan acceso a la página de login |
| **Resultado Esperado** | La página deberá visualizarse y funcionar para todos los usuarios registrados |
| **Resultado Obtenido** | Se dirigió al inicio de la página del sistema |
|  | Se le dio click al botón de login |
|  | La página de login se mostró a todos los usuarios |
| **Responsable** | Pedro Picapiedra |

# 4 Anexo

Anexo 1: Wireframes

Anexo 2: Casos de Uso por responsable

Anexo 3: Estándar-HTML.pdf

Anexo 4: Estándar-CSS.pdf

Anexo 5: Estándar-JavaScript.pdf

Anexo 6: Casos de uso por responsable

# 5 Referencias

Incluye documentación de apoyo. (Gráficos, tablas, imágenes, etc.)

* <http://www.bbc.co.uk/schools/gcsebitesize/dida/using_ict/onscreen_publicationsrev3.shtml>
* <https://medium.com/@amherd/the-difference-between-information-architecture-ia-sitemap-and-navigation-64eba19296c>
* <http://www.uxbooth.com/articles/complete-beginners-guide-to-information-architecture/>
* <https://es.wikipedia.org/wiki/Mapa_de_sitio_web>
* <http://pmonestel.esy.es/proyectosoft/prototipado_y_wireframes/>
* <https://www.axure.com>
* <http://theuxreview.co.uk/wireframes-beginners-guide/>
* <https://www.uxpin.com/studio/ui-design/what-is-a-wireframe-designing-your-ux-backbone/>
* <https://www.nngroup.com/articles/screen-resolution-and-page-layout/>
* <https://www.smashingmagazine.com/2009/06/fixed-vs-fluid-vs-elastic-layout-whats-the-right-one-for-you/>

# 6 Glosario

Se definen los términos para la correcta comprensión del documento.

* **Arquitectura Cliente-Servidor:** modelo de aplicación distribuida en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes. Un cliente realiza peticiones a otro programa, el servidor, quien le da respuesta.
* **Arquitectura N Capas:** conocida también como arquitectura de tres capas o multicapa, es una técnica para crear aplicaciones generalmente divididas en capas de servicios de usuario, de negocios y de datos.
* **Codebase**: o base de código, es usado en el desarrollo de software con el significado de la colección completa de código fuente usada para construir una aplicación o componente particular.
* **CSS**: siglas en inglés de *Cascading Stylesheets* (hojas de estilo en cascada) es un lenguaje de diseño gráfico para definir y crear la presentación de un documento estructurado escrito en un lenguaje de marcado.
* **ECMAScript:** es una especificación de lenguaje de programación publicada por ECMA International; el desarrollo empezó en 1996 y estuvo basado en el popular lenguaje JavaScript
* **ES5:** versión 5, que es la más usada actualmente, de ECMAScript. En junio de 2015 se cerró y publicó la versión ECMAScript 6.
* **HTML:** siglas en inglés de HyperText Markup Language (lenguaje de marcas de hipertexto), hace referencia al lenguaje de marcado para la elaboración de páginas web.
* **JavaScript:** (abreviado comúnmente JS) es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.
* **Lado del cliente**: (client-side en inglés) es el ámbito en el cual un cliente ejecuta sus operaciones en una relación cliente-servidor dentro de una red informática. Un cliente se entiende como una aplicación (por ejemplo un navegador web), que se ejecuta en el computador local del usuario y se conecta un servidor cuando es necesario.
* **LAMP**: acrónimo usado para describir un sistema de infraestructura de internet que usa las siguientes herramientas: Linux, el sistema operativo; En algunos casos también se refiere a LDAP. Apache, el servidor web; MySQL/MariaDB, el gestor de bases de datos; Perl, PHP, o Python, los lenguajes de programación.
* **Layout líquido:** se refiere a la estructura de un sitio web donde la mayoría de los componentes internos tienen anchos en porcentaje y, por lo tanto, se ajustan a la resolución de pantalla del usuario.
* **PHP:** es un acrónimo recursivo que significa *PHP: Hypertext Preprocessor*; es un lenguaje de programación de uso general de código del lado del servidor originalmente diseñado para el desarrollo web de contenido dinámico.
* **Programación por capas:** es un modelo de desarrollo software en el que el objetivo primordial es la separación (desacoplamiento) de las partes que componen un sistema de software o también una arquitectura cliente-servidor: lógica de negocios, capa de presentación y capa de datos.
* **Servidor:** es una aplicación en ejecución (software) capaz de atender las peticiones de un cliente y devolverle una respuesta en concordancia.
* **Wireframe:** también conocido como un esquema de página o plano de pantalla, es una guía visual que representa el esqueleto o estructura visual de un sitio web. El wireframe esquematiza el diseño de página u ordenamiento del contenido del sitio web, incluyendo elementos de la interfaz y sistemas de navegación, y cómo funcionan en conjunto.