



UNIVERSIDAD DE CHILE  
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS  
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

FRAMEWORK E-COMMERCE

MEMORIA PARA OPTAR AL TÍTULO DE  
INGENIERO CIVIL EN COMPUTACIÓN

ÁLVARO MAURICIO CRISTIAN BALMACEDA CELEDÓN

PROFESOR GUÍA:  
NELSON BALOIAN TATARYAN

MIEMBROS DE LA COMISIÓN:  
JOCELYN SIMMONDS WAGEMANN  
ALEJANDRO HEVIA ANGULO

SANTIAGO DE CHILE  
2016

RESUMEN DE LA MEMORIA  
PARA OPTAR AL TÍTULO DE  
INGENIERO CIVIL EN COMPUTACIÓN  
POR: ÁLVARO BALMACEDA CELEDÓN  
FECHA: 2016  
PROF. GUÍA: NELSON BALOIAN TATARYAN

## FRAMEWORK E-COMMERCE

Una de las actividades más populares en la *web* es comprar. Los sitios web dedicados al comercio tienen una oportunidad única totalmente nueva para desarrollar negocios. Existe la oportunidad de desarrollar ventajas competitivas significativas en sus respectivos mercados creando experiencias gratas para el usuario. El éxito dependerá en perfeccionar los esfuerzos para abordar las experiencias de clientes centrados en el usuario.

Los *Frameworks Open Source* disponibles permiten desarrollar una gran variedad de soluciones *e-Commerce*, además de tener grandes comunidades que las respaldan así como muchísimos usuarios satisfechos. Sin embargo, todas estas opciones fueron construidas en un contexto en donde el poder de procesamiento de los clientes era limitado. Muy distinto al actual panorama, en donde la *web* ha madurado alcanzando una serie de características que mejoran las experiencias de los usuarios.

El presente trabajo propone una solución tecnológica base para apoyar el desarrollo de diferentes soluciones *e-Commerce* donde es posible (en comparación a los *Frameworks* actuales): resultados más rápidos, ideal para prototipos y productos entregables mínimos; una solución integral que desarrolla características para *servidores*, *browsers* y *dispositivos móviles*; contruido nativamente con capacidades *tiempo-real* para minimizar el esfuerzo en desarrollo; e integración de herramientas de desarrollo para hacer que la configuración, desarrollo, e instalación sea extremadamente rápido.

Durante el trabajo de título se logró desarrollar una plataforma con base sólida construida a partir de las buenas prácticas obtenidas por la experiencia de los actuales sitios *web e-Commerce*, pero con características de las aplicaciones *web* modernas tales como reactividad y tiempo-real, lo que le entrega ventajas competitivas. El mayor aporte de este trabajo fue el desarrollo de una arquitectura genérica útil para cualquier aplicación *web*. Finalmente se plantean transformaciones útiles en la arquitectura que beneficiarán también al framework.

# Tabla de Contenido

<b>1. Introducción</b>	<b>1</b>
1.0.1. <i>e-Commerce</i> en la actualidad . . . . .	2
1.0.2. <i>e-Commerce</i> en el futuro . . . . .	3
1.1. Motivación . . . . .	4
1.2. Alcances y objetivo general . . . . .	6
1.3. Objetivos específicos . . . . .	7
<b>2. Estado del Arte</b>	<b>8</b>
2.1. <i>e-Commerce</i> . . . . .	8
2.1.1. Categorías de <i>e-Commerce</i> . . . . .	8
2.1.2. Ejemplos de <i>Frameworks e-Commerce</i> existentes . . . . .	9
2.2. Tecnologías . . . . .	15
2.2.1. <i>Base de datos</i> . . . . .	15
2.2.2. <i>Server-side</i> . . . . .	16
2.2.3. <i>Client-side</i> . . . . .	19
2.2.4. <i>Full-Stack JavaScript Frameworks</i> . . . . .	19
2.2.5. Herramientas para desarrollo . . . . .	21
2.3. Aplicaciones <i>web</i> . . . . .	21
<b>3. Justificación del proyecto</b>	<b>22</b>
3.1. <i>Base de datos</i> . . . . .	22
3.1.1. SQL y el Modelo Relacional . . . . .	23

3.1.2. NoSQL . . . . .	24
3.1.3. MongoDB y <i>e-Commerce</i> [117] . . . . .	26
3.2. <i>Node.js</i> . . . . .	31
3.2.1. Desempeño de Node.js frente a sus competidores . . . . .	33
3.2.2. <i>Full-Stack JavaScript</i> . . . . .	36
3.2.3. <i>Node.js</i> y <i>e-Commerce</i> . . . . .	37
3.3. <i>Isomorphic JavaScript: el futuro de las aplicaciones web</i> . . . . .	37
3.3.1. Single Page Application . . . . .	38
3.3.2. No todo lo que brilla es oro . . . . .	39
3.3.3. Un enfoque híbrido . . . . .	40
3.3.4. <i>Isomorphic JavaScript Frameworks</i> . . . . .	41
3.4. <i>Meteor</i> . . . . .	42
3.4.1. Principios de <i>Meteor</i> . . . . .	42
3.4.2. ¿Qué es <i>Meteor</i> ? . . . . .	43
3.4.3. El <i>Stack Meteor</i> . . . . .	44
3.4.4. <i>Framework Isomorphic – Full-Stack JavaScript</i> . . . . .	45
3.4.5. ¿Por qué <i>Meteor</i> ? . . . . .	45
3.4.6. ¿Por qué son necesarios los <i>Frameworks</i> ? . . . . .	46
<b>4. Arquitectura</b>	<b>48</b>
4.1. <i>Packages</i> desarrollados por la comunidad . . . . .	49
4.2. Estructura propuesta para el desarrollo de grandes aplicaciones . . . . .	50
<b>5. Solución implementada</b>	<b>57</b>
5.1. Arquitectura del <i>Framework</i> . . . . .	57
5.1.1. <i>Packages</i> . . . . .	57
5.1.2. <i>Workflows</i> . . . . .	59
5.2. Control de Acceso . . . . .	60

5.2.1. Creación de cuenta de un usuario . . . . .	61
5.2.2. Autenticarse . . . . .	62
5.2.3. Cerrar sesión . . . . .	63
5.2.4. Recuperar contraseña . . . . .	64
5.2.5. <i>Log in</i> con servicio de autenticación <i>third-party</i> . . . . .	65
5.3. Localización . . . . .	66
5.4. <i>Themes</i> . . . . .	67
5.5. La información persiste . . . . .	71
5.6. Productos . . . . .	71
5.6.1. Vista global de productos . . . . .	71
5.6.2. Descripción de un producto . . . . .	72
5.6.3. Creación de un producto . . . . .	72
5.6.4. Edición de un producto . . . . .	78
5.6.5. Visibilidad de un producto . . . . .	80
5.7. Carro Compra . . . . .	82
5.7.1. Agregar elementos al carro . . . . .	82
5.7.2. Consultar los elementos del carro . . . . .	83
5.8. Profile . . . . .	86
5.8.1. Actualizar contraseña . . . . .	86
5.8.2. Libreta de direcciones . . . . .	87
5.9. <i>Dashboard</i> . . . . .	90
5.9.1. <i>E-Commerce Framework Core</i> . . . . .	92
5.9.2. <i>Órdenes</i> . . . . .	96
5.9.3. <i>Cuentas</i> . . . . .	97
5.9.4. <i>Shipping</i> . . . . .	99
5.9.5. Pagos . . . . .	102
5.10. <i>Checkout</i> . . . . .	104

5.10.1. Cuenta de usuario . . . . .	106
5.10.2. Detalles de dirección . . . . .	106
5.10.3. Opciones de <i>Shipping</i> . . . . .	107
5.10.4. Resumen de la compra . . . . .	108
5.10.5. Pagos . . . . .	108
5.11. Órdenes . . . . .	111
5.12. Seguridad en un sitio <i>e-Commerce</i> . . . . .	111
5.13. Seguridad en <i>eframework</i> . . . . .	112
5.13.1. Autorización . . . . .	112
5.13.2. Autenticación . . . . .	112
5.13.3. Políticas de navegación . . . . .	113
5.13.4. Encriptación de información sensible . . . . .	113
5.13.5. Seguridad de escritura para las <i>Collections</i> de MongoDB . . . . .	113
<b>6. Conclusiones</b>	<b>114</b>
<b>7. Trabajos Futuros</b>	<b>115</b>
7.1. Mejora en la arquitectura genérica . . . . .	115
7.2. Actualizar el <i>Framework</i> a la nueva arquitectura . . . . .	116
7.3. <i>analytics</i> . . . . .	116
7.4. Opciones de <i>shipping</i> . . . . .	117
7.5. Seguridad . . . . .	117
<b>Glosario</b>	<b>118</b>
<b>Bibliografía</b>	<b>127</b>
<b>A. Gestión de Catálogos <i>handled</i> con una <i>Base de dato</i> relacional</b>	<b>I</b>
<b>B. ¿Qué es <i>Hadoop</i>® <i>B</i>? [104]</b>	<b>II</b>

<b>C. Conexiones <i>Stateful</i> y <i>Stateless</i></b>	<b>IV</b>
<b>D. <i>Pseudo-schemas e-Commerce</i></b>	<b>V</b>
<b>E. <i>UI</i> de los formularios</b>	<b>IX</b>
<b>F. Soluciones de otras plataformas <i>e-Commerce</i></b>	<b>XIV</b>
<b>G. Ejemplos de Formularios</b>	<b>XXIII</b>

# Índice de figuras

1.1. <i>e-Commerce</i> como porcentaje de las ventas totales en <i>Estados Unidos</i> [165]. . . . .	2
2.1. Categorías de <i>e-Commerce</i> . . . . .	9
2.2. <i>OpenCart website</i> [12]. . . . .	10
2.3. <i>PrestaShop website</i> [15]. . . . .	10
2.4. <i>Magento website</i> [8]. . . . .	11
2.5. <i>Zen Cart website</i> [23]. . . . .	11
2.6. <i>Spree Commerce website</i> [17]. . . . .	12
2.7. <i>Drupal Commerce website</i> [5]. . . . .	12
2.8. <i>osCommerce website</i> [13]. . . . .	13
2.9. <i>simpleCart website</i> [16]. . . . .	13
2.10. <i>WooCommerce website</i> [19]. . . . .	14
2.11. <i>WP e-Commerce website</i> [22]. . . . .	14
2.12. <i>Jigoshop website</i> [7]. . . . .	15
3.1. Esquemas de un producto. . . . .	27
3.2. <i>Vertical Scaling</i> o <i>Scaling Up</i> . . . . .	30
3.3. <i>Horizontal Scaling</i> o <i>Scaling Out</i> . . . . .	30
3.4. Diagrama de Técnica tradicional vs. <i>Node.js servidor thread</i> . . . . .	32
3.5. <i>Performance</i> de la aplicación en <i>Java</i> y <i>Node.js</i> . . . . .	34
3.6. <i>Performance</i> de la aplicación <i>Node.js</i> . . . . .	35
3.7. <i>Performance</i> de la aplicación en <i>PHP/Apache™</i> . . . . .	35



3.8. <i>Client-side MVC</i> . . . . .	39
3.9. MVC <i>Isomorphic</i> cliente servidor. . . . .	40
3.10. Arquitectura de <i>Mainframe</i> con <i>dumb terminal</i> . . . . .	42
3.11. El <i>Stack Meteor</i> corre aplicaciones potenciadas por <i>Packages</i> inteligentes sobre <i>Node.js</i> y <i>MongoDB</i> . . . . .	44
3.12. Un <i>wrapper</i> provee valor agregado. . . . .	47
3.13. Una <i>Arquitectura associates/contains objects</i> . (La <i>interface</i> es un <i>wrapper</i> de la <i>Arquitectura</i> ). . . . .	47
3.14. Una <i>Metodología</i> define las interacciones entre <i>Arquitectura</i> , <i>componentes</i> y <i>objects</i> . . . . .	47
4.1. Estructura simple de una aplicación en <i>Meteor</i> . . . . .	51
4.2. Estructura global para una aplicación en <i>Meteor</i> . . . . .	52
4.3. Estructura de la carpeta <b><i>client</i></b> para una aplicación en <i>Meteor</i> . . . . .	52
4.4. Estructura de la carpeta <b><i>template</i></b> para una aplicación en <i>Meteor</i> . . . . .	53
4.5. Estructura de la carpeta <b><i>common</i></b> para una aplicación en <i>Meteor</i> . . . . .	54
4.6. Estructura de la carpeta <b><i>docs</i></b> para una aplicación en <i>Meteor</i> . . . . .	54
4.7. Estructura de la carpeta <b><i>lib</i></b> para una aplicación en <i>Meteor</i> . . . . .	54
4.8. Estructura de la carpeta <b><i>private</i></b> para una aplicación en <i>Meteor</i> . . . . .	55
4.9. Estructura de la carpeta <b><i>tests</i></b> para una aplicación en <i>Meteor</i> . . . . .	55
4.10. Estructura de la carpeta <b><i>server</i></b> para una aplicación en <i>Meteor</i> . . . . .	56
5.1. Estructura de los <i>Packages</i> del <i>eframework</i> . . . . .	57
5.2. Máquina de estados de carro de compras. . . . .	60
5.3. Máquina de estados del estado de una orden. . . . .	60
5.4. Ejemplos de formularios de creación. El formulario de la izquierda pide confirmación de <i>email</i> y contraseña, implicando en un mayor número de campos a llenar. . . . .	61
5.5. Formulario de creación de una cuenta. . . . .	62
5.6. Formulario de autenticación en la aplicación. . . . .	63
5.7. Desvincular la cuenta de la aplicación. . . . .	64

5.8. Formulario de restauración de contraseña. . . . .	64
5.9. La aplicación ofrece autenticarse utilizando un servicio <i>third-party</i> ( <i>Facebook</i> ) o directamente en el sitio. . . . .	65
5.10. Formulario de autenticación para utilizar uno de varios servicios <i>third-party</i> . . . . .	66
5.11. Selección de idioma para el <i>website</i> . . . . .	67
5.12. Descripción de un producto utilizando el <i>Theme united</i> del sitio <i>Bootswatch</i> [139]. . . . .	68
5.13. Descripción de un producto utilizando el <i>Theme journal</i> del sitio <i>Bootswatch</i> [139]. . . . .	68
5.14. Descripción de un producto utilizando el <i>Theme cerulean</i> del sitio <i>Bootswatch</i> [139]. . . . .	69
5.15. Descripción de un producto utilizando el <i>Theme superhero</i> del sitio <i>Bootswatch</i> [139]. . . . .	69
5.16. Descripción de un producto utilizando el <i>Theme paper</i> del sitio <i>Bootswatch</i> [139]. . . . .	70
5.17. Descripción de un producto utilizando el <i>Theme simplex</i> del sitio <i>Bootswatch</i> [139]. . . . .	70
5.18. Descripción de un producto utilizando el <i>Theme yeti</i> del sitio <i>Bootswatch</i> [139]. . . . .	71
5.19. Vista global de los productos. Productos con problemas de <i>stock</i> dan información. . . . .	72
5.20. Interfaz de creación de un producto. . . . .	73
5.21. Título principal del producto. Aparece también en la lista de búsqueda de productos. . . . .	74
5.22. Campo para la descripción del producto. . . . .	74
5.23. Campo para el <i>Vendor</i> del producto. . . . .	74
5.24. Sección de las imágenes del producto. . . . .	75
5.25. Campo para las <i>Tags</i> del producto. . . . .	75
5.26. Sección de las opciones del producto. . . . .	77
5.27. Interfaz de la edición de un producto. . . . .	78
5.28. Campo descripción seleccionado para edición. . . . .	79
5.29. Campo título seleccionado para edición. . . . .	80
5.30. Producto con visibilidad limitada. Notar que no tiene título, lo que implica que no es un producto válido aún. . . . .	81
5.31. Resultado tras dar visibilidad a un producto sin título. Se genera un <i>auto-foco</i> en el campo título, dado que este es requerido y el estado de visibilidad del producto continúa igual. . . . .	81
5.32. Producto con visibilidad global. Con el botón se oculta el producto. . . . .	82

5.33. Botón para agregar productos al carro. En la figura se han seleccionado 4 productos Vintage Desktop para agregar al carro de compra. . . . .	82
5.34. <i>Feedback</i> del nuevo producto agregado al carro. . . . .	83
5.35. Mala práctica en el diseño del botón para agregar elementos al carro. Entre otros detalles el botón no tiene un diseño que lo distinga del botón agregar a la lista de deseos. . . . .	83
5.36. Icono que muestra la cantidad de elementos que tiene el carro. El icono del carro es un botón para la vista del carro. . . . .	83
5.37. Información global del carro. Tiene un resumen del costo total de los elementos seleccionados. . . . .	84
5.38. Vista básica de un producto seleccionado. La [x] permite eliminar ese producto del carro de compra. . . . .	85
5.39. El botón del carro a cambiado de color. Esta informando que un producto esta con <i>Limited Stock</i> . . . . .	85
5.40. El contenedor del producto <i>T-shirt</i> esta en amarillo. Esto indica que el producto esta con <i>Limited Stock</i> . . . . .	86
5.41. Formulario de actualización de contraseña. . . . .	87
5.42. Formulario para agregar una nueva dirección. . . . .	88
5.43. Lista con todas las direcciones configuradas del usuario. Cada una de estas puede ser editada o eliminada con los botones que se encuentran en el costado derecho. . . . .	89
5.44. Formulario para agregar una nueva dirección, aparece en caso de no existir configurada ninguna previa. Notar que no existe un botón para cancelar dado que en este contexto no tiene sentido. . . . .	90
5.45. <i>Dashboard</i> con elementos configurables de la aplicación. . . . .	91
5.46. <i>Dashboard</i> con elementos configurables de la aplicación. . . . .	91
5.47. Menú general de <i>E-Commerce Framework Core</i> . . . . .	92
5.48. Mensaje de confirmación de éxito en la actualización del formulario. Este mensaje se esconde después de un breve intervalo de tiempo. . . . .	92
5.49. Mensaje para informar sobre un error en el proceso de actualización del formulario. . . . .	92
5.50. Componente para permitir <i>checkout</i> desde una cuenta <i>Guest</i> . . . . .	93
5.51. Formulario de información general de la tienda. . . . .	93
5.52. Formulario de la dirección física de la tienda. . . . .	94

5.53. Formulario con la información de la configuración del Mail. . . . .	95
5.54. Vista general de todas las <i>Órdenes</i> ingresadas por clientes. . . . .	96
5.55. Orden ingresada por un cliente. . . . .	97
5.56. Interfaz con los usuarios del sistema. . . . .	97
5.57. Interfaz para administrar los permisos del sistema. . . . .	98
5.58. Tabla con todas las opciones de <i>Shipping</i> disponibles. . . . .	99
5.59. Formulario para la creación de <i>Shipping</i> . . . . .	100
5.60. Formulario de actualización un <i>Shipping</i> . . . . .	101
5.61. <i>Shipping</i> es un factor clave en el abandono de carros de compra [62]. . . . .	102
5.62. Formulario para agregar la configuración necesaria para el uso de <i>PayPal PayFlow</i> . . . . .	104
5.63. Detalle de todos los pasos del <i>workflow Shipping</i> . . . . .	105
5.64. Estado actual del <i>workflow</i> del <i>Shipping</i> . En este caso se encuentra en la selección del tipo de <i>Shipping</i> . . . . .	106
5.65. Despliegue automático del formulario de direcciones cuando no se tiene ninguna dirección agregada. . . . .	107
5.66. Opciones de <i>Shipping</i> configuradas en el sistema. . . . .	108
5.67. Mensaje cuando no hay opciones de <i>Shipping</i> configuradas en el sistema. . . . .	108
5.68. Detalle de la compra dentro del proceso de <i>Checkout</i> . . . . .	109
5.69. Formulario de la tarjeta de crédito para realizar el pago utilizando <i>PayPal Payment Pro</i> . . . . .	111
A.1. <i>Schemas de Magento e-Commerce Framework</i> . . . . .	I
E.1. Errores al enviar el formulario de creación de una nueva cuenta. . . . .	IX
E.2. Errores al enviar vacío el formulario de autenticación. . . . .	X
E.3. Errores del formulario de actualización de contraseña tras enviarlo vacío. . . . .	X
E.4. Ingreso de nueva <i>contraseña fácil</i> . . . . .	X
E.5. Ingreso de contraseña actual incorrecta. . . . .	XI
E.6. Errores en el formulario tras enviarlo. El campo <i>Name</i> es obligatorio. . . . .	XI
E.7. Comentarios del formulario tras error de actualización. . . . .	XII

E.8. Formulario de creación de <i>Shipping</i> tras enviarlo vacío. . . . .	XIII
F.1. Formulario para agregar una dirección en <i>Amazon</i> . . . . .	XIV
F.2. Formulario de <i>Amazon</i> para agregar una tarjeta. . . . .	XV
F.3. Formulario de <i>PayPal</i> para agregar una tarjeta. . . . .	XV
F.4. Resumen del carro de compra del sitio <i>Shopify</i> . . . . .	XV
F.5. Formulario de <i>Shopify</i> para agregar un producto. . . . .	XVI
F.6. Sección del formulario de <i>Shopify</i> para agregar un título y una descripción. . . . .	XVI
F.7. Sección del formulario de <i>Shopify</i> para agregar imágenes de producto. . . . .	XVII
F.8. Sección del formulario de <i>Shopify</i> para agregar inventario al producto. . . . .	XVII
F.9. Sección <i>Inventory</i> del formulario de <i>Shopify</i> con opción <i>Tracking</i> aceptada. . . . .	XVII
F.10. Sección del formulario de <i>Shopify</i> para agregar información de la organización del producto. . . . .	XVIII
F.11. Sección del formulario de <i>Shopify</i> para agregar información del precio. . . . .	XVIII
F.12. Sección del formulario de <i>Shopify</i> para agregar información de <i>shipping</i> al producto. . . . .	XIX
F.13. Sección del formulario de <i>Shopify</i> para agregar variantes de un producto. . . . .	XIX
F.14. Sección <i>Variants</i> del formulario de <i>Shopify</i> con dos variantes del producto. . . . .	XX
F.15. Sección del <i>website</i> de <i>leif</i> para agregar elementos al carro. . . . .	XX
F.16. Sección <i>Visibility</i> del formulario de <i>Shopify</i> del producto. . . . .	XXI
F.17. Modelo de Datos para <i>e-Commerce</i> obtenido desde el sitio <i>DataBase Answers</i> [51]. . . . .	XXI
F.18. Lista de <i>Órdenes</i> para el <i>website dealextrême DX</i> . . . . .	XXII
F.19. Detalles de una <i>Orden</i> para el <i>website dealextrême DX</i> . . . . .	XXII
G.1. Formulario de creación de usuario para <i>Twitter</i> . . . . .	XXIII
G.2. Formulario de creación de usuario para <i>Instagram</i> . . . . .	XXIII
G.3. Formulario de creación de usuario para <i>Netflix</i> . . . . .	XXIV
G.4. Formulario de creación de usuario para <i>LinkedIn</i> . . . . .	XXIV
G.5. Formulario de creación de usuario para <i>Dropbox</i> . . . . .	XXV

# Índice de tablas

2.1. Resumen <i>Full-Stack JavaScript Frameworks</i> . . . . .	21
3.1. Resumen NoSQL vs. SQL. . . . .	26
3.2. Ubicación del <i>Framework Meteor</i> [178]. . . . .	46
5.1. Resumen de restricciones para formulario creación de cuentas de usuario. . . . .	62
5.2. Resumen de restricciones para formulario de autenticación de usuario. . . . .	63
5.3. Resumen de restricciones para recuperación de contraseña. . . . .	65
5.4. Resumen restricciones para formulario creación de un producto. . . . .	76
5.5. Resumen restricciones para formulario creación de <i>Details</i> . . . . .	76
5.6. Resumen restricciones para formulario creación de <i>Options</i> . . . . .	77
5.7. Resumen restricciones formulario edición de contraseña (Figura E.3 y Figura E.4). . . . .	87
5.8. Resumen restricciones formulario para libreta de direcciones. . . . .	88
5.9. Restricciones formulario <i>General</i> . . . . .	93
5.10. Restricciones formulario <i>Address</i> . . . . .	95
5.11. Restricciones formulario <i>Mail</i> . . . . .	95
5.12. Resumen restricciones formulario para los permisos. . . . .	98
5.13. Resumen restricciones formulario para <i>Shipping</i> . . . . .	99
5.14. Resumen restricciones formulario <i>PayPal PayFlow</i> . . . . .	104

# Capítulo 1

## Introducción

Una de las actividades más populares en la *web* es comprar. La razón es porque tiene un encanto especial: puedes comprar en tiempo libre, en cualquier momento. Literalmente cualquiera puede tener páginas para mostrar sus productos y servicios, como ejemplo se encuentran *Amazon* [25], *Ebay* [35] y *BEST BUY* [27].

La historia de *e-Commerce* se remonta a la invención de la básica noción de *comprar y vender*, electricidad, cables, computadores, *modems*, y la *Internet*. *e-Commerce* se hace posible en 1991 cuando la *Internet* estuvo disponible para uso comercial. Desde entonces miles de negocios se han establecido en sitios *web*.

En un inicio, el término *e-Commerce* se refería al proceso de ejecución de transacciones electrónicas comerciales con la ayuda de tecnologías líderes, tales como *Electronic Data Interchange* y *Electronic Funds Transfer*, las cuales dieron la oportunidad a los usuarios para intercambiar información de negocios y realizar transacciones electrónicas. La oportunidad para utilizar estas tecnologías surgió a finales de 1970s [39] y permitió a los negocios de las compañías y organizaciones enviar documentación electrónica comercial.

Aunque la *Internet* comenzó a ganar popularidad entre el público general en 1994, tomó aproximadamente cuatro años desarrollar protocolos de seguridad (por ejemplo *HTTP*) y *DSL (modem)*, los cuales permitieron un acceso rápido y conexiones persistentes a la *Internet*. Con el rápido crecimiento de la *Internet*, las personas comenzaron a considerar maneras de obtener un beneficio. Al mismo tiempo, las compañías entendieron que podrían utilizar *Internet* como una forma de promover sus negocios a potenciales *clientes* al rededor del mundo [65]. Al principio, solo mostraban información sobre sus productos, pero los *clientes* aun tenían que llamar a la compañía para realizar su *orden*. Las compañías rápidamente comprendieron la necesidad de integrar la venta de sus productos y servicios sobre *Internet* a fin de alcanzar tantos *clientes* como fuera posible. Con la mejora de la tecnología, las compañías tuvieron la posibilidad de no solo poner información de sus productos *online*, además pudieron vender sus productos a los *clientes* también [126].

Para finales de 2001, el modelo de negocio más grande de *e-Commerce*, *B2B*, había ganado

alrededor de \$700 billones en transacciones [60].

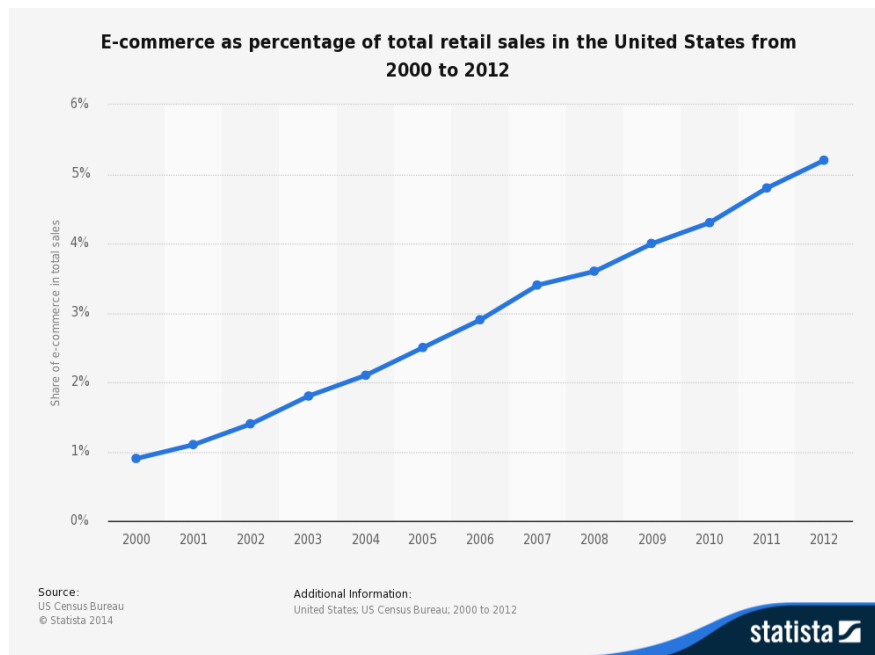


Figura 1.1: *e-Commerce* como porcentaje de las ventas totales en *Estados Unidos* [165].

Acorde a toda información disponible, las ventas *e-Commerce* continuaron creciendo en los siguientes años y en 2012 las ventas *e-Commerce* representaron el 5.2% de las ventas totales en *Estados Unidos*. Como se puede observar en Figura 1.1.

*e-Commerce* tiene muchas ventajas sobre tiendas *Brick-and-Mortar* y catálogos de venta por correo. Los consumidores pueden fácilmente buscar a través de una *Base de datos* muy extensa de productos y servicios. Pueden ver los precios reales, definir una orden de compra en varios días y enviar un correo como un *wish list* esperando que alguien pague por sus productos seleccionados [56]. Los clientes pueden comparar precios con un simple *click* del *mouse* y comprar los productos seleccionados al mejor precio.

Proveedores *online*, en sus turnos, también tienen claras ventajas. La *web* y sus motores de búsqueda proveen una manera para encontrar clientes sin campañas de publicidad costosas. Incluso tiendas *online* pequeñas pueden alcanzar mercados globales.

La tecnología *web* también permite realizar un seguimiento sobre las preferencias de los clientes para ofrecer una comercialización personalizada.

### 1.0.1. *e-Commerce* en la actualidad

En la actualidad, *e-Commerce* es una experiencia destacable. Transformó las compras tradicionales más allá de lo reconocible. La experiencia es mucho mejor que cualquier otra manera de comprar, seduciendo a una gran cantidad de *e-Commerce lovers*.



Si hace algunos años *e-Commerce* fue una palabra de moda, ahora se ha convertido en la orden del día. Al parecer las personas compran literalmente en cualquier parte, en sus lugares de trabajo, durante el almuerzo, en las horas punta cuando no hay nada más por hacer salvo encender el computador y comenzar a navegar.

En el presente, *e-Commerce* ha ganado tanta popularidad debido a que su tecnología subyacente está evolucionando a pasos agigantados. Incluso se está ofreciendo "*sentir*" el producto con un *mouse 3D* para comprender mejor su forma, tamaño y textura. ¿Para qué salir cuando todo lo que se debe hacer es realizar un pedido, elegir la forma de envío, pararse y esperar hasta que la orden sea entregada en la puerta de la casa?

*e-Commerce* hoy ofrece tanta comodidad que incluso las tiendas convencionales han encendido las alarmas. Aunque, cada uno está de acuerdo en que hay un gran camino para que *e-Commerce* reemplace las tiendas, la posibilidad existe que ocurra en el futuro. *e-Commerce* del cual somos actualmente testigos trae tanta aventura a nuestras vidas que es disfrutado por toda la comunidad *online*.

En la actualidad *e-Commerce* tiene algunos inconvenientes, sin embargo, muchos consumidores están dispuestos a aguantar estas desventajas, ya que confían en el mundo *online* y desean que sea un mejor lugar.

*e-Commerce* refleja lo que hemos creado desde un inicio para el comercio electrónico *online*. Fue creado por nosotros y pensado para nosotros.

### **1.0.2. *e-Commerce* en el futuro**

Expertos predicen un futuro prometedor para *e-Commerce*. En un presumible futuro *e-Commerce* se confirmará por sí misma como una herramienta importante para las ventas. El éxito de *e-Commerce* se convertirá en una noción inseparable de la *web*, ya que *e-Shopping* se está volviendo cada vez más y más popular y natural. Al mismo tiempo la competitividad en el mundo de los servicios *e-Commerce* intensificará su crecimiento. Así la tendencia de que prevalecerá *e-Commerce* será el crecimiento de las ventas por *Internet* y la evolución.

Cada año el número de ofertas de *e-Commerce* crece enormemente. El volumen de ventas de las tiendas *online* es más que comparable con esos *Brick-and-Mortar*. Y la tendencia va a continuar [134], porque hay mucha gente *limitada* por las obligaciones laborales y de los hogares, mientras que *Internet* permite ahorrar mucho tiempo y dinero al permitir elegir los productos a los mejores precios.

La tendencia de *cantidad a calidad* del *e-Commerce* se está haciendo cada vez más evidente, ya que *Internet* ha incluido el factor geográfico de la venta. Así que no importa si su tienda se encuentra en *New York*, *Londres* o en una pequeña ciudad. Para sobrevivir, los comerciantes tendrán que adaptarse rápidamente a las nuevas condiciones. Para atraer a más clientes, los propietarios de las tiendas *online* tendrán no solo que incrementar el número de servicios disponibles, además deberán poner

más atención a elementos como *design* atractivo, *user friendliness*, presentación atractiva; tendrán que oportunamente emplear tecnología moderna para que sus negocios sean parte del futuro del *e-Commerce*.

Claramente, aquellos que adquieren *e-stores* antes, tienen mejores oportunidades para el éxito y prosperidad, aunque un sitio *e-Commerce* por sí mismo no garantiza nada. Solo una apropiada solución *e-Commerce* en combinación con *e-Marketing* y publicidad pueden asegurar éxito.

*e-Commerce* está cambiando fundamentalmente la economía y la manera en que los *negocios* son conducidos. *e-Commerce* obliga a las compañías a encontrar nuevas maneras para expandir los mercados en los que compite, para atraer y retener *clientes* adaptando los productos y servicios para sus necesidades, y para reestructurar sus procesos de *negocios* para entregar productos y servicios de forma más eficiente y efectiva. Sin embargo, a pesar de los rápidos y sostenidos desarrollos de *e-Commerce*, muchas empresas que realizan negocios electrónicos aún permanecen en la fase de inversión y creación de marca. Muchos *e-business* se han centrado en el atractivo visual y la facilidad de uso del *website* como la manera para aumentar la base de *clientes*. Sin embargo, tal como *e-business* cambia su enfoque de construir una base para el *cliente* para incrementar el *revenue* y la rentabilidad, deben re-evaluar sus sus estrategias para proveer uuna clara rentabilidad [158].

En el mundo empresarial, el tamaño importa, pero ni siquiera eso garantiza un futuro en un mercado de constante cambio y con una competencia feroz. Innumerables son las empresas que murieron en el intento, entre las cuales podemos destacar: *Sega, Kodak, Daewoo, Nokia, Blockbuster*, entre otros.

En la actualidad existe una gran variedad de *Frameworks Open Source* disponibles que permiten desarrollar una gran variedad de soluciones *e-Commerce*. Todas estas opciones tienen grandes comunidades que las respaldan, así como muchos usuarios satisfechos [2, 5, 7, 8, 12, 13, 15–17, 19, 22, 23].

Todas estas opciones fueron construidas en un escenario en donde el poder de procesamiento en los clientes era limitado, pero en estos días un solo *iPhone* tiene un poder de cómputo superior a la mayoría de los súper computadores en los inicios de la *web*.

Después de años de avance en computadores personales, tecnologías creativas han surgido, y los *web browsers* han evolucionado para mantenerse al día. En la actualidad, la *web* ha madurado alcanzando una serie de características que permiten aplicaciones enriquecidas que mejoran las experiencias de los usuarios. Es entonces factible desarrollar *Frameworks e-Commerce* mejores tanto del punto de vista del usuario final como de los desarrolladores.

## 1.1. Motivación

Los *online retailers* están en la cúspide de una forma totalmente nueva de hacer negocios. Ellos tienen una oportunidad única para obtener ventajas competitivas significativas en sus respectivos

mercados, siempre que entreguen consistentemente una experiencia grata para el consumidor y permitan características de comercio *Multi Channel* únicos antes que sus competidores. El éxito dependerá en perfeccionar los esfuerzos para abordar las experiencias de clientes centrados en el usuario, reducir el enfoque a los programas más valiosos, y eligiendo estrategias de tecnología adecuada que permitan a los equipos internos ofrecer experiencias escalables optimizadas. Las áreas emergentes a observar son *tiempo-real*, *retail analytics*, incorporar *red social* en *e-Commerce*, el continuo éxito de aplicaciones móviles, y herramientas que permitan a *retailers* escalar experiencias que dirijan de manera más efectiva el *Merchandising*.

Existe una amplia variedad de escenarios en donde se encuentran soluciones *e-Commerce* o similares con la potencialidad de ser mejorados permitiendo entrar al mercado con ventajas competitivas. A continuación se describen variadas situaciones que tienen gran potencial latente.

- **Shopping Cart:** software *e-Commerce* en un servidor *web* que permite a los visitantes del *website* seleccionar productos para eventualmente comprarlos.
- **Reclamos ciudadanos:** Un sistema para realizar y gestionar reclamos que estén relacionados principalmente con temas sociales con el fin de alertar rápidamente a los organismos públicos responsables. Ejemplos de uso serían: calles en mal estado, semáforos dañados, semáforos mal sincronizados, cruces peligrosos para peatones, aceras en mal estado, lugares de ruido excesivo, zonas de robos comunes, plazas en mal estado, accesos cerrados a playas. La idea básica es que un usuario pueda crear contenido (quejas), permitiendo a otros usuarios apoyarlas, para informar oportunamente a las autoridades responsables.
- **Reservas de horas médicas:** Un sistema para gestionar las horas médicas que muestre sugerencias de acuerdo a la distancia que se encuentran los especialistas desde la *geo-position* del sistema que realiza la solicitud.
- **Consultas de libros en una biblioteca:** Un sistema para consultar la disponibilidad, la cantidad, la posibilidad de reservarlo, e incluso cuándo será devuelto.
- **Catálogo de moteles:** Un catálogo de moteles de acuerdo a la *geo-position* pueden ser listados para determinar las mejores opciones disponibles. Adicionalmente se podría mostrar los horarios disponibles de las habitaciones, e incluso poder generar reservas y pagar la habitación. De esta manera se optimiza el tiempo de los clientes, y los recursos de los proveedores.
- **Catálogo de medicamentos:** Un catálogo completo de los medicamentos oficialmente aceptados podrían ser listados para entregar información relevante de acuerdo a las dosis así como de efectos adversos, si es necesaria una receta, etc. Otra cosa interesante es dar la opción de mostrar todos los remedios genéricos que existen en el mercado. El escenario ideal sería además mostrar el precio de estos, así como la distancia a las droguerías en donde el producto se encuentra disponible.
- **Catálogo de Fiestas:** Un catálogo con las fiestas disponibles tanto a la brevedad posible como en el futuro. Que muestre información relevante. Y que permita la compra de entradas.

- **Gestor de detalle de cuenta en restaurante:** Permite gestionar el detalle de la cuenta de un restaurante. La idea básica es tener una mejor visión tanto de mis órdenes como las del total de la mesa en la cual me encuentro, evitando así problemas en relación con el ajuste total de la cuenta, sorpresas en el detalle del consumo, cobros erróneos, etc. Además de permitir prepagos que no necesariamente se realizarán al final de la reunión (posibles clientes que se retiren antes que el grupo en el cual se encuentran).

Todas estas situaciones tienen algo en común, y es que los usuarios desean consultar información para tomar una decisión. El sistema idealmente permite contrastar datos entre diferentes proveedores de servicios similares, así como de aportar con información clara y relevante, para que futuros usuarios puedan tomar una decisión aún más informados.

Como se puede concluir, el modelo de negocio de estas situaciones es en su mayoría el mismo, solo cambiando la presentación para hacerlo *ad-hoc* a cada situación [2,5,7,8,12,13,15–17,19,22,23].

En la actualidad, existe una gran variedad de *Frameworks* disponibles que gracias a sus funcionalidades genéricas, agrupadas selectivamente, permiten desarrollar estas y otras soluciones específicas agregando código escrito por un desarrollador. Sin embargo, estas herramientas fueron adoptadas cuando muchos de los conceptos actuales en los que se mueve la *Internet* no existían, como por ejemplo:

- Desarrollo de aplicaciones *tiempo-real*, *reactive*, las cuales permiten deseables *características* en un *Framework* de esta naturaleza.
- Soporte completo para desarrollo en *dispositivos móviles*, permitiendo nuevas ventajas para *e-Commerce* [65].

Se propone desarrollar un *Framework free Open Source* utilizando tecnología *web* que permita crear aplicaciones enriquecidas con características no presentes en los *Frameworks e-Commerce free Open Source* disponibles en la actualidad.

## 1.2. Alcances y objetivo general

Implementar un *Framework e-Commerce* utilizando tecnología *free Open Source* que permita (en comparación a los *Frameworks* actuales): resultados más rápidos, ideal para *prototypes* y *MVP*; una solución completa para crear *características* para *servidores* [128], *browsers* y *dispositivos móviles*; *built-in* con capacidades *tiempo-real* para minimizar esfuerzo en *Desarrollo*; e integración de herramientas de desarrollo para hacer que *setup*, *Desarrollo*, y *deployment* sea extremadamente rápido.

### 1.3. Objetivos específicos

- Especificar los requerimientos del diseño del *Framework*.
- Escoger la tecnología más adecuada para el desarrollo del proyecto.
- Determinar la arquitectura adecuada para el desarrollo del *Framework*.
- Implementar la solución.
- Desarrollar una solución utilizando EE-Commerce, que permite contrastar las potencialidades con respecto a *Frameworks* existentes.

## Capítulo 2

# Estado del Arte

### 2.1. e-Commerce

Para muchos *negocios*, *e-Commerce* significa utilizar la *Internet* como un *channel* de distribución para comercializar y vender productos y servicios a los *clientes*. Esta definición solo incluye *Internet Commerce*. *e-Commerce* abarca aún más. *e-Commerce* es el intercambio electrónico de información, productos, servicios, y pagos e incluye la creación y mantenimiento de relaciones *web-based*. Por lo tanto, *e-Commerce* incluye, pero no está limitado a *Internet*, *Intranets*, *Extranets*, *Electronic Data Interchange*, y otros. Ejemplos de *e-Commerce* incluyen proceso de *Transactions* con *Electronic Payments*, coordinación con *partner negocios* tal como *Manejo* de inventario, *self-service* del *cliente* como *tracking* el estado de la *orden*, la investigación de resolución de problemas, y utilizar un *Intranet* de la corporación para la distribución de la información ubicua [83].

#### 2.1.1. Categorías de e-Commerce

Tecnología digital avanzada, combinada con las empresas y los clientes de estas tecnologías, impulsan *e-Commerce*. De igual forma que las tecnologías digitales, *e-Commerce* no pudo alcanzar su meta en un movimiento. En cuanto a las empresas y clientes, *e-Commerce* de diferentes tipos y niveles implica diferentes oportunidades [200].

En relación con categorías de transacciones, *e-Commerce* se divide en 5 tipos:

- **B2B.** Es simplemente definido como *e-Commerce* entre compañías. Este es el tipo de *e-Commerce* que tiene que ver con las relaciones entre y dentro de los negocios. Cerca del 80 % de *e-Commerce* que existe, corresponde a esta categoría, y cada vez son más los expertos que predicen que *B2B e-Commerce* continuará su crecimiento a una velocidad mayor que el segmento *B2C*.
- **B2C.** En este caso, *Internet* es utilizado por negocios y empresas para proveer bienes y servicios

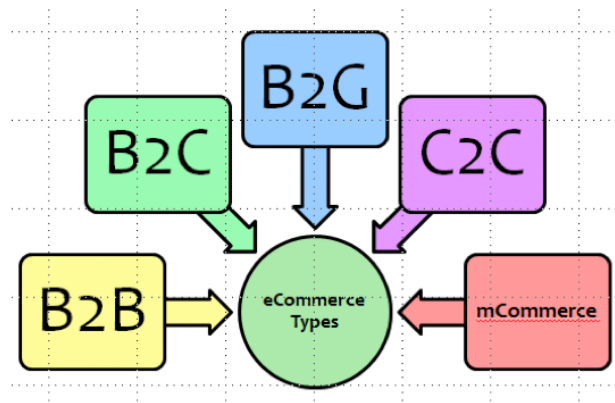


Figura 2.1: Categorías de *e-Commerce*.

por sitios *web* a los clientes. Actualmente varios tipos de sitios *web B2C* están repartidos por *Internet* para suministrar a los clientes una variedad de bienes y servicios que van desde flores y libros hasta ordenadores y autos, etc.

- **B2G.** El gobierno, como administrador nacional, desempeña un papel importante en la orientación, administración y ajuste de la economía. El advenimiento de la era de *e-Commerce* planteó fundamentos *e-Commerce* a las nuevas solicitudes de las funciones originales de los gobiernos. Por un lado, el gobierno debería administrar *e-Market* efectivamente y prestar un mejor servicio a las empresas y al público a través de *e-Government*. Por otra parte, los gobiernos, como los "*grandes clientes*" en economía deberían tomar la iniciativa para adoptar *e-Commerce* y ofrecer maneras eficientes a través de invitación de licitación electrónica para las contrataciones públicas.
- **C2C.** Es un comercio simple entre individuos privados o consumidores. Este tipo de *e-Commerce* está caracterizado por el crecimiento de mercados electrónicos y las subastas *online*, particularmente en industrias verticales donde empresas/negocios pueden pujar por lo que quieren de entre varios proveedores.
- **m-Commerce.** Corresponde a comprar y vender productos y servicios a través de tecnología *wireless*, *i.e.*, dispositivos *handheld*, tales como *smartphones*, *PDA*s, *tablets*, etc. Mientras la entrega de contenido por *dispositivos wireless* se vuelve *Faster*, más *Secure*, y *Scalable*, se cree que *m-Commerce* superará a *wireline e-Commerce* como la elección para las *Transactions Commerce*.

### 2.1.2. Ejemplos de *Frameworks e-Commerce* existentes

He aquí una lista de 11 soluciones *e-Commerce Open Source*. El *core* de todas las aplicaciones es *free*. Cada aplicación tiene extensiones *free* y *premium*, y opciones de soporte para mejorar el desarrollo de la *store*.

- **OpenCart.** Es una aplicación *Open Source*, basada en *PHP*, es una solución *e-Commerce* para comerciantes *online*. *OpenCart* tiene una comunidad activa y leal para el apoyo de usuarios, así como una lista de socios comerciales para instalación y customización profesional. En *OpenCart* existen más de 20 medios de pago y más de 8 métodos de envío para la descarga por defecto, con cientos de métodos adicionales para el envío y el pago en su directorio de extensiones. *OpenCart* también fue diseñado para un manejo sencillo de múltiples compras desde una interfaz de administración. Tiene más de 2700 *Themes*.

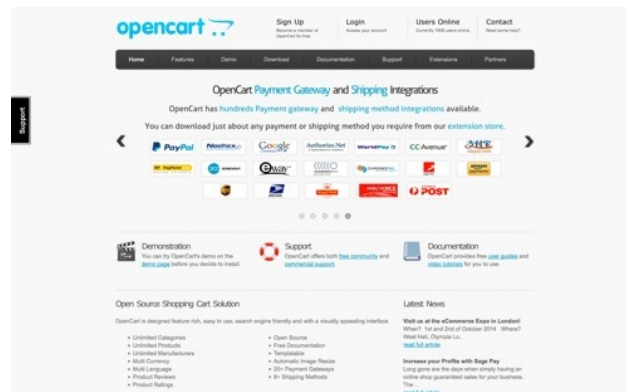


Figura 2.2: *OpenCart* website [12].

- **PrestaShop.** Es una solución *e-Commerce Open Source*, escrita en *PHP* y basada en *Smarty Template Engine*. *PrestaShop* viene con más de 310 características integradas y 3500 módulos y *templates*. Cuenta con ventas cruzadas, productos descargables, la exportación de productos, una página de pago, envío, descuentos y mucho más. Descargado más de 4 millones de veces, *PrestaShop* es usado en 160 países y traducido a 63 idiomas. Tiene más de 600000 miembros en su comunidad.

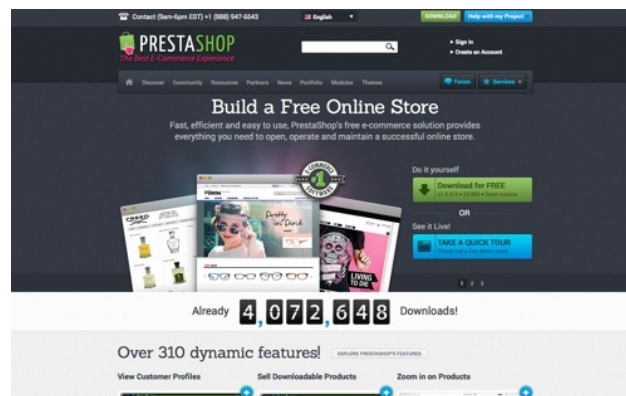


Figura 2.3: *PrestaShop* website [15].

- **Magento Community Edition** es una versión *free* y *Open Source* de una plataforma *e-Commerce*. Los comerciantes pueden acceder a características adicionales instalando las extensiones desde el gran *Magento Connect Marketplace*. No existe soporte para *Magento Community Edition*, así que las respuestas a las preguntas técnicas deben ser resueltas a través



del foro de usuarios. Un detalle, *Magento* ha anunciado el cierre de su *hosted solution*, *Magento Go*, por ahora no habría problemas con *Community Edition*. *Magento Community Edition* soporta más de 200000 sitios de clientes.

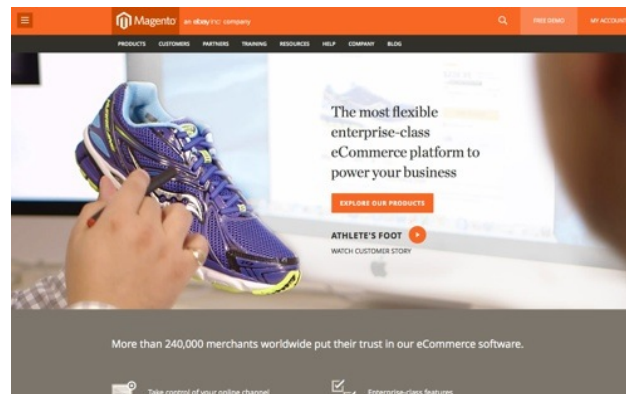


Figura 2.4: *Magento website* [8].

- **Zen Cart** es una aplicación *e-Commerce Open Source* escrita en *PHP*. *Zen Cart* branched desde el código *osCommerce* en 2003, con una solución que era más *template-based*. Tiene más de 1800 *add-ons* en 16 categorías. La comunidad de apoyo de *Zen Cart* tiene aproximadamente 150000 miembros y 200000 *threads*.

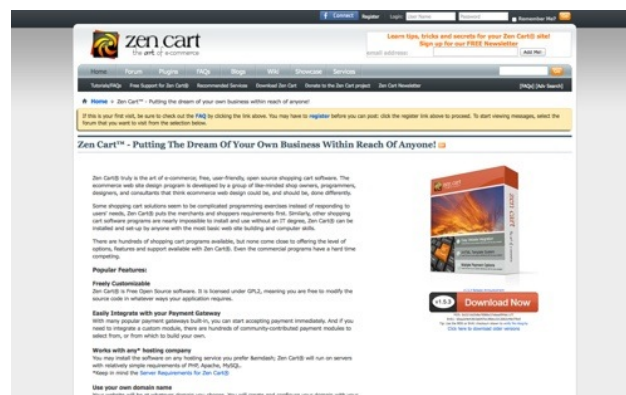


Figura 2.5: *Zen Cart website* [23].

- **Spree Commerce** es una solución *e-Commerce Open Source* basado en *Ruby on Rails*. La plataforma modular permite configurar, complementar, o reemplazar cualquier funcionalidad que necesites. *Spree Commerce* tiene más de 45000 tiendas; la plataforma alrededor del mundo, incluyendo *Chipotle* [2]. *Spree Commerce* ha sido traducido a más de 30 idiomas.



Figura 2.6: *Spree Commerce website* [17].

- **Drupal Commerce** es una aplicación *e-Commerce* desarrollado por *Commerce Guys*. Construido sobre *Drupal Content Manager System*, *Drupal Commerce* ofrece un sistema de administración de productos completo, carro de compra en varios lenguajes y monedas, y forma de pago. La lista de extensión de *Drupal Commerce* es una integración completamente en *third-party* para formas de pago, servicios de cumplimiento, aplicaciones de contabilidad, redes sociales y mucho más. Paquetes de soporte técnico están disponibles por *Commerce Guys*.

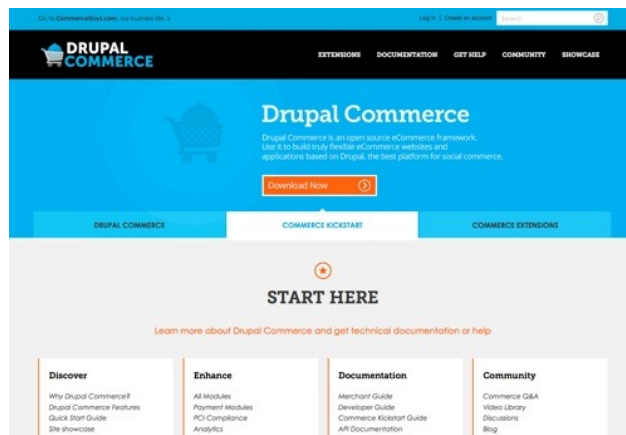


Figura 2.7: *Drupal Commerce website* [5].

- **osCommerce** (i.e., *Open Source Commerce*) es una de las primeras aplicaciones *e-Commerce Open Source*. Más de 7000 *free add-ons* han sido subidos por su comunidad para customizar una *store online*. *osCommerce* es usado por cerca de 13000 sitios registrados. La comunidad de apoyo tiene aproximadamente 280000 miembros, los cuales han contribuido con 1.5 millones de *posts* en los foros. La comunidad directa junto con miembros de otra comunidad están disponibles en vivo en el *Chat Room*.

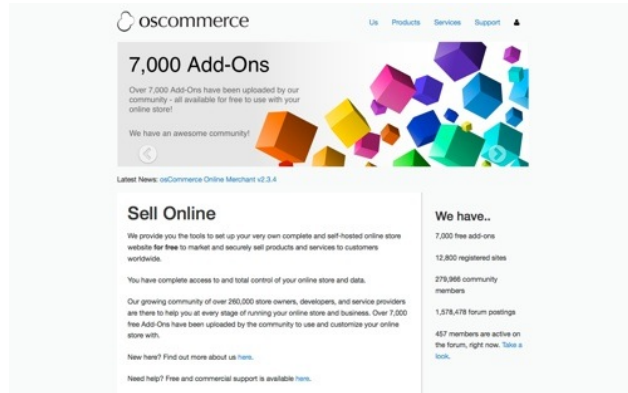


Figura 2.8: osCommerce website [13].

- **simpleCart** es un *free y Open Source JavaScript Shopping Cart*. Con su pequeño tamaño, *simpleCart* está diseñado para mantener simple y sitios con alto tráfico funcionando rápidamente. *simpleCart* tiene la habilidad de pagar con *PayPal Express*, *Google Checkout*, y *Amazon Payments*. *email checkout* e integración con *Authorize.Net* llegarán pronto.

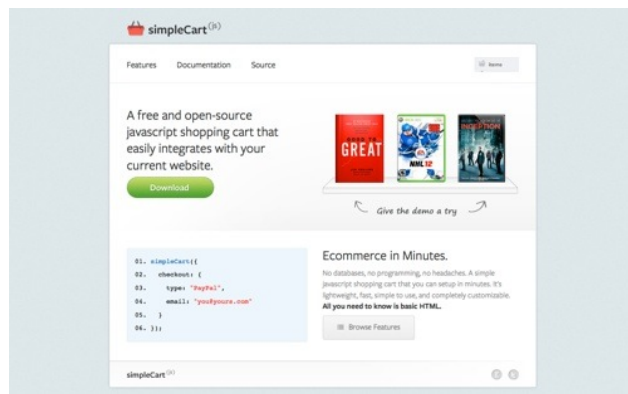


Figura 2.9: simpleCart website [16].

- **WooCommerce** es una aplicación *e-Commerce free Open Source* que permite a los comerciantes transformar *WordPress sites* en *stores*. *WooCommerce* fue desarrollado por *WooThemes* desde un *fork* de *Jigoshop*. *WooCommerce* tiene una larga variedad de *plugins* y *Themes* de *WooThemes* como de sitios *third-party*, tales como *ThemeForest* [18] y *CodeCanyon* [3]. Con cerca de 4.5 millones de descargas desde *WordPress.org* [20], *WooCommerce* es una solución *e-Commerce* muy popular para *WordPress*. Para obtener el soporte oficial de *WooThemes*, es necesario comprar un producto. De otra manera, obtener ayuda desde la comunidad activa del foro.

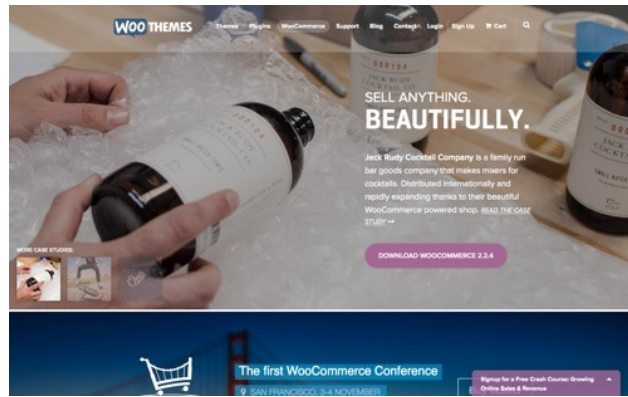


Figura 2.10: *WooCommerce website* [19].

- **WP e-Commerce** es otra aplicación popular obtenida desde la conversión del sitio *WordPress* a una *e-Commerce store*. *WP e-Commerce* tiene cerca de 3 millones de descargas de *plugin* desde *WordPress.org* [20]. Usa el propio *HTML* y *CSS* y obtén el control completo sobre la vista y la experiencia de tu *online store*. *WP e-Commerce* tiene una gran variedad de características estándar, incluyendo *Multi-Tier Pricing* para descuentos por cantidad e integración con redes sociales para *marketing*. Para soporte, hay tutoriales en vídeo y un foro en *WordPress.org*, también como consultantes de características para ayuda profesional.



Figura 2.11: *WP e-Commerce website* [22].

- **Jigoshop** es una solución *e-Commerce free* y *Open Source* basado en *WordPress*. Liberado en 2011, *Jigoshop* es el predecesor para *WooCommerce*. *Jigoshop* tiene más de 30 *Themes*, 100 extensiones y tres *Theme Frameworks*. *Jigoshop* es *free*, así como el soporte a *WordPress.org*. Sin embargo, el acceso a la comunidad de *Jigoshop* comienza desde \$40 por mes.



Figura 2.12: *Jigoshop website* [7].

## 2.2. Tecnologías

A continuación se describen un conjunto de tecnologías que son agrupadas lógicamente de acuerdo a su función en el desarrollo de aplicaciones *web*.

Estas tecnologías tienen en común que son herramientas que se ocupan en la actualidad además de tener la característica de ser *Open Source*.

### 2.2.1. Base de datos

- **MongoDB** (de "humongous") es una *Base de datos Document Oriented Open Source*, escrita en *C++* [9]. MongoDB evita las estructuras de *Base de datos* tradicionales *table-based* en favor de *documents Json-Like* con *Schemas* dinámicos (MongoDB llama al formato *BSON*), haciendo la integración de *Data* en cierto tipo de aplicaciones más fácil y rápido. MongoDB es la *NoSQL* líder, y ha alcanzado una popularidad que supera a *PostgreSQL* [73].
- **MySql** es una *RDBMS Open Source* que se posiciona como la segunda más utilizada a nivel mundial [73] [120]. *MySql* es una opción popular de *Base de datos* para aplicaciones *web*, ampliamente utilizado en el *Stack LAMP* (y otros *Stacks*, *AMP*, etc.).
- **PostgreSQL** es una *ORDBMS* con énfasis en *Extensibility* y normas de cumplimiento. Como *servidor de Base de datos*, su primera función es *store Data*, de forma segura, y apoyar las mejores prácticas, y recuperarlo más tarde, a petición de otras aplicaciones de *software*, ya sea que estén corriendo en el mismo o en otro computador a través de una *network* (incluido *Internet*). Puede manejar las cargas de trabajo que van desde pequeñas aplicaciones de una sola máquina a grandes aplicaciones orientadas a *Internet* con muchos usuarios concurrentes.
- **SQLite** es una *RDBMS* contenida en una librería *C programming*. A diferencia de otras *Data Base Management System*, *SQLite* no está implementado como un proceso independiente del cliente. En contraste, es parte del programa en uso [100].

- **Cassandra** es un NoSQL *Wide-Column store Open Source* diseñado para manejar gran cantidad de *Data* sobre muchos *commodity server*, proporcionando *High Availability* sin *Single Point of Failure*. *Cassandra* ofrece soporte robusto para *clusters* que abarcan múltiples *Data Centers* [66], con *masterless Asynchronous Replication*, permitiendo *Low Latency Operations* para todos los clientes. *Cassandra* es la *Wide-Column store* más popular, y el segundo NoSQL más utilizado [73].
- **Redis** es un NoSQL *Open Source, Key-Value cache y store*. Usualmente referido como *servidor* con *Data* estructurada dado que *keys* pueden contener *strings, hashes, lists, sets, sorted, bitmaps* y *hyperloglogs* [40].
- **Solr™** es *highly Reliable, Scalable y Fault tolerant*, proporcionando *indexing* distribuido, *replication y querying Load-Balanced, Failover y Recovery* automático, configuración centralizada y más [26]. Sus principales características corresponden a: soporte para búsqueda de expresiones complejas, búsqueda de texto completa; *Stemming; ranking* y agrupar los resultados de búsqueda; búsquedas *Geospatial*; y búsqueda distribuida para *High Scalability* [72].

## 2.2.2. Server-side

### Webserver software

- **Node.js** es una plataforma construida sobre *Chrome's JavaScript runtime* para construir fácilmente aplicaciones *network* rápidas y escalables. *Node.js* utiliza un modelo *event-driven, non-blocking I/O* que hace esto liviano y eficiente, muy adecuado para aplicaciones *data-intensive tiempo-real* que corren sobre dispositivos distribuidos [10].
- **Apache™ HTTP servidor**, coloquialmente llamado *Apache™*, es el *software* del *servidor web* más utilizado en el mundo [80].
- **Apache™ Tomcat** es un *Webserver* y contenedor de *Servlet Open Source*. *Tomcat* implementa muchas de las especificaciones de *Java EE*, incluyendo *Java Servlet, JavaServer Pages, Java EL*, y *Websockets*, y proveer un ambiente *Webserver HTTP* para código *Java run in* [81].

### Server-side scripting

- **PHP** es un lenguaje *scripting Server-side* diseñado para desarrollo *web*, pero también utilizado como un lenguaje de programación de uso general [96]. *PHP* es soportado por varios *Webserver*, incluyendo *Apache™ HTTP servidor*.
- **JavaScript** es un lenguaje *Lightweight*, interpretado, *object-oriented* con *funciones first-class*, mayoritariamente conocido como *scripting* para páginas *web*, pero utilizado en muchos ambientes *non-browser* también, tales como *Node.js* y *Apache™ couchDB*. Es un *scripting prototype-based*,

*multi-paradigm* dinámico, y soporta estilos de programación *object-oriented*, *imperative*, y *functional* [135].

- **Java** es un lenguaje de programación de propósito general que es *object-oriented*, *concurrent class-based* [135]. *Java* tiene la filosofía de *write once, run anywhere*, lo que significa que si se quiere cambiar de plataforma, no es necesario volver a compilar el código [196].
- **Ruby** es un lenguaje de programación dinámico y *Open Source*, con énfasis en *Simplicity* y *productivity*. Tiene una sintaxis elegante que es natural para leer y fácil para escribir. *Ruby* es un lenguaje de programación *object-oriented*, *imperative*, y *functional* [41].
- **Perl** es un lenguaje de programación con más de 27 años de desarrollo. *Perl 5* corre en más de 100 plataformas desde *Portables* a *Mainframes*, y es adecuado tanto para un rápido *prototyping* y desarrollar proyectos a gran escala [143].
- **Python** es un lenguaje de programación *high-level* de propósito general, permitiendo desarrollar conceptos en unas pocas líneas, lo cual sería imposible en lenguajes como *Java* o *C++*. *Python* es un lenguaje de programación *multi-paradigm* dinámico, soportando paradigmas de programación *object-oriented*, *imperative*, *functional*, *procedural* y *reflective* [38].

## Frameworks

- **Laravel** es un *Framework* de aplicaciones *web PHP free* y *Open Source*, designado para el desarrollo de aplicaciones *web* utilizando el modelo de arquitectura MVC. De acuerdo a una encuesta realizada a desarrolladores de *PHP Laravel* fue elegido como el *Framework PHP* más popular del 2013 [163]. En la actualidad ( Febrero 2015 ), *Laravel* es el proyecto *PHP* en *GitHub* más popular [177].
- **Phalcon** es un *Framework web* para *PHP high-performance* basado en el patrón arquitectónico MVC. Es un *Framework Open Source* implementado como una extensión de *C* para optimizar *Performance* y bajo consumo de *resources* [182]. *Phalcon* ha sido considerado como el segundo *Framework PHP* más popular [177].
- **Symfony** es un conjunto de componentes *PHP*, un *Framework* de aplicaciones *web*, y una comunidad, todo trabajando en armonía [186]. *Symfony* es un *software free* para desarrollar aplicaciones con el patrón arquitectónico MVC. Es el tercer *Framework* más popular de *PHP* [177].
- **Drupal** es una plataforma *Open Source* manejadora de contenido escrita en *PHP*, alimentando a millones de aplicaciones *websites*. Es construido, usado, y apoyado por una comunidad diversa de personas alrededor del mundo [4].
- **Zend Framework Framework 2** es un *Open Source Framework* para desarrollo de aplicaciones *web* y servicios utilizando *PHP 5.3+*. *Zend Framework Framework 2* usa código 100%

*object-oriented* y utiliza muchas de las nuevas características de *PHP 5.3*, namely *namespaces*, *late static binding*, *lambda functions* y *closures* [24].

- **Yii** es un *Framework* para el desarrollo de aplicaciones *web free* y *Open Source*, escrito en *PHP 5* que fomenta limpios, *DRY design*, y apoya el desarrollo rápido. Esto funciona para agilizar el desarrollo de la aplicación y ayuda a asegurar un producto final extremadamente *Efficient*, *Extensible* y *Maintainable*.
- **WordPress** es un *software web* basado en *PHP* y *MySQL*, con el cual se puede crear hermosos *website* o *Blog*. *WordPress* es *free* e invaluable al mismo tiempo. El *software core* es construido por cientos de voluntarios, y hay miles de *plugins* y *Themes* disponibles para transformar un *website* en casi todo lo que imaginas [21].
- **Ruby on Rails** es un *Framework* para el desarrollo de aplicaciones *web* escrito en lenguaje *Ruby*. Está diseñado para programar fácilmente aplicaciones *web* haciendo supuestos que necesita cada desarrollador para iniciar. Esto permite escribir menos código que muchos otros lenguajes y *Frameworks* [97].
- **Sinatra** es un *domain-specific Language* para construir *websites*, *web service* y aplicaciones *web* en *Ruby*. Enfatiza un enfoque para desarrollo *Minimalistic*, ofreciendo solo lo esencial para manejar *HTTP request* y entregar *responses* a los clientes. *Sinatra* no fuerza a seguir el patrón *model-view-controller*, u otro patrón que importe. Es un *wrapper Lightweight* alrededor de *Rack Middleware* y fomenta una relación cercana entre *Service endpoints* y los *HTTP verbs*, transformando esto particularmente ideal para *web services* y *APIs* [49].  
*Sinatra* no es un *Framework*; no hay herramientas *built-in* ORM, no hay archivos de configuración *pre-fab*; incluso no hay una carpeta de proyecto a menos que sea creada. Ciertamente esto puede parecer como algo no deseable, pero sí es algo completamente liberador. Las aplicaciones en *Sinatra* son muy flexibles por naturaleza, típicamente no son más grandes de lo que necesitan para ser y pueden ser fácilmente distribuidas como *Gems* [49].
- **Padrino** es un *Framework* construido sobre la librería *web Sinatra*. *Padrino* fue creado para realizar código de forma sencilla y entretenida para aplicaciones *web* avanzadas mientras mantiene el espíritu que hace a *Sinatra* excelente [181].
- **Django** es una *Framework Open Source* escrito en *Python*, diseñado para perder *Coupling* y obligar la separación entre diferentes partes de la aplicación [45].
- **Express.js** es un *Framework* de aplicaciones *web Node.js* minimalistas y flexibles, proporcionando un conjunto robusto de características para crear aplicaciones *web single*, *multi-page*, e híbridas [175].
- **Koa** es un *Framework web* diseñado por el equipo detrás de *Express.js*, cuyo objetivo es ser la base más pequeña, más *Expressive*, y más *Robust* para las aplicaciones *web* y *APIs*. El generador de *Koa* permite deshacerse de *callbacks* e incrementar considerablemente el *error-handling*. *Koa* no incorpora ningún *Middleware* en su *core*, y provee una elegante *Suite* de métodos que hacen la escritura en el *servidor Fast* y *Enjoyable* [180].



### 2.2.3. Client-side

- **Angular.js** es un *Framework Client-side* desarrollado por *Google*. Está escrito en *JavaScript*, con una librería reducida de *jQuery*. La teoría detrás de *Angular.js* es proveer de un *Framework* que hace sencilla la implementación de aplicaciones y *web page well-design* y *Structured*, usando un *Framework MVC* [71].

*Angular.js* es utilizado para abordar muchos de los problemas encontrados en el desarrollo de *Single Page Application* [89].

- **Ember.js** [109] es un *Framework Open Source Client-side JavaScript* para aplicaciones *web* basado en el patrón de arquitectura *MVC*. Esto permite a los desarrolladores crear *Single Page Application Scalables* [46], con la incorporación de expresiones idiomáticas comunes y las mejores prácticas dentro de un *Framework* que provee un enriquecido *object model*, *Data Binding Declarative two-way*, *Computed Properties*, *templates automatically-updating powered por Handlebars.js* y un *router* para manejar los estados de la aplicación [54].
- **Backbone.js** da estructura a aplicaciones *web* ofreciendo **Models** con asociación *key/value* y *events custom*, **Collections** con una *API* enriquecida de funciones, **Views handling event** declarativos, y conecta todo a su *API* existente sobre una interfaz *JSON RESTful* [52].
- **PhoneGap** es un *Framework free y Open Source* que permite crear *apps móvil* usando *web APIs* para las plataformas que interesan, en lugar de confiar en una *APIs platform-specific* como aquellas de *iOS*, *Windows Phone* o *Android* [78].
- **jQuery** es una librería *JavaScript Fast, Small y feature-rich*. Esto hace que cosas como la manipulación y recorrido del *HTML Document*, *event handling*, animación y *Ajax* mucho más simple con una *APIs* fácil de utilizar que funciona a través de multitud de *browsers*. Con una combinación de *Versatility* y *Extensibility*, *jQuery* ha cambiado la manera que millones de personas escriben *JavaScript* [179].
- **bootstrap** es el *Framework* más popular de *HTML*, *CSS*, y *JavaScript* para el desarrollo del primer proyecto *móvil* crítico en la *web* [1].

### 2.2.4. Full-Stack JavaScript Frameworks

#### Stack de JavaScript Frameworks

- **MEAN** (*MongoDB*, *Express.js*, *Angular.js*, *Node.js*) es un *Full-Stack JavaScript Framework* que simplifica y acelera el desarrollo de aplicaciones *web* [131]. Existen varias variantes al *Stack MEAN*, tales como *MEEN*. En ocasiones el término es utilizado simplemente para indicar infraestructura corriendo en *Node.js* en combinación con una *Base de datos NoSQL*.
- **MEEN** [164](*MongoDB*, *Ember.js*, *Express.js*, *Node.js*) es un *Full-Stack JavaScript Framework* equivalente a *MEAN*.

- **SocketStream** es un *Full-Stack JavaScript* para construir Single Page Application. Aunque aún se encuentra bajo desarrollo, ofrece algunas extremadamente útiles *características: Websockets, rendering Client-side, soporte HTTP y authentication built-in* [185].
- **SANE** es un *Full-Stack JavaScript* que permite rápidamente crear aplicaciones *web production-ready* utilizando *Sails* y *Ember.js*. Tiene soporte *Docker, generators* y más [118].
- **COKE** es un *Node.js MVC Framework Lightweight* que *speed up* el desarrollo *web*. Es *modularized* [172].
- **Sleekjs** es un *Framework MVC* implementado de *Node.js, built-in* con dependencias base *Handlebars.js, Express.js, mongoose*. La arquitectura de *Sleekjs* sigue el formato común de MVC, lo que hace sencillo manejar y construir nuevos *websites* [124].
- **Danf** es un *Full-Stack JavaScript/Node.js OOP Framework*, permitiendo escribir código de la misma manera tanto en el *Server-side* y *Client-side*. Provee muchas características produciendo aplicaciones *Scalable, Maintainable, Testable* y *Performance* [87].

### **Isomorphic JavaScript Frameworks**

- **Meteor** es una *Framework Open Source tiempo-real* para aplicaciones *web JavaScript* escritas sobre *Node.js* [75]. *Meteor* permite *prototyping* muy veloz [133] y produce código *cross-platform* (*web, Android, iOS*) [132].
- **Derby** [173] es un *Framework Full-Stack, Isomorphic*, con patrón arquitectónico MVC que hace sencillo escribir aplicaciones *Collaborative, tiempo-real* que *Run* tanto en *Node.js* como en *browsers* [198].
- **Mojito** es un *Framework* para aplicaciones MVC construido sobre *YUI 3* que permite el desarrollo ágil de aplicaciones *web*. *Mojito* permite a los desarrolladores usar una combinación de configuraciones y una arquitectura MVC para crear aplicaciones. *Mojito* fue diseñado para correr tanto en el *Server-side (Node.js)* como en el cliente (*browser*) [188].
- **Rendr** es una pequeña librería de *Airbnb* [171] que permite *Run* aplicaciones *Backbone.js* sin problemas tanto en *Client-side* como en *Server-side*. Permite al *Webserver* servir páginas *HTML fully-formed* a cualquier profundidad de *Link* de la aplicación, preservando al mismo tiempo la sensación ágil de una aplicación MVC *Client-side Backbone.js* tradicional [198].
- **Flatiron** es un *Framework* adaptable para construir aplicaciones *web* modernas. Esto fue construido desde cero para ser utilizado con *JavaScript* y *Node.js* [176].

	Tipo Framework
<i>Derby</i>	<i>Isomorphic JavaScript Frameworks.</i>
<i>SocketStream</i>	<i>SocketStream, Ember.js, Node.js.</i>
<i>MEAN</i>	<i>MongoDB, Express.js, Angular.js, Node.js.</i>
<i>MEEN</i>	<i>MongoDB, Ember.js, Express.js, Node.js.</i>
<i>Meteor</i>	<i>Isomorphic JavaScript Frameworks.</i>
<i>Mojito</i>	<i>Isomorphic JavaScript Frameworks.</i>
<i>SANE</i>	<i>Sails, Ember.js, Node.js.</i>
<i>Sleekjs</i>	<i>Handlebars.js, Express.js, mongoose, Node.js.</i>

Tabla 2.1: Resumen *Full-Stack JavaScript Frameworks*.

### 2.2.5. Herramientas para desarrollo

- **Grunt** es una herramienta *command-line task-based* construida para proyectos *JavaScript*, que corren sobre *Node.js* y es instalado vía *NPM* [6].
- **Browserify** permite *require módulos* en el *browsers bundling up* todas las dependencias [30]. *browsers* no necesitan definir el método *request*, pero *Node.js* sí. Con *Browserify* es posible escribir código que use *require* de la misma manera que se usaría en *Node.js* [30].
- **Docker** es una plataforma *Open Source* y *sysadmins* para construir, enviar y ejecutar aplicaciones distribuidas. Consiste en el motor *Docker*, un portable, *Lightweight runtime* y *Packaging tool*, y *Docker hub*, un servicio de la nube para compartir aplicaciones y *Workflows* automatizados, *Docker* permite que las aplicaciones sean rápidamente ensambladas desde componentes y elimina la fricción entre *Desarrollo*, *QA* y *Production environments*. Como resultado, puede ser entregado rápidamente y correr la misma aplicación, sin cambios, en *laptops*, *Data Center VMs*, y cualquier otro recurso provisto por *Internet* [76].
- **Bower** <http://bower.io/>
- **Bunyan** `install bunyan npm install bunyan`

## 2.3. Aplicaciones web

- **Single Page Application** se distingue por su habilidad de *redraw* en cualquier parte de la *UI* sin requerir intervención del *servidor*. Esto se logra separando la *Data* de la presentación de *Data* teniendo un *Model Layer* que maneje la *Data* y una *Views Layer* que lea desde los *models*.
- **Aplicaciones Isomorphic JavaScript** son aplicaciones escritas en *JavaScript* que pueden correr tanto en el cliente como en el *servidor*. Por consecuencia, es posible escribir código y ejecutarlo en el *servidor* para desplegar páginas estáticas y en el cliente para permitir interacciones rápidas.

## Capítulo 3

# Justificación del proyecto

A continuación se discutirán las razones para considerar el desarrollo de una plataforma *e-Commerce*, como un aporte en relación con las opciones *Open Source* actualmente vigentes. Para esto se explicará por qué los cambios de tecnología propuestos son un beneficio con respecto al actual estado del arte del tema, así como un aporte para potenciales funcionalidades que serán agregadas en el futuro.

### 3.1. *Base de datos*

Bases de datos relacionales se encuentran en la mayoría de las organizaciones desde hace muchísimo tiempo, y por buenas razones. Las bases de datos relacionales soportan las aplicaciones del pasado que cumplen con las necesidades de negocio actuales. Estas son apoyadas por un extenso ecosistema de herramientas; y hay una gran cantidad de mano de obra calificada para implementar y mantener estos sistemas.

Pero las compañías están cada vez más considerando la opción de alternativas para la infraestructura relacional heredada. En algunos casos la motivación es técnica, tal como la necesidad de escalar o actuar más allá de las capacidades de sus sistemas actuales. Mientras que en otros casos las compañías están motivadas por el deseo de identificar alternativas viables a *softwares* propietarios de alto costo. Una tercera motivación es la agilidad y velocidad del desarrollo, dado que las compañías ansían adaptarse más rápido al mercado y adoptar metodologías de desarrollo ágil.

Estas opciones se aplican tanto para aplicaciones *analytics* como *procesamiento transaccional*. Las compañías están cambiando *workloads* a *Hadoop*® *B* para sus *offline, analytical workloads*, y están construyendo *online*, aplicaciones operacionales con una nueva clase de tecnología de manejo de datos llamada NoSQL, como por ejemplo MongoDB.

### 3.1.1. SQL y el Modelo Relacional

SQL es un lenguaje declarativo de consulta de datos. Un lenguaje declarativo es uno en el cual un programador especifica lo que desea y el sistema lo ejecuta, en lugar de definir proceduralmente cómo el sistema debería hacerlo. Unos pocos ejemplos incluyen: encontrar el registro del *estudiante* 39, mostrar solo el *nombre* y el *número de teléfono* del estudiante de la totalidad de su registro, filtrar los registros de los *estudiantes* a aquellos que cursan contabilidad, contar la cantidad de *estudiantes* en cada curso, unir la información de la tabla de los *estudiantes* con la tabla de *profesores*.

En una primera aproximación, SQL permite consultar sobre aquellas preguntas sin pensar sobre cómo la información es expuesta en el disco, cuáles índices utilizar para acceder a la información o qué algoritmo utilizar para procesar la información. Un componente arquitectural significativo para la mayoría de las bases de datos relacionales es un optimizador de consultas, que decide cuál de los muchos equivalentes lógicos planea ejecutar para obtener la respuesta más rápida a una consulta. Estos optimizadores son usualmente mejores que los promedios de los usuarios de la base de datos, pero en algunas ocasiones estos no tienen la suficiente información o tienen un modelo muy simple del sistema con el fin de generar la ejecución más eficiente.

Bases de datos relacionales, las bases de datos más utilizadas en la práctica, siguen el modelo de datos relacional. En este modelo, diversas entidades del *mundo real* son guardadas en diferentes tablas. Por ejemplo, todos los *estudiantes* podrían ser guardados en una tabla *Estudiante*, y todos los  *cursos* podrían ser almacenados en la tabla *Cursos*. Cada fila de una tabla tiene varias propiedades guardadas en columnas. Por ejemplo, *Estudiante* podría tener un *estudiante id*, *dirección*, *edad*, *fecha nacimiento*, y *nombres/apellidos*. Cada una de estas propiedades sera guardada en una columna de la tabla *Estudiante*.

El modelo relacional va mano a mano con SQL. Consultas simples de SQL, tales como filtrar, recuperar todos los registros donde sus campos hagan *match* en algunos tests (ejemplo, *estudiante id* = 3, o *edad* > 20 años). Construcciones más complejos causan que la base de datos haga trabajo extra, tal como unir información sobre múltiples tablas (ejemplo, ¿cuáles son los nombres de los cursos que actualmente tiene inscritos el *estudiante id* 3 ?). Otras estructuras complejas tal como *agregación* (ejemplo, ¿cuál es la cantidad promedio de ramos que tienen inscritos los estudiantes?) pueden conducir a realizar un *scan* completo de la tabla.

Un modelo de datos relacional define entidades altamente estructuradas con relaciones estrictas entre ellos. Consultar este modelo con SQL permite recorridos de datos complejos sin demasiado desarrollo personalizado. La complejidad del modelo y las consultas tienen sus límites, aunque:

*complexity* guía a *unpredictability*. La expresividad en SQL implica un desafío en relación con el costo de cada consulta, y así el costo de *workload*. Mientras lenguajes de consultas simples pueden complicar la lógica, al mismo tiempo hacen que sea sencillo proveer de almacenamiento de datos, cuando solo responde a consultas simples.

Hay muchas maneras de modelar un problema. El modelo de datos relacional es estricto: El

esquema asignado para cada tabla especifica los datos en cada fila. Si se están almacenando menos datos estructurados, o filas con más diferencia en las columnas que se guardan, el modelo relacional puede ser innecesariamente restrictivo. Similarmente, aplicaciones desarrolladas podrían no encontrar el modelo relacional idóneo para modelar cada tipo de dato. Por ejemplo, una gran cantidad de aplicaciones lógicas son escritas en lenguaje *object-oriented* e incluye conceptos *high-level*, tales como *lists*, *queues*, y *sets*, y algunos programadores desearán *persistence layer* para modelar esto.

Si los datos crecen más allá de la capacidad de un servidor, entonces las tablas en la base de datos tendrán que ser particionadas a través de varios computadores. Para evitar *joins* a través de la red para obtener todas las tablas requeridas, será necesario desnormalizarla. Desnormalizar consiste en guardar todos los datos de diferentes tablas que se desean consultar en el mismo lugar. Esto hace que la base de datos simule una *Key Lookup store system*, dejando abiertas preguntas sobre la posibilidad de adaptarse mejor a los datos.

Generalmente no es inteligente descartar años de consideraciones de diseño arbitrariamente. Cuando se considera el almacenamiento de los datos en una base de datos, considera SQL y el modelo relacional, los cuales son respaldados por décadas de investigación y desarrollo, ofreciendo enriquecidas capacidades de modelamiento, y proporcionan garantías *fácil-de-entender* sobre operaciones complejas. NoSQL es una buena opción cuando se tiene un problema específico, tal como gran cantidad de datos, un masivo *workload*, o una difícil decisión de modelamiento para la cual SQL y bases de datos relacionales podrían no haber sido optimizadas.

### 3.1.2. NoSQL

NoSQL engloba una gran variedad de diferentes tecnologías de bases de datos que fueron desarrolladas en respuesta al creciente volumen de datos guardados de los usuarios, objetos y productos, la frecuencia en que los datos son accedados, y la *Performance* en las necesidades de los procesos. Bases de datos relacionales, por otra parte, no fueron diseñadas para hacer frente a los desafíos de escalabilidad y de agilidad que enfrentan las aplicaciones modernas, no fueron construidas para tomar ventaja del almacenamiento barato y al poder de procesamiento disponible en la actualidad.

#### ***Document Model***

Mientras las bases de datos relacionales guardan la información en filas y columnas, *Base de datos* orientada al documento se guardan los datos en documentos.

Estos documentos típicamente usan una estructura equivalente a JSON, un formato muy popular entre desarrolladores. Los documentos proveen una manera intuitiva y natural para modelar datos que está cercanamente alineada con la programación *object-oriented*, en la cual cada documento es efectivamente un objeto. Los documentos contienen en uno o mas campos, donde cada campo contiene un tipo de valor, tales como *string*, *date*, *binary*, ó *array*. En lugar de extenderse un registro entre múltiples tablas y columnas, cada registro y su dato asociado son típicamente almacenados juntos en

un solo documento. Esto simplifica el acceso a la información y reduce e incluso elimina la necesidad de *joins* y *Transactions* complejas.

En una *Base de datos* orientada al documento, la noción de esquema es dinámica: cada documento puede contener diferentes campos. Esta flexibilidad puede ser particularmente útil para modelar datos sin estructura y *Polymorphic*. Esto también hace posible la evolución de una aplicación durante su desarrollo, simplemente agregando nuevos campos. Adicionalmente, *Base de datos* orientada al documento generalmente proveen consultas robustas que los desarrolladores esperan en las bases de datos relacionales.

Aplicaciones : *Base de datos* orientada al documento son de propósito general y útiles para una amplia variedad de aplicaciones, debido a su flexibilidad del modelo de datos, la habilidad para consultar sobre cualquier campo y el *mapping* natural del *Base de datos* orientada al documento a objetos en programación de lenguajes modernos.

Ejemplos: MongoDB y CouchDB.

### **Modelo graph**

Basado en *Graph Theory* [193] , usa estructuras de grafos con nodos, bordes y propiedades para representar los datos. En esencia, los datos son modelados como una red de relaciones entre elementos específicos. Si bien es cierto Modelo *graph* puede ser contra intuitivo y tomar tiempo para entenderlo, puede ser utilizado extensamente para numerosas aplicaciones. Su principal característica es que modela fácilmente las relaciones entre entidades en una aplicación.

Aplicaciones: *Base de datos graph* son útiles en escenarios donde las relaciones son el *core* de la aplicación, como redes sociales.

Ejemplos: Neo4j y HyperGraph.

### **key/value**

Desde una perspectiva de modelo de datos, *key/value stores* son el tipo más básico de las NoSQL *Base de datos*. Cada producto en la base de datos es guardado como un atributo *name*, o *key*, junto con su *value*. El *value*, sin embargo, es totalmente opaco al sistema; los datos solo pueden ser requeridos desde la *key*. Este modelo puede ser útil para representar datos *Polymorphic* y sin estructura, dado que la *Base de datos* no define un esquema al conjunto *key/value*.

### **Modelos Wide-Column**

*Wide-Column stores*, o *column family stores*, usan un mapa ordenado *multi-dimensional* distribuido y *sparse* para guardar datos. Cada registro puede variar en el número de *columns*

que son guardadas, y las *columns* pueden ser anidadas dentro de otras *columns* llamadas *super columns*. *columns* pueden ser agrupadas juntas por acceso en *column families*, o *columns* pueden ser propagadas a través de múltiples *column families*. Los datos son recuperados utilizando una *primary key* por cada *column family*.

Aplicación: *key value stores* y *Wide-Column stores* son útiles para un reducido conjunto de aplicaciones que solo consultan datos para un único *key/value*. Lo llamativo de estos sistemas es su *Performance* y *Scalability*, los cuales pueden ser *highly optimized* debido a su simplicidad de los patrones de acceso de datos.

	SQL Base de datos	NoSQL Base de datos
Tipos	Un tipo (SQL Base de datos) con variaciones menores.	Muchos tipos distintos incluyendo <i>key/value</i> , <i>Document Model</i> , Modelos <i>Wide-Column</i> y Modelo <i>graph</i> .
Historia de desarrollo	Desarrollado en 1970s para tratar con la primera ola de aplicaciones de almacenamiento de <i>Data</i> .	Desarrollado en 2000s para tratar con las limitaciones de las bases de datos SQL, en particular en relación a escalar, <i>replication</i> y guardar datos sin estructura.
Ejemplos	<i>MySQL</i> , <i>PostgreSQL</i> , <i>Oracle Base de dato</i>	<i>MongoDB</i> , <i>Cassandra</i> , <i>Hbase</i> , <i>Neo4j</i>
Modelo <i>Data storage</i>	Registros individuales (ej., <i>empleados</i> ) son guardados como filas en la tabla, con cada columna guardando un dato específico sobre el registro (ej., <i>jefe</i> , <i>perfil</i> , etc.), similar a un <i>spreadsheet</i> . Tipos de datos separados son guardados en tablas separadas, y entonces uniéndose cuando consultas más complejas son ejecutadas. Por ejemplo, <i>oficinas</i> podrían estar guardadas en una tabla, y <i>empleados</i> en otra. Cuando un usuario quiere buscar la dirección de trabajo de un <i>empleado</i> , el motor de la <i>Base de datos</i> unir las tablas <i>EMPLEADOS Y OFICINAS</i> juntas para obtener toda la información necesaria.	Varia de acuerdo al tipo de base de datos. Por ejemplo, <i>key/value store</i> funciona de manera similar a la base de datos SQL, pero tiene solo dos <i>columns</i> ( <i>key</i> y <i>value</i> ), con información más compleja en algunas ocasiones guardada dentro de <i>columns value</i> . <i>Document Data Base</i> acabó con el modelo <i>table-and-row</i> completamente, guardando todos los datos relevantes juntos en un solo <i>document</i> en <i>JSON</i> , <i>XML</i> , u otro formato, el cual puede anidar valores jerárquicamente.
<i>Schemas</i>	Estructura y tipo de datos son fijos por adelantado. Para guardar información sobre un nuevo <i>data item</i> , la <i>Base de dato</i> entera debe ser alterada, tiempo durante el cual la base de datos debe ser tomada <i>offline</i> .	Típicamente dinámica. Registros pueden agregar nueva información <i>on the fly</i> , y diferente a las <i>rows</i> de las tablas SQL, datos distintos pueden estar guardados juntos como sea necesario. Para algunas Bases de datos (ej., <i>Wide-Column stores</i> ), es algo más desafiante para agregar nuevos <i>fields</i> dinámicamente.
<i>scaling</i>	<i>Vertically</i> , significa que un solo servidor debe incrementar <i>powerful</i> con el fin de hacer frente al incremento de demanda. Es posible propagar la <i>Base de datos</i> SQL sobre muchos servidores, pero ingeniería adicional significativa será generalmente requerida.	<i>Horizontally</i> , significa que para agregar capacidad, un administrador de <i>Base de datos</i> puede simplemente agregar más servidores o <i>cloud instances</i> .
Modelo Desarrollo	<i>mix de Open Source</i> (ej., <i>PostgreSQL</i> , <i>MySQL</i> ) y <i>Closed Source</i> (ej., <i>Oracle Base de dato</i> ).	<i>Open Source</i> .
Soporta <i>Transactions</i>	Sí, actualizaciones pueden ser configuradas para completar enteramente o no del todo.	En ciertas circunstancias y en ciertos niveles (ej., <i>Document Level</i> vs. <i>Data Base Level</i> ).
Manipulación de datos	Lenguaje especial usando declaraciones <i>Select</i> , <i>insert</i> , y <i>update</i> , ej., <i>SELECT fields FROM table WHERE...</i>	A través de <i>APIs object-oriented</i> .
<i>Consistency</i>	Puede ser configurada para <i>Strong Consistency</i> .	Depende del producto. Algunos proveen <i>Strong Consistency</i> (ej., <i>MongoDB</i> ) mientras otros ofrecen <i>Eventual Consistency</i> (ej., <i>Cassandra</i> ).

Tabla 3.1: Resumen NoSQL vs. SQL.

### 3.1.3. MongoDB y e-Commerce [117]

Demostraciones de almacenamientos de datos de la siguiente generación giran típicamente en torno a las redes sociales: *Twitter*, *Facebook*, *Foursquare*, etc. Desafortunadamente, tales aplicaciones generalmente tienen modelos de datos complejos. *e-Commerce*, por ejemplo, tiene la ventaja de incluir un largo número de patrones familiares para modelar datos. Además no es complicado imaginar cómo productos, *categorías*, *itemReviews* y órdenes son típicamente modelados en RDBMS.

*e-Commerce* ha sido usualmente un dominio exclusivo de RDBMS, y esto es así por un par de razones. La primera es que los sitios *e-Commerce* generalmente requieren *Transactions*, y *Transactions*



son una operación básica en RDBMS. Lo segundo es que, hasta hace poco, dominios que requieren de un modelo de dato enriquecido y consultas sofisticadas han sido presupuestos que se ajustan mejor en RDBMS. **En las siguientes secciones se cuestionará la segunda afirmación.**

### **Manejo de Catálogo**

Si es necesaria información sobre cómo el *Manejo de Catálogo* se maneja con *Base de datos* relacionales, es necesario dar una mirada rápida a los esquemas del popular *Magento e-Commerce Framework* [Figura A.1]. En la Figura 3.1 se observa un conjunto de *tables* trabajando a la par para proveer un esquema flexible sobre un fundamentalmente inflexible estilo de sistema de *Base de datos*.

Esto significa que los datos de cualquier producto se extienden a través de una docena de *tables*. Esto incrementa la complejidad del código requerido para la persistencia de consultas de productos individuales y hacer una consulta *Shell-Based* es casi imposible. Simplemente considere escribir un SQL *join* para reunir el modelo de un producto como el de la Figura 3.1.

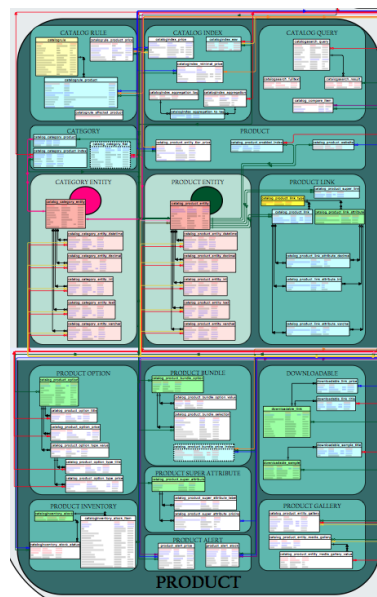


Figura 3.1: Esquemas de un producto.

O realizar una sencilla búsqueda desde la *shell* MongoDB *JavaScript* para obtener un objeto JSON como el Código D.1.

Claramente en Código D.1 no hay representación completa de un producto, pero esto demuestra cuántas de estas *tables* triviales que existen en una representación relacional pueden prescindir en una representación *document*.

Para datos *object-oriented*, **los documents tienen mayor sentido**, tanto en concepto como rendimiento. Una representación *Document Oriented* de un dato de un producto se traduce a unas pocas entidades (un puñado de *Collections* vs. una docena de *tables*), mejor *Performance* en consultas (sin *Server-side joins*), y estructuras que corresponden precisamente al producto. **Ya no existe la**

**necesidad** de diseñar un *Master Schema* que pueda considerar a cada tipo de producto concebible.

*Manejo de Catálogo* es esencialmente *Manejo de Contenido*, un campo en donde MongoDB sobresale.

## Carros de compra y órdenes

Permitir que un carro de compra sea simplemente una orden en un estado de carro, el modelo de carro de compra y las órdenes en MongoDB se tornan muy sencillas. Un ejemplo de esto se puede observar en Código D.2.

Notar que es posible presentar los pedidos como un *array* de productos. Como es usual con *documents*, esto hace el despliegue del carro de compra más sencillo, dado que no hay *joins* envueltos. Pero esto también resuelve el problema del versionamiento de productos. Usualmente es necesario tener el estado de un producto cuando este es comprado. Esto puede ser logrado en una RDBMS estableciendo un vínculo a versiones particulares de un producto. Aquí, sin embargo, simplemente se almacenó el estado de un producto dentro de la misma *orden*.

## Consultando órdenes

Dado que MongoDB soporta consultas dinámicas y *Secondary indexing*, las consultas para las órdenes son automáticas. Es posible, por ejemplo, definir un *index* en un producto *SKU*, lo que permite consultas eficientes en todas las órdenes para un producto dado. En Código D.3 se observa un ejemplo.

Con MongoDB, es posible realizar consultas en atributos arbitrarios, de esa manera cualquier consulta en la *Collection* órdenes es posible. Y para consultas comunes, es posible definir *indexes* para una mejor eficiencia.

## Aggregation

Claramente, *Aggregation* también es necesario. Se desea reportar órdenes de diferentes maneras, y para ese propósito, *map-reduce* está disponible. A modo de ejemplo, Código D.4 tiene comando *map-reduce* que *Aggregates* el total de órdenes por *zip code*.

## Actualizando órdenes

## Incrementando Calidad

Una manera de ajustar la cantidad es usando un *position operator*, el cual permite aplicar *atomic operations*, a un único objeto dentro de un *array*. Código D.5 muestra cómo cambiar el número de álbumes que se están ordenando.

## Agregar y remover productos

Igualmente, *atomic operations* resuelve el problema de agregar y remover productos desde el carro. Código D.6 ejemplifica el uso de *\$push atomic operator* para agregar un producto al carro.

Al ajustar la cantidad y el cambio de los mismos productos en el carro, es necesario actualizar el total de la orden. Notar el uso del operador *\$inc* para manejar esto.

## Inventario

No todos los sitios *e-Commerce* necesitan manejar el inventario. Pero para aquellos que sí lo hacen, MongoDB funciona a la altura de las circunstancias.

Una manera para manejar el inventario, es guardar un *document* separado por cada *physical item* en la bodega. Así, por ejemplo, si la bodega tiene veinte copias del álbum Coltrane, se traduce en veinte *documents* distintos en la *Collection* de *Inventario*. Cada *document* tiene una estructura como lo muestra Código D.7.

Cuando un usuario intenta agregar un producto al carro, un comando *findAndModify* puede ser facilitado para automáticamente marcar el producto en *in-Cart*, asociando el producto con una *orden* dada, y estableciendo un tiempo de espiración (Código D.8).

Si se obtiene un *item back* desde la operación *findAndModify*, se sabe que tenemos un único *lock* en el producto, y es posible guardarlo en el carro. Cuando el usuario desea realizar el *checkout*, el estado del producto puede cambiar a *purchased*, o cualquiera sea el caso de la llamada.

Mientras, se puede ejecutar un *script* en *background* que libere el inventario del carro que no ha sido *purchased* en la ventana de quince minutos. La actualización es trivial y se observa en Código D.9.

## **Transactions, Consistency y Durability**

Muchos argumentos impuestos contra NoSQL en *e-Commerce* se centran en *Transactions*, *Consistency* y *Durability*. En relación con esto se mencionan algunos puntos.

Con relación a *Transactions*, ciertamente MongoDB no soporta el tipo *multi-object*; sin embargo, soporta *atomic operations* sobre *documents* individuales, y esto combinado con el modelo *Document Oriented* recién descrito, y creatividad, es suficiente para muchos problemas *e-Commerce*. Ciertamente, si se necesita *debit one account y credit another* en la misma operación, o si se desea *rollback*, sera necesario *full-fledged Transactions*. No obstante, *transactionality* provista por MongoDB debería ser suficiente en la mayoría de los casos, si no en todos, para operaciones *e-Commerce*.

Si la preocupación está sobre *Consistency* y *Durability*, operaciones escritas

en MongoDB pueden ser realizadas *Consistency* sobre conexiones. Además, MongoDB 1.5 soporta *near-real-time replications*, así que es posible asegurarse que una operación ha sido replicada antes de retornar.

## Scalability

La manera más sencilla para escalar la mayoría de las Bases de Datos, es *upgrading el hardware*. Si la aplicación está corriendo en un único nodo, es usualmente posible agregar una combinación de *disk IOPS*, *memoria* y *CPU* para eliminar los cuellos de botella de la *Base de datos*. La técnica de mejorar el *hardware* de un solo nodo para escalar se conoce como *Vertical Scaling* o *Scaling Up* (Figura 3.2). *Vertical Scaling* tiene la ventaja de ser simple, seguro, y *costo-efectividad* hasta un cierto punto. Si se está ejecutando sobre un *hardware* virtualizado (tal como *Amazon's EC2*), entonces puedes encontrar que una instancia lo suficientemente larga no está disponible. Si estás ejecutando sobre *physical Hardware*, habrá un punto donde el costo de un *servidor* más poderoso se vuelve prohibitivo.

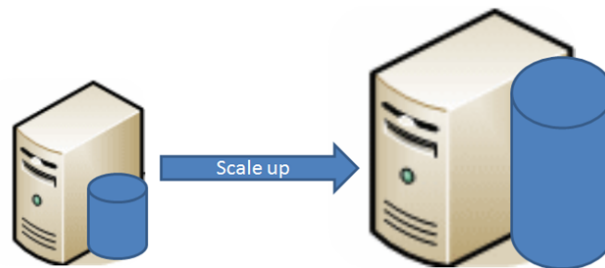


Figura 3.2: *Vertical Scaling* o *Scaling Up*

Entonces tiene sentido considerar *Horizontal Scaling* o *Scaling Out* (Figura 3.3). En lugar de reforzar un único nodo, *Horizontal Scaling* significa distribuir la base de datos sobre múltiples máquinas. Dado que la arquitectura *Horizontally Scaled* puede utilizar *Commodity Hardware*, el costo de *Hosting* el total de los datos puede ser reducido significativamente. Incluso, la distribución de los datos sobre máquinas mitiga las consecuencias de falla. Las máquinas inevitablemente fallarán de algún momento a otro. En el caso de *Vertical Scaling*, si la máquina falla, entonces es necesario tratar con una falla en una máquina de la cual la mayoría del sistema depende. Podría no considerarse un tema si una copia de los datos existe en un *replicated slave*, pero aun está el caso en que solo un único *servidor* es necesario para bajar el sistema completo. En contraste con la falla dentro de una arquitectura *Horizontally Scaled*. Esto podría ser menos catastrófico dado que una sola máquina representa un porcentaje menor del sistema completo.

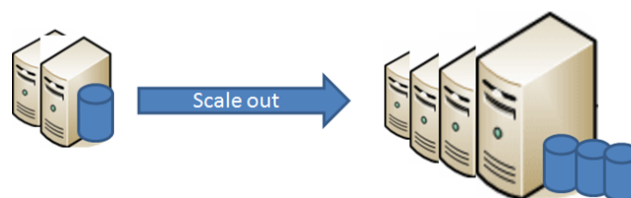


Figura 3.3: *Horizontal Scaling* o *Scaling Out*.

## Conclusión

Es cierto que la mayoría de las bases de datos NoSQL no fueron construidas considerando *e-Commerce*. Las bases de datos que carecen de modelos de datos enriquecidos, consultas dinámicas, y de la noción de *transactionality* no puede esperarse que compitan en el espacio de *e-Commerce*, entonces tampoco es comprensible que no se considere MongoDB.

Pero para las partes en donde el sitio *e-Commerce* comprende el manejo de contenido, MongoDB es claro vencedor. E incluso para más componentes *transactional* del sistema, MongoDB tiene características que hacen de la posibilidad de correr un sistema completo de *e-Commerce* una realidad. Si bien es cierto que esto solo corresponde a un *sketch*, este permite obtener una noción de que **Bases de datos orientadas al documento lo hacen bien.**

## 3.2. Node.js

En una frase: *Node.js* brilla en aplicaciones *web tiempo-real* utilizando tecnología *push* sobre *Websockets*. ¿Qué es tan revolucionario en relación a esto? Después de 20 años de *stateless-web* en el paradigma *request-response* finalmente existen aplicaciones *web tiempo-real*, conexiones *two-way*, donde cliente y *servidor* pueden iniciar la comunicación, permitiendo intercambiar datos libremente. Esto es un contraste enorme al paradigma típico de *web response*, donde el cliente siempre iniciaba la comunicación. Adicionalmente, está todo basado en *open web stack* (*HTML*, *CSS* y *JavaScript*) corriendo sobre el *standar port* 80.

Se puede discutir que estas características existían hace años a través de *Flash* y *Java Applets*, pero en realidad esos fueron *Sandboxed Environment* utilizando la *web* como un protocolo de transporte para entregar al cliente. Además, ellos corrían aislados y generalmente sobre *non-standard ports*, los cuales podrían requerir permisos extras.

Con todas estas ventajas, *Node.js* ahora juega un rol crítico en el *Stack* de tecnología de las principales compañías *High-Profile* [169], **las cuales dependen de sus beneficios únicos.**

La principal idea de *Node.js*: usar *non-blocking*, *event-driven I/O* para permanecer liviano y eficiente para enfrentar aplicaciones *data-intensive tiempo-real* que corren a través de dispositivos distribuidos.

Lo que esto realmente significa es que *Node.js* no es una plataforma *Panacea* que dominará el mundo *web Desarrollo*. **En lugar de eso, es una plataforma que llena una necesidad particular.** Esto es absolutamente esencial. Definitivamente no se deseará utilizar *Node.js* para operaciones *CPU-Intensive*; de hecho, utilizando esto para *heavy computation* anulará prácticamente todas sus ventajas. Donde *Node.js* realmente brilla es construyendo aplicaciones *Scalable network* rápidamente, ya que es capaz de manejar un gran número de conexiones simultáneas con un alto rendimiento, lo que equivale a una *High Scalability*.

Cómo trabaja *under-the-hood* es muy interesante. Comparado con la técnica tradicional *web-serving* donde cada conexión (*request*) engendra un nuevo *thread*, ocupando memoria *RAM* del sistema y eventualmente *maxing-out* la cantidad de *RAM* disponible, *Node.js* opera en un *single-thread*, usando llamadas *non-blocking I/O*, permitiendo el apoyo de decenas de miles de conexiones concurrentes (retenidas en el *event loop*). Figura 3.4.

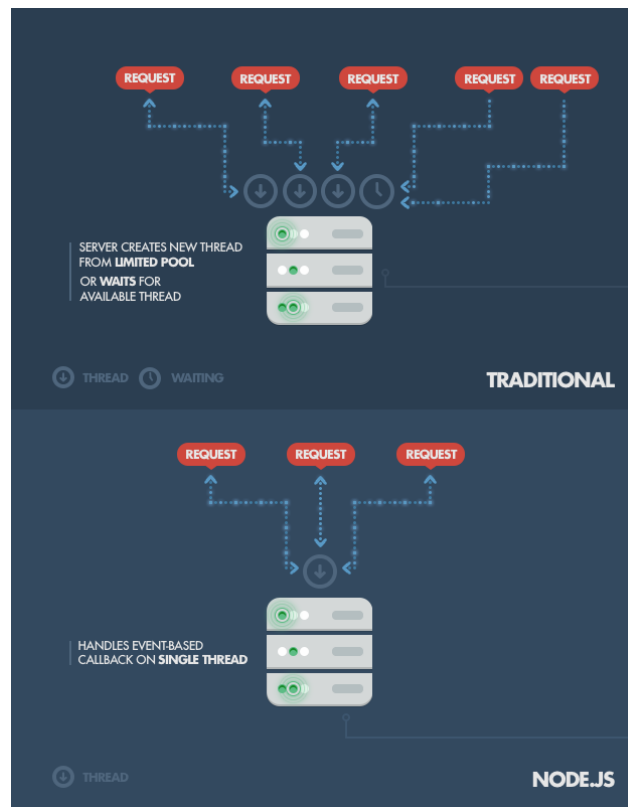


Figura 3.4: Diagrama de Técnica tradicional vs. *Node.js* servidor *thread*.

Un cálculo rápido: asumiendo que cada *thread* potencialmente tiene asignado 2 MB de memoria con ella, corriendo en un sistema con 8 GB de *RAM* nos coloca en un teórico máximo de 4000 conexiones concurrentes, además del costo por cambio de contexto entre *threads*. Ese es el escenario en el cual típicamente se encuentran técnicas tradicionales de *web-serving*. Evitando todo eso, *Node.js* logra niveles de *Scalability* sobre 1M de conexiones concurrentes (como *proof-of-concept*).

Hay, por supuesto, la cuestión de compartir un solo *thread* entre todos los *request* de *clientes*, y es una potencial trampa de escribir aplicaciones *Node.js*. Primeramente, *heavy computation* puede asfixiar el único *thread* de *Node.js* y causar problemas en todos los clientes como *incomming request* podrían ser bloqueados hasta que dicho *computation* sea completado. En segundo lugar, desarrolladores necesitan ser realmente cuidadosos en no permitir una excepción *bubbling up* al *core* del *event loop* de *Node.js*, el cual puede causar que la instancia *Node.js* termine (efectivamente *crashing* el programa).

La técnica utilizada para evitar excepciones *bubbling up to the surface* es pasando el error de regreso al *caller* como parámetro de *callback* (en lugar de lanzarlos, como en otros ambientes). Incluso si alguna excepción no controlada conduce a *bubbling up*, hay múltiples paradigmas y herramientas

disponibles para monitorear el proceso *Node.js* y realizar lo necesario para recuperarse de una *crashed instance* (aunque no esté disponible para recuperar sesiones de usuarios), siendo el más común el módulo *Forever* [145].

### 3.2.1. Desempeño de Node.js frente a sus competidores

A continuación se realizan comparaciones en el desempeño de la plataforma con algunos de sus similares más conocidos, los cuales corresponden a:

- Java
- Ruby
- PHP

#### ***Node.js vs. Java* [99]**

La primera adopción que hizo *PayPal* con *Node.js* no fue una aplicación menor; esto fue su *account overview page* y una de las más *trafficked apps* en el *website*. Ellos tomaron un gran riesgo, pero lo hicieron con la finalidad de determinar si realmente la nueva plataforma representaba una mejora con respecto al sistema actual.

Una vez que la aplicación con *Node.js* logró las mismas funcionalidades que la aplicación actual, percibieron el siguiente resultado:

- Fue construida casi **el doble de rápido con menos personal**.
- Escrito con un **33 % menos de líneas de código**.
- Construida con **40 % menos de archivos**.

Por sí solo, esta evidencia mostraba a los equipos desarrollar más rápido utilizando *JavaScript*.

*Performance* fue un tópico interesante. Se pudo analizar 2 aplicaciones con exactamente las mismas funcionalidades: una con *Framework Java* interno basado en *Spring* y la otra construida en *Kraken.js* usando *Express.js*, *Dust.js* y otros códigos *Open Source*.

Se ejecutaron los tests adecuados utilizando *hardware* de producción que probaran las rutas y recolectaran datos de rendimiento y tiempo de respuesta. Los resultados se aprecian en Figura 3.5.

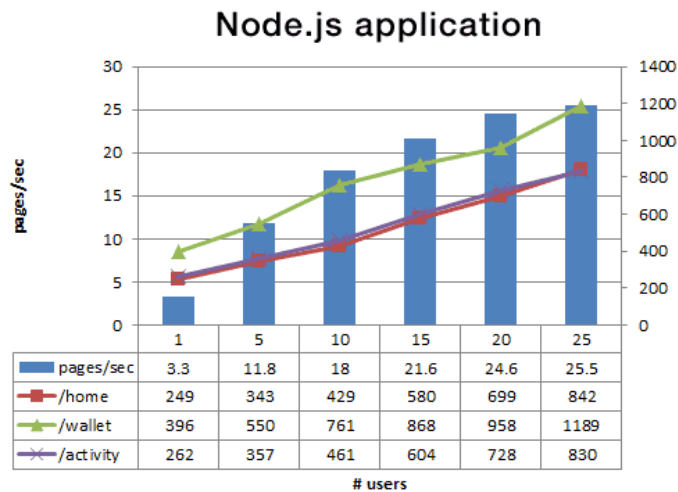
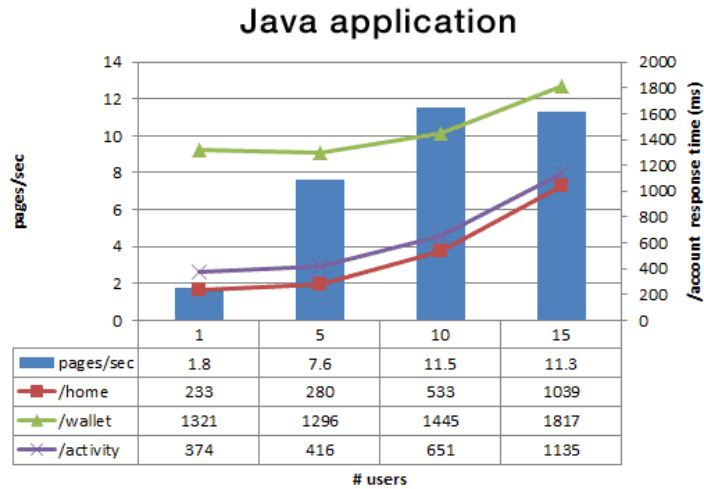


Figura 3.5: Performance de la aplicación en Java y Node.js.

Se observa que la aplicación utilizando *Node.js* tiene:

- El doble de solicitudes por segundo vs. la aplicación *Java*. Esto es incluso más interesante porque los resultados iniciales de rendimiento estaban utilizando un solo núcleo para la aplicación *Node.js*, en comparación con cinco núcleos en *Java*. Esperan aumentar aun más esta brecha en el futuro.
- **Disminución en un 35 % en el tiempo promedio de respuesta** para la misma página. Esto resultó ser **200ms más rápido** en el *servidor*. Algo que los usuarios definitivamente notan.

Más ejemplos de *benchmark* pueden ser encontrados en [14].



## Node.js vs. Rails [53]

De acuerdo a un estudio que realizó *LinkedIn* en el cual evaluó tres posibles soluciones: *Rails/Event Machine*; *Python/Twisted* y *Node.js*, se determinaron los siguientes beneficios de parte de *Node.js*:

- Mejor *Performance*, *Node.js* alcanzando hasta 20 veces más rápido que *Rails* en ciertos escenarios.
- Usando solo 3 *servidores* en lugar de 30, dejando espacio para un crecimiento de 10 veces el tráfico.
- Ingenieros *JavaScript Front-End* pudieron ser utilizados para código *Back-End*, y los dos equipos se transformaron en uno.

## Node.js vs. PHP [108]

No es justo comparar *Node.js* con *PHP*. Lo que realmente se compara es *Node.js* y *PHP + Apache2* ( u otro servidor *HTTP*). En este caso particular, las pruebas fueron realizadas usando *Apache2* y *mod\_php*, dado que es por lejos la configuración más común.

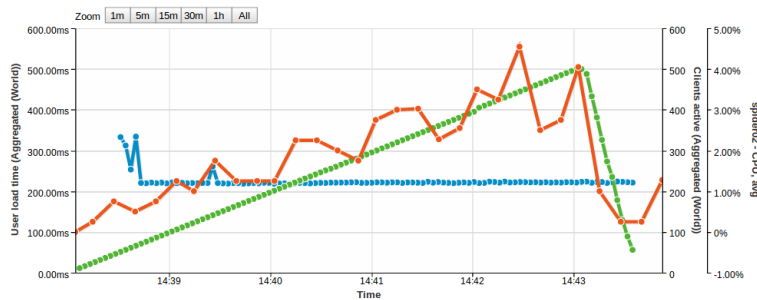


Figura 3.6: Performance de la aplicación *Node.js*.

El gráfico de Figura 3.6 muestra lo que sucede cuando se carga el test en el *servidor* utilizando *Node.js*. La respuesta de tiempo (azul) es mucho más constante durante todo el test.

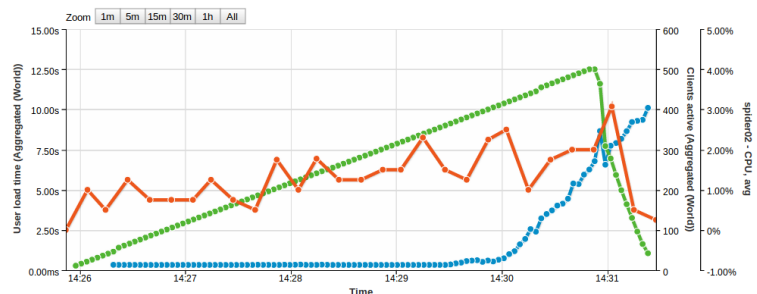


Figura 3.7: Performance de la aplicación en *PHP/Apache™*

Muy diferentes los resultados para el caso de *PHP* (Figura 3.7). No es sencillo observar qué sucede, pero las líneas azules están inicialmente estables a 320ms hasta cerca de 340 usuarios activos simultáneos. Después de eso, se observa un pequeño incremento en el tiempo de respuesta, pero después de agregar más usuarios activos simultáneos, el tiempo de respuesta se dispara por completo.

¿Qué problema tiene *PHP/Apache™*?

La diferencia no es de sorprender. Esto es producto de la diferencia de arquitectura entre ambas soluciones.

Cuando *Apache2* sirve a una página *PHP*, este deja la ejecución *PHP* a un *child process* específico. Ese *child process* puede solo manejar una solicitud *PHP* al mismo tiempo, así que si hay más solicitudes, las otras deben esperar. En el *servidor* hay un máximo de 256 *clientes* (*MaxClients*) configurados vs. 150 que vienen por defecto. Incluso si es posible aumentar el *MaxClients* hasta más allá de 256, habrá problemas con la memoria interna (*RAM*). Al final, se necesita encontrar el balance correcto entre el máximo número de solicitudes simultáneas y los recursos disponibles del *servidor*.

Pero para *Node.js*, es sencillo. Después de todo, cada solicitud es cerca de 30 % más rápido que *PHP*, así que en *Performance*, el cual es una configuración extremadamente básica, *Node.js* es más rápido. Además, en *Node.js* todo está en un solo proceso en el *servidor*. Un proceso manejando una solicitud activa. Así que no existe una comunicación interna de procesos entre diferentes instancias y el proceso madre. Incluso, por solicitud, *Node.js* es mucho más eficiente. *PHP/Apache™* necesita muchísimo *PHP* y sobrecarga de procesos por cada *worker/client* concurrente mientras *Node.js* compartirá la mayoría de su memoria entre las solicitudes.

Más ejemplos de *benchmark* pueden ser encontrados en [101].

### 3.2.2. Full-Stack JavaScript

La elección del uso de *Node.js* trae como consecuencia inmediata el uso de *JavaScript* en *Server-side*. Hay una gran cantidad de ventajas al utilizar un único lenguaje a lo largo del *Stack*. Al programar con *JavaScript* existe la posibilidad de realizar mejoras de rendimiento tanto en el propio *software* como en la productividad de los desarrolladores. Con MongoDB, se pueden guardar los *documents* en un formato *Json-Like*, escribir consultas JSON en los servidores basados en *Node.js*, y similarmente pasar documentos JSON al *Front-End*. *Debugging* y la administración de la base de datos se vuelve una tarea muchísimo más sencilla cuando los objetos guardados en la base de datos son esencialmente idénticos a los objetos que el cliente *JavaScript* ve. Incluso mejor, alguien trabajando en *Client-side* puede fácilmente entender el código del *Server-side* y las consultas a la *Base de datos*; usando la misma sintaxis y objetos todo el proceso libera a los desarrolladores la consideración de varios conjuntos de buenas prácticas de lenguajes y reduce la barrera de entrada para la comprensión de su código base. Esto es especialmente importante en una *hackathon*: el equipo puede no tener mucha experiencia trabajando juntos, y con tan poco tiempo para integrar todas las piezas de su proyecto, cualquier detalle que haga el proceso de desarrollo más sencillo es oro.

### 3.2.3. *Node.js* y e-Commerce

*e-Commerce* es un excelente ejemplo de sistema que se puede ver totalmente beneficiado con el uso de *Node.js* en el lado del *servidor*.

- **Front-End se movió desde el *Server-side* al *Client-side* (al menos en *móvil*).** Una de las grandes consecuencias es que el *Server-side* ya no es más *CPU Bound* ahora es *Memory Bound* e *I/O Bound*. *Node.js* es *event-driven*, lo cual implica eficiencia en *I/O Bound* y es extremadamente eficiente en el uso de memoria.
- Permite velocidad de desarrollo y ejecución. En otras palabras iteraciones rápidas.
- **Comunidad:** Existe una activa comunidad, la cual está constantemente solucionando dudas, además de proporcionar módulos para resolver problemas conocidos.
- **Reutilizar código:** *Server-side* y *Client-side* usan el mismo lenguaje, permitiendo reutilización de componentes.
- **Optimiza el uso de recursos:** Los desarrolladores de *Client-side* pueden desarrollar en *Server-side*, ya que son el mismo lenguaje.
- **Scalability:** Los sitios *web* se encuentran en constante crecimiento. *Scalability* es una consecuencia de la eficiencia que tiene *Node.js* para *Memory Bound* e *I/O Bound*.

### 3.3. *Isomorphic JavaScript*: el futuro de las aplicaciones *web*

Cuando se utiliza un *Back-End* corriendo una aplicación *Java*, *PHP* o *Rails*, el proceso ocurre muy lejos del usuario. Los clientes solicitan datos llamando una *uri*. En respuesta, la aplicación trae datos desde una *Base de dato*, realiza algún procedimiento para crear *HTML* y envía los resultados al cliente. Mientras mayor sea el número de clientes que realizan la misma solicitud, los mecanismos inteligentes de *caching* pueden ser usados por el *servidor*. *sites* de noticias funcionan particularmente bien en este tipo de paradigma.

En un escenario en donde cada usuario realiza solicitudes para crear vistas altamente individualizadas, un único proceso de instancia rápidamente se convierte en un cuello de botella. Considerando a *Facebook* como un ejemplo: Dos personas nunca verán exactamente un mismo *Wall*, esto es calculado individualmente para cada usuario. Esto agrega una gran cantidad de *stress* en el servidor mientras el cliente está en su mayoría de tiempo *idle*, esperando una respuesta.

Cuando el poder de procesamiento de un cliente era relativamente limitado, este enfoque tenía sentido, pero en la actualidad un solo *smartphone* tiene más poder de cómputo que muchos de los súper computadores en los primeros años de la *web*. Los nuevos enfoques toman ventaja de ese poder y delegan la mayoría del proceso a los clientes. *Front-Ends* inteligentes solicitan información al *servidor* y montan el *DOM* en el *browser*.

Este enfoque *client-centric* implica dos ventajas significativas:

- Menos necesidad de transferencia de información entre el *servidor* y el cliente, lo que esencialmente se traduce como tiempos de respuesta mas rápidos.
- Existe una menor probabilidad de que el procesamiento sea bloqueado por otros usuarios debido a peticiones de duración prolongada, ya que la mayor parte del trabajo se realiza individualmente en cada cliente.

La arquitectura *client-server* se basa en *stateless-connections* [C]. clientes solicitan información una vez, el *servidor* responde y la conexión es cerrada nuevamente. Actualizaciones en otros clientes pueden suceder, a menos que explícitamente realiza una solicitud al *servidor* nuevamente, no verán actualizaciones. No existe un canal de *feedback* del *servidor* al cliente para enviar actualizaciones de contenido.

Mover el proceso desde un solo *servidor* a múltiples clientes envuelve moverse en dirección a una plataforma de cómputo distribuida. En tal situación, los datos deben ser enviados en ambas direcciones.

Después de muchos años de avance, se desarrollaron soluciones muy creativas que impulsaron a los *browsers* a mejorar para mantenerse a la par. Actualmente la *web* se ha transformado en una plataforma de aplicaciones *fully-featured*, *runtimes JavaScript* rápidos y *standard HTML5* que permite crear aplicaciones que antes eran solo posibles en plataformas nativas.

### 3.3.1. Single Page Application

No pasó mucho tiempo hasta que los desarrolladores comenzaron a crear aplicaciones completas en el *browser* utilizando las nuevas capacidades de *JavaScript* y *HTML5* para crear Single Page Application. Una SPA es una aplicación *web* completa, la cual solo tiene una página, que es usada como un contenedor para todas las páginas *web* de la aplicación y utiliza *JavaScript*, *HTML5* y *CSS* para todas las interacciones *Front-Ends*. En SPAs no hay *posts* completos de vuelta al *servidor*, no hay refrescos completos de una página *web*, y no hay objetos embebidos. La principal diferencia con las aplicaciones *web* tradicionales es la naturaleza de las *requests* y *responses* que siguen al *request HTTP* inicial. Con SPA se utiliza *Ajax* para *request* datos y se obtiene un *responses* correspondiente a datos JSON. Una vez que la información es recibida por el cliente, este hará un despliegue parcial de *HTML* para representar los cambios. Incluso, moviéndonos de una página a otra en un SPA ocurre en *Client-side*, lo que es diferente a lo que sucede en aplicaciones tradicionales y RIA.

Librerías como *Backbone.js*, *Ember.js*, y *Angular.js* usualmente son referidas como librerías *Client-side MVC* o *MVVM*. La arquitectura típica *Client-side MVC* se ve similar a la Figura 3.8.

## Client-side MVC

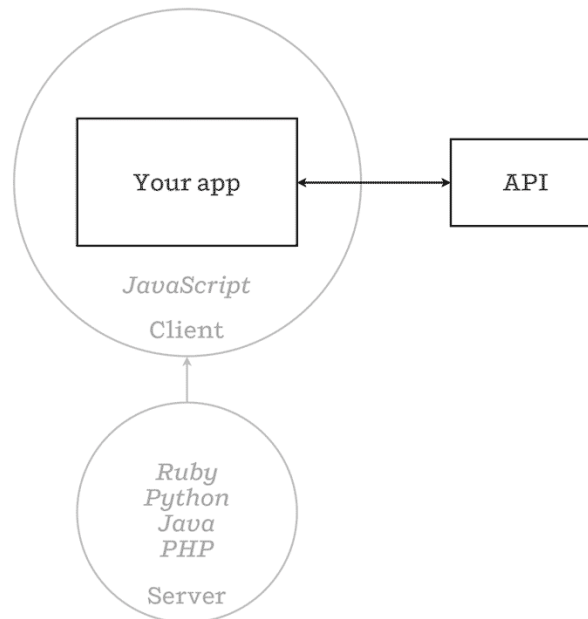


Figura 3.8: *Client-side MVC*.

Otro cambio entre SPA y aplicaciones *web* tradicionales es el manejo de estados. En SPA, dado que no se abandona o refresca la página *web* principal se puede persistir el estado en la memoria del *browser*. Incluso, si se desea guardar el estado de la aplicación en escenarios *offline* o cuando los usuarios cierran el *browser*, se puede utilizar *HTML5 storage* para mantener el estado. Después cuando el usuario regrese a *online*, puede regresar al último estado de la aplicación sin involucrar al *servidor*.

Esta arquitectura es genial para los desarrolladores, porque la idealizada Single Page Application tiene una separación de conceptos clara entre el cliente y el *servidor*, promoviendo un hermoso *workflow* y previene la necesidad de compartir lógica entre ambos, lo que usualmente es escrito en diferentes lenguajes.

### 3.3.2. No todo lo que brilla es oro

En la práctica, sin embargo, hay algunos defectos fatales con este enfoque que impide que sea adecuado para muchos casos de uso.

- **SEO.** Una aplicación que puede solo *Run* en el *Client-side* no puede entregar *HTML* a los *Crawlers*, lo que implica que el *web* tendrá un SEO mediocre por *default*. Los *Crawlers* actúan enviando un *request* a un *Webserver* e interpretando el resultado; pero si el *servidor response* una página en blanco, esto no tiene mucho valor. Existen algunas soluciones, pero estas requieren algo de trabajo [58].

- **Performance.** Por la misma razón, si el *servidor* no despliega una página completa de *HTML*, en lugar de eso el *Client-side* espera para que *JavaScript* lo realice, los usuarios experimentarán críticos segundos de una página en blanco o un *loading spinner* antes de ver el contenido de la página. Hay una abundante cantidad de estudios que muestran los drásticos efectos que un *site* lento tienen en los usuarios, y por lo tanto su *revenue* [183]. **Amazon afirma** que cada 100ms que se reduce el *load time* de una página aumenta el *revenue* en 1 % [116] [184]. **Twitter invirtió un año y 40 ingenieros** para *rebuilding* su *site* para desplegar en el *servidor* en lugar del cliente, afirmando una mejora x5 en la percepción de *loading time* [187].
- **Maintainability.** Mientras que en el caso ideal se podría dirigir hacia un linda, y limpia separación de conceptos, inevitablemente algunos *bits* de la lógica de la aplicación o de la vista terminen duplicados entre el cliente y el *servidor*, usualmente en diferentes lenguajes. Ejemplos comunes son el formateo de *date* y *currency*, validación de formularios, y la *routing logic*. Esto hace que *maintenance* sea una pesadilla, especialmente para aplicaciones más complejas [58].

### 3.3.3. Un enfoque híbrido

La experiencia muestra que la situación más deseable es un híbrido entre el enfoque clásico y el enfoque moderno: Se desea entregar *HTML fully-formed* desde el *servidor* para *Performance* y *SEO*, pero también la *Velocidad* y *Flexibilidad* de la lógica de aplicación de *Client-side*.

Aplicaciones *Isomorphic JavaScript* son aquellas en que la aplicación *JavaScript* puede *Run* tanto en *Client-side* como en *Server-side*.

Una aplicación *Isomorphic* debería verse como se representa en Figura 3.9.

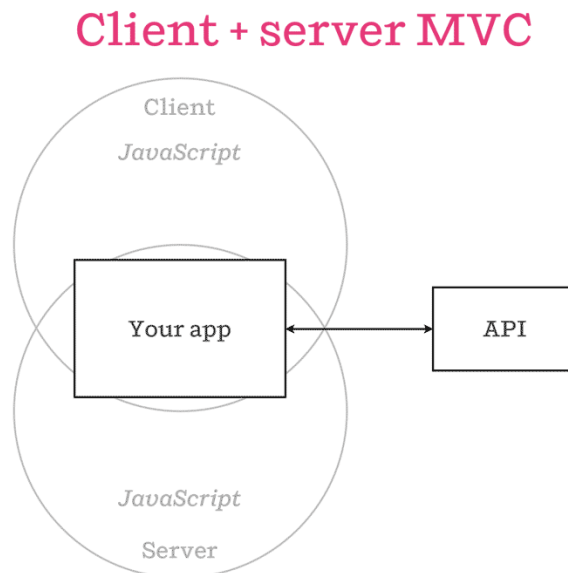


Figura 3.9: MVC *Isomorphic* cliente servidor.

Algunas de las aplicaciones y *view logic* que son utilizadas recurrentemente, pueden ser ejecutadas tanto en el *servidor* como en cliente. Esto permite nuevas posibilidades: *Performance optimization*, mejor *Maintainability*, SEO por *default*, y más aplicaciones *web Stateful*.

Con la aparición de *Node.js*, un *Fast, Stable Server-side JavaScript runtime*, es factible hacer este sueño realidad. Creando las abstracciones apropiadas, es posible escribir *lógica* de aplicación que corra en el *servidor* y el cliente; justamente la definición de *Isomorphic JavaScript*.

### 3.3.4. *Isomorphic JavaScript Frameworks*

*Nodejitsu* escribió una gran descripción de una arquitectura *Isomorphic JavaScript* en 2011 [146], pero fue adoptado lentamente.

El cliente y el *servidor* son *environments* muy distintos, y por lo tanto es necesario un conjunto de abstracciones que desacoplen la *lógica* de la aplicación desde su implementación subyacente, así se puede exponer una sola *API* para los desarrolladores.

- **Routing.** Se desea un solo *set* de *routes* para *mapear* patrones *uri* que *manejar route*. Los *manejadores* de *route* necesitan disponibilidad de acceso a *HTTP headers*, *cookies*, e información *uri*, y especificar redireccionamiento sin acceder directamente a *window.location* (*browser*) o *req* y *res* (*Node.js*) [59].
- **Fetching y Persisting Data.** Se desea describir los *resources* necesarios para desplegar una página particular o *componente* independiente del mecanismo *Fetching* [59].
- **View rendering.** Ya sea que se decida manipular directamente *DOM*, seguir con un *templating HTML string-base*, o bien optar como una de las *componentes UI* con una abstracción *DOM*, es necesaria la capacidad de generar *Markup Isomorphically*. Es necesaria la factibilidad de desplegar cualquier *View* desde el *servidor* o el cliente, dependiendo solo de las necesidades de la aplicación [59].
- **Building y Packaging.** Escribir código de aplicaciones *Isomorphic* es la mitad de la batalla. Herramientas como *Grunt* y *Browserify* son partes esenciales del *workflow* para obtener una aplicación corriendo. Puede existir un número de *build steps: compiling templates*, incluyendo dependencias *Client-side*, aplicando transformaciones, *minification*, etc. El caso sencillo es combinar todo el código de la aplicación, *Views* y *templates* en un único *bundle*, para aplicaciones grandes, esto puede resultar en cientos de *kilobytes* para *download*. Un enfoque más avanzado es crear *dynamic bundles* e introducir *lazy loading assets*, sin embargo, esto rápidamente se torna complicado. Herramientas *static-analysis* como *Esprima* permiten a desarrolladores ambiciosos intentar *optimization* y *metaprogramming* para reducir *boilerplate code*.

Todo *Framework* que resuelve estos problemas al mismo tiempo es considerado un *Isomorphic JavaScript Frameworks*. Lo interesante de esto es que este tipo de *Frameworks* **permite a los desarrolladores enfocarse en la *business logic***.

### 3.4. Meteor

*Meteor* es un *environment* ultra-simple para construir *websites* modernos. Lo que antes demoraba semanas, incluso con las mejoras herramientas, ahora toma horas con *Meteor*.

La *web* fue diseñada originalmente para trabajar de la misma manera que los *Mainframes* funcionaban en los 70s. La aplicación del *servidor* *rendered* una *screen* y la envía a través de la *network* a un *dumb terminal* (Figura 3.10). Si el usuario realizó alguna acción, el *servidor* *rendered* una *screen* completamente nueva. Este modelo funcionó bien para la *web* por una década. Esto permitió la existencia de LAMP, *Rails*, *Django*, *PHP*.

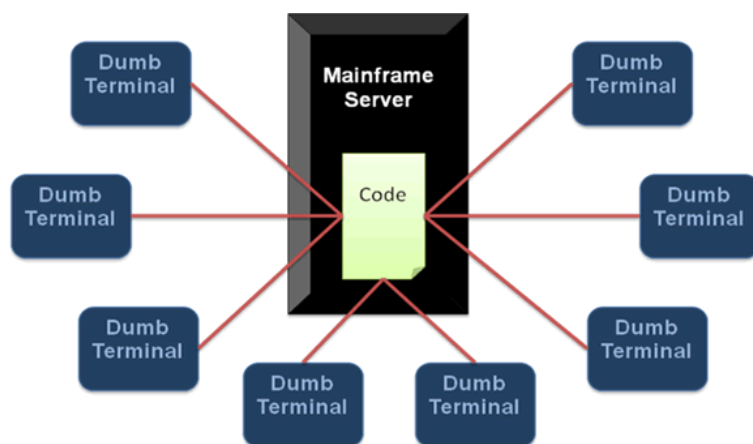


Figura 3.10: Arquitectura de *Mainframe* con *dumb terminal*.

Pero los mejores equipos, con los más grandes presupuestos y más largos *schedules*, ahora construyen aplicaciones en *JavaScript* que corren en el cliente. Estas aplicaciones tienen interfaces estelares. Estas aplicaciones no *reload pages*, son *reactive*: cambios de cualquier cliente aparecen inmediatamente en las *screen* de todos.

La aplicación fue construida de la manera difícil. *Meteor* hace esto con la intención de ofrecer simpleza, y muchísima más entretención. Es posible construir una aplicación completa en un fin de semana, o en una *hackathon* con la suficiente cafeína. Ya no es necesario los *resources* del *servidor*, o *deploy API endpoints* en la *cloud*, o manejar una *Base de datos*, o desgastarse con una *ORM layer*, o intercambiar *back and forth* entre *JavaScript* y *Ruby*, o *broadcast* datos de invalidación a los *clientes*.

*Meteor* es un trabajo en desarrollo, pero se espera que muestre la dirección del equipo desarrollador. Los desarrolladores están deseosos de recibir *feedback*.

#### 3.4.1. Principios de *Meteor*

- **Datos on the wire.** *Meteor* no envía *HTML* a través de la *network*. El *servidor* envía datos y permite al cliente desplegar la información.



- **Un lenguaje.** *Meteor* permite escribir el código correspondiente al cliente y al *servidor* de la aplicación en *JavaScript*.
- **Base de datos en todas partes.** Se pueden utilizar los mismos métodos para acceder a la *Base de datos* desde el cliente o el *servidor*.
- **Compensación de latency.** En el cliente, *Meteor prefetch* es datos y simula modelos para aparentar que los métodos del *servidor* retornan instantáneamente.
- **Full-Stack reactivity.** En *Meteor*, *tiempo-real* es el *default*. Todas las *layers*, desde la *Base de datos* hasta el *template*, *update* por sí mismo automáticamente cuando es necesario.
- **Embrace the Ecosystem.** *Meteor* es un *Open Source* e integra herramientas y *Frameworks Open Source*.
- **Simplicity igual productivity.** La mejor manera para hacer algo que parezca sencillo es tener lo que realmente sea simple. La principal funcionalidad de *Meteor* tiene limpias y hermosas *APIs* clásicas.

### 3.4.2. ¿Qué es Meteor?

*Meteor* son dos cosas:

- Una librería de *Packages*: *módulos pre-written* y *self-contained* que probablemente serán necesarios en las aplicaciones.

Hay cerca de una docena de *Packages core Meteor* que prácticamente cualquier aplicación usará. Dos ejemplos: *webapp*, la cual *maneja* conexiones *incoming HTTP*, y *templating*, lo que permite hacer *templates HTML* que automáticamente actualiza en vivo cuando los datos cambien. Entonces hay *Packages* opcionales como *email*, que permite a una aplicación enviar *emails*, o la serie de cuentas de *Meteor*, las cuales proporcionan un sistema de cuentas de usuarios *full-featured* que puede ser *drop right into* la aplicación. Adicionalmente a estos *Packages "core"*, hay miles de *Packages* escritos por la comunidad, los cuales pueden ser necesarios para alguna aplicación propia.

- Una herramienta *command-line* llamada *meteor*.

*meteor* es una herramienta de construcción análoga para hacer, examinar, o las partes no-visuales de *Visual Studio*. Reúne todos los archivos y *assets* en la aplicación, lleva a cabo cualquier paso de construcción necesario (tal como *minifying CSS*, construir *módulos NPM*, o generar *source maps*), trae los *Packages* usados por la aplicación, y entrega un paquete independiente de aplicaciones *ready-to-run*. En modo desarrollador puede hacer todo esto de manera interactiva, así que siempre que se cambie un archivo, inmediatamente se reflejarán los cambios en el *browser*. Esto es muy sencillo de utilizar *out of the box*, pero también es extensible: es posible agregar soporte para nuevos lenguajes y compiladores incluyendo *build plugin Packages* a la aplicación.

La idea clave de *Meteor package system* es que todo debería funcionar exactamente igual en el *browser* y en el *servidor* (siempre que esto tenga sentido, claramente *browsers* no pueden enviar *emails* y *servidores* no pueden capturar eventos del *mouse*). El ecosistema completo ha sido construido desde cero para soportar esto.

### 3.4.3. El Stack Meteor

El *Stack Meteor* (Figura 3.11) es un miembro de la familia *MEAN*, esto significa que está impulsado por *Node.js* en el *Server-side*. Es el mismo propósito que tiene el *Webserver Apache™* en el *Stack LAMP*.

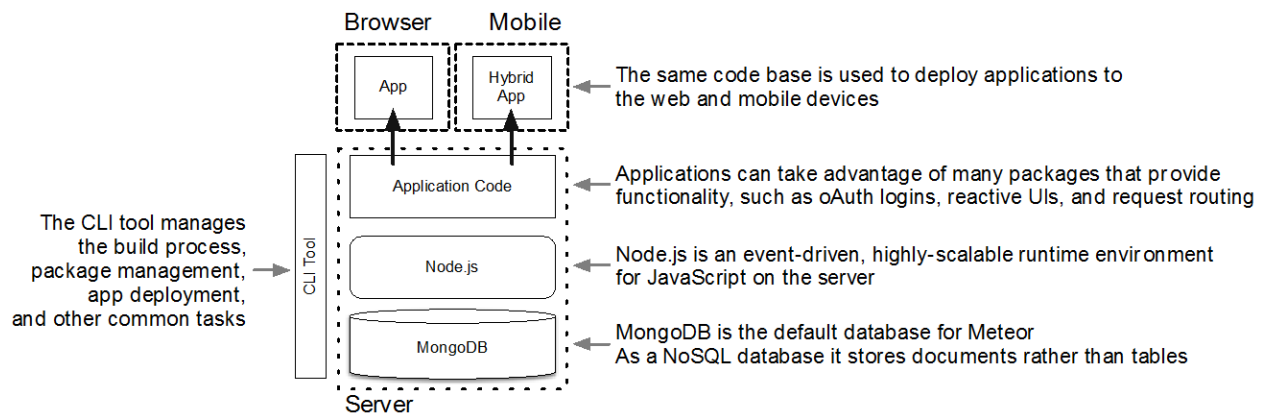


Figura 3.11: El *Stack Meteor* corre aplicaciones potenciadas por *Packages* inteligentes sobre *Node.js* y *MongoDB*.

Toda la información es típicamente guardada dentro de *MongoDB*. Provee una *API JavaScript* que da acceso a todo el contenido almacenado en forma de *documents* o *objects*. El mismo lenguaje dentro del *browser* puede ser usado para acceder a los datos, del cual *Meteor* toma ventaja para implementar un verdadero desarrollo *Full-Stack*.

Todo el *software* y librerías requeridos para crear aplicaciones *web* desde cero son agrupados en forma de *Packages* inteligentes, así los desarrolladores pueden empezar de inmediato. Estos *Packages* incluyen una librería de interfaz de usuario (*Blaze*), un manejador de cuentas de usuarios (*accounts*), y mantener a todos los *clientes* *reactively updated* en *tiempo-real* (*Tracker*).

La herramienta *CLI* de *Meteor* permite a los desarrolladores rápidamente *setup* un *environment* completo de desarrollo. No es necesario saber cómo instalar o configurar algún *software* de *servidor*. *Meteor* cuida literalmente los aspectos de la infraestructura. Esto es también una herramienta para construir, comparable a *make* o *Grunt*, y un manejador de *package*, tal como *APT* o *NPM*. Finalmente la herramienta *CLI* agrupa una aplicación para correr en diferentes plataformas de *clientes*, dentro de un *browser web* o como una aplicación *móvil* nativa.

Todas las partes del *Stack* se integran sin problemas, todos los *Packages core* son diseñados y probados para trabajar bien en conjunto. Por otro lado, es perfectamente posible cambiar partes

del *Stack* a otros, en caso de ser necesario. En lugar de utilizar *Meteor* en su totalidad podría decidir utilizarse sólo los componentes de *servidor* y utilizar por ejemplo *Angular.js* para el *Client-side*, o utilizar un *Java Backend* que utilice *Meteor* en el *Front-End* para proveer *updates tiempo-real* para todos los *clientes*.

#### **3.4.4. Framework Isomorphic – Full-Stack JavaScript**

*Meteor* corre sobre *Node.js* y mueve la lógica de la aplicación hacia el *browser*, lo que es usualmente referido a Single Page Application. El mismo lenguaje es usado sobre todo el *Stack*, lo que transforma a *Meteor* en una plataforma *Isomorphic*. Como resultado el mismo código *JavaScript* puede ser usado en el *servidor*, en el cliente, e incluso en la *Base de datos*.

Mientras muchos *Frameworks* usan el mismo lenguaje tanto en cliente como en *servidor*, la mayoría del tiempo estos no pueden compartir código porque los *Frameworks* no son íntimamente integrados, por ejemplo el uso de *Angular.js* en el *Front-End* y *Express.js* en el *Back-End*. *Meteor* es un *Full-Stack* verdadero porque usa una simple y unificada *API* expuesta para todas las funcionalidades *core* y pueden ser utilizadas en el *servidor*, en el *browser*, e incluso para acceder a la *Base de datos*. Para comenzar, no es necesario aprender múltiples *Frameworks* y da como resultado una mejor *re-usability* del código solo utilizando el mismo lenguaje.

Para acceder a la *Base de datos* desde el *browser*, *Meteor* incluye una mini *Base de datos*. Esta simula exactamente la misma *API* de una *Base de datos*. Dentro del *browser* *Minimongo* permite a los desarrolladores utilizar los mismos comandos como si estuvieran en una consola de *MongoDB*.

Típicamente los desarrolladores necesitan saber cómo escribir código que tome una completa ventaja del *event loop* y qué funciones corren *synchronously* y cuáles *asynchronously* (*Node.js*). Mientras más funcionalidades *asynchronously* son utilizadas, tanto mayor serán los *callbacks* envueltos y las cosas se vuelven bastante sucias.

Afortunadamente, *Meteor* aprovecha toda la potencia de *event loop*, pero facilita el proceso evitando cierta preocupación sobre escribir código *Asynchronous*. Utiliza el concepto llamado *Fibers behind the scenes*. *Fibers* proveen una capa de abstracción para el *event loop* que ejecuta funciones *Asynchronous* (*tasks*) en secuencia. Esto elimina la necesidad de *callbacks* explícitos de manera que se pueda utilizar un estilo *Synchronous* familiar.

#### **3.4.5. ¿Por qué Meteor?**

No importa lo bueno que una aplicación sea, la cantidad de prestaciones. Incluso puede ser superior a las que los competidores presentan; todo esto no significa nada si finalmente la aplicación no es utilizada por los usuarios objetivos.

Esta misma afirmación se aplica para el caso de los *Frameworks*; no importa cuan interesante

pueda ser, cuántas herramientas provee, etc. Si no existe una comunidad activa creando nuevo contenido, solucionando infinidad de problemas, incluso encontrando *bugs* para lograr una versión más estable. O lo que es lo mismo, si existe una gran comunidad activa, entonces tienes un *Framework* con un gran respaldo.

*HotFrameworks* es un *website* que se dedica a posicionar los *Frameworks* en un *ranking* de acuerdo a lo popular que estos sean, o lo que es lo mismo, de acuerdo a cuan activa es su comunidad. Dicho estudio, que se aprecia en Tabla 3.2, posiciona a *Meteor* en la sexta posición.

<i>Framework</i>	<i>Git Hub Score</i>	<i>Stack Overflow Score</i>	<i>Overall Score</i>
<i>ASP.NET</i>	-	100	100
<i>Ruby on Rails</i>	96	98	97
<i>Angular.js</i>	100	90	95
<i>ASP.NET MVC</i>	-	93	93
<i>Django</i>	89	91	90
<i>Meteor</i>	95	74	84

Tabla 3.2: Ubicación del *Framework Meteor* [178].

Importante recordar que este *ranking* incluye todos los *Frameworks* sin discriminar. Entonces, no solo es el sexto *Framework* con la mayor comunidad activa, sino que es el primero solo considerando los *Framework JavaScript Isomorphic*.

### 3.4.6. ¿Por qué son necesarios los *Frameworks*?

Un *Framework* es un conjunto de bloques de *software* comunes y prefabricados que los programadores pueden utilizar, extender o customizar para soluciones específicas. Con *Frameworks* los desarrolladores no necesitan comenzar a escribir una aplicación en cada oportunidad desde el principio. *Frameworks* son construidos desde una colección de objetos tal que tanto el diseño como el código del *Framework* pueden ser reutilizados [82].

Un esqueleto de una aplicación en la cual los desarrolladores agregan en sus códigos y proveen la mayoría de las funcionalidades comunes [85].

Un *Framework* es un directorio con jerarquía que encapsula recursos compartidos, tales como *dynamic shared library*, documentación de referencia, imágenes, en un solo *package*. Múltiples aplicaciones pueden utilizar todos los recursos de manera simultánea. El sistema los carga en *memoria* según necesidad y comparte una copia de los recursos entre todas las aplicaciones posibles [121].

De acá se difiere que no existe una definición única para lo que significa el concepto *Framework*. Sin embargo, todo *Framework* debería ser simultáneamente:

- Un *wrapper* es una manera de *repackaging* una función o un conjunto de funciones (no necesariamente relacionadas) para lograr al menos uno de los siguientes objetivos: Simplificar

uso; Consistencia en Interfaz; Mejora de las funcionalidades *core*; Recolectar procesos discretos dentro de una asociación lógica (un *object*) [61].

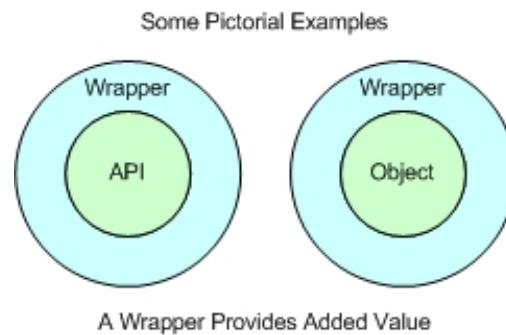


Figura 3.12: Un *wrapper* provee valor agregado.

- Una **Arquitectura** es un estilo que incorpora elementos de diseño específicos [61].

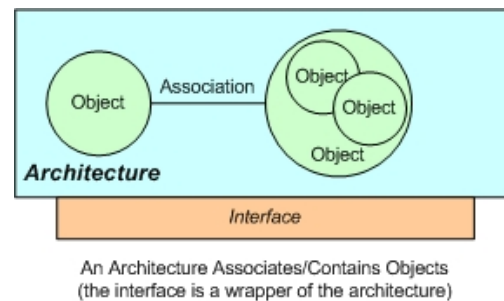


Figura 3.13: Una *Arquitectura* associates/contains *objects*. (La *interface* es un *wrapper* de la *Arquitectura*.)

- **Metodología**. Es la manera para realizar algo. En este caso, define las interacciones entre *Arquitectura*, *componentes* y *objects* [61].

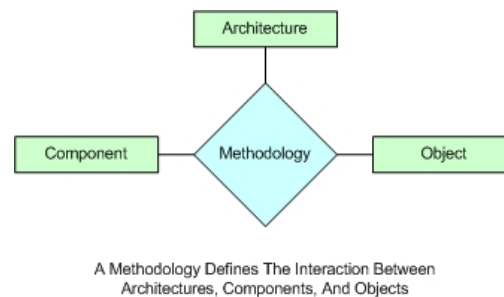


Figura 3.14: Una *Metodología* define las interacciones entre *Arquitectura*, *componentes* y *objects*.

Elegir el *Framework* correcto para un proyecto puede tener un impacto tremendo en la habilidad de entregar a tiempo y la habilidad para mantener el código en el futuro. Es ciertamente deseable un *Framework* sólido, estable y probado, pero sin estar limitado por la opción.

## Capítulo 4

# Arquitectura

*Packages* es una estrategia de programación para *code separation*, *modularity*, y *re-usability*. Al estructurar el código de cada *característica* en un *Packages* separado, el código de una *característica* no tendrá acceso al código de otra *característica* excepto utilizando *export*, haciendo cada dependencia explícita. Además esto permite la manera más sencilla para realizar *testing* independientes a cada *característica*. Incluso es posible *publish* los *Packages* y utilizarlos en otros proyectos.

Es esta forma de estructurar el código la parte principal del desarrollo del *Framework e-Commerce*. Cada vez que se requiera desarrollar una nueva solución específica utilizando el *Framework*, simplemente se incluirán todos aquellos *módulos* necesarios para el desarrollo de la aplicación, y con un poco de desarrollo, obtener las *características* específicas buscadas.

Como este *Framework* será completamente *Open Source*, la comunidad podrá crear nuevos *módulos* con el fin de crear aquellas *características* que serán requeridas en el futuro.

Como se ha mencionado, *Meteor* utiliza *librerías* y herramientas; las combina con nuevas consideraciones y nuevas *librerías*, servicios y *standards*; y las agrupa para proveer un ecosistema completo de desarrollo de aplicaciones *web* y *móvil* que da gusto utilizar. Sin embargo una de las debilidades que tiene *Meteor* está relacionada con las pocas convenciones para la estructura de aplicaciones y el código. Esta libertad o flexibilidad es aceptable para desarrolladores únicos, quienes pueden rápidamente escribir código, pero esto requiere una muy buena coordinación entre miembros de un equipo cuando la aplicación crece en tamaño. Todo depende de la preferencia de los desarrolladores si utilizan un solo archivo o cientos de carpetas y archivos. Para resolver este problema se propondrá una estructura, apoyada por una variedad de *Packages* para el desarrollo de grandes aplicaciones, las que usualmente serán desarrolladas por equipos de trabajo.

## 4.1. Packages desarrollados por la comunidad

Una gran cantidad de soluciones han sido desarrolladas por la comunidad para solucionar problemas altamente comunes a la hora de desarrollar aplicaciones. A continuación dichos *Packages* serán nombrados y descritos.

- **Collection2 [48]** . Permite asociar un *Schema* a un *Mongo.Collection*. Automáticamente se valida la estructura cuando se inserta o actualiza desde el cliente o el *servidor*. Esto es altamente deseable principalmente porque asegura que solo propiedades y valores aceptables puedan ser insertados desde el cliente. De esta manera, inserciones y actualizaciones *Client-side* pueden ser permitidas sin comprometer seguridad o integridad de la información.
- **Router [111]** . Un enrutador que funciona en el *servidor* y en el *browser*, diseñado específicamente para *Meteor*.
- **Collection Hooks [130]** . Extiende *Mongo.Collection* con *hooks before/after* para *insert*, *update*, *remove*, *find* y *findOne*.
- **Bower [136]** . *Bower.io* [29] es un repositorio popular de librerías *JavaScript Client-side*.
- **Security [138]** . Es un *package* de *Meteor* que proporciona un lenguaje *API* simple y lógico para definir seguridad para las *Collections* de MongoDB. Agrega funcionalidades al *core* de seguridad *allow/deny*.
- **Velocity [195]** . El *test-runner reactive* para *Meteor*.
- **Jasmine [154]** . Corresponde a la integración de *Meteor Velocity* [194] para el *Framework* de *testing Jasmine* [119] versión 2.3. Facilita el proceso de *testing* en la aplicación y los *package* de *Meteor* con *tests* unitarios e integrados.
- **Bunyan [32]** . Exporta y agrega el *módulo* de *logging Bunyan* [192] . También agrega el cliente *Browserify* y *bunyan-prettystream*.
- **D3.js [68]** . *Data-Driven Documents* es una *librería JavaScript* para la manipulación de *documents* basados en datos. *D3.js* [68] facilita el proceso de publicación de datos utilizando *HTML*, *SVG* Y *CSS*. *D3.js* [68] enfatiza el uso de *web standard* y permite el uso de todas las opciones disponibles en los *browsers* modernos, sin la necesidad de utilizar un *Framework* propietario, combinando componentes visuales y un acercamiento a *data-driven* para la manipulación de *DOM*.
- **undersore-string-latest [69]** . *underscore.string package* para *Meteor*. Es una librería para la manipulación de *strings*.
- **LESS [34]** . *LESS* extiende *CSS* con comportamientos dinámicos tales como variables, *mixins*, operaciones y funciones. Esto permite *stylesheets* más compactos y ayuda a la reducción de duplicación en los archivos *CSS*.

- **bootstrap** [137] . Este *package* integra *bootstrap* en *Meteor* permitiendo configurar lo que realmente se necesita.
- **Browser-Policy** [33] . Permite establecer políticas de seguridad que se promoverán en los nuevos *browsers*. Estas políticas ayudan a prevenir y mitigar ataques comunes como *cross-site-scripting* y *clickjacking*.
- **Roles** [47] . *package* de autorización para *Meteor*. Es compatible con el *package* de cuentas oficial de *Meteor*.

## 4.2. Estructura propuesta para el desarrollo de grandes aplicaciones

Cuando se comienza a trabajar en *Meteor* es importante entender qué código es ejecutado y en qué ambiente al momento de escribir aplicaciones. En teoría, todo el código puede correr en cualquier parte en el *Stack*, pero existen algunas limitaciones. Hay *APIs* que no deberían enviarse al cliente; los *events* que manejan los *clicks* del *mouse* no son útiles en el *servidor*. Para indicar a *Meteor* dónde ejecutar código específico, es posible organizar código en carpetas dedicadas o utilizar un *check* para verificar en qué contexto están corriendo.

En *Meteor* existen directorios especiales que separan lógicamente el alcance de su contenido. Estos directorios se pueden apreciar en la Figura 4.1.

- **client**. Todo el código que se encuentre dentro de la carpeta **client** solo será visible desde el *Client-side*.
- **server**. Todo el contenido de la carpeta **server** estará disponible solo para el *Server-side*.
- **public**. Los archivos dentro de esta carpeta son visibles desde el *Client-side* y *Server-side*.
- **private**. A los archivos dentro de esta carpeta se pueden acceder solo desde el código del *servidor* a través de un *API* de *assets*.

El código compartido es especialmente útil para el desarrollo de aplicaciones. A modo de ejemplo, en el caso de las validaciones, el mismo método es utilizado tanto en el *Client-side* como en el *Server-side*. En el *Client-side* utilizado para mostrar un mensaje de error en el *browser* (un rut mal escrito), y en el *Server-side* para no persistir información incorrecta en la *Base de datos*.

Para definir una estructura de desarrollo, se debe considerar, entre otras cosas, qué paradigma de programación se está utilizando, cuales son sus principios, y el *API* que este proporciona. Se procede entonces a definir ciertos conceptos relevantes:

- **templates**. En *Meteor*, las *Views* son definidas en *templates*. Un *template* es una pequeña porción de *HTML* que puede incluir datos dinámicos. Incluso es posible interactuar con el *template* desde código *JavaScript* para insertar información y escuchar *events* [75].



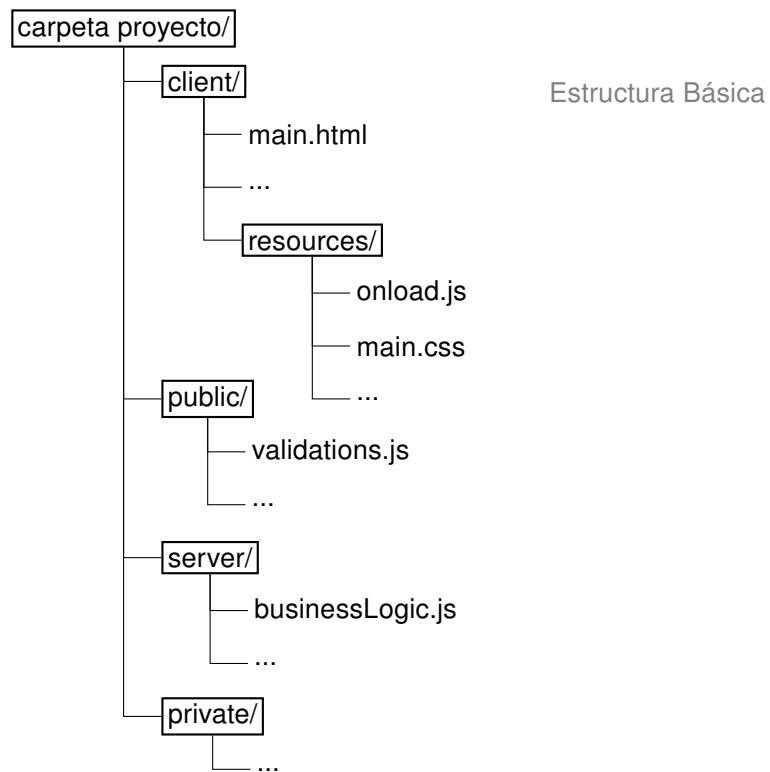


Figura 4.1: Estructura simple de una aplicación en *Meteor*.

- **Session.** Provee un objeto global en el cliente que puede utilizarse para guardar un arbitrario par *Key-Value*. [75].
- **Tracker.** *Meteor* tiene sistema simple de dependencia para hacer *tracking*, permitiendo así automáticamente *re-run templates* y otras funciones siempre que variables *Session*, consultas a la *Base de datos*, y otros recursos de datos cambien [75].
- **Collections.** *Meteor* guarda la información en *Collections*. Objetos *JavaScript* almacenados en *Collections* son denominados *documents* [75].
- **Publish and subscribe.** El *servidor* de *Meteor* puede publicar un conjunto de *documents*, y los clientes pueden suscribirse a esas publicaciones [75].
- **Methods.** Son funciones del *Server-side* que pueden ser llamadas desde el *Client-side*. Son útiles en situaciones que se requiere realizar acciones más complicadas que *insertar*, *actualizar* y *remover*; o cuando es necesario realizar ciertas validación de datos [75].

Teniendo estas consideraciones en mente, se define la siguiente estructura global para el desarrollo de una aplicación cualquiera.

Como se mencionó, **client** es la carpeta que es solo visible desde el *Client-side*, lo que significa que contiene solo código útil para el cliente. Su estructura básica se puede observar en Figura 4.3; y cada una de sus partes corresponde a:

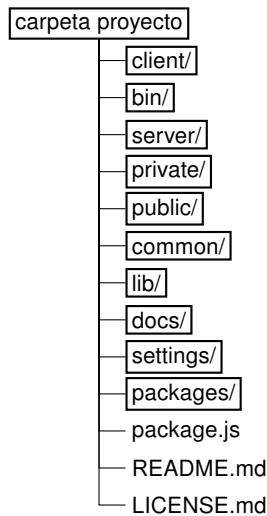


Figura 4.2: Estructura global para una aplicación en *Meteor*.

- **helpers/**. Carpeta donde se encuentran todos los *helpers* globales del sistema.
- **templates/**. Contiene los *templates* y sus respectivos *helpers*. Se utiliza el mismo nombre en ambos archivos, con el fin de identificarlos. En Figura 4.4 se muestra con detalle esta carpeta.
- **app.js**. Lugar donde están todas aquellas funciones y variables globales del sistema.
- **subscriptions.js**. Lugar donde se especifican todas las subscripciones a las diferentes publicaciones que proporciona el *servidor*.

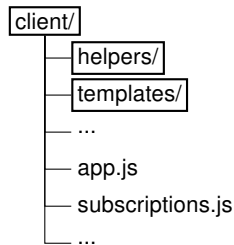


Figura 4.3: Estructura de la carpeta **client** para una aplicación en *Meteor*.

La carpeta *template* tiene el suficiente detalle y relevancia como para indicar sus partes.

- **account/**. Contiene los *templates* y *helpers* necesarios para manejar las cuentas de usuarios.
- **dashboard/**. Aquí están todos los archivos necesarios (*templates* y *helpers*) para el manejo de la interfaz de administrador. Dada su naturaleza, es esperable que se agreguen una gran cantidad de carpetas.
- **layout/**. Contiene todos aquellos *templates* con sus respectivos *helpers* que son globales en el sistema. Como por ejemplo las alertas, el encabezado, el pie de página, *loading*, e interfaces estáticas tales como: acceso no autorizado, página no encontrada, recurso no encontrado, etc.

Es importante agregar que estas interfaces estáticas no cuentan con *helpers*; la razón tras esto es que el encargado de manejar esos *templates* es el *Packages Router* [111] .

- **Carpetas exclusivas del proyecto.** Acá deberían ir las carpetas que separen lógicamente las diferentes funcionalidades de la aplicación que se desea realizar. A modo de ejemplo, en el proyecto que se está realizando en este memoria se han creado carpetas para el carro de compras y los productos, respectivamente.

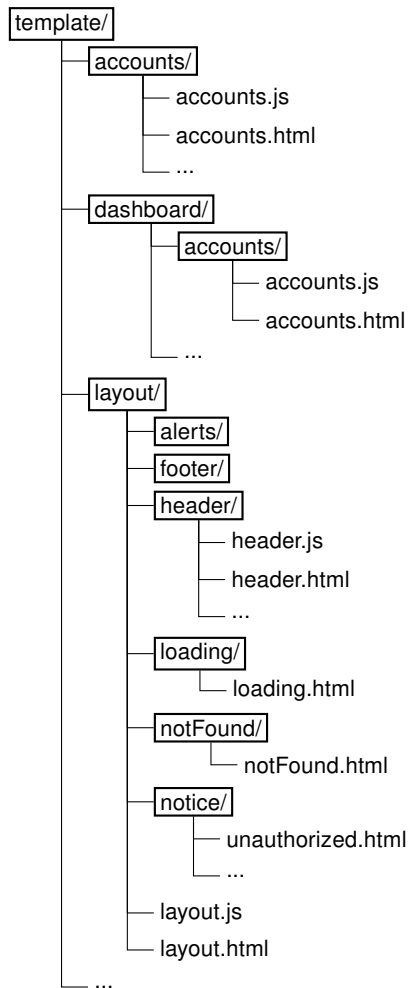


Figura 4.4: Estructura de la carpeta **template** para una aplicación en *Meteor*.

- **collections/.** Acá se crean las variables de todas las *Collections* definidas en la aplicación. Aunque es totalmente posible crear variables propias para el cliente como para el *servidor*; se considera una buena práctica hacerlo de esta manera.
- **hooks/.** Contiene los *hooks* relacionados con los *Collections*. Código escrito en base al *Packages Collection Hooks* [130] .
- **schemas/.** Contiene todos los *Schemas* que se han creado utilizando el *Packages Collection2* [48] .

- **packageGlobals.js**. Contiene todas las variables globales del sistema. En este caso, también son variables que se encuentran en el *environment* del cliente y del *servidor*.
- **routing.js**. Contiene todo el código relacionado con el enrutamiento. El código es desarrollado utilizando el *Packages Router* [111] .

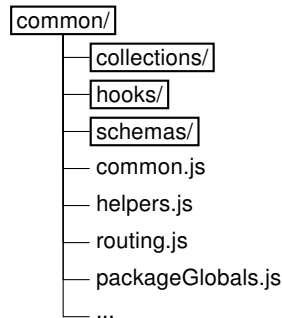


Figura 4.5: Estructura de la carpeta **common** para una aplicación en *Meteor*.

La carpeta **docs** (Figura 4.6) contiene toda la documentación de la aplicación, lo que incluye información sobre *deployment*, *Packages*, *Routing*, *templates*, agradecimientos, convenciones, etc.

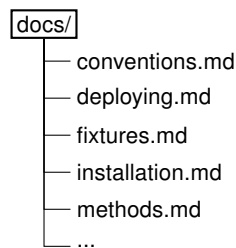


Figura 4.6: Estructura de la carpeta **docs** para una aplicación en *Meteor*.

La carpeta **lib** contiene todas las librerías. Como se observa en Figura 4.7, existe una carpeta y un archivo JSON llamados **bower**. A través del *Packages Bower* [136] se manejan las diferentes librerías requeridas por el sistema.

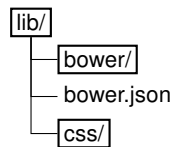


Figura 4.7: Estructura de la carpeta **lib** para una aplicación en *Meteor*.

La carpeta **private** se utiliza principalmente para mantener archivos JSON que contienen *Collections* con la información suficiente para realizar la *localización*, además de guardar los archivos necesarios para test fixtures.

La carpeta **tests** (representada en la Figura 4.9) contiene los archivos para realizar *testing* a la aplicación. Las pruebas se realizan apoyados en el *package Jasmine* [154] .

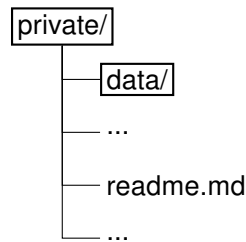


Figura 4.8: Estructura de la carpeta **private** para una aplicación en *Meteor*.

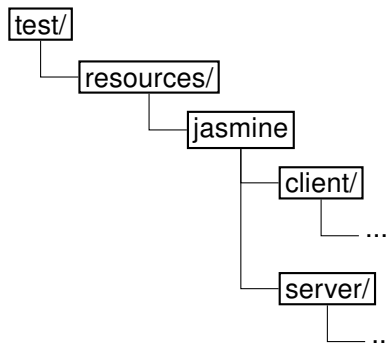


Figura 4.9: Estructura de la carpeta **tests** para una aplicación en *Meteor*.

La carpeta **server**, como su nombre lo indica, contiene la implementación de la aplicación solo visible por el *Server-side*. De los archivos existentes, podemos destacar:

- **methods/**. Se tienen todos los *Methods* definidos.
- **app.js**. Contiene información relacionada con los niveles de *Bunyan* [192] (*package Bunyan* [32]) que se utilizan para hacer *logging*; junto con algunas funciones principales del servidor.
- **browserPolicy.js**. Se definen cuáles son los *websites* permitidos para mostrar contenido. Es importante permitir mostrar contenido solo a sitios confiables, a modo de ejemplo, *websites* maliciosos pueden atacar a los usuarios con ataques *clickjacking*.
- **fixture.js**. Su principal función es ejecutar el código de inicio de test fixtures. Este código es ejecutado solo la primera vez, para poblar la *Base de datos* con las *Collections* requeridas como mínimo, para realizar *testing* y/o poder ejecutar la aplicación. Algo muy relevante que se crea, es el usuario **Administrador**. Esta ejecución se inicia con el evento de inicio de *Meteor*.
- **packageDescription.js**. Entregar la información necesaria para permitir la configuración del *package* agregado.
- **publications.js**. Archivo que contiene todas las publicaciones de *documents*. Cada una de estas publicaciones puede incluir definiciones de seguridad relacionadas con roles para limitar la entrega.
- **security.js**. Archivo que contiene todas las reglas de seguridad para *inserts*, *updates* y *removes* iniciadas desde código inseguro (clientes). De igual manera, existen otras acciones para roles

específicos que tienen permitido realizar. Dichas acciones no necesariamente aparecen aquí si la operación de *Base de datos* es ejecutada desde un método del *servidor*. Las definiciones de seguridad aquí descritas utilizan el *package Security* [138] .

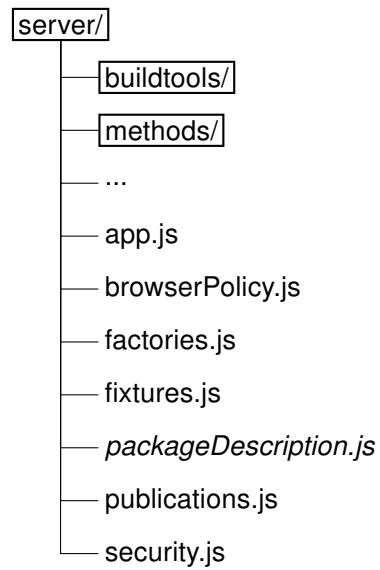


Figura 4.10: Estructura de la carpeta **server** para una aplicación en *Meteor*.

# Capítulo 5

## Solución implementada

Como se explicó en la Sección 4.2, fue necesario desarrollar una arquitectura genérica para el desarrollo de aplicaciones en *Meteor* con el fin de generar código de calidad. Acto seguido se comenzó con la implementación de las siguientes características:

### 5.1. Arquitectura del *Framework*

#### 5.1.1. *Packages*

A continuación se describen las funcionalidades de cada uno de los *Packages* que actualmente existen. La estructura se encuentra en Figura 5.1.

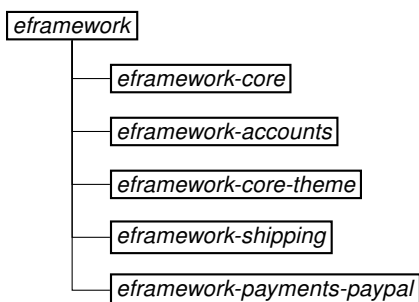


Figura 5.1: Estructura de los *Packages* del *eframework*.

La arquitectura descrita en la Sección 4 fue diseñada para facilitar las implementaciones de cada uno de los *Packages* que se desarrollen en los proyectos de *Meteor*. Cada uno de los *Packages* que se aprecian en la Figura 5.1 hacen uso de dicha arquitectura.

### ***eframework-core***

Corresponde al *core* de la solución. Contiene varias características básicas de una plataforma *e-Commerce*, además de algunas funcionalidades genericas como:

- Manejo de productos.
- Manejo de *sessions*.
- Proceso de *checkout*.
- Manejo de órdenes de Compra.
- Manejo del carro de Compra.
- Manejo de las funcionalidades (*Dashboard*).
- Localización.

Se hablará en detalle de cada una de estas componentes.

### ***eframework-accounts***

Este *Packages* esta enfocado en el manejo de cuentas de los usuarios. Permite entre otras cosas:

- Crear cuentas y editar cuentas.
- Asignar permisos a los usuarios del sistema.
- Recuperación de contraseña.
- etc.

### ***eframework-shipping***

Este *Packages* se encarga de las características relacionadas con *Shipping*, en otras palabras, permite la creación, edición y eliminación de opciones de *Shipping*.

### ***eframework-payments-paypal***

Este *Packages* permite al sistema utilizar una de las varias opciones de pago que permite *PayPal*. En la Sección 5.9.5 se hablará en detalle sobre el método de compra disponible.



## ***eframework-core-theme***

Este *Packages* es la base del *bootstrap Theme* del *Framework* de *e-Commerce*. Este contiene todos los archivos *LESS* utilizados para el proceso que genera los archivos *LESS* personalizados para el *Framework*. Este *Packages* puede ser copiado para crear *Theme* adicionales para el *Framework e-Commerce*.

La implementación de este *Packages* está soportada sobre *bootstrap* [137] permitiendo sencillamente hacer todas aquellas configuraciones que se desean. *bootstrap* [137] permite, entre otras cosas, indicar exactamente cuáles son las componentes de *bootstrap* que se desean utilizar, así como personalizar e importar el *Theme*.

### **5.1.2. Workflows**

Existen una serie de flujos que se pueden experimentar al visitar un sitio *e-Commerce*, entre los cuales podemos destacar:

1. Experiencia de *shopping*.
2. Proceso de una orden
3. *shipping*.
4. Sistema de pago.
5. Servicio al Cliente
6. Retorno de artículos.
7. Entre otros.

Cada uno de los pasos dentro de un *Workflows* puede ser considerado como un estado, el cual cambiará dependiendo de los eventos que estén involucrados. Por lo tanto cada uno de estos *Workflows* puede ser modelado utilizando una máquina de estados finitos. Para el caso particular del *Framework* para *e-Commerce*, se han modelado 2 *Workflows* que se implementan utilizando la librería *JavaScript javascript-state-machine* [112] .



Existe un sistema completo de cuentas de usuarios, el cual permite hacer *login* (Figura 5.6), *logout* (Figura 5.7), creación de cuentas (Figura 5.5), y recuperación de contraseña (Figura 5.8).

### 5.2.1. Creación de cuenta de un usuario

La creación de una cuenta se ha pensado realizando la menor cantidad de acciones posibles. Agregar acciones innecesarias puede significar que los usuarios desistan y dejen el sitio [88]. Una de las acciones innecesarias más común es llenar dos campos de un formulario con la misma información: ejemplos de esto son escribir dos veces el *email* y la contraseña (ver Figura 5.4). Históricamente esto ha sido diseñado para prevenir errores de tipeo. Esta última situación no es el escenario más común, pero aunque efectivamente se haya ingresado un error, siempre se puede recuperar la contraseña usando el *email*. El usuario podrá gastar tiempo extra recuperando su cuenta, pero este escenario es mejor en oposición al tiempo que se ahorrará a miles de usuarios que llenarán el formulario correctamente [74].

The image shows two registration forms side-by-side. The left form, outlined in red, is titled 'Registration' and contains five input fields: 'Username:', 'Email:', 'Confirm Email:', 'Password:', and 'Confirm Password:'. Below these fields is a 'Submit' button. The right form, outlined in green, is also titled 'Registration' and contains three input fields: 'Username:', 'Email:', and 'Password:'. Below these fields is a 'Submit' button.

Figura 5.4: Ejemplos de formularios de creación. El formulario de la izquierda pide confirmación de *email* y contraseña, implicando en un mayor número de campos a llenar.

Este estilo de creación de cuenta está actualmente implementando en sitios populares como *Twitter*, *Dropbox*, *Netflix*, *Instagram* y *LinkedIn*. Los formularios se pueden ver en las Figuras G.1, G.5, G.3, G.2 y G.4 respectivamente.

Finalmente se presenta el formulario de creación de usuario en la Figura 5.5.

Figura 5.5: Formulario de creación de una cuenta.

El formulario de creación tiene ciertas restricciones, las cuales deben cumplirse para la creación exitosa de una cuenta. La Tabla 5.1 resume todas las restricciones del formulario. En la Figura E.1 pueden verse los errores que muestra el formulario para guiar al usuario.

Campo	Requerido	Restricción
<i>Email Address</i>	✓	<ul style="list-style-type: none"> <li>- Debe ser un <i>email</i> válido.</li> <li>- El <i>email</i> no debe pertenecer a un usuario previamente ingresado.</li> </ul>
<i>Password</i>	✓	<ul style="list-style-type: none"> <li>- Debe tener al menos 8 caracteres.</li> <li>- Debe contener al menos un número o símbolo.</li> </ul>

Tabla 5.1: Resumen de restricciones para formulario creación de cuentas de usuario.

### 5.2.2. Autenticarse

Al igual que el formulario de creación de usuario, el formulario de autenticación tiene solo dos campos (Figura 5.6).

Figura 5.6: Formulario de autenticación en la aplicación.

Las restricciones del formulario de autenticación pueden observarse en la Tabla 5.2. Un formulario con errores puede verse en Figura E.2.

Campo	Requerido	Restricción
<i>Email Address</i>	✓	- Debe ser un <i>email</i> válido. - El <i>email</i> debe estar en la base de datos.
<i>Password</i>	✓	- Debe corresponder al <i>email</i> .

Tabla 5.2: Resumen de restricciones para formulario de autenticación de usuario.

### 5.2.3. Cerrar sesión

Una vez autenticado, el formulario de autenticación desaparecerá y en su lugar habrá un botón para cerrar sesión (Figura 5.7).

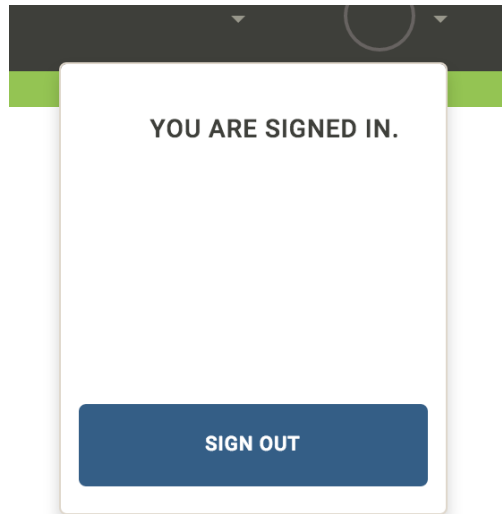


Figura 5.7: Desvincular la cuenta de la aplicación.

#### 5.2.4. Recuperar contraseña

Olvidar las contraseñas es algo recurrente. De hecho, más del 81 % ha olvidado una contraseña utilizada en un sitio web [113]. Esto demuestra lo importante que es contar con mecanismos de recuperación de contraseña. El procedimiento es muy sencillo; se envía el *email* usando el formulario de recuperación de contraseña (Figura 5.8). El sitio genera una contraseña temporal y es enviada al correo.

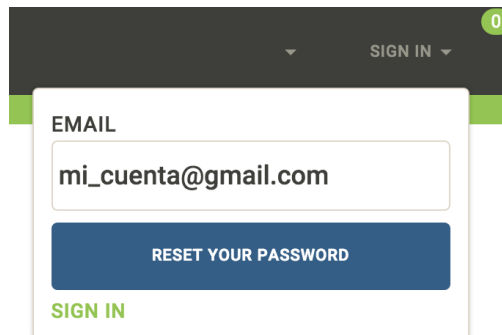


Figura 5.8: Formulario de restauración de contraseña.

Las restricciones del formulario de recuperación de contraseña se ven en la Tabla 5.3.

Campo	Requerido	Restricción
<i>Email Address</i>	✓	- Debe ser un <i>email</i> válido. - El <i>email</i> debe estar en la base de datos.

Tabla 5.3: Resumen de restricciones para recuperación de contraseña.

### 5.2.5. Log in con servicio de autenticación *third-party*

La aplicación permite hacer *login* a través de servicios de autenticación *third-party*, tales como *Facebook*, *Google*, *Twitter*, *GitHub*, entre otros. En la Figura 5.9 se observa la existencia de un servicio *third-party* para la autenticación en la aplicación.

The image shows a mobile application login screen. At the top right, there is a 'SIGN IN' button. Below it is a blue button labeled 'CONFIGURE FACEBOOK'. Underneath is the word 'OR'. There are two input fields: 'EMAIL' and 'PASSWORD', both with asterisks indicating they are required. Below the input fields is a blue 'SIGN IN' button. At the bottom, there are two green links: 'CREATE AN ACCOUNT' and 'RESET PASSWORD'.

Figura 5.9: La aplicación ofrece autenticarse utilizando un servicio *third-party* (*Facebook*) o directamente en el sitio.

Si se hace click sobre el botón *Facebook* de la Figura 5.9, la aplicación irá a una página de *Facebook* y mostrará el formulario para autenticarse.

Es importante destacar que el sistema permite agregar o esconder las opciones *third-party* de la interfaz. En la Figura 5.10 se ven los proveedores *OAuth Facebook*, *GitHub*, *Google*, *Twitter*, respectivamente. Cada uno de estos botones enviara la aplicación a autenticarse en su respectivo sitio oficial.

0

SIGN IN

CONFIGURE FACEBOOK

CONFIGURE GITHUB

CONFIGURE GOOGLE

CONFIGURE TWITTER

OR

EMAIL \*

PASSWORD \*

SIGN IN

CREATE AN ACCOUNT

RESET PASSWORD

Figura 5.10: Formulario de autenticación para utilizar uno de varios servicios *third-party*.

### 5.3. Localización

El inglés no solo ostenta el primer lugar como el idioma más utilizado, *ranking* que considera además del porcentaje de población, la distribución geográfica, sino que además es el idioma más usado en los *websites* alcanzando la cifra de 59,4 %, seguido tímidamente por Rusia con un 5,9 % [166]. Mirando estas cifras se podría concluir que la inclusión de múltiples idiomas en un *site e-Commerce* no representaría ingresos importantes. El potencial económico *online* es de \$45 trillones, de acuerdo a un estudio realizado por *Common Sense Advisory* [152]. Sin una sólida estrategia de localización que considere *websites e-Commerce* con *multi-idiomias*, será un desafío tener al alcance de la mano ese *revenue commerce*. De hecho, el mismo estudio vislumbró que si solo se tiene una versión en inglés del *site*, se estará limitado solo a una tercera parte del total.

¿Cuántos serán los idiomas necesarios para permanecer competitivo en el mundo *online*?. Los investigadores dicen que un mínimo de 14. Las marcas mundiales que aspiran a un 95 % de las billeteras *online* necesitan de 20 idiomas. Si bien es cierto no es siempre posible mantener contenido *online* en 20 idiomas, es difícil negar los beneficios de la localización para regiones específicas alrededor del mundo [152]. Estas son las razones de por qué se considera muy importante la localización en el *Framework* y de por qué se aborda esta característica desde los inicios.

Actualmente la aplicación cuenta con un sistema robusto para la localización, permitiendo nuevos idiomas simplemente agregando un archivo JSON (Figura 5.11).



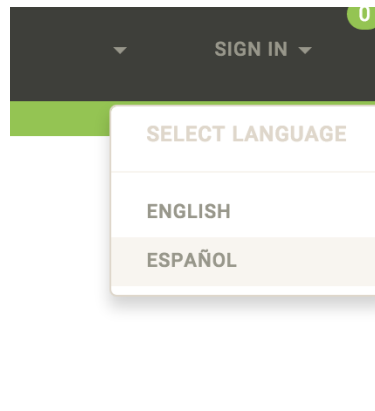


Figura 5.11: Selección de idioma para el *website*.

## 5.4. Themes

La aplicación permite cambiar el *Theme* fácilmente. Una de las partes más importantes del *Framework e-Commerce* corresponde justamente a la customización de la aplicación, dado que esto logra generar identidad. A modo de ejemplo, se mostrarán importaciones de *Themes* en el *Framework*. Para esto se utilizará los *Themes* gratuitos que proporciona *Bootstrap* [139]. Esta tarea es muy sencilla, y basta con agregar el Código 5.1 al final del archivo generado por *bootstrap* [137] llamado **custom.reaction.import.less**.

```
1 @import "http://bootswatch.com/THEME-NAME/bootswatch.less";  
2 @import "http://bootswatch.com/THEME-NAME/variables.less";
```

Código 5.1: Código genérico para importar *Themes* desde *Bootstrap* [139].

En donde **THEME-NAME** corresponde a algún *Theme* de los disponibles en el sitio de *Bootstrap* [139]. En particular, los *Themes* **united**, **journal**, **cerulean**, **superhero**, **paper**, **simplex** y **yeti** se pueden apreciar en las Figuras 5.12, 5.13, 5.14, 5.15, 5.16, 5.17 y 5.18, respectivamente.

## Vintage Desktop



### Tags

[vintage](#) [Desktop](#)

\$150.00

Vintage Desktop  [ADD TO CART](#)

Xerox

When people talk about computer workstations, they are usually referring to a single computer, such as a PC, that is linked to several other single computers in a work environment. Each terminal shares information, equipment like printers, software, or all three items with the other computers in the work group. Other people use the term "workstation" to mean a single PC where an individual works. A vintage workstation may be either one of these -- just a single desktop computer or a PC that is linked to a network.

Figura 5.12: Descripción de un producto utilizando el *Theme united* del sitio *Bootswatch* [139].

## Vintage Desktop



### Tags

[vintage](#) [Desktop](#)

\$150.00

Vintage Desktop  [ADD TO CART](#)

Xerox

When people talk about computer workstations, they are usually referring to a single computer, such as a PC, that is linked to several other single computers in a work environment. Each terminal shares information, equipment like printers, software, or all three items with the other computers in the work group. Other people use the term "workstation" to mean a single PC where an individual works. A vintage workstation may be either one of these -- just a single desktop computer or a PC that is linked to a network.

Figura 5.13: Descripción de un producto utilizando el *Theme journal* del sitio *Bootswatch* [139].



Figura 5.14: Descripción de un producto utilizando el *Theme cerulean* del sitio *Bootswatch* [139] .



Figura 5.15: Descripción de un producto utilizando el *Theme superhero* del sitio *Bootswatch* [139] .



## Vintage Desktop



### Tags

[vintage](#) [Desktop](#)

\$150.00

Vintage Desktop

1

ADD TO CART

Xerox

When people talk about computer workstations, they are usually referring to a single computer, such as a PC, that is linked to several other single computers in a work environment. Each terminal shares information, equipment like printers, software, or all three items with the other computers in the work group. Other people use the term "workstation" to mean a single PC where an individual works. A vintage workstation may be either one of these -- just a single desktop computer or a PC that is linked to a network.

Figura 5.16: Descripción de un producto utilizando el *Theme paper* del sitio *Bootswatch* [139] .



## Vintage Desktop



### Tags

[vintage](#) [Desktop](#)

\$150.00

Vintage Desktop

1

ADD TO CART

Xerox

When people talk about computer workstations, they are usually referring to a single computer, such as a PC, that is linked to several other single computers in a work environment. Each terminal shares information, equipment like printers, software, or all three items with the other computers in the work group. Other people use the term "workstation" to mean a single PC where an individual works. A vintage workstation may be either one of these -- just a single desktop computer or a PC that is linked to a network.

Figura 5.17: Descripción de un producto utilizando el *Theme simplex* del sitio *Bootswatch* [139] .



Figura 5.18: Descripción de un producto utilizando el *Theme yeti* del sitio *Bootswatch* [139] .

## 5.5. La información persiste

Aunque se cambie de vista, al regresar regreso, la información debería seguir viéndose. Por ejemplo, al llenar un formulario y dejar la página, al regresar, este formulario estará igual.

## 5.6. Productos

### 5.6.1. Vista global de productos

La interfaz inicial de la aplicación corresponde a la vista de todos los productos que el sistema tiene. En la Figura 5.19 se pueden observar 4 productos.

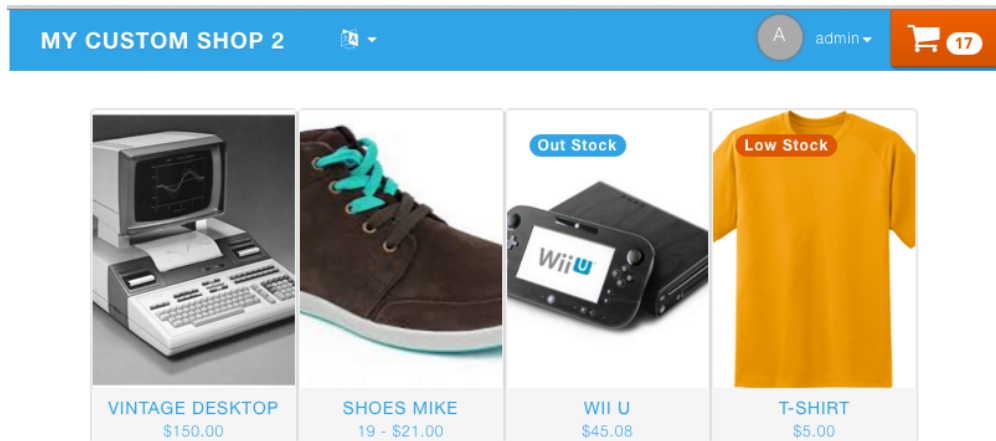


Figura 5.19: Vista global de los productos. Productos con problemas de *stock* dan información.

De la interfaz, se observa que hay un producto *Out of Stock* y otro producto *Limited Stock*. Esto ocurre porque la plataforma tiene un sistema de *tracking* de productos la cual puede ser modificada desde la vista Edición de un producto.

Esta vista depende de los permisos del usuario que ha ingresado. Si el usuario tiene permisos de edición de productos, tendrá la posibilidad de ver todos los productos que existen en el sistema, incluso esos que no estan publicados. En el caso de un usuario sin permisos de edición, solo tendrá visibilidad de aquellos productos que han sido publicados.

### 5.6.2. Descripción de un producto

La aplicación permite consultar la descripción de un producto. Dicha interfaz puede observarse en la Figura 5.18. Se puede apreciar algunas de las carecterísticas de las que ya dispone cada producto.

- Un título del producto.
- Un subtítulo para dar un resumen.
- Un espacio para la foto del producto.
- Un lugar para la descripción del producto.
- Un precio.
- *Tags*.

### 5.6.3. Creación de un producto

Cuando se pensó en la interfaz de creación/edición de un producto, la base era tener una interfaz idéntica a la vista de Descripción de un producto, y por lo tanto, interactuar con los diferentes elementos

directamente en la interfaz de vista. Cuando se ingresa a la interfaz de vista de un producto con los permisos adecuados, todas las componentes de la vista del producto cambian a editables.

La interfaz de creación de un producto se observa en la Figura 5.20. Es importante destacar la ayuda que brindan los *place-holders* para guiar al usuario.

This product is not available to buy.

# Product title

Image placeholder: DROP FILE TO UPLOAD

Price: \$0.00

Quantity: 1 ADD TO CART

Product vendor

Add a product description

Tags

Variants

Weight	Quantity	Price
		0

Figura 5.20: Interfaz de creación de un producto.

Para determinar los campos utilizados en Figura 5.20, de inspiración se utilizó principalmente la interfaz de creación de productos desarrollada para el *website* de *Shopify* Figura F.5. *Shopify* es un servicio que proporciona todas las herramientas necesarias para desarrollar una tienda *online*, razón por la cual cuenta con una gran cantidad de herramientas, las cuales no pueden ser abordadas en su totalidad en un contexto de una memoria. A continuación se explica cada una de las componentes utilizadas en el *Framework* obtenidas desde *website Shopify*:

**Title** Título principal del producto (Figura 5.21). Idealmente debe ser conciso porque se utiliza tanto en la interfaz de descripción del producto como en la pantalla de búsqueda de productos. El título aparece claramente en la sección principal de la interfaz de creación Figura F.6.

## Vintage Desktop

Figura 5.21: Título principal del producto. Aparece también en la lista de búsqueda de productos.

**Description** Permite agregar una descripción del producto (Figura 5.22). Se encuentra en la sección principal al igual que el *Title* Figura F.6. Esta sección es particularmente importante (junto con *Multimedia*), porque ayudan en la decisión de compra del producto. Es importante agregar descripciones que expliquen el producto, que entreguen *lindas* palabras que fomenten la compra, no una lista de términos técnicos [67].

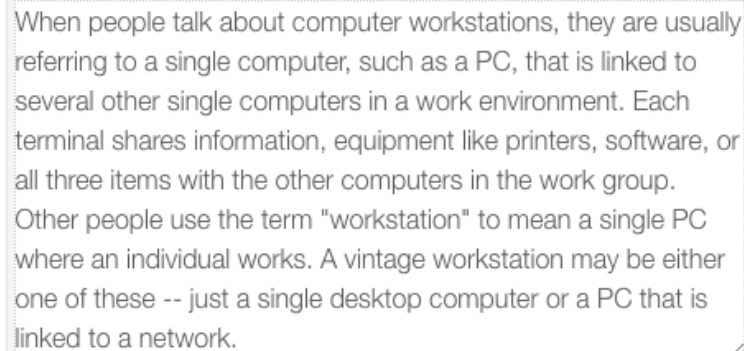
A screenshot of a text input field for a product description. The text inside the field reads: "When people talk about computer workstations, they are usually referring to a single computer, such as a PC, that is linked to several other single computers in a work environment. Each terminal shares information, equipment like printers, software, or all three items with the other computers in the work group. Other people use the term 'workstation' to mean a single PC where an individual works. A vintage workstation may be either one of these -- just a single desktop computer or a PC that is linked to a network." The text is in a light gray font on a white background with a thin border.

Figura 5.22: Campo para la descripción del producto.

**Vendor** Permite agregar un *Vendor* a la descripción del producto. Aparece en la sección de organización de *Shopify* Figura F.10.

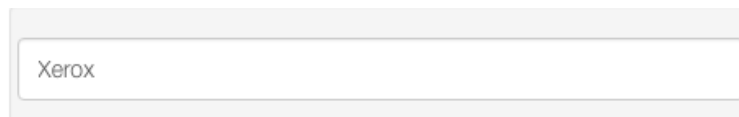
A screenshot of a text input field for a vendor name. The text inside the field is "Xerox". The field has a light gray border and a white background.

Figura 5.23: Campo para el *Vendor* del producto.

### Options

**Multimedia** Al igual como en *Shopify* Figura F.7, esta sección permite agregar imágenes a través de 2 métodos: *drag & drop* y selección, utilizando el explorador de archivos (Figura 5.24). Las imágenes siempre han sido lo primero que el cliente ve. La mayoría de los *websites* ubican las imágenes en la parte superior izquierda de la página [67]. A diferencia de los compraderos en las tiendas, las imágenes de los productos son la única forma real de determinar si un producto se ajusta a sus necesidades y a sus preferencias. Es una manera poderosa de darles vitrina a los productos. Por lo tanto es muy importante proveer de imágenes de buena calidad entre otros detalles [67]. Esto explica el tamaño de imagen que se acepta en la plataforma.





Figura 5.24: Sección de las imágenes del producto.

**Tags** Provenientes de la sección organización Figura F.10, permite la categorización utilizando palabras clave relevantes.

### Tags

Figura 5.25: Campo para las *Tags* del producto.

También se tuvo en consideración la siguiente información al momento de desarrollar esta interfaz:

- Los visitantes prefieren imágenes de los productos a la izquierda y la descripción a la derecha [67].
- Es improbable que un usuario haga *scroll* (en el caso de interfaces *desktop*) para ver más contenido en la página [67].
- Usuarios nunca o muy rara vez leen toda la información disponible especialmente cuando hay muchísimo contenido. Usualmente se ignora la parte inferior o recorren sin poner atención a

todas las secciones. E incluso si hacen *scroll*, ellos no necesariamente leen o ponen atención a todo el contenido inferior de la página [67].

- Agregar solo la información importante del producto que se venderá [67].
- Proveer un espacio razonable y no sobreutilizar colores [67].
- Evitar el uso de *clicking* si se desea que el usuario lea [67].

De los campos que se observan en la Figura 5.20, *Title*, *Page title*, *Vendor*, *Description*, *Tags* y *Details*, tienen *Feedback Reactivo*. Las restricciones en la creación de un producto se encuentran en la Tabla 5.4.

Campo	Requerido	Restricción
<i>Title</i>	✓	- Debe tener al menos un caracter.
<i>Page title</i>		
<i>Vendor</i>		
<i>Options</i>	✓	
<i>Description</i>		
<i>Multimedia</i>		
<i>Tags</i>		

Tabla 5.4: Resumen restricciones para formulario creación de un producto.

Como se aprecia en la Tabla 5.4, el campo *Details* no es requerido, sin embargo, cada *Detail* en realidad es un objeto compuesto cuyos campos sí son obligatorios (Tabla 5.5).

Campo	Requerido	Restricción
<i>Anónimo key</i>	✓	- Debe tener al menos un caracter.
<i>Anónimo Value</i>	✓	- Debe tener al menos un caracter.

Tabla 5.5: Resumen restricciones para formulario creación de *Details*.

Si se enfoca la atención en los campos de la Figura 5.26 se observa que ellos se encuentran repartidos entre las secciones *Pricing* (Figura F.11), *Inventory* (Figura F.8), *Shipping* (Figura F.12) y *Variants* (Figura F.13).

**Label** Corresponde al nombre de la *Variant* del producto que se utiliza.

**Option** Se utiliza para identificar cada una de las *Variants* al momento de agregarlas al carro.

**Weight** Corresponde al peso (masa), del producto que se está ofreciendo. Este valor se utiliza para *shipping*, razón por la cual aparece esta componente en la sección *Shipping* del formulario de *Shopify* Figura F.12.

**Tracking** Permite tomar acciones dependiendo del inventario que se tenga. Ver Figura F.8.

**Deny** Al igual que en la sección *Inventory* de *Shopify* (Figura F.8), esta componente aparece cuando se ha seleccionado la opción de *Tracking*. Esta opción evita que los clientes sigan comprando el producto una vez que se ha terminado el inventario.

**Taxable** Aplica Cargos de impuesto al producto.

**Quantity** Cantidad de elementos para este producto en particular.

**Price** Precio de este producto en particular.

## Variants

Label  
T-shirt M

Weight: 3      Quantity: 10      Price: 5

Taxable  
 Track this product's inventory  
 Deny when out of stock

Warn @  
14

Figura 5.26: Sección de las opciones del producto.

Campo	Requerido	Restricción
<i>Label</i>	✓	- Debe tener al menos un carácter.
<i>Weight</i>	✓	- Número racional mayor o igual a 0.
<i>Quantity</i>	✓	- Número entero.
<i>Price</i>	✓	- Número racional mayor o igual a 0.
<i>Deny</i>	✓	- Boolean.
<i>Warn</i>		- Número racional mayor o igual a 0.
<i>Taxable</i>	✓	- Boolean.
<i>Tracking</i>	✓	- Boolean.

Tabla 5.6: Resumen restricciones para formulario creación de *Options*.

#### 5.6.4. Edición de un producto

Al igual que para la creación de productos, la edición requiere de permisos especiales. Si se cuenta con dichos permisos especiales, la interfaz de la vista de un producto inmediatamente permite editar los campos. Para editarlos, simplemente es necesario seleccionar la componente con el *mouse* y hacer un *click*. La componente cambiará su aspecto para dar *feedback* al usuario de que está seleccionada dicha componente y que puede realizar sobre él ediciones. En las Figuras 5.28 y 5.29 es posible apreciar la componente que se ha seleccionado para editar.

**T-Shirt**

**\$5.00**

T-Shirt  **ADD TO CART**

Mike

Nice T-Shirts  
Best Prices

DROP FILE TO UPLOAD

**Tags**

**Variants**

Label  
T-shirt M

Weight	Quantity	Price
<input type="text" value="3"/>	<input type="text" value="10"/>	<input type="text" value="5"/>

Taxable


Track this product's inventory

Deny when out of stock

Warn @

Figura 5.27: Interfaz de la edición de un producto.

# T-Shirt



DROP FILE TO UPLOAD

**\$5.00**

T-Shirt 1 ADD TO CART

Mike

Nice T-Shirts  
Best Prices

### Tags

### Variants

Weight	Quantity	Price
3	10	5

Taxable

Track this product's inventory


Deny when out of stock

Warn @

14

Figura 5.28: Campo descripción seleccionado para edición.

## T-Shirt



DROP FILE TO UPLOAD

**\$5.00**

T-Shirt ▼

ADD TO CART

Mike

Nice T-Shirts  
Best Prices

**Tags**

**Variants**

Label

T-shirt M

Weight	Quantity	Price
3	10	5

Taxable

Track this product's inventory

Deny when out of stock

Warn @

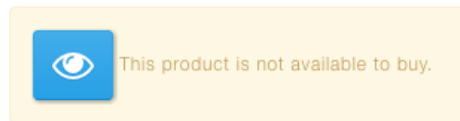
14

Figura 5.29: Campo título seleccionado para edición.

Es importante mencionar que la edición de producto es *reactive*, por lo tanto todos los cambios que se realicen serán inmediatamente visibles por parte de todos los usuarios que estén mirando la descripción del producto incluso sin la necesidad de que el usuario haga un *refresh* del *website*.

### 5.6.5. Visibilidad de un producto

Todo producto válido puede ser visible para todos los usuarios (visibilidad global), o solo visible para aquellos que tienen permiso de edición sobre dicho producto. Esta importante característica se encuentra disponible en el servicio de *Shopify* (Figura F.16). En la Figura 5.30 se ve el caso particular de un producto que está en proceso de creación (no tiene título, el cual es requerido).



Product title

Figura 5.30: Producto con visibilidad limitada. Notar que no tiene título, lo que implica que no es un producto válido aún.

Después de llenar todos los campos mínimos requeridos para la creación de un producto, es posible presionar el *Link you can make it visible* y dar visibilidad global al producto. En el caso que no estén llenos todos los campos requeridos, entonces se genera un *auto-foco* del primer campo no completado requerido. Suponiendo que se intenta dar visibilidad al producto de la Figura 5.30, se hará un *auto-foco* del campo título, el cual puede observarse en la Figura 5.31; además de mantener el estado de visibilidad (no visible para todos). Esto ocurre para mostrar claramente al usuario qué está sucediendo [88,91].

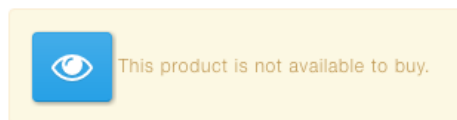


Figura 5.31: Resultado tras dar visibilidad a un producto sin título. Se genera un *auto-foco* en el campo título, dado que este es requerido y el estado de visibilidad del producto continúa igual.

Si todos los campos requeridos de un producto están completos, y se presiona el *Link you can make it visible*, se tendrá el resultado de la Figura 5.32; en donde se observa cómo el texto superior cambió. Ahora aparece un *Link* con el texto *make invisible*, para dar *feedback* al usuario de lo que ha ocurrido [88,90].

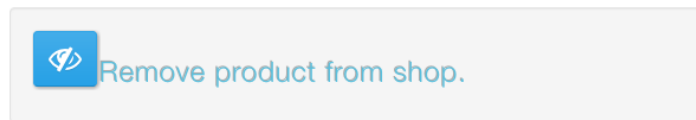


Figura 5.32: Producto con visibilidad global. Con el botón se oculta el producto.

## 5.7. Carro Compra

El carro de compra agrupa todos los productos seleccionados para efectuar la compra. Actualmente del carro de compra se pueden desprender dos acciones: Agregar elementos al carro y Consultar los elementos del carro.

### 5.7.1. Agregar elementos al carro

El proceso de agregar elementos es muy sencillo, simplemente hay que ir a la vista de un producto, ir donde esta el botón *add to cart*, seleccionar la cantidad de productos que se desean agregar y finalmente apretar el botón. En la Figura 5.33 se han seleccionado 4 productos para agregar al carro. Este diseño ha sido inspirado también por el *website* de *Shopify* (Figura F.15).

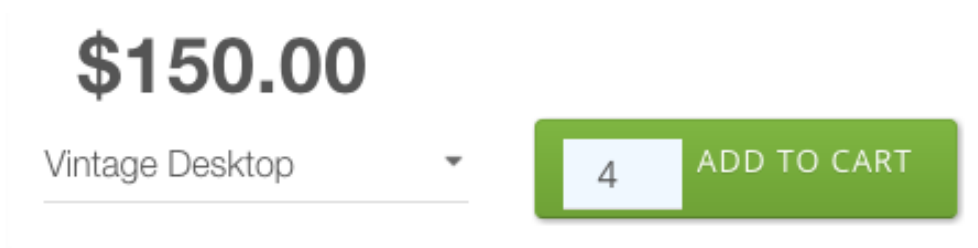


Figura 5.33: Botón para agregar productos al carro. En la figura se han seleccionado 4 productos Vintage Desktop para agregar al carro de compra.

Es importante agregar que la opción *Vintage Desktop* es la única opción disponible del sistema. Actualmente la plataforma no permite el ingreso de *Variants* para el mismo producto. Solo se construyo de esta manera para dar soporte cuando dicha funcionalidad sea implementada.

Mostrar la confirmación de objeto agregado y permanecer en la misma página es algo fundamental. Las personas no han pedido moverse a otra página, de esta manera no hay sorpresas y se mantiene la buena experiencia. Además es posible que consideren agregar más productos al carro antes de *checkout* [63]. Consideranto esto, se muestra una notificación en la parte superior de la pantalla sobre el ingreso del nuevo producto al carro sin abandonar la página actual (Figura 5.34).



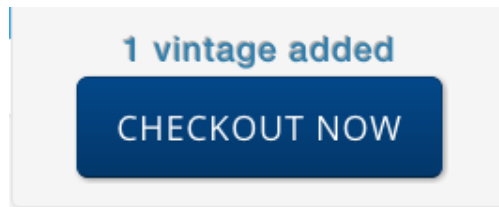


Figura 5.34: *Feedback* del nuevo producto agregado al carro.

Usualmente el *agregar al carro*, el botón con la acción final *website e-Commerce* cae en una de estas dos opciones: no está bien diseñado o no está ubicado estratégicamente para el uso del cliente [92]. Por esta razón se creó un botón evidente, claro y prominente, en comparación a otras funcionalidades dentro de la misma página. Un ejemplo de lo que no se debe hacer se observa en la Figura 5.35.



Figura 5.35: Mala práctica en el diseño del botón para agregar elementos al carro. Entre otros detalles el botón no tiene un diseño que lo distinga del botón agregar a la lista de deseos.

Una vez agregado al carro, el número de elementos presentes se actualiza. Este número es visible en la parte superior izquierda de la interfaz (Figura 5.36).



Figura 5.36: Icono que muestra la cantidad de elementos que tiene el carro. El icono del carro es un botón para la vista del carro.

El estado del carro de compra se guarda en la base de datos, esto quiere decir que aunque haga *logout*, el estado seguirá vigente para la próxima vez que haga *Log in*.

## 5.7.2. Consultar los elementos del carro

Existen dos claves para un buen despliegue de contenido en un carro de compras [63]:

- **Claridad.** Que sea sencillo y obvio entender qué es lo que se tiene en el carro y su costo final, incluyendo *shipping* e impuestos. Costos sorpresivos producen que los clientes abandonen los

carros de compra [63].

- **Control.** Es sencillo realizar cambios [63]. Esto incluye actualización de cantidad y eliminar productos [64].

El icono del carro de compras que se aprecia en la Figura 5.36 es a su vez un botón, el cual muestra todos los elementos agregados al carro en una lista horizontal. Además muestra un resumen de la compra, inspirado en el que utiliza el sitio de *Shopify* (Figura F.4), mostrando la siguiente información:

- Subtotal.
- Costos de envío.
- Impuestos
- Total

Todos estos detalles, que se plasman en la Figura 5.37 fueron diseñados con el fin de mantener **Claridad**.

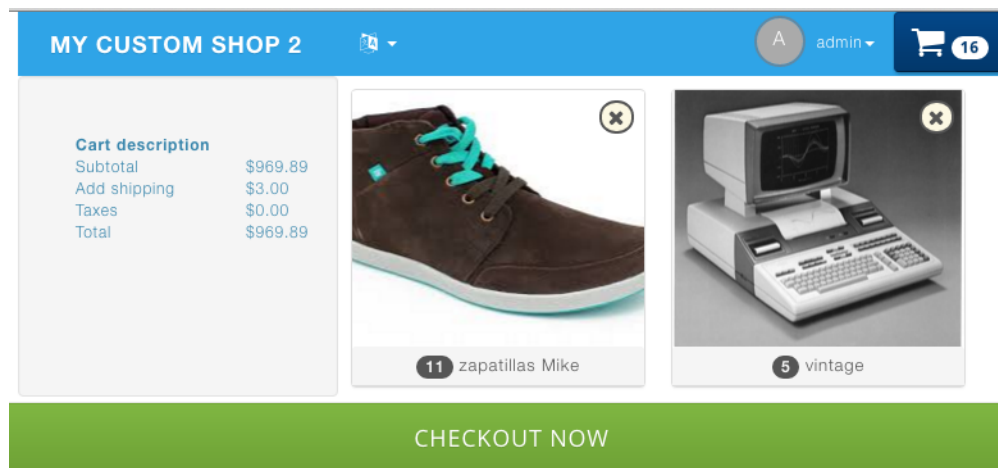


Figura 5.37: Información global del carro. Tiene un resumen del costo total de los elementos seleccionados.

Cada uno de los diferentes elementos agregados al carro pueden ser eliminados simplemente presionando sobre la [x] que tiene asociada cada producto (Figura 5.38). Esto permite tener **control** sobre el contenido del carro. Es importante recordar como concepto general de diseño de interfaces, que es inevitable que las personas cometan errores, o incluso cambien de parecer durante algún procedimiento. Es necesario entonces permitir correcciones [88].



Figura 5.38: Vista básica de un producto seleccionado. La [x] permite eliminar ese producto del carro de compra.

El botón *Checkout Now* que se ubica en la parte inferior de la vista Consultar los elementos del carro nos envía a la interfaz de *Checkout*.

De momento no es posible actualizar la cantidad de un determinado producto dentro del carro. En el caso particular del producto de la Figura 5.38, solo se pueden eliminar los 4 productos simultáneamente.

Otra característica interesante que ha sido integrada, es de notificar al cliente cuando uno de sus productos esta bajo en *stock* (*Limited Stock*) y sin *stock* (*Limited Stock*).

En la Figura 5.39 se aprecia como el botón del carro de compras a cambiado su color en relación al que aparece en la Figura 5.37.

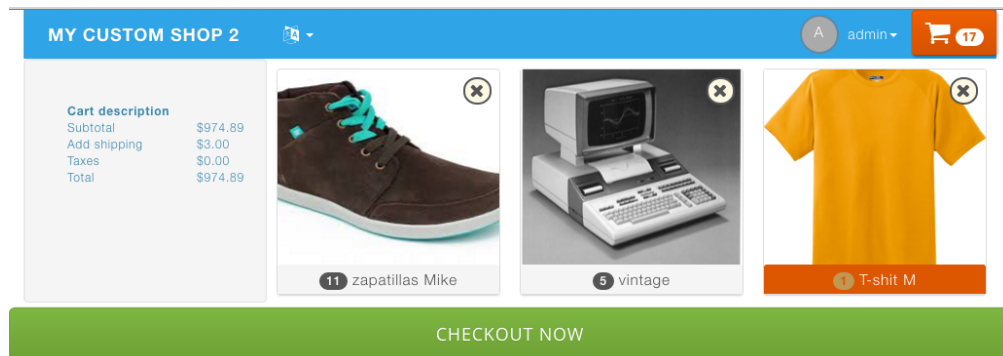


Figura 5.39: El botón del carro a cambiado de color. Esta informando que un producto esta con *Limited Stock*.

También se aprecia que el producto *T-shirt* esta dentro de un contenedor de color distinto en relación a los otros dos productos. Esto ocurre para identificar que ese producto es aquel que se encuentra con *Limited Stock*. En la Figura 5.40 Se aprecia mejor el producto *T-shirt*.



Figura 5.40: El contenedor del producto *T-shirt* esta en amarillo. Esto indica que el producto esta con *Limited Stock*.

## 5.8. Profile

La *UI* de profile permite ver y editar información relacionada directamente con el usuario. Esta información está dividida en dos partes: actualización de contraseña y libreta de direcciones.

### 5.8.1. Actualizar contraseña

El formulario de actualización de contraseña permite al usuario cambiar tanto como desee la contraseña que tiene actualmente. Después de todo, el cambio periódico de una contraseña es una medida de seguridad básica [201].

Como se puede observar en la Figura 5.41, el formulario de actualización de contraseña cuenta con dos campos obligatorios para el proceso. El primero corresponde al campo de solicitud de contraseña actual (a modo de confirmación de que la persona que está actualizando la contraseña, sea el dueño de la cuenta) y el segundo campo, para ingresar la nueva contraseña. Al apretar el botón de confirmación esta se cambia. Este formulario fue diseñado utilizando las mismas consideraciones que para el formulario Creación de cuenta de un usuario (5.2.1).

Existen diversas situaciones en donde el formulario notifica sobre errores que se cometen al

Figura 5.41: Formulario de actualización de contraseña.

momento de llenar el formulario. Por ejemplo, al intentar enviar el formulario sin información ambos campos tienen errores y pueden observarse en la Figura E.3.

Campo	Requerido	Restricción
<i>Current Password</i>	✓	Debe ser la contraseña actual.
<i>Password</i>	✓	- Debe tener al menos 8 caracteres. - Debe tener al menos un número o símbolo.

Tabla 5.7: Resumen restricciones formulario edición de contraseña (Figura E.3 y Figura E.4).

Tanto en la Figura E.3 y en la Figura E.4 se puede observar que la nueva contraseña no cumple con estos requerimientos. Los requerimientos solicitados no responden a ninguna política de seguridad actual. Simplemente se eligieron esos requerimientos para evitar contraseñas aún más débiles que las que se pueden generar.

Existe un último error, y corresponde simplemente cuando el usuario no logra identificarse correctamente, o lo que es equivalente, la contraseña actual ingresada no es correcta. En la Figura E.5 puede observarse el mensaje que el formulario entrega en este caso.

### 5.8.2. Libreta de direcciones

Corresponde a todas las direcciones que el usuario ha agregado al sistema con el fin de utilizarlas para la recepción del o los productos y de las facturas. Cada uno de los parámetros del formulario para una dirección fue elegido a partir del formulario de direcciones que utiliza *Amazon* (Figura F.1).

Como buena práctica, se debe utilizar la dirección de *Shipping* como dirección *default* para la dirección de facturación. Los clientes típicamente ordenan productos para la dirección del hogar. Así que por *default*, se debe utilizar la misma dirección para ambos casos [102]. Dejando la dirección de facturación por *default* como la de *Shipping*, el proceso de *checkout* tendrá algunos campos menos, haciendo el proceso menos intimidante para los clientes. Los usuarios también disminuyen el riesgo de cometer un error al escribir sus direcciones solo una vez; ellos no se moverán a través del formulario de manera tan rápida y de existir errores, el usuario solo tendrá que resolverlos una vez [102].

Teniendo en consideración estos detalles, se desarrolla el formulario que se observa en la Figura 5.42.

**Address Book**

Country  
(Select One) ▼

---

Full name

---

Address 1 Address 2

---

Postal City Region

---

Phone

---

Set as shipping address?

Set as billing address?

SAVE AND CONTINUE
CANCEL

Figura 5.42: Formulario para agregar una nueva dirección.

Este formulario cuenta con las restricciones descritas en la Tabla 5.8, y por lo tanto muestra mensaje de error según corresponda.

Campo	Requerido	Restricción
<i>Country</i>	✓	Debe ser una de las alternativas.
<i>Full Name</i>	✓	
<i>Address 1</i>	✓	
<i>Address 2</i>		
<i>Postal</i>	✓	
<i>City</i>	✓	
<i>Region</i>	✓	Para ciertos países, existen alternativas definidas.
<i>Phone</i>	✓	
<i>Shipping address</i>	✓	Boolean.
<i>billing address</i>	✓	Boolean.

Tabla 5.8: Resumen restricciones formulario para libreta de direcciones.

Notar que en la parte superior de la Figura 5.42 está el botón que permite ingresar nuevas direcciones. Cada una de estas nuevas direcciones es agregada a la lista que aparece en la Figura 5.43, lugar en donde pueden ser editadas y eliminadas de acuerdo a las necesidades del cliente. En caso de editar alguna dirección, esta debe cumplir las mismas restricciones impuestas para el caso de la creación de una dirección (Tabla 5.8).

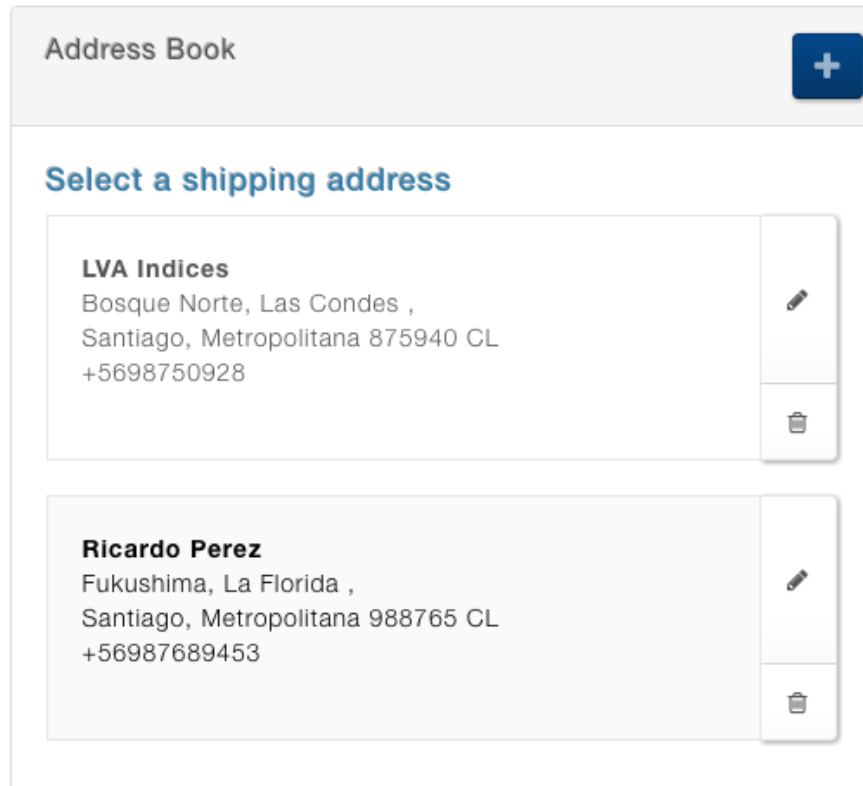


Figura 5.43: Lista con todas las direcciones configuradas del usuario. Cada una de estas puede ser editada o eliminada con los botones que se encuentran en el costado derecho.

Es importante agregar que en la situación en que el usuario no tenga configurada ninguna dirección, aparecerá por *default* el formulario de creación. Esto se observa en la Figura 5.44.

**Address Book**

Country  
Chile

Full name  
Ricardo Soto

Address 1  
Fukushima

Address 2

Postal  
748375

City  
Santiago

Region  
Metropolitana

Phone  
+5696472653

Set as shipping address?

Set as billing address?

SAVE AND CONTINUE

Figura 5.44: Formulario para agregar una nueva dirección, aparece en caso de no existir configurada ninguna previa. Notar que no existe un botón para cancelar dado que en este contexto no tiene sentido.

## 5.9. *Dashboard*

Corresponde al administrador de características de la aplicación. Básicamente muestra todas las funcionalidades personalizables que existen en la plataforma. La información que se muestra de cada componente corresponde a:

- Nombre de la componente a agregar.
- Un icono.

En la Figura 5.45 se observa el *Dashboard* 5 elementos configurables de la aplicación.

Existen ciertos *Packages* que tienen la opción de ser deshabilitados. Esta opción es especialmente útil para agregar o quitar ciertas funcionalidades al *website*. A modo de ejemplo, en la Figura 5.46 se



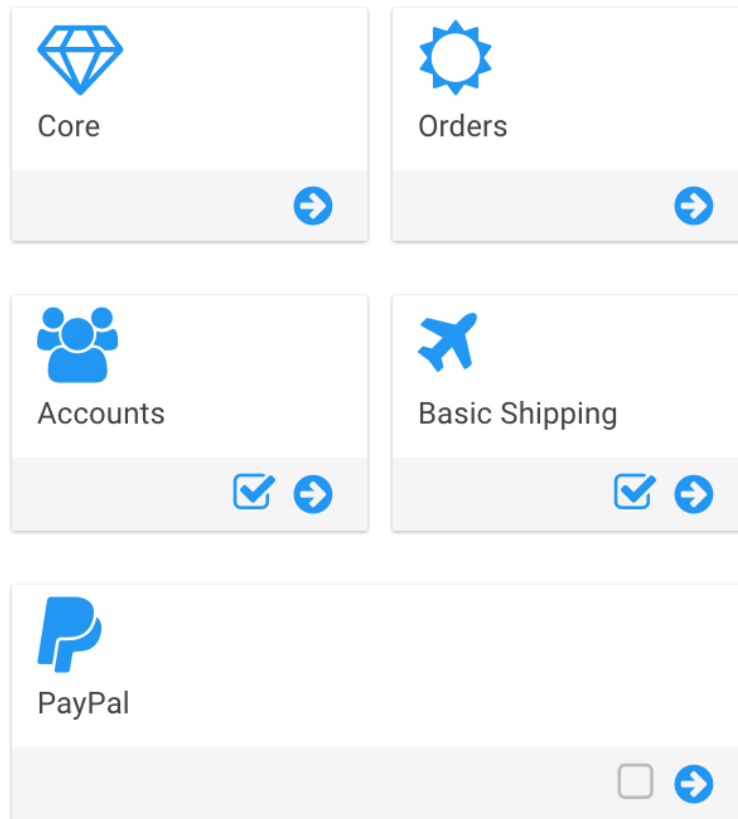


Figura 5.45: *Dashboard* con elementos configurables de la aplicación.

observa que el *package* de *PayPal* está deshabilitado, en contraste con el *package* de *Shipping*. En el caso de que existieran más opciones de pago como *Google Wallet* y *Amazon Payments*, se podrían ocultar los métodos sencillamente utilizando este método.

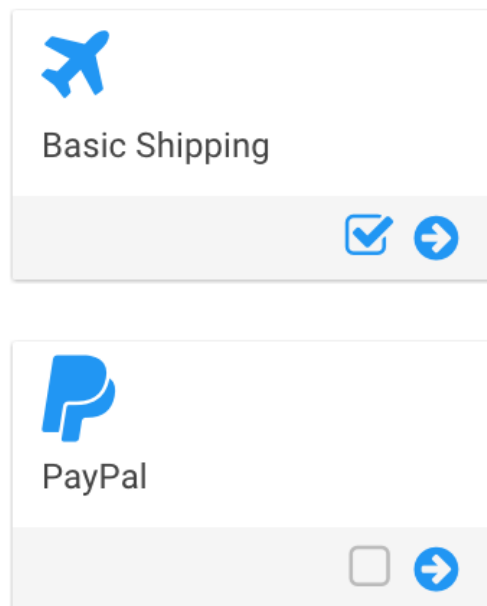


Figura 5.46: *Dashboard* con elementos configurables de la aplicación.

### 5.9.1. E-Commerce Framework Core

Corresponde a la customización de los elementos relacionados directamente con la tienda. La información se ha jerarquizado en diferentes paneles de contenido plegable (conocidos como menú acordeón), los cuales se observan en la Figura 5.47.



Figura 5.47: Menú general de *E-Commerce Framework Core*.

Cada uno de estos elementos tiene información configurable; además del *feedback* habitual correspondiente a la información entregada de validación de cada campo, tiene una ventana emergente para indicar si la actualización del formulario fue o no exitosa. La razón de esta información extra es muy sencilla, si no existieran estos mensajes emergentes, estaría la ambigüedad de si el resultado fue exitoso o no dado que no hay ningún otro elemento gráfico con el cual se podría inferir el resultado. Por ejemplo, el formulario no se *limpia*, el formulario no se cierra, no se agrega un nuevo elemento a una lista, etc.

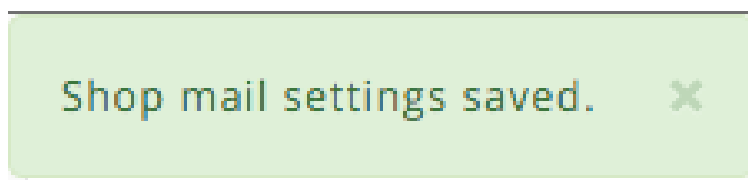


Figura 5.48: Mensaje de confirmación de éxito en la actualización del formulario. Este mensaje se esconde después de un breve intervalo de tiempo.

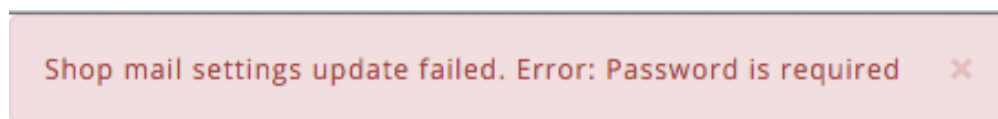


Figura 5.49: Mensaje para informar sobre un error en el proceso de actualización del formulario.

En el caso de ser exitoso, aparece un mensaje, el cual eventualmente desaparecerá después de un breve intervalo de tiempo (Figura 5.48). Los mensajes de error persisten en el tiempo. Por lo tanto se deben cerrar para que desaparezcan (Figura 5.49).

## Panel *General*

El panel general está formado por dos formularios; el primero solo contiene un campo y es un checkbox para permitir que un usuario *Guest* pueda realizar un *checkout* (Figura 5.50). Un usuario es considerado *invitado*, cuando no tiene su cuenta confirmada.

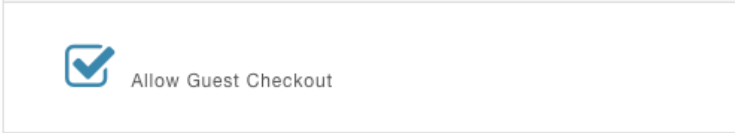
A rectangular box containing a blue checkmark icon on the left and the text "Allow Guest Checkout" to its right.

Figura 5.50: Componente para permitir *checkout* desde una cuenta *Guest*.

Este formulario tiene la particularidad que se envía automáticamente al cambiar su estado. El otro formulario está relacionado con la información más general de la empresa (Figura 5.51).

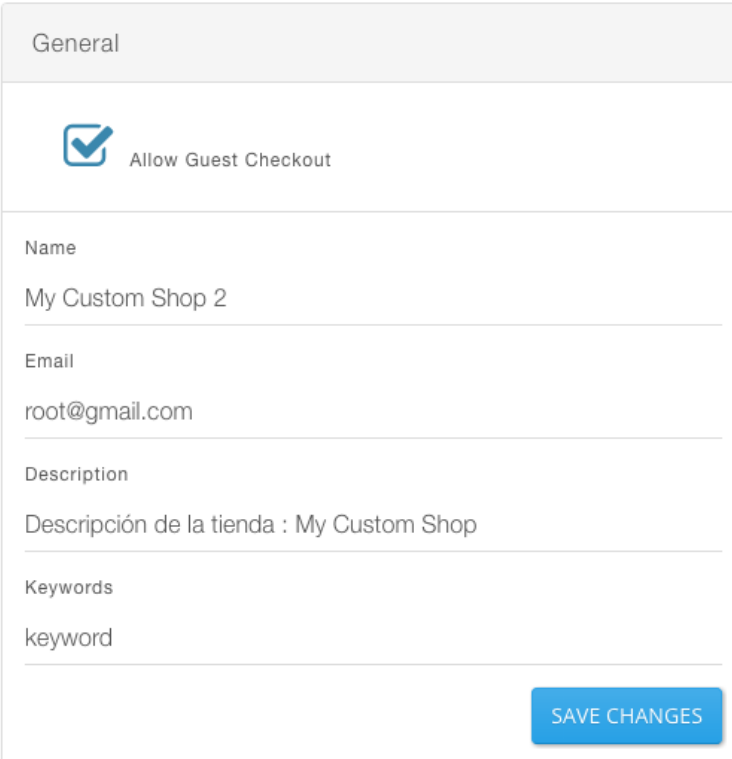
A form titled "General" with a header bar. Below the header is a blue checkmark icon and the text "Allow Guest Checkout". The form contains several input fields: "Name" with the value "My Custom Shop 2", "Email" with the value "root@gmail.com", "Description" with the value "Descripción de la tienda : My Custom Shop", and "Keywords" with the value "keyword". A blue "SAVE CHANGES" button is located at the bottom right of the form.

Figura 5.51: Formulario de información general de la tienda.

Campo	Requerido	Restricción
<i>Name</i>	✓	
<i>Email</i>		Debe ser un email válido.
<i>Description</i>		
<i>Keywords</i>		

Tabla 5.9: Restricciones formulario *General*.

Este formulario solo tiene un campo obligatorio, y corresponde a *Name*. Al igual que otros formularios, dicho campo se destaca de color rojo, además de entregar un mensaje a modo de identificar el error (Figura E.6).

### Panel *Address*

Este formulario se utiliza para agregar información física de la tienda, como algún teléfono de contacto.

The image shows a web form titled "Address" with the following fields and values:

- Company:** DCC
- Full name:** Universidad de Chile
- Address 1:** Beauchef
- Address 2:** Blanco Encalada
- City:** Santiago
- Region:** Metropolitana
- Postal:** 870000
- Country:** CL
- Phone:** +5696745367

A "Save Changes" button is located at the bottom right of the form.

Figura 5.52: Formulario de la dirección física de la tienda.

En relación con los campos del formulario, se tienen las siguientes restricciones (Tabla 5.10):

Campo	Requerido	Restricción
<i>Full name</i>	✓	
<i>Address 1</i>	✓	
<i>Address 2</i>		
<i>City</i>	✓	
<i>Region</i>	✓	
<i>Postal</i>	✓	
<i>Country</i>	✓	
<i>Phone</i>	✓	

Tabla 5.10: Restricciones formulario *Address*.

### Panel *Mail*

Corresponde a la información básica necesaria para configurar un servicio de *email*. Agregar un *email* no es requerido, pero si se configura el servicio deben agregarse todos los campos (Tabla 5.11).

Mail

**Username**

**Password**

**Host**

**Port**

**Save Changes**

Figura 5.53: Formulario con la información de la configuración del Mail.

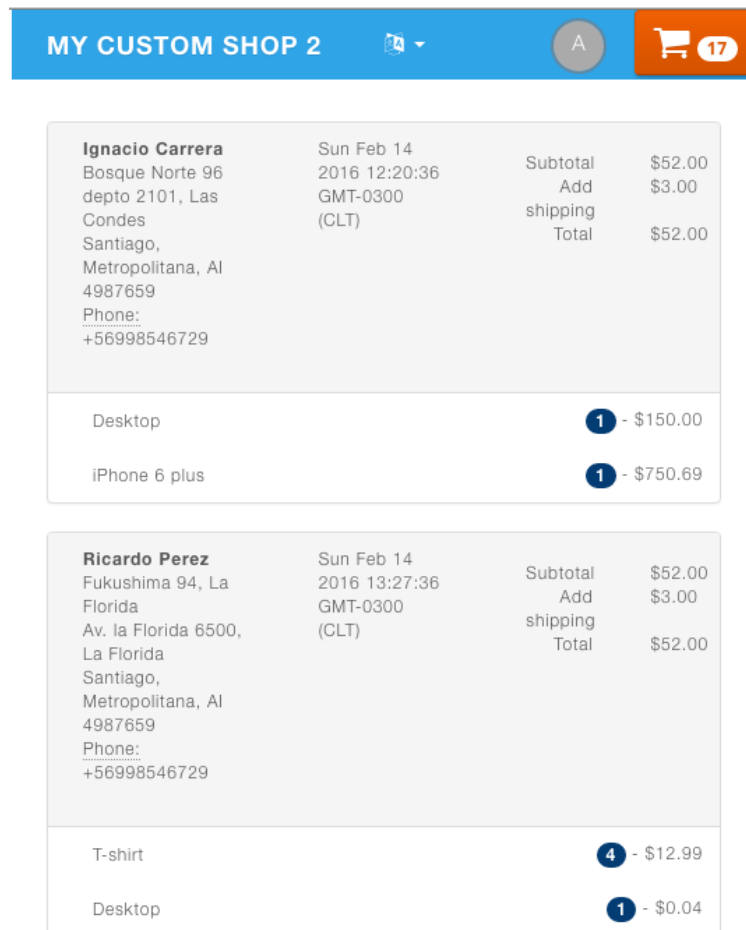
Campo	Requerido	Restricción
<i>User Name</i>	✓	
<i>Password</i>	✓	
<i>Host</i>	✓	
<i>Port</i>	✓	Número mayor que 0

Tabla 5.11: Restricciones formulario *Mail*.

## 5.9.2. Órdenes

Por definición, una orden corresponde a una solicitud confirmada, en esta caso particular, desde un cliente para comprar un producto o servicio bajo unos términos y condiciones específicas. Es importante mencionar que el flujo de compra no ha terminado cuando el cliente a realizado el pago. Es en este escenario en donde surge el concepto de *Order fulfillment*. Esta sección permite al encargado administrar las órdenes generadas por los clientes para posteriormente gestionarlas, conteniendo por lo tanto, toda la información necesaria para llevar a cabo dicho proceso. Dada la complejidad de operaciones que puede alcanzar *Order fulfillment*, en la aplicación, está sección se ha simplificado a lo más fundamental.

Al seleccionar las *Órdenes* desde el panel de *Dashboard*, lo primero que se vera es la lista de todas las *Órdenes* ingresadas por los clientes que se encuentran actualmente en el sistema. En la Figura 5.54 se ve la pantalla de *Órdenes*.



The screenshot shows a user interface for 'MY CUSTOM SHOP 2'. At the top, there is a blue navigation bar with the shop name, a user profile icon 'A', and a shopping cart icon with '17' items. Below this, two order cards are displayed. Each card contains customer information, order date and time, and a summary of items and costs.

Customer Name	Order Date	Order Time	Timezone	Subtotal	Add	shipping	Total
<b>Ignacio Carrera</b> Bosque Norte 96 depto 2101, Las Condes Santiago, Metropolitana, AI 4987659 Phone: +56998546729	Sun Feb 14	2016 12:20:36	GMT-0300 (CLT)	\$52.00	\$3.00		\$52.00
Items for Ignacio Carrera:							
Desktop							1 - \$150.00
iPhone 6 plus							1 - \$750.69
<b>Ricardo Perez</b> Fukushima 94, La Florida Av. la Florida 6500, La Florida Santiago, Metropolitana, AI 4987659 Phone: +56998546729	Sun Feb 14	2016 13:27:36	GMT-0300 (CLT)	\$52.00	\$3.00		\$52.00
Items for Ricardo Perez:							
T-shirt							4 - \$12.99
Desktop							1 - \$0.04

Figura 5.54: Vista general de todas las *Órdenes* ingresadas por clientes.

La Componente visual de una *Orden* se encuentra áltamente influenciada por la interfaz de detalle de una *Orden* del sitio *dealextrême DX*. Con el fin de simplificar espació, se tomó el resumen de la compra y se movió desde el costado inferior derecho, al superior derecho. La Figura 5.55 muestra la

interfáz de una orden.

<b>Ignacio Carrera</b> Bosque Norte 96 depto 2101, Las Condes Santiago, Metropolitana, AI 4987659 Phone: +56998546729	Sun Feb 14 2016 12:20:36 GMT-0300 (CLT)	Subtotal \$52.00 Add \$3.00 shipping Total \$52.00
Desktop		1 - \$150.00
iPhone 6 plus		1 - \$750.69

Figura 5.55: Orden ingresada por un cliente.

### 5.9.3. Cuentas

Permite agregar o quitar permisos a los usuarios. El sistema actualmente admite crear usuarios y modificar sus permisos, dejando o eliminando las componentes relacionadas con sus permisos.

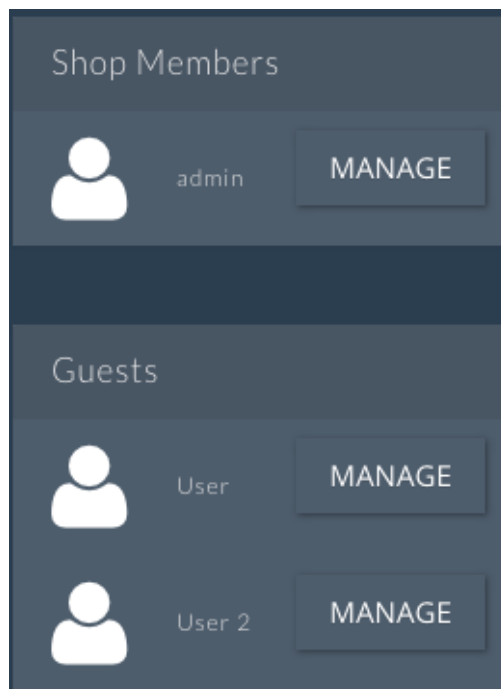


Figura 5.56: Interfaz con los usuarios del sistema.

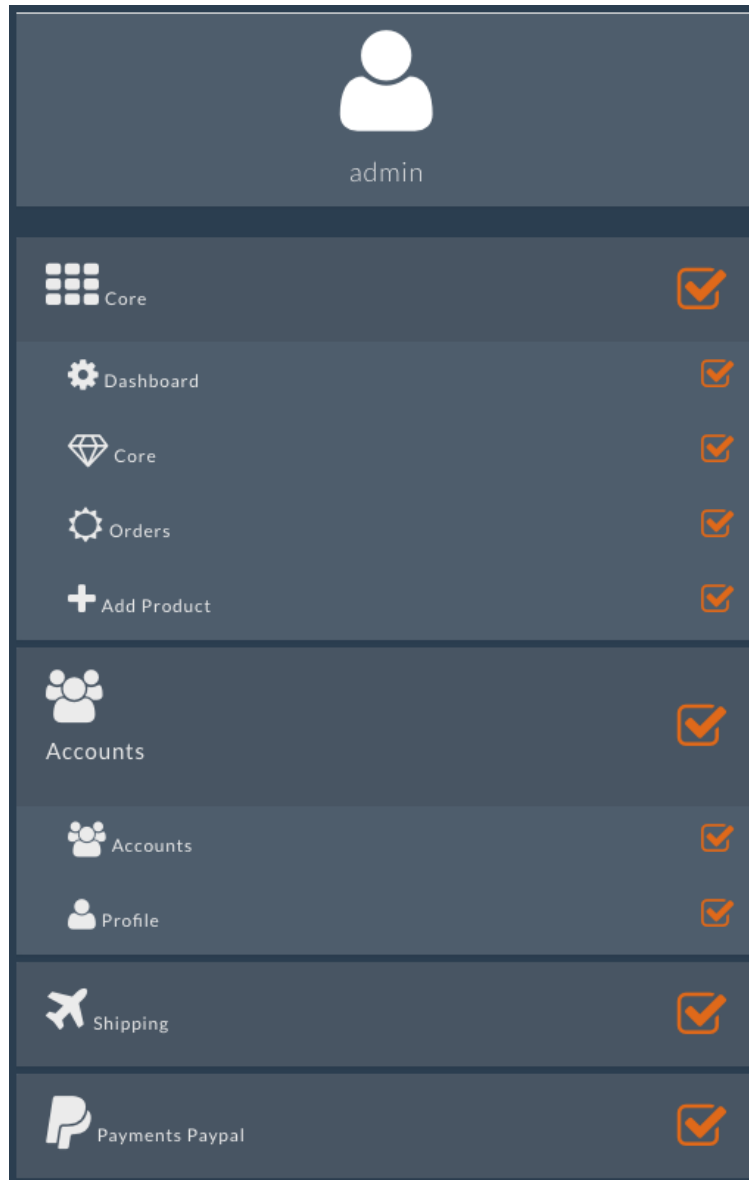


Figura 5.57: Interfaz para administrar los permisos del sistema.

Campo	Requerido	Restricción
<i>Core</i>	✓	Boolean.
<i>Dashboard</i>	✓	Boolean.
<i>Orders</i>	✓	Boolean.
<i>Add Product</i>	✓	Boolean.
<i>Accounts</i>	✓	Boolean.
<i>Profile</i>	✓	Boolean.
<i>Shipping</i>	✓	Boolean.

Tabla 5.12: Resumen restricciones formulario para los permisos.



### 5.9.4. Shipping

*Shipping* es parte del proceso realmente grande y complejo perteneciente al *workflow Order fulfillment*.

Como primer paso, hay que agregar una dirección de origen, la cual es agregada en Panel *Address*.

Segundo, se deben agregar *Shipping Zones*, los cuales tendrán su correspondiente *Shipping Rates*. Existen servicios que permiten calcular en tiempo real el *Shipping Rates*, evitando así que la tienda asuma un costo extra por un error en la estimación de la tarifa de envío. El sitio además debería ser capaz de determinar si la dirección de destino se encuentra dentro de alguna de las *Shipping Zones* definidas.

Por temas de tiempo, se decidió simplificar la solución, permitiendo agregar diferentes opciones de *Shipping* que eventualmente representarían diferentes criterios de envío.

Name	Label	Group	Cost	Enabled	+
Free	Free Shipping	Ground		✓	⚙️ 🗑️
Standard	Standard	Ground	5	✓	⚙️ 🗑️
Priority	Priority	Priority	20	✓	⚙️ 🗑️

Figura 5.58: Tabla con todas las opciones de *Shipping* disponibles.

Campo	Requerido	Restricción
<i>Method Name</i>	✓	
<i>Public Label</i>	✓	
<i>Group</i>	✓	
<i>Cost</i>		
<i>Handling</i>		
<i>Rate</i>	✓	Número mayor que 0.
<i>Enabled</i>	✓	Boolean.

Tabla 5.13: Resumen restricciones formulario para *Shipping*.

Add rate

**Method Name**

**Public Label**

**Group**

Enabled

**Cost**

Submit Cancel

Figura 5.59: Formulario para la creación de *Shipping*.

Update Free rate

**Method Name**

**Public Label**

**Group**

**Enabled**

Enabled

**Cost**

Figura 5.60: Formulario de actualización un *Shipping*.

### Tarifas de *Shipping* y carros abandonados

Aunque no es un tema que involucre a la finalidad de esta memoria, es importante mencionar la importancia que tiene el factor *Shipping* dentro del abandono del carro de compra. El desafío real que aparece cuando se desea abordar la estrategia de *Shipping*, es determinar la solución que impacte lo mínimo posible los márgenes, pero que aún así siga siendo atractivo para el cliente. Y esto es algo en lo que se desea estar en lo correcto. Estudios demuestran que el costo de *Shipping* es la principal razón del abandono de carros de compra (Figura 5.61) [62].

**“Thinking of the last time you put items in your shopping cart but did not finish the online purchase, which of the following best describes why you did not complete the transaction?”**



Base: 2,921 Web buyers who have abandoned an online shopping cart  
(multiple responses accepted)

Source: North American Technographics® Retail Online Survey, Q3 2009 (US)

Figura 5.61: *Shipping* es un factor clave en el abandono de carros de compra [62].

### 5.9.5. Pagos

En establecimientos tradicionales *Brick-and-Mortar*, un cliente ve un producto, lo examina, y entonces paga utilizando dinero, cheques, o tarjetas de crédito.

En el mundo *e-Commerce*, en la mayoría de los casos un cliente no ve físicamente el producto al momento de realizar la transacción, y el método de pago es realizado electrónicamente. Por lo tanto, temas de confianza y aceptación juegan un rol más importante en el mundo *e-Commerce* que en el mundo de negocios tradicionales en lo que a sistemas de pago se refiere [55].

Los sistemas de pago electrónico (*EPS*, siglas en inglés) utilizan *hardware* y *software* que permiten a los clientes pagar por productos y servicios en línea. Aunque estos sistemas podrían ser considerados inmaduros por los más críticos, no puede negarse el constante avance que viven periódicamente. El principal objetivo de un *EPS* son incrementar eficiencia, mejorar seguridad, y lograr una mayor comodidad y facilidad de uso para los clientes [55].

Es importante agregar varios métodos de pagos. Puede sonar obvio, pero existen métodos que aún ofrecen solo una manera de realizar un pago. Un estudio demuestra que el 56 % de usuarios espera una variedad de opciones de pagos en la página de *checkout* [115].

Aunque en la práctica es innecesario e impracticable ofrecer todos los métodos de pagos existentes, se deseará observar la preferencia del público para determinar cuáles son los métodos que ellos utilizan. Entonces se agregarán aquellos para capturar a la mayoría de los visitantes al *website* [115].

En el contexto de esta memoria, solo se implementará *PayPal* como servicio de pago.

## ***PayPal***

*PayPal* es un servicio global que mueve los montos de pago desde una tarjeta de crédito hacia el vendedor sin compartir la información financiera. *PayPal* ofrece una variedad de productos y soluciones para aceptar pagos. De estos uno ha sido elegido para permitir el flujo completo de compra, pero siempre considerando la opción de agregar nuevos métodos en un futuro.

## ***PayPal Payment Pro***

*PayPal Payment Pro* es una solución customizable que permite a los vendedores mantener a los compradores dentro de su *website* durante el proceso completo de pagos y *checkout*. Los vendedores pueden mantener sus propios sitios de *checkout* personalizados y enviar transacciones a *PayPal*, o es posible también tener un *host* de *PayPal* que tenga páginas de *checkout* e incluso manejar la seguridad para las ventas y autorización. *PayPal Payment Pro* puede aceptar pagos de *PayPal* y *PayPal Credit*, al igual que con tarjetas de crédito de pago [141].

Para el uso de este servicio, es necesaria la creación de una *PayPal app*. Toda la información necesaria se encuentra en la documentación para los desarrolladores [142], pero en resumen, lo relevante acá para lograr una configuración adecuada son 3 elementos:

- Existen dos ambientes de trabajo: *Sandbox* y *Live*.
- *Client ID*.
- *Secret*.

Tanto el *Client ID* como el *Secret* dependen del ambiente en el cual me encuentro.

Desde el *Dashboard* es posible acceder al menú configurable de *PayPal*, el cual permite agregar esta información (ver Figura 5.62).

Figura 5.62: Formulario para agregar la configuración necesaria para el uso de *PayPal PayFlow*.

El formulario de *PayPal* se somete a las restricciones visibles en la Tabla 5.14.

Campo	Requerido	Restricción
<i>Payflow enabled</i>	✓	Boolean.
<i>Client ID</i>		
<i>Secret</i>		
<i>PayFlow Mode</i>	✓	Boolean.

Tabla 5.14: Resumen restricciones formulario *PayPal PayFlow*.

## 5.10. Checkout

Una de las peores violaciones de usabilidad para un *website e-Commerce* corresponde a un proceso de *Checkout* que no sea lineal. *website* que no tengan un proceso de *Checkout* lineal deja a muchos de los usuarios confusos e intimidados. Al momento de realizar este estudio, tanto *Walmart* como *Zappos* tenían un proceso *Checkout* no lineal [102].

Afortunadamente, hacer el proceso completamente lineal no es difícil. En el caso de necesitar, por ejemplo, un *sub-paso* tal como *creación de una cuenta*, esta nunca debe redirigir a un paso previo en el proceso de *Checkout*, en lugar de eso, debe dirigir al cliente al siguiente paso del proceso [102].

Esto es crítico porque el modelo mental de la mayoría de los compradores dicta que el proceso de *Checkout* debería ser lineal. Al ver la misma página dos veces, la mayoría de los clientes concluye que el sitio tiene un error, porque esto ocurre cuando hay errores de validación. [102].

Para este *Framework* se ha implementado un proceso de *Checkout* lineal, el cual tiene los pasos que se aprecian en la Figura 5.63.

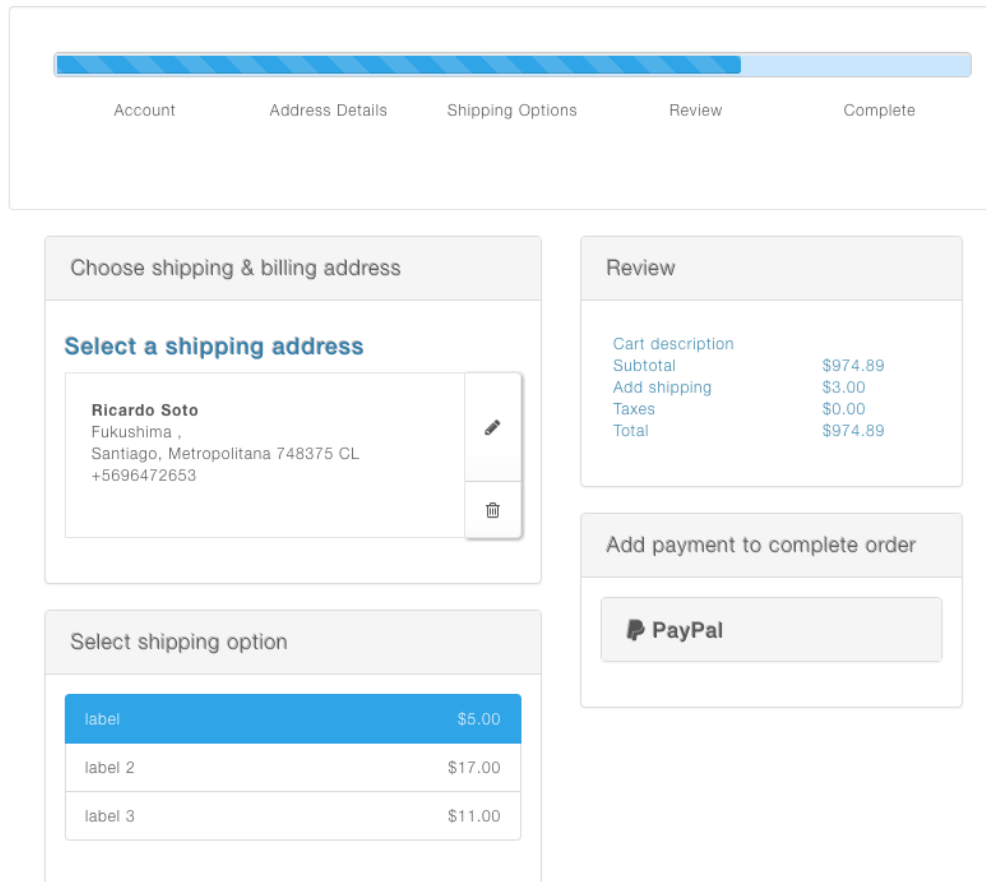


Figura 5.63: Detalle de todos los pasos del *workflow Shipping*.

Otro detalle importante a considerar es que mientras sea posible hay que limitar el proceso de *Checkout* a una sola página. En el caso de separar el proceso en algunas páginas, se debe incluir una barra de proceso para mostrar cuanto falta para terminar [147].

Afortunadamente, ambas restricciones son una manera para lograr la otra. Por lo tanto se desarrolló el proceso incluyendo a ambas. Esto se observa en la Figura 5.64.

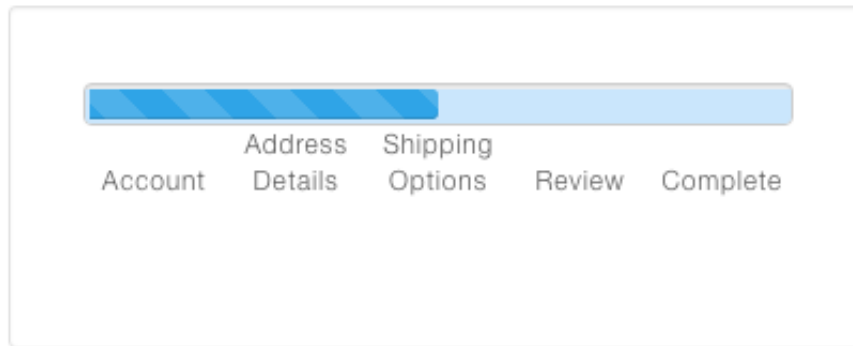


Figura 5.64: Estado actual del *workflow* del *Shipping*. En este caso se encuentra en la selección del tipo de *Shipping*.

Cada uno de los pasos que aparecen en la Figura 5.63 son descritos a continuación.

### 5.10.1. Cuenta de usuario

Nada irrita más a un cliente que ser forzado a crear una cuenta para lograr comprar algo *online*. Después de todo, comprar *online* se supone debería ser sencillo. Es cierto que la creación de una cuenta hará que las comprar en el futuro sean más sencillas, pero no aporta nada para la situación en cuestión [147].

Es cierto que este *Framework* requiere la creación de una cuenta para lograr la compra de productos, sin embargo no es necesario confirmarla para poder comprar. Es cierto que esta no es la situación ideal, pero esta situación tampoco es del todo mala, dada la simplicidad en la Creación de cuenta de un usuario (5.2.1).

### 5.10.2. Detalles de dirección

Esta sección funciona exactamente igual a la libreta Libreta de direcciones (5.8.2). En otras palabras, aparte de seleccionar una dirección permite:

- Ver todas las direcciones configuradas en el sistema.
- Editar y/o eliminar las direcciones disponibles del sistema.
- Configurar una nueva dirección.
- En caso de no tener ninguna dirección configurada, muestra el formulario de creación.

Como se indica en otras partes, es altamente deseable limitar el proceso de *checkout* a una sola página [147]. Estas funcionalidades están alineadas con este principio. En caso de necesitar hacer alguna gestión en las direcciones, no será necesario cambiar de página. Como ejemplo, en la Figura



5.65 se observa el formulario de creación de una dirección dado que no existe ninguna configurada previamente.

2 Choose shipping & billing address

Create your default address

**Country**  
United States

**Full name**  
[Text input field]

**Address 1** [Text input field] **Address 2** [Text input field]

**Postal** [Text input field] **City** [Text input field] **Region** [Text input field]

**Phone**  
[Text input field]

Make this your default shipping address?

Make this your default billing address?

This is a commercial address.

Save and continue

Figura 5.65: Despliegue automático del formulario de direcciones cuando no se tiene ninguna dirección agregada.

### 5.10.3. Opciones de *Shipping*

Como se describió en la sección *Shipping* (5.9.4) la implementación de *Shipping* ha sido simplificada considerablemente. Sin embargo, la implementación existente permite la elección entre diferentes métodos de *Shipping* previamente definidos (Figura 5.66).

Select shipping option	
Free Shipping	\$0.00
Normal Shipping	\$10.00
Priority Shipping	\$59.00

Figura 5.66: Opciones de *Shipping* configuradas en el sistema.

En caso de no tener ningún método de *Shipping* disponible, o si el *package* a sido deshabilitado, entonces aparecerá un mensaje ( Figura 5.67) que indicará que no hay métodos disponibles.

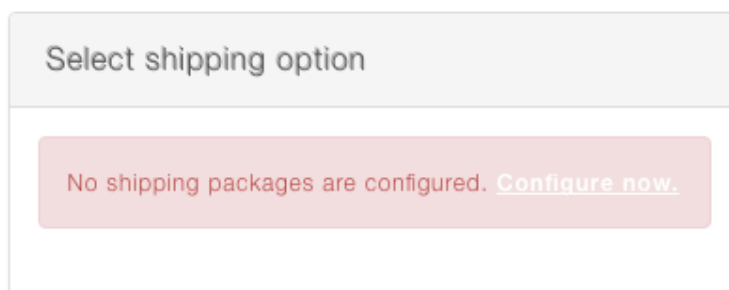


Figura 5.67: Mensaje cuando no hay opciones de *Shipping* configuradas en el sistema.

#### 5.10.4. Resumen de la compra

Esta sección muestra un resumen de la compra que se está realizando (ver Figura 5.68). Esta información es parte del detalle completo que debe mostrar esta sección para dejar claro al cliente qué es exactamente lo que está comprando. Esto debe ser mejorado como trabajo futuro.

#### 5.10.5. Pagos

Como se mencionó en la sección 5.9.5 Pagos, es relevante considerar varios métodos de pagos, sin embargo en el contexto de esta memoria solo se ha integrado a *PayPal Payment Pro*.

Review	
Cart description	
Subtotal	\$974.89
Add shipping	\$3.00
Taxes	\$0.00
Total	\$974.89

Figura 5.68: Detalle de la compra dentro del proceso de *Checkout*.

### PayPal Payment Pro

Anteriormente se había indicado que este método permite mantener a los clientes dentro del *website* durante el proceso completo de pagos y *checkout*. Básicamente se ha implementado la aceptación de pago de tarjetas de crédito a través de un formulario dentro del *website*. La información necesaria para la transacción se ve en Código 5.2.

```

1 {
2   "intent": "sale",
3   "payer": {
4     "payment_method": "credit_card",
5     "funding_instruments": [
6       {
7         "credit_card": {
8           "number": "5500005555555559",
9           "type": "mastercard",
10          "expire_month": 12,
11          "expire_year": 2018,
12          "cvv2": 111,
13          "first_name": "Betsy",
14          "last_name": "Buyer"
15        }
16      }
17    ]
18  },
19  "transactions": [
20    {
21      "amount": {
22        "total": "7.47",
23        "currency": "USD"
24      },
25      "description": "This is the payment transaction description."
26    }
27  ]

```

Código 5.2: Estructura de un objeto para la aceptación de tarjeta para *PayPal PayFlow* [140].

De acá se desprende la información de la tarjeta:

- Número.
- Tipo de Tarjeta.
- Mes en que expira la tarjeta.
- Año en que expira la tarjeta.
- *CVV2*.
- Nombre.
- Apellido.

Toda esta información debe ser entregada por el cliente, con el fin de gestionar el pago. Sin embargo, se sabe que el tipo de tarjeta puede ser determinado a partir de los 6 primeros dígitos del número de tarjeta [110].

Los campos de expiración de tarjeta de crédito pueden ser confusos para decifrar si no son escritos exactamente como están en las tarjetas de crédito. Algunos *websites* usan nombres de meses, mientras otros emplean una combinación de *nombre-número*, mientras y usan solo números. La manera correcta de dar los campos de formato es simplemente poner los campos de la misma manera en que aparecen en la tarjeta de crédito (solamente números). Esto minimiza la confusión y mala interpretación porque el usuario puede fácilmente verificar los campos contra los de la tarjeta de crédito [102].

Finalmente, se desarrolla el formulario de la Figura 5.69, para que el cliente pueda efectuar el pago del producto y/o servicio.

The image shows a PayPal credit card payment form. At the top left is the PayPal logo. Below it are several input fields: 'Cardholder name', 'Card number', 'Choose month', 'Chose year', and 'CW'. A large green button at the bottom contains the text 'COMPLETE YOUR ORDER'.

Figura 5.69: Formulario de la tarjeta de crédito para realizar el pago utilizando *PayPal Payment Pro*.

Una vez finalizado este proceso, se creará una orden de compra, la cual podría ser consultado en la interfaz de ordenes.

## 5.11. Órdenes

Muestra información relevante relacionada con cada una de las compras realizadas en el sitio por el cliente autenticado. Esta interfaz es equivalente a la que utiliza el administrador desde el *Dashboard* con la gran diferencia que acá el usuario solo ve las *Órdenes* que el ha generado en el sistema, en cuanto el administrador, ve todas las *Órdenes* generadas en el sistema. (ver Figura 5.54). Este escenario no es el correcta, dado que las necesidades de estos usuarios son distintas, por lo tanto ambas interfaces no pueden ser exactametne igual. Sin embargo, esta plataforma ha sido desarrollada para el contexto de una Memoria, por lo que no es posible la implementación total de cada Interfaz.

## 5.12. Seguridad en un sitio e-Commerce

Seguridad en una aplicación *web* es sobre todo entender seguridad de los dominios (*security domain*) y entender la superficie de ataque (*attack surface*) entre esos dominios. En una aplicación de *Meteor*, las cosas son muy simples:

- El código que se ejecuta en el servidor es confiable.

- Todo lo demás: código que se ejecuta en el cliente, la información enviada a través métodos y publicación de argumentos, etc; no puede ser confiable.

En la práctica, esto significa que la mayoría de la seguridad y las validaciones se encuentran en la interacción entre estos dominios. En términos simples:

- Validar y revisar todos los *input* que provienen desde el cliente.
- No liberar ninguna información secreta al cliente [168].

La seguridad en *Internet* se ha tornado un problema constante y creciente al mismo tiempo que nuevas tecnologías basadas en *Internet* y aplicaciones son desarrolladas.

## 5.13. Seguridad en *eframework*

### 5.13.1. Autorización

Se integra la asignación de permisos a los usuarios, para posteriormente verificar si cuenta o no con accesos a ciertos métodos de *Meteor* o publicar información.

La aplicación permite la flexibilidad de asignar permisos de la manera que parezca más útil. Se pueden utilizar roles tradicionales, como *admin* o *webmaster*, o incluso asignar permisos con una mayor granularidad tales como *ver-mensajes-secretos*, *vista-usuarios*, *lectura-planillas*, etc. En general, una mayor granularidad es mejor, porque permite el manejo de aquellos casos particulares que existen en la *vida-real* de forma sencilla, sin la necesidad de incluir una gran cantidad de *roles*.

En ocasiones es útil permitir la asignación de conjuntos independientes de permisos. La aplicación utiliza grupos de *roles* que se denominan *sets*.

Para conocer en detalle las opciones permitidas, consulte la documentación del *package* de *Meteor Roles* [47] .

### 5.13.2. Autenticación

*Meteor Account* es un sistema modular del cual todos pueden escribir un *package* para proveer un nuevo método de *login*. Algunos de estos *package* son oficialmente mantenidos por *Meteor Project*. *package* tales como *accounts-google* [94] , *accounts-facebook* [93] , y *accounts-twitter* [95] proveen la habilidad de hacer *login* con servicios de autenticación *third-party*.

Password son condicionados utilizando *bcrypt*, el cuál es considerado como buena práctica de acuerdo a la industria [167]. Esto ayuda a la protección contra vergonzosas filtraciones de contraseña en el caso de que la *Base de datos* este comprometida.

### 5.13.3. Políticas de navegación

Nuevos *browsers* permiten imponer políticas relacionadas con la seguridad. Estas políticas ayudan a prevenir y mitigar ataques comunes, como *cross-site-scripting* y *clickjacking*.

Básicamente, lo que se hace es entregar instrucciones de configuración al encabezado *HTTP X-Frame-Options* y *Content-Security-Policy*.

*X-Frame-Options* le indica al *browser* cuáles *websites* tienen permitido desplegar la aplicación. Solo se debe permitir a sitios de confianza incrustar la aplicación, porque sitios maliciosos podrían dañar a los usuarios con *clickjacking* [114].

*Content-Security-Policy* indica al *browser* desde donde la aplicación puede cargar contenido, lo que promueve prácticas seguras y mitiga el daño de *cross-site-scripting* [197]. Las políticas de navegación pueden ser configuradas en el caso de que la opción por defecto no sea adecuada.

Para obtener detalles de cómo configurar las diferentes opciones de políticas disponibles, consulte la documentación del *package* de *Meteor Browser-Policy* [33] .

### 5.13.4. Encriptación de información sensible

Se encripta la información sensible guardada en la *Base de datos* tales como una *key* secreta de aplicación para servicio *login* y *tokens* de acceso de usuarios.

### 5.13.5. Seguridad de escritura para las *Collections* de MongoDB

La única manera de realizar cambios a la información que existe en la *Base de datos* es exclusivamente a través de *Methods* de *Meteor* (definidos en Sección 4.2). Al inicio de cada uno de estos métodos se confirma si el cliente que esta accediendo cuenta con los permisos necesarios para realizar dicha acción. Dependiendo de sus permisos, se acepta o deniega la petición.

## Capítulo 6

# Conclusiones

La plataforma desarrollada no constituye una solución completa de un *Framework e-Commerce*, pero sí una sólida base para esta finalidad dado que cada una de sus componentes fueron seleccionadas a partir de las mejores prácticas que se han identificado producto de la experiencia en el mercado y diversos estudios realizados.

La tecnología utilizada mostró ser una elección correcta dado las facilidades que esta entregó, junto con la gran comunidad aportando soluciones, permitieron enfocarse principalmente en el problema de negocio en lugar de enfocar los esfuerzos para el uso de la tecnología. Sin duda esta característica es altamente relevante en donde la velocidad de desarrollo presenta una gran ventaja competitiva para adaptarse rápidamente a las futuras necesidades del mercado.

Para desarrollar una aplicación de grandes proporciones se necesitó la construcción de una arquitectura genérica la cual puede ser utilizada como punto de inicio de una gran cantidad de aplicaciones *web*. Aunque no fue planteado como un objetivo inicial, uno de los logros más significativos fue el desarrollo de esta arquitectura, la cual es un punto sólido de inicio para prácticamente cualquier aplicación *web* de la actualidad.

De los objetivos planteados inicialmente, no se cumplió el referente a la evaluación del uso del *Framework* para el desarrollo de una solución particular de servicio *e-Commerce*. El respecto de los demás objetivos planteados al comienzo, sí se cumplieron puesto que se logró desarrollar un *Framework* que si bien es cierto, no constituye una solución equivalente en funcionalidades a los *Framework* descritos en un inicio dada la gran cantidad de características que estos tienen, si representa el *core* de una plataforma con características nuevas, útiles para el desarrollo de plataformas modernas.



# Capítulo 7

## Trabajos Futuros

### 7.1. Mejora en la arquitectura genérica

Durante el desarrollo de la aplicación se comprendió el fuerte concepto que representa *package* para *Meteor*. Esto sugirió mejoras en la arquitectura las cuales básicamente corresponden en desarrollar un proyecto base que administre todos los módulos (*package*) que se agregan al sistema para incorporar características a la aplicación en desarrollo.

Cada uno de estos módulos, contendría un archivo descriptivo que expondría:

- **Permisos.** Se describen los permisos necesarios que el usuario necesita para utilizar los recursos del módulo. Se utiliza el *package Roles* [47] para soportar esta funcionalidad.
- **Enrutamientos.** Los enrutamientos para acceder a los servicios del módulo.
- **Configuración.** El enrutamiento para acceder al Menú. Este debería ser accesible desde el *Dashboard* de la arquitectura base.
- **Tipo de Módulo.** Agregar etiquetas que permitan distinguir y agrupar diferentes módulos.
- **Etiqueta descriptiva.** Básicamente un nombre con el cual podría distinguir el módulo. Por ejemplo podría utilizarse en la interfaz de *Dashboard* para acceder a su menú de configuración.
- **Icono.** Algún icono que podría utilizarse para distinguir el módulo de los otros. Se podría utilizar en el *Dashboard* al igual que lo hace **Etiqueta descriptiva**.
- **etc.**

Otro detalle importante, es que la información de todos los módulos registrados serían accesibles desde cualquier módulo integrado a la arquitectura. Esto permitiría por ejemplo:

- Crear una vista que agrupe los módulos por tipo.

- Crear un módulo de cuentas que consulte todos los permisos que existen para todos los módulos integrados y, a través de una interfaz, asociarlos a los distintos usuarios registrados en el sistema. Con esto tendría un módulo genérico de asociación de permisos para usuarios.
- etc.

Notar que esta nueva funcionalidad no debería excluir ningún tipo de aplicación que se pueda desarrollar con la anterior arquitectura. Es más, la otra arquitectura también debe utilizarse para todos los módulos que se integren a esta arquitectura.

## 7.2. Actualizar el *Framework* a la nueva arquitectura

Las ventajas:

- **Generalizar.** Permitiría la integración de una infinidad de módulos con nuevas características sin la necesidad de hacer reestructuraciones.
- **Desacoplar.** Disminuir la superficie de contacto entre el módulo y la arquitectura, el cual será solo a través de una interface (archivo de registro).
- **Agrupar.** Se podrían agrupar tipos de módulos. Ejemplo: módulos de pago.
- **Dashboard.** Permite poblar el *Dashboard*. Cada uno de los elementos en el dashboard podría tener características únicas provistas por los módulos que representan, como su nombre, utilizar el icono que lo representa. Finalmente dicho elemento sería un acceso directo al menú de configuración de dicho módulo. Lo interesante de todo esto es que sería sin agregar ninguna línea de código.
- **Módulos compatibles.** Los módulos ya creados solo deberían agregar su archivo de descripción. Todo lo demás debería permanecer básicamente igual.
- etc.

## 7.3. *analytics*

Hay una considerable evidencia que decisiones basadas en *analytics* tienen más probabilidad de resultar correctas que aquellas basadas en la intuición. Es, al menos, mejor saber dentro de los límites de datos y análisis que *crear, pensar o sentir*, y la mayoría de las compañías pueden beneficiarse desde decisiones tomadas analíticamente. Claramente, existen circunstancias en las cuales las decisiones no pueden o no deberían realizarse analíticamente [70].

## **7.4. Opciones de *shipping***

Cuando se inicia un sitio *e-Commerce* es importante considerar una estrategia para *shipping* dado que es un hecho que el mundo del *shipping* es complejo y confuso. Sin embargo, una buena estrategia de *shipping* es de vital importancia y un elemento fundamental para cada negocio *e-Commerce* con éxito [159].

## **7.5. Seguridad**

En estricto rigor, todas las áreas desarrolladas en esta plataforma requieren de mejoras. En particular la seguridad, a pesar de su gran relevancia a la hora de entregar una buena imagen a los clientes, no pudo ser abordada a una profundidad aceptable.

# Glosario

## A

**AMP** *Apache<sup>TM</sup>, MySql, PHP.*

**analytics** *Analytics processing* es caracterizado por pocos usuarios (análisis de negocio en lugar de operaciones de clientes y POS) *submitting* pocas *request*, pero *queries* pueden ser muy complejas y con intenso uso de recursos. El tiempo de respuesta es generalmente medido en decenas a cientos de segundos.

**API** Application Programming Interfaces.

**Application Programming Interfaces** Es un conjunto de *routines*, protocolos, y herramientas para el desarrollo de aplicaciones. La *API* especifica cómo la componente de *software* debería interactuar y son utilizados en la programación de las componentes GUI. Una buena *API* facilita el desarrollo de un programa proporcionando todos los bloques *build*. Un programador luego pone los bloques juntos [129].

**Asynchronous Replication** Replicado *Asynchronous* es un enfoque *store and forward* para *Data Backup*. Replicado *Asynchronous* escribe datos primero para el *primary storage array* y luego, dependiendo del enfoque de la implementación, *Commits* datos para ser replicada a *memoria* o en *Journal disk-based*. Entonces esto copia los datos en *tiempo-real* o *Scheduled* intervalos para la replicación de los objetivos [148]. En contraste a replicación *Synchronous*, replicación *Asynchronous* está diseñado para trabajar sobre grandes distancias. Esto no requiere tanta *Ancho de Banda* como en el caso de replicación *Synchronous* y puede tolerar bajas en la *Conectividad* [148].

## B

**B2B** Business to Business.

**B2C** *negocios to cliente.*

**B2G** *negocios to Gobierno.*

**Berkeley Software Distribution** 3-clause license es una *permissive Open Source license*, que impone mínimas restricciones para el uso del *software* [31].

**boilerplate code** En tecnología de la información, *boilerplate* es una unidad de escritura que puede ser reutilizada una y otra vez sin cambios. Por extensión, la idea es en ocasiones aplicada a programación *reusable* como *boilerplate code* [28].

**Bot** ver Crawler.

**Brick-and-Mortar** (También conocido como bricks-and-mortar o B&M) es simplemente utilizado para describir presencia física de edificios o otras estructuras. El término *brick-and-mortar negocios* es también utilizado para referirse a empresas que poseen edificios, instalaciones de producción, o el almacén para las operaciones.

**BSD 3-Clause License** Berkeley Software Distribution.

**Business to Business** Describe transacciones comerciales entre empresas, como entre un fabricante y un mayorista, o entre un mayorista y un *retailer*.

## C

**C2C** *cliente to cliente*.

**clickjacking** Es un término primeramente introducido por Jeremiah Grossman y Robert Hansen el año 2008 para describir la técnica para la cual ataques cross-domain son realizados hijacking *clicks* del *mouse* para realizar acciones que el usuario no tienen conocimiento [103].

**Commodity Hardware** Dispositivo o componente de dispositivo que es relativamente barato, altamente disponible y más o menos intercambiable con otro *hardware* de su tipo [149].

**Computed Properties** En *Ember.js*, computed properties permiten declarar funciones y propiedades. Es posible crear uno definiendo una computed property como una función, la cual *Ember.js* llamará automáticamente cuando se solicite la propiedad. Es posible usar entonces de la misma manera que se usaría, *static property* [174]. Es súper útil para tomar una ó más propiedades y transformando ó manipulando su *Data* para crear un nuevo valor [174].

**Conversion Rate** *Conversion Rate* de un sitio web es la medida de éxito de una campaña pagada de inclusión. Una *Conversion Rate* es medida por el número de visitantes potenciales realizando la acción deseada, sea la acción de comprar un producto, completar un formulario, o algún otro objetivo del sitio web. Por ejemplo, si hay 100 visitantes para una página web particular a través de publicidad de pago por *click*, y solo uno de esos 100 compra el producto que vende el sitio web, entonces la *Conversion Rate* para esta publicidad en particular es del 1% [127].

**Crawler** Es un programa que visita *websites* y lee sus páginas y otras informaciones con la intención de crear entradas para un search engine index. Los principales *search engines* en la *web* tienen este tipo de programa, el cual también es conocido como Spider o un Bot. Crawlers son típicamente programados para visitar *sites* que han sido publicados por sus dueños como contenido nuevo o actualizado. *sites* o páginas específicas pueden ser visitadas e *indexed*. Crawlers aparentemente reciben su nombre porque ellos *crawl* a través de un *sites* de una página en algún momento, siguiendo los *Links* a otras páginas en el *sites* hasta que todas las páginas sean revisadas [150].

**Crawlers** ver Crawler.

**cross-site-scripting** .

**CVV** [86].

**CVV2** [86].

## D

**DBMS** Database Management System.

**Digital Subscriber Line** Es una tecnología para permitir información *high-bandwidth* hacia hogares y pequeños *negocios* sobre líneas telefónicas ordinarias [37].

**domain-specific Lenguaje** Son lenguajes pequeños, focalizados en un aspecto particular de un sistema de *software*. Es posible construir una programa completo con DSL.

**don't repeat yourself** En *software engineering*, es un principio de desarrollo de *software*, cuyo objetivo es reducir la repetición de información de todo tipo, especialmente útil en arquitecturas *Multi-Tier*.

**DRY** *don't repeat yourself*.

**DSL** Siglas de *domain-specific Lenguaje*.

**DSL (modem)** ver *Digital Subscriber Line*.

## E

**e-Commerce** Electronic Commerce.

**Easy e-Commerce** Easy e-Commerce, es un *Framework* e-Commerce para el desarrollo flexible y ágil de soluciones e-Commerce a fin de disminuir la complejidad en el desarrollo, así como los costos recursos humanos y duración del proyecto.

**EE-Commerce** Easy e-Commerce.

## **Electronic Data Interchange**

Es el intercambio de documentación de *negocios computer-to-computer* en un formato electrónico *standard* entre *partner negocios* [43].

**Electronic Funds Transfer** Es un sistema de transferencia de dinero desde una cuenta bancaria a otra directamente sin ningún tipo de papel moneda que intercambie de manos [36].

**EPS** Siglas de Electronic Payment System.

## **G**

**G2G** *Gobierno to Gobierno.*

**Gems** Son *Packages* que contienen tanto su información, como dependencias necesarias a instalar [42].

**GNU General Public License** Licencia de *software* utilizada para la mayoría de los programas de GNU y para más de la mitad de los paquetes de software libre. La última es la versión 3 [44].

**GNU General Public License version 3** Última versión de GNU General Public License. Notar que GPLv3 no es compatible GPLv2 por si mismo. Sin embargo, muchos de los programas liberados en GPLv2 permite el uso en términos de la última versión de GPL [44].

**GNU GPL** GNU General Public License.

**GNU GPLv3** GNU General Public License version 3.

## **H**

**HDD** Hard Disk Drives.

**HDFS** Hadoop Distributed File System, un sistema de archivos distribuido que almacena datos en las máquinas , proporcionando un muy alto ancho de banda agregado a través de clústers.

**hijacking** Corresponde al ingreso total que una firma recibe [77].

**HTTP verbs** .

## **I**

**IOPS** (Input/Output Operations Per Second, pronounced eye-ops) es una medida de performance usada para benchmark dispositivos de almacenamiento para computadores como HDD, SSD y SAN.

## J

**JSON** *JavaScript* Object Notation.

## L

**LAMP** *Linux, Apache™, MySQL, PHP*.

**Luhmann's theory** [125].

## M

**MapReduce** Modelo de programación para el procesamiento de datos a gran escala.

**merchant account** [160].

**minifying** *Minification*, en lenguajes de programación y especialmente en *JavaScript*, es el proceso de remover todos los caracteres innecesarios desde *source code* sin cambiar su funcionalidad.

**Minimongo** Una implementación *Client-side* de MongoDB que soporta consultas básicas, incluyendo algunas geo-espaciales.

**Minimum viable product** Es aquel producto que tiene solo aquellas *características* necesarias, y no más que permitan un producto entregable para que los usuarios lo vean, al menos aquellos que tienen un interés monetario, y empiecen a entregar *feedback* [57]. ¿Pero por que mínimo? Porque el tiempo y dinero son severamente limitados. Se desea el mejor resultado de inversión. Aprendizaje máximo con el mínimo esfuerzo [57].

**MongoDB** Base de datos *Document-Oriented*.

**Multi Channel** *Multi channel retailing* es el conjunto de actividades involucradas en la venta de mercancías o servicios a los consumidores a través de mapas de un canal. Esta definición distingue *Multi channel retailing* de *Multimedia Marketing* en que típicamente involucra el use de de múltiples *channels* para simplificar la comunicación con *clientes* [199].

**Multi-Tier Pricing** Differential ó Tiered Pricing es la práctica de establecer diferentes precios para diferentes *markets*, típicamente precios más altos en *markets* más ricos, y precios más bajos en *markets* más pobres [144].

**MVC** Se refiere al patrón arquitectónico Model-View-Controller.

**MVP** Siglas de *Minimum viable product*.

**MVVM** Se refiere al patrón arquitectónico Model-View-ViewModel.



## N

**NoSQL** Not Only SQL.

## O

**Object-Relational Mapper** Es una técnica de programación en la cual un descriptor de *Metadata* es utilizado para conectar *object code* a una *Base de dato* relacional. *object code* es escrito en programación *object-oriented*; lenguajes como *Java* o *C#*. ORM convierte *Data* tipos de sistemas que no pueden coexistir dentro de la *Base de dato* relacional y lenguajes *OOP* [189].

**Object-relational mapping** En ciencias de la computación es una técnica de programación para convertir datos entre *type system* incompatibles en lenguajes de programación *object-oriented*.

**OLTP** Online Transaction Processing.

**Open Software License** Open Software *License* es una licencia tipo *copyleft*, con una clausula de rescisión provocada por la presentación de una demanda alegando violación de patentes [11].

**ORDBMS** Object-Relational Database Management System.

**Order fulfillment** Es definido como aquellos pasos envueltos en recibir, procesar y entregar las ordenes a los clientes finales [157].

**ORM** Object-Relational Mapper.

**ORM** Object-relational mapping.

**OSI** Open System Interconnection.

**OSL** Open Software License.

**Out of the Box** An out of the box *característica* ó *functionality*, particularmente en *software*, es una *característica* ó *functionality* de un producto que funciona inmediatamente despues de la instalación sin ninguna configuración o modificación [191].

## P

**payment gateway** [161].

**prefetch** En *computer architecture*, *instruction prefetch* es una técnica usada en *microprocessors* para *speed up* la ejecución de un programa reduciendo los estados de espera.

## R

**Rack Middleware** Rack proporciona una interfaz mínima entre *Webservers* soportando *Ruby* y *Ruby Frameworks* [50].

**RAM** Random Acces Memory.

**RDBMS** Relational Database Management System.

**Representational State Transfer** Es un conjunto de restricciones arquitectónicas que intenta minimizar la *latency* y la comunicación *network* mientras al mismo tiempo maximiza la independencia y la *Scalability* de las componentes de la implementación. Esto es logrado agregando restricciones en *connector semantics* donde otros estilos se han focalizado en *component semantics*. Rest permite *caching* y la reutilización de las interacciones, sustitución dinámica de *Hypermedia* [79].

**Resource Descriptor** Son documentos (*Serialization Languages* usualmente bien conocidos tales como *XML*, *RDF* y *JSON*) que proporcionan información *machine-readable* sobre *resources* (*resource Metadata*) para el propósito de promover *interoperability* y asistir en la interacción con *resources unknown* para apoyar *interfaces* conocidas [98].

**REST** Representational State Transfer.

**RESTful** Termino utilizado para referirse a un *web Service* que implementa la arquitectura REST.

**RIA** Single Rich Internet Application.

## S

**SAN** Storage Area Networks.

**search engine index** Es el lugar donde toda la *Data* que el *search engines* a recolectado es *stored*. Es este el que proporciona los resultados para las búsquedas de *queries*, y páginas que son *stored* dentro del search engine index que aparece en la página de los resultados del *search engines* [170].

**Search Engine Optimization** Proceso de obtención de tráfico desde los resultados de búsqueda *free*, *organic*, *editorial* o *natural* en *search engines* [155].

**SEO** Search Engine Optimization.

**Serialization Languages** Serialization Languages.

**Shipping Rates** Corresponde a las tarifas de envío a los diferentes Shipping Zone.

**Shipping Zone** Corresponde a regiones y países donde se realizan entregas.

**Shipping Zones** Ver Shipping Zone.

**Single Page Application** Single Page Application.

**Single Point of Failure** Es una componente de un sistema crítico con la habilidad de cesar las operaciones durante un fallo. SPOF no son deseables para un sistema que requiere *Reliability* y *Availability*, tales como aplicaciones de *software*, *Networks* ó *Supply Chains* [190].

**Single Rich Internet Application** Single Rich Internet Application.

**SKU** Stock-keeping unit.

**source maps** Es un archivo generado por *minifier* el cual comprime un *mapping* del archivo *minified* a la version original sin comprimir [156].

**SPA** Single Page Application.

**Spider** ver Crawler.

**SPOF** Single Point of Failure.

**SQL** Structured Query Language.

**SSD** Solid State Drives.

**Stemming** Es un problema lingüístico que describe el proceso de reducir las palabras a su raíz base. En análisis morfológicos automatizados, la raíz de una palabra puede ser de un menor interés inmediato que sus sufijos, que pueden utilizados como pistas de la estructura gramática [123].

**Supply Chain** Es un sistema de organizaciones, personas, actividades, información, y recursos envueltos en el movimiento de un producto o servicio desde un *Supplier* a un cliente.

## T

**test fixture** Un test fixture es un estado definido utilizado como base para correr *tests*. El proposito de un test fixture es asegurar que existe un *environment* fijo y conocido en el cual *tests* se ejecutan de tal forma que los resultados son repetibles [84]. Ejemplos de test fixture son:

- Cargar una *Base de datos* con un conjunto específico y conocido de *Data*.
- Copiar un conjunto específico de *archivos*.
- Preparación de datos de entrada creación/*setup* de objetos simulados ó *mock objects*.

**test fixtures** Ver test fixture.

**third party payment** [162].

**Tiered Pricing** Differential ó Tiered Pricing es la práctica de establecer diferentes precios para diferentes *markets*, típicamente precios más altos en *markets* más ricos, y precios más bajos en *markets* más pobres [144].

**transactional** *Transactional processing* es caracterizado por un largo número de de pequeñas, discretas, transacciones atómicas. El énfasis de los sistemas OLTP es (a) *high throughput (transactions per second)*, y (b) mantener integridad de datos en ambientes multiusuarios.

## U

**URI** Uniform Resource Identifier.

## V

**Vendor** También conocidos como proveedor, es una individuo o compañía que vende productos o servicios a alguien más en la cadena de producción [151].

## W

**WAF** Web Application Framework.

**workload** Cantidad de trabajo que una maquina produce o puede producir para un periodo específico de tiempo.

## Y

**YARN** Hadoop Yet Another Resource Negotiator, una plataforma responsable de la gestión de los recursos informáticos en clústers y utilizarlos para la programación de aplicaciones de los usuarios.

# Bibliografía

- [1] *Bootstrap*, Noviembre 2014. <http://getbootstrap.com/>.
- [2] *Chipotle Website*, Noviembre 2014. <http://www.chipotle.com/>.
- [3] *CodeCanyon Website*, Noviembre 2014. <http://codecanyon.net/>.
- [4] *Drupal*, Noviembre 2014. <https://www.drupal.org/>.
- [5] *Drupal Commerce Website*, Noviembre 2014. <https://drupalcommerce.org/>.
- [6] *Grunt*, Noviembre 2014. <http://gruntjs.com/>.
- [7] *Jigoshop Website*, Noviembre 2014. <https://www.jigoshop.com/>.
- [8] *Magento Community Edition Website*, Noviembre 2014. <http://www.magentocommerce.com/download>.
- [9] *MongoDB*, Noviembre 2014. <http://www.mongodb.org/>.
- [10] *Nodejs*, Noviembre 2014. <http://nodejs.org/>.
- [11] *The Open Software License 3.0 (OSL-3.0)*, Noviembre 2014. <http://opensource.org/licenses/OSL-3.0>.
- [12] *OpenCart Website*, Noviembre 2014. <http://www.opencart.com/>.
- [13] *osCommerce Website*, Noviembre 2014. <http://www.oscommerce.com/>.
- [14] *Performance Comparison Between Node.js and Java EE*, Noviembre 2014. <http://java.dzone.com/articles/performance-comparison-between>.
- [15] *PrestaShop Website*, Noviembre 2014. <http://www.prestashop.com/>.
- [16] *simpleCart Website*, Noviembre 2014. <http://simplecartjs.org/>.
- [17] *Spree Commerce Website*, Noviembre 2014. <http://spreecommerce.com/>.
- [18] *ThemeForest Website*, Noviembre 2014. <http://themeforest.net/>.
- [19] *WooCommerce Website*, Noviembre 2014. <http://www.woothemes.com/woocommerce/>.

- [20] *WordPress*, Noviembre 2014. <https://wordpress.org/>.
- [21] *WordPress.org Website*, Noviembre 2014. <https://wordpress.org/>.
- [22] *WP e-Commerce Website*, Noviembre 2014. <http://getshopped.org/>.
- [23] *Zen Cart Website*, Noviembre 2014. <http://www.zen-cart.com/>.
- [24] *Zend Framework*, Noviembre 2014. <http://framework.zend.com/>.
- [25] *Amazon*, Marzo 2015. <http://www.amazon.com/>.
- [26] *APACHE SOLR™ 4.10*, Febrero 2015. <http://lucene.apache.org/solr/>.
- [27] *BEST BUY*, Marzo 2015. <http://www.bestbuy.com/>.
- [28] *boilerplate*, Marzo 2015. <http://whatis.techtarget.com/definition/boilerplate>.
- [29] *Bower. A package manager for the web*, Febrero 2015. <http://bower.io/>.
- [30] *Browserify*, Febrero 2015. <http://browserify.org/>.
- [31] *The BSD 3-Clause License*, Marzo 2015. <http://opensource.org/licenses/BSD-3-Clause>.
- [32] *Bunyan*, Septiembre 2015. <https://atmospherejs.com/ongoworks/bunyan-logger>.
- [33] *Configure security policies enforced by the browser.*, Diciembre 2015. <https://atmospherejs.com/meteor/browser-policy>.
- [34] *The dynamic stylesheet language.*, Septiembre 2015. <https://atmospherejs.com/meteor/less>.
- [35] *Ebay*, Marzo 2015. <http://www.ebay.com/>.
- [36] *Electronic Funds Transfer (EFT) definition*, Marzo 2015. <http://searchwindowserver.techtarget.com/definition/Electronic-Funds-Transfer-EFT>.
- [37] *Fast Guide to DSL (Digital Subscriber Line)*, Enero 2015. <http://whatis.techtarget.com/reference/Fast-Guide-to-DSL-Digital-Subscriber-Line>.
- [38] *Functional Programming HOWTO*, Febrero 2015. <https://docs.python.org/2/howto/functional.html>.
- [39] *The History Of Ecommerce*, Marzo 2015. <http://www.ecommerce-web-hosting-guide.com/history-of-ecommerce.html>.
- [40] *Redis*, Febrero 2015. <http://redis.io/>.
- [41] *Ruby*, Febrero 2015. <https://www.ruby-lang.org/en/>.
- [42] *What is a gem?*, Febrero 2015. <http://guides.rubygems.org/what-is-a-gem/>.

- [43] *What is EDI (Electronic Data Interchange)?*, Marzo 2015. <http://www.edibasics.com/what-is-edi/>.
- [44] *¿Qué es GNU?*, Febrero 2015. <https://www.gnu.org/>.
- [45] Adrian Holovaty, Jacob Kaplan Moss: *The Django Book*, Febrero 2015. <http://www.djangobook.com/en/2.0/chapter05.html#the-mtv-or-mvc-development-pattern>.
- [46] Aghassipour, Alexander y Shajith Chacko: *Enterprise Apps Are Moving To Single-Page Design*, Febrero 2015. <http://techcrunch.com/2012/11/30/why-enterprise-apps-are-moving-to-single-page-design/>.
- [47] alanning: *Authorization package for Meteor*, Febrero 2016. <https://atmospherejs.com/alanning/roles>.
- [48] aldeed: *Collection2*, Julio 2015. <https://atmospherejs.com/aldeed/collection2>.
- [49] Allan Harris, Konstantin Haase: *Sinatra up and running*, capítulo Chapter 1, páginas 1–2. y O'Reilly Media, Inc, 2012.
- [50] Amberbit: *Introduction to Rack middleware*, Julio 2011. <https://www.amberbit.com/blog/2011/07/13/introduction-to-rack-middleware/>.
- [51] Answers, DataBase: *e-Commerce*, Febrero 2016. [http://databaseanswers.org/data\\_models/e\\_commerce/index.htm](http://databaseanswers.org/data_models/e_commerce/index.htm).
- [52] Ashkenas, Jeremy: *Backbone.js*, Enero 2015. <http://backbonejs.org/>.
- [53] Avram, Abel: *Ruby on Rails vs. Node.js at LinkedIn*, Marzo 2015. <http://www.infoq.com/news/2012/10/Ruby-on-Rails-Node-js-LinkedIn>.
- [54] Bango, Rey: *Getting Into Ember.js*, Febrero 2015. <http://code.tutsplus.com/tutorials/getting-into-ember-js--net-30709>.
- [55] Bidgoli, Hossein: *Electronic commerce: principles and practice*. Academic Press, 2002.
- [56] Bigcommerce:  
*What are wish lists and why are they important?*, Febrero 2015. <https://www.bigcommerce.com/ecommerce-answers/what-are-wish-lists-and-why-are-they-important/>.
- [57] Blagojevic, Vladimir: *The Ultimate Guide to Minimum Viable Products*, Marzo 2015. <http://scalemybusiness.com/the-ultimate-guide-to-minimum-viable-products/>.
- [58] Brasil, Matheus: *The problems with Single Page Applications*, Marzo 2015. <https://www.npmjs.com/package/eponymous>.
- [59] Brehm, Spike: *Isomorphic JavaScript: The Future of web Apps*, Marzo 2015. <http://nerds.airbnb.com/isomorphic-javascript-future-web-apps/>.

- [60] Clark, Cynthia L: *The American Economy: A Historical Encyclopedia*. ABC-CLIO, 2011.
- [61] Clifton, Marc: *What is Framework?*, Marzo 2015. <http://www.codeproject.com/Articles/5381/What-Is-A-Framework>.
- [62] Consulting, Forrester: *Smarter strategies for free Shipping. Understanding the true cost and benefist to retailers.*, February 2016. [https://www.ups.com/media/en/Smarter\\_Strategies\\_for\\_Free\\_Shipping.pdf](https://www.ups.com/media/en/Smarter_Strategies_for_Free_Shipping.pdf).
- [63] ConversionXL: *How to Design an eCommerce Checkout Flow That Converts*, Febrero 2016. <http://conversionxl.com/how-to-design-an-ecommerce-checkout-flow-that-converts/>.
- [64] ConversionXL: *Shopping Cart Abandonment: Why It Happens & How To Recover Baskets Of Money*, Febrero 2016. <http://conversionxl.com/shopping-cart-abandonment-how-to-recover-baskets-of-money/>.
- [65] Cook, Andrew y Tanya Goette: *Mobile Electronic Commerce: What Is It? Who Uses It? And Why Use It?* Communications of the IIMA, 6(4):4, 2015.
- [66] Csares, Joaquin: *Multi-datacenter Replication in Cassandra*, Febrero 2015. <http://www.datastax.com/dev/blog/multi-datacenter-replication>.
- [67] cxpartners: *What people see before they buy: Design guidelines for e-commerce product pages with eyetracking data*, Enero 2016. <https://www.cxpartners.co.uk/our-thinking/what-people-see-before-they-buy-design-guidelines-for-ecommerce-product-pages-with-eyetracking-data/>.
- [68] d3js: *D3 (official): A JavaScript visualization library for HTML and SVG.*, Septiembre 2015. <https://atmospherejs.com/d3js/d3>.
- [69] d3js: *D3 (official): A JavaScript visualization library for HTML and SVG.*, Septiembre 2015. <https://atmospherejs.com/mrt/underscore-string-latest>.
- [70] Davenport, Thomas H y Jeanne G Harris: *Competing on analytics: The new science of winning*. Harvard Business Press, 2007.
- [71] Dayley, Brad: *Node.js, MongoDB y Angular.js web Desarrollo*, capítulo Introduction. Addison-Wesley.
- [72] DB-Engines: *Article Solr*, Febrero 2015. <http://db-engines.com/en/article/Search+Engines>.
- [73] DB-Engines: *DB-Engines Ranking*, Enero 2015. <http://db-engines.com/en/ranking>.
- [74] designmodo: *5 UX Tips for Designing More Usable Registration Forms.*, Diciembre 2015. <http://designmodo.com/ux-tips-registration-forms/>.



- [75] Development Group *Meteor: Meteor Documentation*, Enero 2015. <http://docs.meteor.com/#/basic/>.
- [76] Docker, Inc: *Docker*, Noviembre 2014. <https://www.docker.com/>.
- [77] EconomicsHelp.org: *Economics – Profit and Revenue*, Septiembre 2015. <http://www.economicshelp.org/microessays/costs/profit-revenue/>.
- [78] Feroso, Jose: *PhoneGap Seeks to Bridge the Gap Between Mobile App Platforms*, Febrero 2015. <https://gigaom.com/2009/04/05/phonegap-seeks-to-bridge-the-gap-between-mobile-app-platforms/>.
- [79] Fielding, Roy T y Richard N Taylor: *Principled design of the modern Web architecture*. ACM Transactions on Internet Technology (TOIT), 2(2):115–150, 2002.
- [80] Foundation, The Apache Software: *Apache*, Enero 2015. <http://www.apache.org/>.
- [81] Foundation, The Apache Software: *Apache<sup>TM</sup> Tomcat*, Febrero 2015. <http://tomcat.apache.org/>.
- [82] FRONTIER: *What is Framework?*, Marzo 2015. [http://www.jfwk.com/what\\_is.html](http://www.jfwk.com/what_is.html).
- [83] Fruhling, Ann L y Lester A Digm: *The Impact of Electronic Commerce on Business Level Strategies*. J. Electron. Commerce Res., 1(1):13–22, 2000.
- [84] FYICenter.com: *What Is a JUnit Test Fixture?*, Septiembre 2015. [http://sqa.fyicenter.com/FAQ/JUnit/What\\_Is\\_a\\_JUnit\\_Test\\_Fixture\\_.html](http://sqa.fyicenter.com/FAQ/JUnit/What_Is_a_JUnit_Test_Fixture_.html).
- [85] Gamma, E.: *Design Patterns*. Addison-Wesley, 1995.
- [86] getcreditcardnumbers: *What is a CVV Number?*, Febrero 2016. <http://www.getcreditcardnumbers.com/cvv-numbers-credit-card>.
- [87] gnodi, Marzo 2015. <https://github.com/gnodi/danf>.
- [88] goodui.org: *A Good User Interface has high conversion rates and is easy to use*, Diciembre 2015. <https://www.goodui.org>.
- [89] Google: *AngularJs*, Noviembre 2014. <https://angularjs.org/>.
- [90] Google: *Material Design*, Diciembre 2015. <https://www.google.com/design/spec/material-design/introduction.html>.
- [91] Google: *Patterns Errors*, Diciembre 2015. <https://www.google.com/design/spec/patterns/errors.html>.
- [92] Grath, Ellis Mc.: *Top 9 E-Commerce Usability Guidelines*, Febrero 2016. <http://usabilitygeek.com/top-9-e-commerce-usability-guidelines/>.

- [93] Group, Meteor Development: *Login service for Facebook accounts*, Septiembre 2015. <https://atmospherejs.com/meteor/accounts-facebook>.
- [94] Group, Meteor Development: *Login service for Google accounts*, Septiembre 2015. <https://atmospherejs.com/meteor/accounts-google>.
- [95] Group, Meteor Development: *Login service for Twitter accounts*, Septiembre 2015. <https://atmospherejs.com/meteor/accounts-twitter>.
- [96] Group, PHP: *PHP*, Enero 2015. <http://php.net/>.
- [97] Group, PHP: *Ruby on Rails*, Enero 2015. <http://rubyonrails.org/>.
- [98] Hammer-Lahav, E.: *Link-based Resource Descriptor Discovery*, Marzo 2015. <https://tools.ietf.org/html/draft-hammer-discovery-03>.
- [99] Harrell, Jeff: *Node.js at PayPal*, Noviembre 2014. <https://www.paypal-engineering.com/2013/11/22/node-js-at-paypal/>.
- [100] Hipp, D. Richard: *Introduction to SQLite*, Enero 2015. <https://www.youtube.com/watch?v=f428dSRkTs4#t=48m15s>.
- [101] Hirschauer, Jim: *An example of how Node.js is faster than PHP*, Noviembre 2014. <http://www.appdynamics.com/blog/nodejs/an-example-of-how-node-js-is-faster-than-php/>.
- [102] Holst, Christian: *Fundamental Guidelines Of E-Commerce Checkout Design*, Febrero 2016. <https://www.smashingmagazine.com/2011/04/fundamental-guidelines-of-e-commerce-checkout-design/>.
- [103] Huang, Lin Shung, Alexander Moshchuk, Helen J Wang, Stuart Schecter y Collin Jackson: *Clickjacking: Attacks and Defenses*. En *USENIX Security Symposium*, páginas 413–428, 2012.
- [104] IBM: *What is Hadoop?*, Diciembre 2014. <http://www-01.ibm.com/software/data/infosphere/hadoop/>.
- [105] IBM: *What is Hive?*, Diciembre 2014. <http://www-01.ibm.com/software/data/infosphere/hadoop/hive/>.
- [106] IBM: *What is Pig?*, Diciembre 2014. <http://www-01.ibm.com/software/data/infosphere/hadoop/pig/>.
- [107] IBM: *What is ZooKeeper?*, Diciembre 2014. <http://www-01.ibm.com/software/data/infosphere/hadoop/zookeeper/>.
- [108] Impact, Load: *Node.js vs PHP – using Load Impact to visualize node.js efficiency*, Noviembre 2014. <http://blog.loadimpact.com/2013/02/01/node-js-vs-php-using-load-impact-to-visualize-node-js-eficiency/>.
- [109] Inc, Tilde: *Ember.js*, Enero 2015. <http://emberjs.com/>.

- [110] Investopedia: *Issuer Identification Number (IIN)*, Febrero 2016. <http://www.investopedia.com/terms/i/issuer-identification-number-iin.asp>.
- [111] iron: *Router*, Julio 2015. <https://atmospherejs.com/iron/router>.
- [112] jakesgordon: *A finite state machine javascript micro framework*, Septiembre 2015. <https://github.com/jakesgordon/javascript-state-machine>.
- [113] Jason Hong, Hesham Kamel, John Kodumal Francis Li James Lin: *Attitudes and Behavior Towards Password Use on the World Wide Web*, Diciembre 2015. <http://courses.ischool.berkeley.edu/i271a/f00/Passwords/index.html>.
- [114] Kantor, Ilya: *The Clickjacking attack, X-Frame-Options*, Febrero 2016. <http://javascript.info/tutorial/clickjacking#x-frame-options>.
- [115] Kissmetrics: *9 Ways to Make the Payment Process Easy for Online Customers*, Enero 2016. <https://blog.kissmetrics.com/easy-payment-process/>.
- [116] Kohavi, Ron y Roger Longbotham: *Online experiments: Lessons learned*. *Computer*, 40(9):103–105, 2007.
- [117] Kyle: *MongoDB and E-commerce*, Abril 2010. <http://web.archive.org/web/20110713174947/http://kylebanker.com/blog/2010/04/30/mongodb-and-ecommerce/>.
- [118] Labs, Artificial y contributors: *SANE STACK*, Marzo 2015. <http://sanestack.com/>.
- [119] Labs, Pivotal: *Jasmine*, Septiembre 2015. <http://jasmine.github.io/2.3/introduction.html>.
- [120] Lentz, Robin Schumacher & Arjen: *Dispelling the Myths*, Enero 2015. <http://web.archive.org/web/20110606013619/http://dev.mysql.com/tech-resources/articles/dispelling-the-myths.html>.
- [121] Library, Mac Developer: *What is Framework?*, Marzo 2015. <https://developer.apple.com/library/mac/documentation/MacOSX/Conceptual/BPFrameworks/Concepts/WhatAreFrameworks.html>.
- [122] linktionary: *Stateful and Stateless Connections*, Marzo 2015. <http://www.linktionary.com/s/state.html>.
- [123] Lovins, Julie B: *Development of a stemming algorithm*. MIT Information Processing Group, Electronic Systems Laboratory, 1968.
- [124] Ltd, Cubet Techno Labs Pvt.: *Sleekjs. Get ready to magic!*, Marzo 2015. <http://sleekjs.com/>.
- [125] Luhmann, Niklas, Howard Davis, John Raffan y Kathryn Rooney: *Trust; and, Power: two works by Niklas Luhmann*. Wiley Chichester, 1979.
- [126] Maamar, Zakaria: *Commerce, E-Commerce, and M-Commerce: What Comes Next?* 2003.

- [127] Marketing, Brick: *WHAT IS A CONVERSION RATE?*, Diciembre 2015. <http://www.brickmarketing.com/define-conversion-rate.htm>.
- [128] Marston, Tony: *An end-to-end eCommerce solution requires more than a fancy website*, Marzo 2012. <http://www.tonymarston.net/php-mysql/an-end-to-end-ecommerce-solution-requires-more-than-a-fancy-website.html>.
- [129] Marston, Tony: *API - application program interface*, Marzo 2015. <http://www.webopedia.com/TERM/A/API.html>.
- [130] matb33: *Meteor Collection Hooks*, Julio 2015. <https://atmospherejs.com/matb33/collection-hooks>.
- [131] MEAN.io: *The Friendly & Fun Javascript Fullstack for your next web application*, Diciembre 2014. [http://mean.io/#!/.](http://mean.io/#!/)
- [132] Meteorpedia: *Running your app on Android or iOS*, Enero 2015. <https://www.meteor.com/try/7>.
- [133] Meteorpedia: *Why Meteor?*, Enero 2015. [http://www.meteorpedia.com/read/Why\\_Meteor](http://www.meteorpedia.com/read/Why_Meteor).
- [134] Morrison, Kimberlee: *The Growth of E-commerce [Infographic]*, Marzo 2015. <http://www.adweek.com/socialtimes/data-growth-e-commerce-infographic/199692>.
- [135] mozilla: *JavaScript*, Febrero 2015. <https://developer.mozilla.org/en-US/docs/Web/JavaScript>.
- [136] mquandalle: *Bower for Meteor*, Julio 2015. <https://atmospherejs.com/mquandalle/bower>.
- [137] nemo64: *Highly configurable bootstrap integration.*, Septiembre 2015. <https://atmospherejs.com/nemo64/bootstrap>.
- [138] ongoworks: *Logical security for client-originated MongoDB collection operations*, Septiembre 2015. <https://atmospherejs.com/ongoworks/security>.
- [139] Park, Thomas: *Free themes for Bootstrap*, 2015. <http://bootswatch.com/>.
- [140] PayPal: *Accept credit card payments*, Febrero 2016. <https://developer.paypal.com/docs/integration/direct/accept-credit-cards/>.
- [141] PayPal: *Accept payments*, Febrero 2016. <https://developer.paypal.com/docs/accept-payments/>.
- [142] PayPal: *My Apps & Credentials*, Febrero 2016. <https://developer.paypal.com/developer/applications>.
- [143] Perl.org: *About Perl*, Febrero 2015. <https://www.perl.org/about.html>.
- [144] Rights, Intellectual Property: *Integrating intellectual property rights and development policy*. Londres, sèptanbr, 2002.

- [145] Robbins, Charlie: *A simple CLI tool for ensuring that a given script runs continuously (i.e. forever)*, Noviembre 2014. <https://github.com/nodejitsu/forever>.
- [146] Robbins, Charlie: *Scaling Isomorphic Javascript Code*, Marzo 2015. <http://blog.nodejitsu.com/scaling-isomorphic-javascript-code/>.
- [147] Robbins, Thom: *Best practices for your e-commerce shopping cart*, Febrero 2016. <http://www.imediaconnection.com/content/36794.asp>.
- [148] Rouse, Margaret: *Asynchronous Replication*, Febrero 2015. <http://searchdatabackup.techtarget.com/definition/asynchronous-replication>.
- [149] Rouse, Margaret: *Commodity Hardware*, Enero 2015. <http://whatis.techtarget.com/definition/commodity-hardware>.
- [150] Rouse, Margaret: *Crawler*, Marzo 2015. <http://searchsoa.techtarget.com/definition/crawler>.
- [151] Rouse, Margaret: *Vendor*, Enero 2015. <http://whatis.techtarget.com/definition/vendor>.
- [152] SAJAN: *4 tips to step up your global e-commerce localization strategy*, Septiembre 2015. <http://www.sajan.com/4-tips-step-global-e-commerce-localization-strategy/>.
- [153] Sandhu, Ravi S y Pierangela Samarati: *Access control: principle and practice*. Communications Magazine, IEEE, 32(9):40–48, 1994.
- [154] sanjo: *Velocity integration of the Jasmine testing framework*, Septiembre 2015. <https://atmospherejs.com/sanjo/jasmine>.
- [155] searchengineland: *What is SEO?*, Febrero 2015. <http://searchengineland.com/guide/what-is-seo>.
- [156] Sentry: *JavaScript Source Maps*, Diciembre 2014. <https://www.getsentry.com/docs/sourcemaps/>.
- [157] service efulfillment: *What is Order Fulfillment?*, Febrary 2016. <http://www.efulfillmentservice.com/2013/01/what-is-order-fulfillment/>.
- [158] Shin, Namchul: *Strategies for Competitive Advantage in Electronic Commerce*. J. Electron. Commerce Res., 2(4):164–171, 2001.
- [159] Shopify: *Ecommerce Shipping Toolbox: Apps, Tools, and Resources to Help You Streamline Your Business*, Febrary 2016. <https://www.shopify.com/blog/14881397-ecommerce-shipping-toolbox-apps-tools-and-resources-to-help-you-streamline-your>
- [160] shopify: *What is a merchant account?*, Febrary 2016. <https://www.shopify.com/faq/what-is-a-merchant-account>.

- [161] shopify: *What is a payment gateway?*, February 2016. <https://www.shopify.com/faq/what-is-a-payment-gateway>.
- [162] shopify: *What is a "third-party payment processor"?*, February 2016. <https://www.shopify.com/faq/what-is-a-third-party-processor>.
- [163] Skvorc, Bruno: *Best PHP Frameworks for 2014*, Febrero 2015. <http://www.sitepoint.com/best-php-frameworks-2014/>.
- [164] slapner: *MEEN stack using Ember.js based off MEAN*, Marzo 2015. <https://github.com/slapner/meen>.
- [165] statista: *E-commerce as percentage of total retail sales in the United States from 2000 to 2012*, Enero 2015. <http://www.statista.com/statistics/185351/share-of-e-commerce-in-total-value-of-us-retail-wholesale-trade-sales/>.
- [166] Surveys, Web Technology: *Usage of content languages for websites*, Septiembre 2015. [http://w3techs.com/technologies/overview/content\\_language/all](http://w3techs.com/technologies/overview/content_language/all).
- [167] Team, Meteor: *Meteor, the official guide. Accounts*, Febrero 2016. <https://www.meteor.com/accounts>.
- [168] Team, Meteor: *Meteor, the official guide. Security*, Febrero 2016. <http://guide.meteor.com/security.html>.
- [169] Team, Node.js: *Node plays a critical role in the technology stack of many high-profile companies*, Diciembre 2014. <http://nodejs.org/industry/>.
- [170] Team, Brick Marketing: *WHAT IS SEARCH ENGINE INDEX?*, Marzo 2015. <http://www.brickmarketing.com/define-search-engine-index.htm>.
- [171] Team Airbnb: *Airbnb*, Febrero 2015. <https://www.airbnb.com/>.
- [172] Team COKE: *COKE. Node.js MVC*, Marzo 2015. <http://coke-js.org/>.
- [173] Team Derby: *Derby*, Febrero 2015. <http://derbyjs.com/>.
- [174] Team Ember.js: *What are computed properties?*, Febrero 2015. <http://emberjs.com/guides/object-model/computed-properties/>.
- [175] Team Express.js: *Express.js. Fast, unopinionated, minimalist web Framework for Node.js*, Febrero 2015. <http://expressjs.com/>.
- [176] Team Flatiron: *Flatiron for Node.js*, Febrero 2015. <http://flatironjs.org/>.
- [177] Team GitHub: *PHP projects*, Febrero 2015. <https://github.com/search?l=PHP&q=stars%3A%3E0&ref=searchresults&type=Repositories>.
- [178] Team HotFrameworks: *ranking*, Marzo 2015. <http://hotframeworks.com/>.

- [179] Team jQuery: *What is jQuery?*, Febrero 2015. <http://jquery.com/>.
- [180] Team Koa: *Koa, next generation web Framework for Node.js*, Febrero 2015. <http://koajs.com/>.
- [181] Team Padrino: *The Elegant Ruby web Framework*, Febrero 2015. <http://www.padrinorb.com/>.
- [182] Team Phalcon: *The fastest PHP Framework*, Febrero 2015. <http://www.phalconphp.com/en/>.
- [183] Team radware: *FastView - Web Performance Optimization and Acceleration*, Marzo 2015. [http://www.radware.com/Products/FastView/?utm\\_source=strangeloop&utm\\_medium=slforward&utm\\_campaign=slmoving](http://www.radware.com/Products/FastView/?utm_source=strangeloop&utm_medium=slforward&utm_campaign=slmoving).
- [184] Team radware: *The Psychology of Web Performance*, Marzo 2015. <http://www.websiteoptimization.com/speed/tweak/psychology-web-performance/>.
- [185] Team SocketStream: *The future is Realtime*, Marzo 2015. <http://socketstream.org/>.
- [186] Team Symfony: *What is Symfony?*, Febrero 2015. <http://symfony.com/what-is-symfony>.
- [187] Team Twitter: *Improving performance on twitter.com*, Marzo 2015. <https://blog.twitter.com/2012/improving-performance-on-twittercom>.
- [188] Team YAHOO!: *Introducing Mojito*, Febrero 2015. <https://developer.yahoo.com/cocktails/mojito/docs/intro/>.
- [189] techopedia: *Object-Relational Mapping (ORM)*, Marzo 2015. <http://www.techopedia.com/definition/24200/object-relational-mapping--orm>.
- [190] techopedia: *Single Point of Failure (SPOF)*, Febrero 2015. <http://www.techopedia.com/definition/4351/single-point-of-failure-spod>.
- [191] TechTarget: *¿ What is out of the box ?*, Diciembre 2014. <http://searchcio.techtarget.com/definition/out-of-the-box>.
- [192] trentm: *Bunyan*, Septiembre 2015. <https://github.com/trentm/node-bunyan>.
- [193] Trudeau, Richard J.: *Introduction to Graph Theory*.
- [194] Velocity: *Velocity*, Septiembre 2015. <https://velocity.readme.io/>.
- [195] Velocity: *Velocity, a Meteor specific test-runner*, Septiembre 2015. <https://atmospherejs.com/velocity/core>.
- [196] Weekly, Computer: *Write once, run anywhere?*, Febrero 2015. <http://www.computerweekly.com/feature/Write-once-run-anywhere>.
- [197] West, Mike: *An Introduction to Content Security Policy*, Febrero 2016. <http://www.html5rocks.com/en/tutorials/security/content-security-policy>.
- [198] Zannr: *Isomorphic JavaScript Frameworks*, Febrero 2015. <http://www.zannr.com/isomorphic-javascript-frameworks.html>.

- [199] Zhang, Jie, Paul W Farris, John W Irvin, Tarun Kushwaha, Thomas J Steenburgh y Barton A Weitz: *Crafting integrated multichannel retailing strategies*. Journal of Interactive Marketing, 24(2):168–180, 2010.
- [200] Zheng, Qin, Shundong Li, Yi Han, Jinchun Dong, Lixiang Yan y Jun Qin: *Fundamentals of E-commerce*. En *Introduction to E-commerce*, páginas 3–76. Springer, 2009.
- [201] Zviran, Moshe y William J Haga: *Password security: an empirical study*. Journal of Management Information Systems, páginas 161–185, 1999.



# Apéndice A

## Gestión de Catálogos *handled* con una *Base de dato* relacional

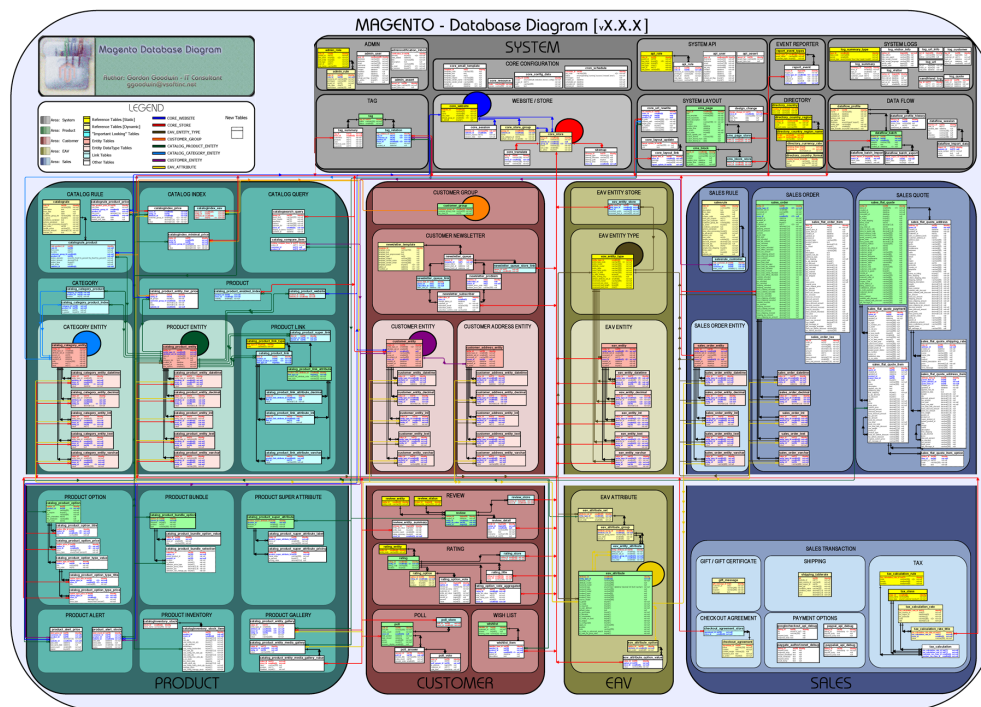


Figura A.1: Schemas de *Magento e-Commerce Framework*.

## Apéndice B

### ¿Qué es *Hadoop*® *B*? [104]

*Apache*<sup>TM</sup> *Hadoop*® *B* es un proyecto software *Open Source* que permite el procesamiento distribuido de gran cantidad de datos a través de *clusters* de *servidores*. Esta diseñado para *scale up* desde un único *servidor* a miles de máquinas, con un muy alto grado de tolerancia al fallo. En lugar de confiar en *high-end hardware*, la flexibilidad de estos *clusters* proviene de la habilidad del *software* en detectar y manejar fallos en la *Application Layer*.

#### Arquitectura *high-level*

*Apache*<sup>TM</sup> *Hadoop*® *B* tiene dos pilares:

- YARN asigna *CPU*, memoria y almacenamiento para las aplicaciones corriendo en *clusters Hadoop*® *B*. La primera generación de *Hadoop*® *B* podía correr solo aplicaciones MapReduce. YARN permite que otros entornos de aplicaciones (como *SPARK*) para ejecutarse en *Hadoop*® *B*, lo que abre un mundo de posibilidades.
- HDFS es un sistema de archivos que se extiende por todos los nodos en su *cluster Hadoop*® *B* para almacenamiento de datos. Se conecta entre si a los sistemas de archivos en muchos nodos locales para convertirlos en un sistema de archivos grandes.

*Hadoop*® *B* se complementa con un ecosistema de proyectos de *Apache*<sup>TM</sup>, como Pig [106], Hive [105] y Zookeeper [107], para extender el valor de *Hadoop*® *B* y mejorar su *usability*.

#### Entonces, ¿Cuál es el problema?

*Hadoop*® *B* cambia la economía y la dinámica de la computación a gran escala. Su impacto puede reducirse a cuatro características sobresalientes.

*Hadoop*® *B* permite una solución de computación que es:

- **Scalable**- Nuevos nodos pueden ser agregados según sean necesarios, y agregados sin la necesidad de cambiar el formato de los datos, como los datos son cargados, como los *jobs* son escritos, o las aplicaciones en la parte superior.
- **Cost effective**- *Hadoop*® B trae computación paralela masiva a las maquinas de los servidores. El resultado es una considerable disminución en los costos por *terabyte* de almacenamiento. lo cual hace asequible para modelar todos sus datos.
- **Flexible**- *Hadoop*® B no tiene esquema, y puede absorber cualquier tipo de datos, estructurados o no, desde cualquier numero de fuentes. Los datos de múltiples fuentes pueden ser unidos y se agregan arbitrariamente para permitir análisis que ningún otro sistema puede proporcionar.
- **Fault tolerant**- Cuando se pierde un nodo, el sistema redirige el trabajo a otra ubicación de los datos para seguir con el procesamiento sin perder el ritmo.

## Apéndice C

# Conexiones *Stateful* y *Stateless*

Mantener el estado o ser *Stateful* significa que algunos dispositivos mantienen *track* de otro dispositivo o una conexión, ya sea temporal o largo periodo de tiempo. Cuando se agrega el nombre de una persona en una libreta de direcciones y se observa su cumpleaños y teléfono, se podría decir que se esta manteniendo el estado de esa persona. En la *web* una *cookie* es un mecanismo *Stateful* que permite a los *Webservers* mantener *track* de información de las personas, tal como se describe en un momento [122].

Una conexión *Stateful* es una en la cual cierta información sobre una conexión entre dos sistemas es retenida para uso futuro. En algunos casos, la conexión es mantenida abierta incluso a través de dos sistemas que podrían no estar transmitiendo información(*i.e.*, la conexión en si misma mantiene el estado) [122].

En contraste, una conexión *Stateless* es aquella en que no hay información retenida por el *sender* ó *receiver*. El *sender* transmite un paquete al *receiver* sin esperar ninguna confirmación de recepción. El *receiver* recibe el paquete sin ninguna configuración de la conexión anterior [122].

## Apéndice D

### *Pseudo-schemas e-Commerce*

```
1
2 db.products.find({'_id': ObjectId("4bd87bd8277d094c458d2fa5")});
3
4 {
5   _id      : ObjectId("4bd87bd8277d094c458d2a43"),
6   title    : "A Love Supreme [Original Recording Reissued]",
7   author   : "John Coltrane",
8   author_id : ObjectId("4bd87bd8277d094c458d2fa5"),
9   details  : {
10     number_of_discs: 1,
11     label: "Impulse Records",
12     issue_date: "December 9, 1964",
13     average_customer_review: 4.95,
14     asin: "B0000A118M"
15   },
16   pricing  : {
17     list: 1198,
18     retail: 1099,
19     savings: 99,
20     pct_savings: 8
21   },
22   categories : [
23     ObjectId("4bd87bd8277d094c458d2a43"),
24     ObjectId("4bd87bd8277d094c458d2b44"),
25     ObjectId("4bd87bd8277d094c458d29a1")
26   ]
27 }
```

Código D.1: Búsqueda en MongoDB

```
1
2 {
3   '_id': objectid('4b980a6dea2c3f4579da141e'),
4   'user_id': objectid('4b980a6dea2c3f4579a4f54'),
5   'state': 'cart',
```

```

6   'line_items': [
7     {
8       'sku': 'jc-432',
9       'name': 'John Coltrane: A Love Supreme',
10      'retail_price': 1099
11    },
12    {
13      'sku': 'ly-211',
14      'name': 'Larry Young: Unity',
15      'retail_price': 1199
16    }
17  ],
18  'shipping_address': {
19    'street': '3333 Greene Ave.',
20    'city': 'Brooklyn',
21    'state': 'NY',
22    'zip': '11216'
23  },
24  'subtotal': 2199
25 }

```

Código D.2: Estructura de una *orden*.

```

1
2   db.orders.ensureIndex({'line_items.sku': 1});
3   db.orders.find({'line_items.sku' => 'jc-431'});

```

Código D.3: Consulta eficiente con *Secondary indexes*.

```

1
2   map = "
3     function(){
4       emit(this['shipping_address']['zip'], {total: this.total})
5     }
6   "
7
8   reduce = "
9     function(key, values) {
10      var sum = 0;
11      values.forEach(function(doc) {
12        sum += doc.total;
13      })
14
15      return {total: sum};
16    }"
17
18
19   db.orders.mapReduce(map, reduce, {out: 'order_totals_by_zip'});

```

Código D.4: Ejemplo de comando *map-reduce*.

```

1
2 db.orders.update(
3   {
4     '_id': order_id,
5     'line_items.sku': 'jc-431'
6   },
7   {
8     '$set': {'line_items.$.quantity': 2}
9   }
10  );

```

Código D.5: Ejemplo de uso de *position operator*.

```

1 db.orders.update(
2   {'_id': order_id},
3   {
4     '$push': {
5       'line_items': {
6         'sku': 'md-12',
7         'price': 2500,
8         'title': 'Basketball'
9       }
10    },
11    '$inc': {'subtotal': 2500}
12  }
13 );

```

Código D.6: Ejemplo del operador *\$push*.

```

1 {
2   '_id': ObjectId('4b980a6dea2c3f4579da432a'),
3   'sku': 'jc-431',
4   'state': 'available',
5   'expires': null,
6   'order_id': null
7 }

```

Código D.7: Ejemplo de *document* para un producto.

```

1 query = {'sku': 'jc-431', 'state': 'available'};
2
3 update = {
4   '$set': {
5     'state': 'cart',
6     'order_id': order_id,
7     'expires': Date.now() + 15 * 60
8   }
9 };
10
11 item = db.inventory.findAndModify(query: query, update: update);

```

Código D.8: Marcando un producto con tiempo de espiración.

```
1 db.inventory.update(  
2   {  
3     'state': 'cart',  
4     'expires': {'$lt': Date.now()}},  
5   {  
6     '$set': {  
7       'state': 'available',  
8       'expires': null,  
9       'order_id': null  
10    }  
11  },  
12  {multi: true}  
13 );
```

Código D.9: Ejemplo de *script* corriendo *background*.



## Apéndice E

### UI de los formularios

---

Email Address

Invalid email

Password

Password must be at least 8 characters long.

Password must contain at least one number or symbol

REGISTER

Sign In

Figura E.1: Errores al enviar el formulario de creación de una nueva cuenta.

**Email Address**

Invalid email

**Password**

Password is required.

**Sign In**

[Reset password](#)      [Register](#)

Figura E.2: Errores al enviar vacío el formulario de autenticación.

**Current Password**

Password is required.

**password**

Password must be at least 8 characters long.

Password must contain at least one number or symbol

Figura E.3: Errores del formulario de actualización de contraseña tras enviarlo vacío.

**Current Password**

**Password**

Password must contain at least one number or symbol

Figura E.4: Ingreso de nueva *contraseña fácil*.

---

Incorrect password

**Current Password**

**Password**

---

Figura E.5: Ingreso de contraseña actual incorrecta.

Shop general settings update failed. Error: Name is required×

**Name**

Name is required

**Email**

**Description**

**Keywords**

Figura E.6: Errores en el formulario tras enviarlo. El campo *Name* es obligatorio.

## Address

**Company**

**Full name**

Full name is required

**Address 1**

Address 1 is required

**Address 2**

**City**

City is required

**Region**

Region is required

**Postal**

Postal is required

**Country**

Country is required

**Phone**

Phone is required

Save Changes

Figura E.7: Comentarios del formulario tras error de actualización.

**Method Name**

Method Name is required

**Public Label**

Public Label is required

**Group**

Group is required

Enabled

**Cost**

Figura E.8: Formulario de creación de *Shipping* tras enviarlo vacío.

## Apéndice F

# Soluciones de otras plataformas *e-Commerce*

### Add an address

**Full name:**

**Address line 1:**   
Street address, P.O. box, company name, c/o

**Address line 2:**   
Apartment, suite, unit, building, floor, etc.

**City:**

**State/Province/Region:**

**ZIP:**

**Country:**

**Phone number:**  [Learn more](#)

### Additional Address Details [\(What's this?\)](#)

Preferences are used to plan your delivery. However, shipments can sometimes arrive early or later than planned.

**Weekend Delivery:**

**Security access code:**   
For buildings or gated communities

[Save & Add Payment Method](#)

[Save & Continue](#)

Figura F.1: Formulario para agregar una dirección en *Amazon*.

Amazon accepts all major credit & debit cards.



Enter your credit card (Step 1 of 2)

Card number

Name on card

Expiration date  
03 ▾ 2016 ▾

Cancel Next

Figura F.2: Formulario de *Amazon* para agregar una tarjeta.

## Add a card

Card type ▾

Card number

Expiration MM/YY CSC (3 digits)

Select a billing address ▾

Save

Figura F.3: Formulario de *PayPal* para agregar una tarjeta.

Add discount	—
Subtotal	\$345
Add shipping	—
Taxes	\$0
<b>Total</b>	<b>\$345</b>

Figura F.4: Resumen del carro de compra del sitio *Shopify*.

Products / Add product [Cancel](#) [Save product](#)

**Title**

**Description**

Rich text editor toolbar: A, B, I, bulleted list, numbered list, link, unlink, table, image, video, fullscreen.

**Visibility**

No applicable sales channels  
[Review your sales channel settings](#)

---

**Organization**

Product type:

Vendor:

---

**Collections**

Add this product to a collection so it's easy to find in your store.

---

**Tags** [View all tags](#)

**Images** [Upload image](#)

Drop files to upload

Figura F.5: Formulario de *Shopify* para agregar un producto.

**Title**

**Description**

Rich text editor toolbar: A, B, I, bulleted list, numbered list, link, unlink, table, image, video, fullscreen.

Figura F.6: Sección del formulario de *Shopify* para agregar un título y una descripción.



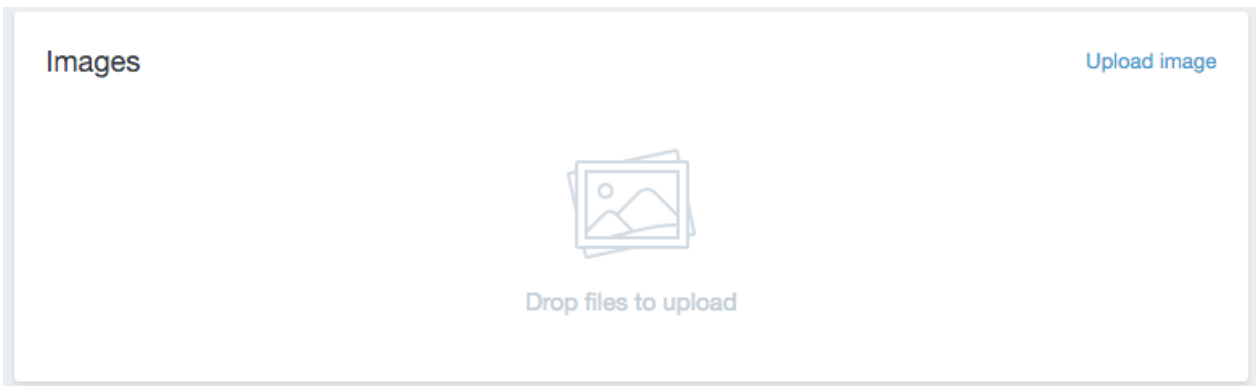


Figura F.7: Sección del formulario de *Shopify* para agregar imágenes de producto.

The image shows a rectangular box representing the 'Inventory' section of a Shopify product form. At the top left, the word 'Inventory' is written in a dark font. Below it, there are two input fields: 'SKU (Stock Keeping Unit)' on the left and 'Barcode (ISBN, UPC, etc.)' on the right. Underneath these fields is a dropdown menu labeled 'Inventory policy' with the selected option being 'Don't track inventory'. A small downward arrow is visible on the right side of the dropdown menu.

Figura F.8: Sección del formulario de *Shopify* para agregar inventario al producto.

This image is a close-up of the 'Inventory policy' and 'Quantity' fields from the previous screenshot. The 'Inventory policy' dropdown menu is highlighted with a blue border and shows the selected option 'Shopify tracks this product's inventory'. To its right, the 'Quantity' input field contains the number '17'. Below these fields, there is a checkbox that is currently unchecked, with the text 'Allow customers to purchase this product when it's out of stock' next to it.

Figura F.9: Sección *Inventory* del formulario de *Shopify* con opción *Tracking* aceptada.

**Organization**

Product type

Shirts

Vendor

Nike

**Collections**

Search for collections

Add this product to a collection so it's easy to find in your store.

Tags [View all tags](#)

Vintage, cotton, summer

Figura F.10: Sección del formulario de *Shopify* para agregar información de la organización del producto.

**Pricing**

Price

\$ 0

Compare at price

\$

Charge taxes on this product

Figura F.11: Sección del formulario de *Shopify* para agregar información del precio.

**Shipping**

Weight

0.0 kg ▾

This product requires shipping

Figura F.12: Sección del formulario de *Shopify* para agregar información de *shipping* al producto.

**Variants** Cancel

Does this product come in multiple variations like size or color?




Option name	Option values	
Size	Separate options with a comma	
Color	Separate options with a comma	
Material	Separate options with a comma	

Figura F.13: Sección del formulario de *Shopify* para agregar variantes de un producto.

**Variants** Cancel

Does this product come in multiple variations like size or color?

Option name	Option values
Size	s x m x l x <span>🗑️</span>
Color	red x white x   <span>🗑️</span>

[Add another option](#)

Modify the variants to be created:

	Variant	Price	SKU	Barcode	Inventory
<input checked="" type="checkbox"/>	s • red	\$ 0			17
<input checked="" type="checkbox"/>	s • white	\$ 0			17
<input checked="" type="checkbox"/>	m • red	\$ 0			17
<input checked="" type="checkbox"/>	m • white	\$ 0			17
<input checked="" type="checkbox"/>	l • red	\$ 0			17
<input checked="" type="checkbox"/>	l • white	\$ 0			17

Figura F.14: Sección *Variants* del formulario de *Shopify* con dos variantes del producto.

## BRASS HEXAGON PLANTER

\$65.00

LARGE [\$65.00] ▼

add to cart

Figura F.15: Sección del *website* de *leif* para agregar elementos al carro.

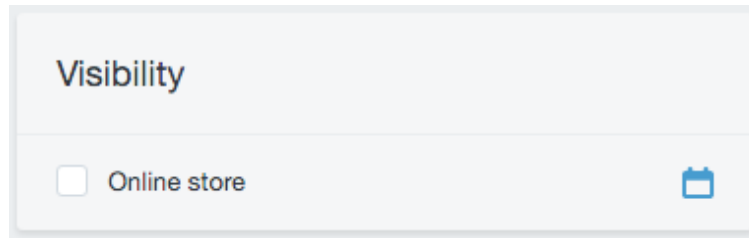


Figura F.16: Sección *Visibility* del formulario de *Shopify* del producto.

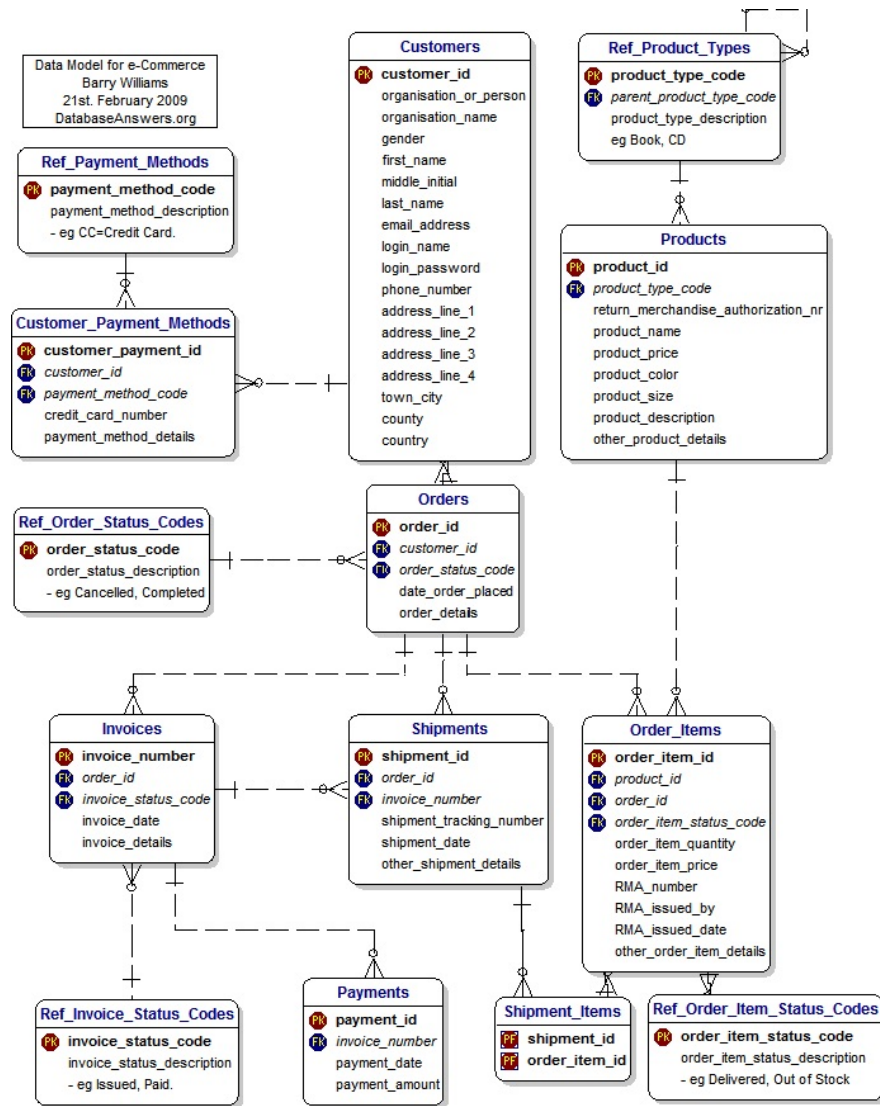


Figura F.17: Modelo de Datos para *e-Commerce* obtenido desde el sitio *DataBase Answers* [51].

Order Number	Order Date	Grand Total	Order Status	Operations
160310001021513679	3/10/2016 12:35 PM	US\$ 14.80	Payment Confirmed	
160201001023593281	2/1/2016 9:46 PM	US\$ 12.48	Full Shipment	Available Actions ▾
160201001022788703	2/1/2016 3:26 PM	US\$ 14.37	Completed	
151208001054199558	12/8/2015 4:06 PM	US\$ 0.00	Completed	
150813001059276752	8/14/2015 1:18 AM	US\$ 11.99	Completed	

Figura F.18: Lista de Órdenes para el website dealextreme DX.




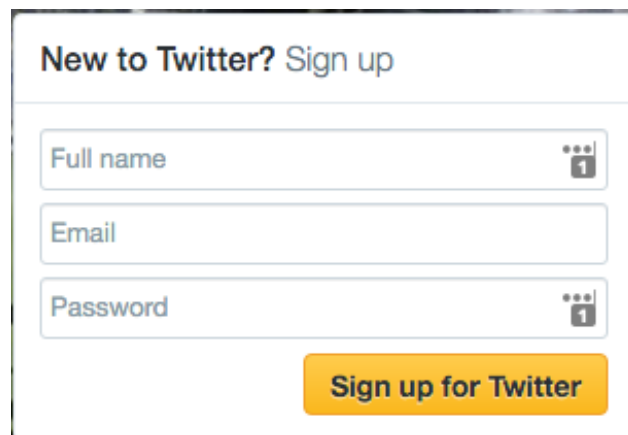
Order Detail							
<b>Shipping Options</b>							
Name: Alvaro Balmaceda							
Street1: Hernando de Aguirre 268 suite 302, Providencia							
Street2:							
City: Santiago							
State/Province: Metropolitana							
Country/Region: CHILE							
PostalCode: 7510047							
Phone: 56998393260							
Shipping Method: Standard							
<b>Memos</b>							
<b>Items</b>							
SKU	Product Name	Purchase Price	Discount	Ordered Qty	Gift Qty	Shipped Qty	Operations
217252	 Bicycle Cycling Quick Release Aluminum Alloy Rear Rack - Black	US\$ 30.19	US\$ 0.00	1	0	1	Review
105432	 Replacement Aluminum Alloy Kickstand for Bicycle Bike	US\$ 17.50	US\$ 0.00	1	0	1	Review
108748	 Ergonomic Outdoor Sports Short Plush Fabric Mask - Black	US\$ 3.69	US\$ 0.00	3	0	3	Review Re-Order
						Order Subtotal: <b>US\$ 58.76</b>	
						Shipping Cost: <b>US\$ 1.70</b>	
						Handling Fee: <b>US\$ 0.00</b>	
						Discount Total: - <b>US\$ 1.70</b>	
						<b>Grand Total: US\$ 58.76</b>	

Figura F.19: Detalles de una Orden para el website dealextreme DX.

## Apéndice G

### Ejemplos de Formularios



New to Twitter? Sign up

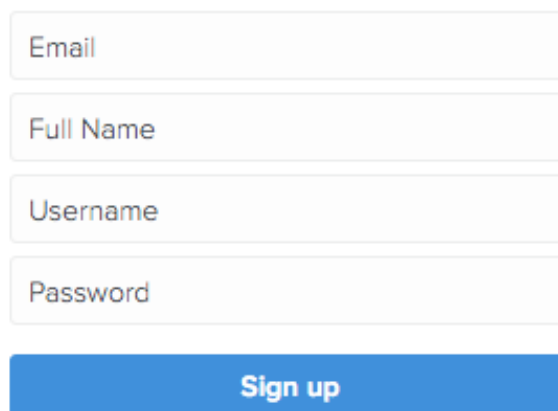
Full name

Email

Password

Sign up for Twitter

Figura G.1: Formulario de creación de usuario para *Twitter*.



Email

Full Name

Username

Password

Sign up

Figura G.2: Formulario de creación de usuario para *Instagram*.

Sign up to start your free month

Create your account:

Email Address

Choose a password (4-50 characters)

**Register**

**Just two more steps and you're done!  
We hate paperwork too.**

Figura G.3: Formulario de creación de usuario para *Netflix*.

Be great at what you do  
Get started - it's free.

First name

Last name

Email

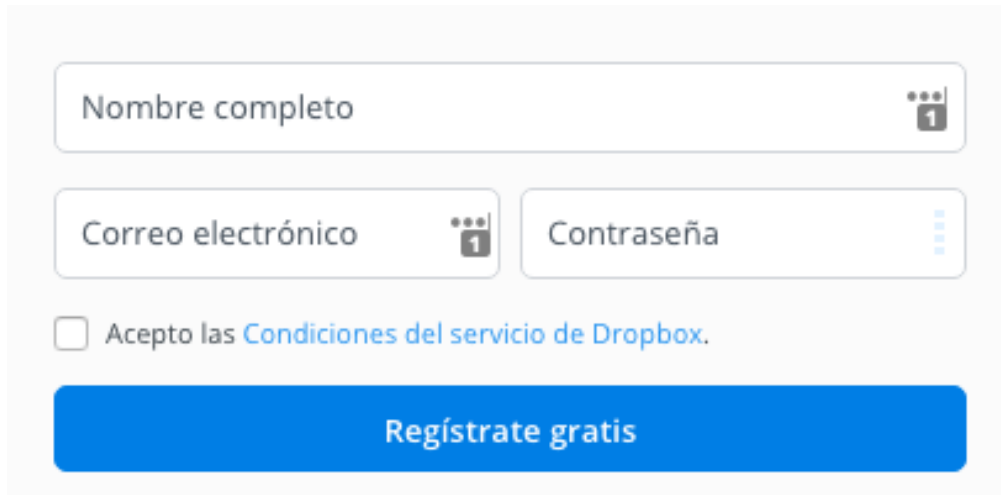
Password (6 or more characters)


By clicking Join now, you agree to LinkedIn's User Agreement, Privacy Policy, and Cookie Policy.



**Join now**

Figura G.4: Formulario de creación de usuario para *LinkedIn*.





Nombre completo 

Correo electrónico  Contraseña 

Acepto las [Condiciones del servicio de Dropbox.](#)

**Regístrate gratis**

Figura G.5: Formulario de creación de usuario para *Dropbox*.