

LAS TRES CAPAS DE LOS SISTEMAS DE INFORMACIÓN WEB CON (UNA) JAVA

Primera Edición

*Alex Santiago Cevallos
Culqui*



Universidad
Técnica de
Cotopaxi

AUTORES

Mgs. Alex Santiago Cevallos Culqui
Universidad Técnica de Cotopaxi

PhD. Gustavo Rodríguez Bárcenas
Universidad Técnica de Cotopaxi

Ing. Mgs. Jorge Bladimir Rubio Peñaherrera
Universidad Técnica de Cotopaxi

Mgs. Fausto Alberto Viscaíno Naranjo
Universidad Técnica de Cotopaxi

Mgs. Segundo Humberto Corrales Beltrán
Universidad Técnica de Cotopaxi

DIRECCIÓN EDITORIAL

Comité Editorial
Universidad Técnica de Cotopaxi

DIAGRAMACIÓN

Wilmer Stalin Chiluisa LLumiquinga

Reservados todos los derechos. Queda prohibida la reproducción total o parcial de esta obra, se deja constancia que el contenido del texto es original y de absoluta responsabilidad de sus autores. La infracción de dichos derechos puede constituir un delito contra la propiedad intelectual.

© Copyright

ISBN DIGITAL 978-9978-395-48-6

PRIMERA EDICIÓN, 2017

Índice

Resumen	11
Sección 1: Overview.....	13
Capítulo I.....	16
Introducción a las Aplicaciones Web en tres capas	16
Patrón de arquitectura MVC (Modelo Vista Controlador)	17
Ventajas del patrón MVC	19
Desventajas del patrón MVC.....	21
Frameworks MVC.....	21
Spring Framework.....	21
Framework Ruby on Rails	23
Framework CakePHP.....	24
Capítulo II	26
Play Framework	26
Historia.....	29
Módulos de Play Framework	31
Capítulo III.....	33
Configurando Play Framework.....	33
Prerrequisitos.....	35
Creación de un proyecto en play framework.....	44
Sección 2: Primera Capa de Presentación	56
Capítulo IV	58
Lenguajes de Interfaz.....	58
El hipertexto en la actualidad	60
¿Qué es Html?.....	62

Hojas de estilo	66
Formas de usar estilos y lenguajes complemento	69
Javascript.....	72
JQuery	74
Html, CSS, JavaScript y JQuery.....	77
Aplicando leguajes interactivos en ambiente Play Framework	78
Capítulo V.....	78
Boceto de la interfaz deseada	79
Html y CSS.....	81
Jquery	90
JavaScript Sección Detalle	94
JavaScript Sección Compras	98
Sección 3: Segunda capa de negocio	102
Lenguajes de capa de negocio (JAVA)	104
Capítulo VI	104
Funciones de la capa de Negocio	105
Historia de Java	106
Qué es JAVA.	108
Características de Java.....	109
Ventajas de Java	111
Historia del paradigma orientado a objetos.....	112
El Paradigma Orientado a Objetos	113
Ventajas del Paradigma Orientado a Objetos	114
Elementos de la P.O.O.	114
Capítulo VII.....	119
Aplicando JAVA en el ambiente Play Framework.....	119

Diseño UML de la arquitectura del Software.....	122
Exportación de la arquitectura a clases Java.....	125
Desarrollo de la estructura de las clases	126
Desarrollo de las funcionalidades.....	131
Sección 4: Tercera Capa de Datos.....	144
Los ORMs (Object Relation Mapping) y las Bases de Datos	146
Capítulo VIII	146
Ventajas del Mapeo de Objetos Relacional.....	148
Desventajas del Mapeo de Objetos Relacional.	148
Tipos de ORMs	150
Bases De Datos Relacionales.....	152
Elementos de una Base De Datos Relacional.	153
Manejo De Bases De Datos Relacionales	154
Capítulo IX.....	157
Configurando ORM con un almacén de datos PostgreSQL.....	157
Creación de un almacén de datos en PostgreSql.....	159
Configuración de la puerta de acceso a la tercera capa.....	161
Sección 5: Interacción entre capas.....	164
Tags para interacción entre segunda y tercera capa.....	166
Capítulo X	166
Asignar los tags o palabras claves a las clases de nuestro modelo.....	168
Asignar los tags o palabras claves de las relaciones entre clases.....	169
Ejecución de prueba para almacenamiento de datos	172
Capítulo XI	175
Tags para interacción entre primera y segunda capa	175
Asignar los tags o palabras claves a las clases de nuestro modelo.....	177

Asignar los tags o palabras claves de las relaciones entre clases.....	178
Ejecución de prueba para almacenamiento de datos	181
Tags para interacción entre primera y segunda capa.....	184
Capítulo XI.....	184
Visualizar la lista de productos extrayéndolos desde la capa de negocio	188
Registrar el producto seleccionado y su cantidad en un carro de compras	192
Eliminar ítems del carro de compras y cálculo del total de la compra...	196
Sección 6: Casos de éxito de la Universidad Técnica de Cotopaxi	201
Primer lugar en concurso nacional de software	203
Orígenes.....	215
Servicios	216
BIBLIOGRAFÍA.....	217

Índice de figuras

Figura 1.1.Prototipos del Patrón MVC. Adaptado de (Martínez, 2013)	18
Figura 1.2.Estructura del Patrón MVC. Adaptado de (helloacm.com, 2017)	18
Figura 1.3 Estructura de Spring Framework. Adaptado de (Ceresola, 2012)	23
Figura 1.5.Estructura del Framework CakePHP. Adaptado de (CakePHP, 2012)	25
Figura 3.1.Archivos de Play Framework.....	36
Figura 3.2.Comando para verificar versión de Java	37
Figura 3.3.Tipos de JDK 7	38
Figura 3.4.Carpetas de instalación Java 7	39
Figura 3.5.VARIABLES DE ENTORNO Java.....	40
Figura 3.6.Variedad de IDEs de Eclipse	42
Figura 3.7.Ruta y ejecución del IDE Eclipse.....	43
Figura 3.8.Versiones del motor de Base de Datos PostgreSql.....	44
Figura 3.9.Creación de un proyecto en Play Framework	46
Figura 3.10.Configuración de atajos para uso de comandos.....	47
Figura 3.11.Comando de ejecución de un proyecto Play Framework	48
Figura 3.12.Ejecución del comando Eclipsify.....	50
Figura 3.13.Importación de un proyecto Play Framework hacia Eclipse (Parte 1)	51
Figura 3.14.Importación de un proyecto Play Framework hacia Eclipse (Parte 2)	52

Figura 3.15.Estructura del proyecto Play Framework y sus opciones para ejecutarlo.....	55
Figura 4.1.Esquema de las partes de un elemento HTML. Adaptado de (Libros Web, 2017)	64
Figura 4.2.Separación de contenidos y presentación. Adaptado de (Libros Web, 2017)	66
Figura 4.3.Reglas CSS. Adaptado de (Libros Web, 2017)	67
Tabla 4.1.Elementos principales de JavaScript. Adaptado de (Desarrollo Web, 2001).....	74
Tabla 4.2.Métodos para envío de información por formularios. Adaptado de (González, 2009)	76
Figura 5.1.Secciones de la aplicación caso de estudio	81
Figura 5.2.Estructura básica de un archivo Html.....	82
Figura 5.3.Código HTML y CSS de la Sección 1	84
Figura 5.4.Código HTML y CSS de la Sección 2	85
Figura 5.5.Código HTML y CSS de la Sección 3	85
Figura 5.6.Código HTML y CSS de la Sección 4	86
Figura 5.7.Código HTML y CSS de la Sección 5	86
Figura 5.8.Código HTML y CSS de la Sección 6	87
Figura 5.8.Código HTML y CSS de la Sección 6	88
Figura 5.8.Código HTML y CSS de la Sección 6	89
Figura 5.10.Interfaz de la aplicación de caso de estudio	89
Figura 5.11.Ubicación para los archivos tipo: imagen, javascript y css	91
Figura 5.12.Llamada a las librerías javascript y css	92
Figura 5.13.Código CSS y JQuery para Slider	92
Figura 5.14.Código HTML para Slider	93

Figura 5.15.Slider de imágenes en la interfaz	94
Figura 5.16.Código HTML de la Sección Detalle del Producto.....	95
Figura 5.17.Llamado a una función JavaScript	96
Figura 5.18.Funció n JavaScript	97
Figura 5.19.Sección del detalle del producto en interfaz	97
Figura 5.20.Llamado a la función eliminar de JavaScript	100
Figura 5.21.Validación de confirmación para la función JavaScript de eliminar.....	101
Figura 5.22.Sección de carro de compras en interfaz.....	101
Figura 6.1.Organización de un programa orientado a objetos. Adaptado de (Edukativos, 2013)	113
Figura 6.2.Ejemplos Clase.	115
Figura 6.3.Ejemplos Objetos.	115
Figura 6.4.Atributos de los Objetos.	116
Figura 6.5.Mensaje en los Objetos.....	117
Figura 7.1.Creando diagramas de clases en Eclipse	123
Figura 7.2.Diseñando un diagrama de clases con AmaterasUML... <td>124</td>	124
Figura 7.3.Exportación de diagrama de clases a código JAVA.....	125
Figura 7.4.Generación de las funciones Getters and Setters en clases Java	127
Tabla 7.1.Relaciones de dependencias en la aplicación de caso de estudio	129
Tabla 7.2.Relaciones de composición en la aplicación de caso de estudio	131
Figura 7.5.Ejecución de la lógica de negocio de la aplicación caso de estudio	137

Figura 8.1.Esquema del patrón repository. Adaptado de (Duarte, 2014)	150
Figura 9.1.Creación de usuarios en PostgreSQL	16
Figura 9.2.Creación de una base de datos en PostgreSQL	161
Figura 9.3.Configuración de la cadena de conexión de Play a PostgreSQL	163
Figura 9.4.Ejeccción de conexión a PostgreSQL exitosa	164
Tabla 10.1.Relaciones de dependencia de la aplicación caso de estudio en modo persistencia.	170
Tabla 10.2.Relaciones de composición de la aplicación caso de estudio en modo persistencia.	172
Figura 10.1. Sentencia para ejecutar persistencia en el almacén de datos	173
Figura 10.2.Tablas creadas en la Base de datos por el ORM de Play Framework.....	174
Tabla 10.1.Relaciones de dependencia de la aplicación caso de estudio en modo persistencia.	179
Tabla 10.2.Relaciones de composición de la aplicación caso de estudio en modo persistencia.	181
Figura 10.1. Sentencia para ejecutar persistencia en el almacén de datos	182
Figura 10.2.Tablas creadas en la Base de datos por el ORM de Play Framework.....	183
Tabla 11.1.Información de la capa de negocio reflejada en la capa de presentación	191
Tabla 11.2.Correlación de datos entre primera y tercera capa	200

Figura 12.1.Estudantes participantes del concurso DevsuCodeJam 2014.....	205
Figura 12.2.Competencia de carros seguidores de línea.....	207
Figura 12.3.Estudante de Ingeniería en Informática y Sistemas Computacionales capacitando.	210
Figura 13.1.Logo de Rebian Sistemas. Adaptado de (Facebook, 2016).	215

Las tres capas de los Sistemas de Información Web con (una) JAVA

Resumen

La WWW (World Wide Web), sus canales de comunicación y su estructura de presentación, han convertido a la información en un elemento importante para nuestra vida cotidiana, así también, la automatización de información es el componente principal para que las organizaciones públicas y privadas brinden un servicio a la colectividad. Es por esta razón que el estudio de la Ingeniería de Sistemas de Información para la Web permitirá la construcción de aplicaciones web: eficientes, ágiles, seguras, escalables e interoperables.

Este libro se ha focalizado en los Sistemas de Información Web y se lo ha dividido en seis secciones para presentar las siguientes temáticas: en la Sección 1, se muestra la base de una estructura moderna, las diferentes tendencias, tecnologías e ideologías en la construcción de aplicaciones web de vanguardia; en la Sección 2, 3 y 4 se realiza un análisis del rol que cumple cada una de las tres capas fundamentales (presentación, negocio, datos) que conforma la estructura web, sus lenguajes, técnicas y paradigmas que sustentan su funcionamiento; en la Sección 5 se muestra la forma de iteración y comunicación entre cada capa, para el efecto del servicio que la web ha sido asignada; y en la Sección 6 se presenta los casos de éxito en el rubro, que ha tenido la Universidad Técnica de Cotopaxi.

Para fortalecer la didáctica práctica de cada una de las temáticas planteadas en cada sección, de forma transversal se plantea el desarrollo de una aplicación ejemplo usando como eje principal la plataforma de software JAVA y Play Framework. El ejemplo fortalece los juicios teóricos de cada una de las secciones, ofreciendo al lector las bases para la construcción

de un Sistema de Información Web, usando herramientas tecnológicas con las que la Universidad Técnica de Cotopaxi ha obtenido logros.

Sección 1: Overview

Berners Lee intelectual del centro de investigación CERN (European Council for Nuclear Research) en el año del 1989 publicó la primera página web, marcando un hito importante en la historia de los medios de comunicación de la humanidad. A partir de ese año hasta la presente fecha, la evolución e innovación de los Sistemas de Información Web ha sido permanente, a tal punto que su consolidación ha influido en algunos fenómenos sociales tales como: la globalización, la proliferación de servicios digitales, entre otros. Dando lugar a la creación de un espacio virtual global donde no existen fronteras y las temáticas de intercambio de información son innumerables, tan solo dependen de nuestra imaginación.

Los servicios digitales que hoy se ofertan alrededor del mundo a través de los sistemas de información web, han llegado a convertirse en una parte importante de nuestra vida cotidiana. Por medio de la continua innovación existente en las carreteras de la WWW (World Wide Web) se ha llegado al punto de poder intercambiar mensajes digitales con familiares y amigos que se encuentran al otro lado del mundo, o también por este medio se ha llegado al punto de poder realizar transacciones interbancarias de forma digital.

Técnicamente para poder conseguir este tipo de servicios la arquitectura de un sistema de información web ha pasado por diversas transformaciones y cada vez se ha buscado brindar mayor eficiencia, seguridad y escalabilidad a este tipo de plataformas. La evolución de este medio de comunicación ha dado lugar a que la información que se intercambie por estos canales en ciertos casos sea información de gran relevancia y/o confidencial, motivo por el cual las arquitecturas de almacenamiento y procesamiento de estas plataformas empiezan a ser diseñadas con robustez, para brindar seguridad a la información del usuario.

Diferentes casas comerciales y no comerciales de software han planteado diversas estructuras de trabajo para enfrentar los retos de la gestión de la información en la modernidad. Es por esta razón que hoy en día existen una diversidad de modelos tecnológicos que buscan ofrecer seguridad y escalabilidad a los sistemas de información web, la elección de una de estas tecnologías muchas de las veces dependen de nuestra ideología y experiencia.

Uno de los modelos tecnológicos que ha tenido acogida en la última década y que se ha convertido en una base para fortalecer los servicios de los sistemas de información web ha sido el modelo tres capas. Como parte de la evolución de este modelo, muchas de las casas de software lo han soportado en un patrón de arquitectura de software denominado MVC (Modelo Vista Controlador), este esquema de trabajo ha apoyado la búsqueda constante de un modelo tecnológico eficiente, seguro y escalable. Sin embargo, por la cantidad de información, las amenazas de la red y las nuevas necesidades de información de los usuarios, los modelos tecnológicos están en constante evolución buscando mejorar su desempeño ante nuevas vulnerabilidades.

El modelo tres capas consiste en segmentar las responsabilidades de los sistemas de información web, el usuario que haga uso de los servicios de una plataforma de este tipo tendrá que superar la seguridad de cada una de las capas hasta llegar a la información que requiere consultar o modificar. La iteración del usuario con las capas del modelo inicia con la Primera Capa, conocida también como capa de presentación o vista; posteriormente con la Segunda Capa, conocida también como capa de negocio; y finalmente con la Tercera Capa, conocida también como capa de datos o capa de persistencia.

La primera capa contiene elementos relacionados a la interfaz de usuario, es la que brinda un diseño con el cual se puede interactuar ya sea para la visualización de información almacenada en el sistema o para el envío de nueva información. La segunda capa está compuesta por una lógica de negocio, que son acciones que se encargan del procesamiento de la información de acuerdo con las necesidades que tenga el usuario. La

tercera capa en cambio, la conforma la estructura del modelo de datos, después del procesamiento de información, el resultado final es el almacenamiento de esta información procesada en un almacén de datos, para esto es importante el uso de gestor de base de datos donde se puedan persistir todos los datos necesarios.

En la presente sección se describe el contexto de un ambiente de trabajo basado en un modelo tres capas, así como también la forma en la que el patrón de arquitectura de software MVC (Modelo Vista Controlador) es usado para soportar este tipo de modelo de trabajo. Posteriormente se presenta las características de un Framework didáctico cuya filosofía de elaboración ha sido el patrón MVC, finalmente para cerrar la sección se presenta una práctica en la que se muestra la configuración de este Framework para levantar la ejecución de un nuevo proyecto que permita la elaboración de un sistema de información web. El detalle conceptual, técnico y práctico de cada una de las tres capas del modelo, se lo irá describiendo a partir de la sección 2 del presente libro.

Capítulo I

Introducción a las Aplicaciones

Web en tres capas

Los sistemas de información web han tenido un gran desarrollo en el mundo de las tecnologías de la información y comunicación, pues su evolución e innovación las han convertido en elementos primordiales de las organizaciones públicas y privadas, actualmente las aplicaciones web que se encuentran disponibles a nivel mundial son un número considerable.

La tecnología estructural de las aplicaciones web avanza constantemente, por esta razón es importante disponer de conocimientos conceptuales y funcionales de la arquitectura de estos sistemas. Al momento, uno de los esquemas de trabajo más cotizados son los ambientes de tres capas que generalmente son soportados por un patrón de arquitectura denominado MVC (Modelo Vista Controlador). El estudio de las arquitecturas de los sistemas de información web permite que la implementación de las aplicaciones web en el mercado se dote de mejor calidad.

Patrón de arquitectura MVC (Modelo Vista Controlador)

El patrón MVC fue diseñado por Trygve Reenskaug en un lenguaje de Programación Orientado a Objetos en los años 70, posteriormente Jim Althoff lo depura en los años 80. Con el pasar de los años varios han sido los prototipos que se han derivado del patrón MVC (ver Figura 1.1), entre estos tenemos a HMVC (MVC Jerárquico), MVA (Modelo Vista Adaptador), MVP (Modelo Vista Presentador), MVVM (Modelo Vista Vista Modelo).

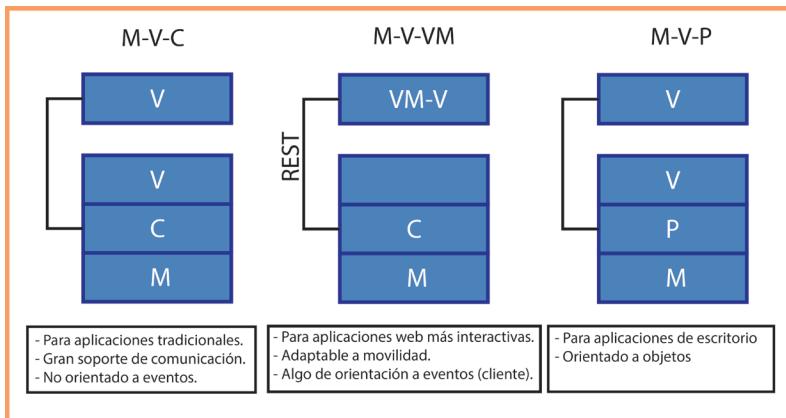


Figura 1.1. Prototipos del Patrón MVC. Adaptado de (Martínez, 2013)

Hoy en día, el desarrollo de aplicaciones web hace mención a técnicas que facilitan el análisis y desarrollo de software, una de estas técnicas son los sistemas de información web soportadas en una arquitectura MVC (Modelo Vista Controlador). Este esquema consiste en una plataforma web dividida en tres capas con funciones específicas cada una. “MVC es un patrón de diseño de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos de forma que las modificaciones al componente de la vista o a cualquier parte del sistema puedan ser hechas con un mínimo impacto en el componente del modelo de datos o en los otros componentes del sistema.” (Reenskaug & Coplien, 2009)

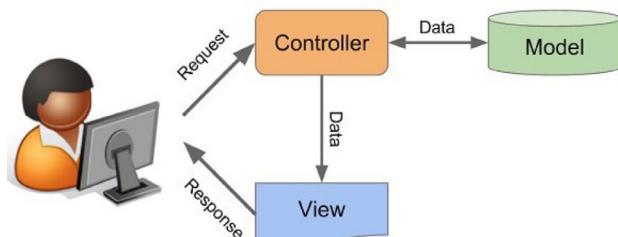


Figura 1.2. Estructura del Patrón MVC. Adaptado de (helloacm.com, 2017)

En la Figura 1.2 se puede apreciar el contexto general del patrón de arquitectura MVC, también se presenta el intercambio de información y la correlación existente entre cada capa. A continuación, se presenta un detalle del aspecto funcional de cada capa:

Modelo (Model): Esta capa posee un conjunto de clases que corresponden al dominio de negocio de la aplicación, dependiendo de los requerimientos funcionales que plantea el usuario se diseña un diagrama de clases que soporte estas necesidades. Generalmente se alude que en esta capa es donde se define las reglas de negocio del sistema, si el cliente requiere agregar, modificar o retirar reglas de negocio, es aquí donde se tiene que reconfigurar los nuevos requerimientos.

Vista (View): Esta capa posee la presentación visual que se ofrece al usuario, esta interfaz es la que permite al usuario entregar o recibir información almacenada en el sistema. Generalmente el desarrollo de esta capa se lo realiza con lenguajes de hipertexto y lenguajes iterativos, su resultado final es el fortalecimiento de la interacción del usuario con el sistema a través de una interfaz.

Controlador (Controller): Esta capa es la intermediaria entre la vista y el modelo, por tal motivo es la que permite la relación y comunicación entre ambas. Aquí es donde se procesan las acciones o peticiones hechas por el usuario en la interfaz para posteriormente enviarlas o extraerlas del modelo.

Ventajas del patrón MVC

- Fomenta la organización del código fuente de la aplicación, esta organización facilita la reutilización del código, característica indispensable en el desarrollo de software ágil.

- Facilita el mantenimiento del sistema de información, la distribución de la aplicación en varias capas permite que su mantenimiento se lo maneje de forma independiente optimizando tiempo y recursos.
- La independencia de la interfaz permite que la sustitución o actualización del entorno gráfico para el usuario sea fácil de rediseñar.
- Al disponer de capas independientes podemos modificar cada una de ellas, sin afectar las demás, es decir que se puede alterar total o parcialmente una interfaz, así como también una base de datos.
- En general el modelo y el controlador son desarrollados con lenguajes de programación orientados a objetos, este hecho hace que la información que se maneje en estas capas pueda ser gestionada bajo las propiedades y fundamentos del paradigma orientado a objetos.
- El uso del paradigma orientado a objetos en el modelo y controlador brinda mayor seguridad a la lógica de negocio que procesa la información del sistema. Pues propiedades de este paradigma como la encapsulación, polimorfismo y otras, entregan a la información un mayor nivel de seguridad.
- Por otro lado, el uso del paradigma orientado a objetos brinda la posibilidad de refactorización en el código fuente de la lógica de negocio. Es decir, si por alguna razón se tiene que cambiar el nombre de una clase, atributo o función el cambio se verá reflejado en cada una de las líneas de código dónde se haga uso dicho elemento.

Desventajas del patrón MVC

- Alta inversión de tiempo en la primera etapa de capacitación, esto a razón de que la configuración y funcionalidad de este tipo de estructuras de sistemas tienen un periodo de asimilación, dependiendo del tamaño del sistema el tiempo de inducción se puede incrementar.
- Se incrementa el nivel de aprendizaje, esto a razón de que la inferencia del paradigma de programación orientado a objetos requiere de un tiempo considerable de entrenamiento, y para desarrollar aplicaciones de calidad se requiere de profesionales con mayor experiencia en el ámbito.

Frameworks MVC

Para el uso del patrón de arquitectura MVC (Modelo Vista Controlador) varios Frameworks o estructuras de trabajo comerciales y no comerciales han sido elaborados, los mismos que circundan en el mercado de software. El uso de un Framework depende de varios factores entre ellos tenemos: ideología, tipo de tecnología, metodología de desarrollo y otros. Para una selección de Framework, factores como estos tienen que ser definidos por el cliente y por el equipo de desarrollo. A continuación, describiremos algunos de los Frameworks MVC más usados en el desarrollo de sistemas de información web.

Spring Framework

Spring Framework es una estructura de trabajo MVC de código abierto, está compuesto por varios módulos de trabajo que brindan más funcionalidades al sistema de información. Constantemente Spring está incorporando nuevos módulos de trabajo que permiten al Framework

ofrecer variedad de servicios al momento de implementar aplicaciones de software. Entre los principales módulos de trabajo de este Framework tenemos: Core, DAO, ORM, JEE, AOP, Web.

Una cualidad de Spring Framework es que no está atado a un modelo de lenguaje de programación, existe versiones que trabajan con el lenguaje de programación JAVA y otras que trabajan con la plataforma .NET. La versión que mayor demanda tiene en la comunidad de software es la que trabaja con JAVA, la diversidad de funcionalidades y la libertad que ofrece a los desarrolladores son sus características más apetecidas.

Spring Framework está basado en los siguientes principios: El buen diseño es más importante que la tecnología subyacente; Los JavaBeans ligados de una manera más libre entre interfaces son un buen modelo; El código debe ser fácil de probar. En la Figura 1.3 se presenta la estructura de trabajo de Spring Framework la cual da inicio cuando el navegador envía una petición y el distribuidor de servlets se encarga de recoger esta petición (1) para pasárselo al controlador de mapeos (2) que comprueba que dicha petición este mapeada y le devuelve el controlador asociado a dicha petición al distribuidor de servlets. Una vez que sabemos que controlador necesitamos el distribuidor de servlets le pasara el control a dicho controlador (3) para que este se encargue de realizar toda la lógica de negocio de nuestra aplicación, este controlador devolverá un objeto modelo y vista (paso 4), el modelo es la información que deseamos mostrar y la vista donde deseamos mostrar dicha información. (Ceresola, 2012)

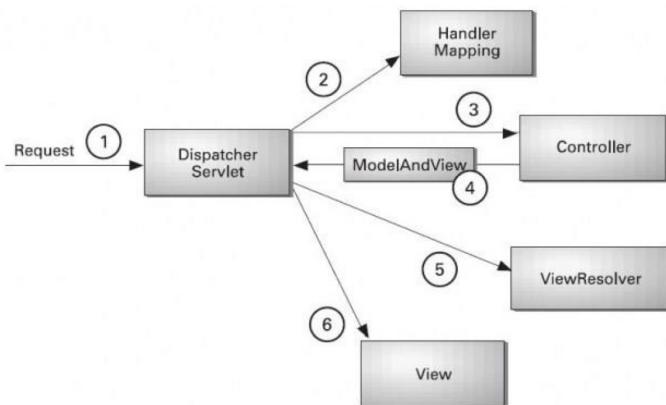


Figura 1.3 Estructura de Spring Framework. Adaptado de (Ceresola, 2012)

Una vez el distribuidor de servlets tiene el objeto modelo y vista tendrá que asociar el nombre de la vista retornado por el controlador con una vista concreta es decir una página jsp, jsf (5). Una vez resulto esto nuestro distribuidor de servlets tendrá que pasar a la vista el modelo, es decir los datos a presentar, y mostrar la vista (6). (Ceresola, 2012)

Framework Ruby on Rails

Es un Framework MVC para aplicaciones web, desarrollado en el lenguaje de programación Ruby. Ha sido considerado un framework para desarrollo de software ágil, esto a razón de la sencillez y eficiencia que brinda al momento de escribir líneas de código. Ruby on Rails es una tecnología relativamente joven, su primera versión fue lanzada en el 2005 y varios han sido los usuarios que han optado por el uso de este framework. Los principales motivos han sido su facilidad de uso y el hecho de ser un lenguaje de programación de origen japonés, símbolo de calidad.

La estructura de Ruby on Rails framework está alineada al esquema general del patrón de arquitectura MVC planteado en la Figura 1.4, sin embargo, para distribuir sus funcionalidades tiene la particularidad de organizar su estructura con los siguientes paquetes: ActiveRecord, ActiveResource, ActionPack, ActiveSupport, y ActiveMailer.

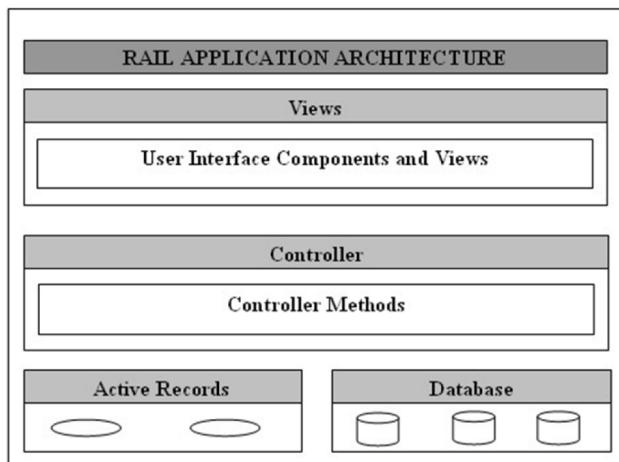


Figura 1.4. Estructura del Framework Ruby on Rails. Adaptado de (Learn Rails, 2017)

En la Figura X se muestra la estructura del Framework Ruby on Rails, siendo el ActiveRecord el elemento más importante pues es el ORM (Object Relation Mapping) del framework que permite correlacionar los objetos de la capa de negocio con la base de datos.

Framework CakePHP

Es un Framework MVC de código abierto, que permite el desarrollo de aplicaciones web con el uso del lenguaje de programación PHP, este framework posee una numerosa comunidad organizada que retroalimenta información relacionada al fortalecimiento del framework

y a la solución de problemas de configuración y uso del framework en el desarrollo de sistemas de información web.

Entre otras características de CakePHP tenemos: la disponibilidad de módulos de integración que permiten la autogeneración de código para facilitar la creación de proyectos; la facilidad que brinda al usuario para trabajar con plantillas que automaticen el proceso de elaboración de una aplicación; la integración de nuevos elementos o plugins que brinden valor agregado al proyecto.

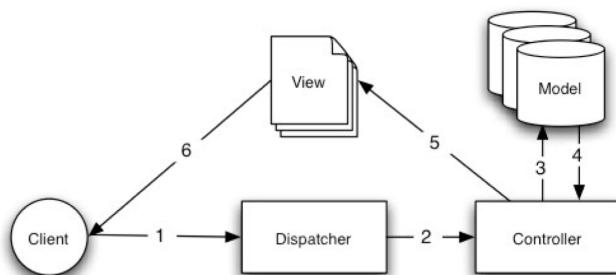


Figura 1.5. Estructura del Framework CakePHP. Adaptado de (CakePHP, 2012)

En la Figura 1.5 se presenta el flujo de trabajo de la estructura del Framework CakePHP: en el Paso 1 el cliente solicita el acceso a una página web; en el Paso 2 el despachador distribuye la petición al controlador que le corresponde; en el Paso 3 se procesa la información solicitada o enviada por el cliente, en el caso de requerir datos del modelo el controlador solicita el acceso a los mismos; en el Paso 4 el modelo entrega al controlador la información solicitada o confirma la ejecución del proceso solicitado; en el Paso 5 el controlador después de haber procesado todos los datos que requería envía a la vista los resultados obtenidos; finalmente en el Paso 6 la vista entrega de forma visual los datos al cliente.

Capítulo II

Play Framework

Luego de revisar algunos de los Frameworks MVC más populares en el mundo del software, a continuación, se describe un framework con similares características y probablemente de mejor calidad. No siempre el término popular es símbolo de calidad, ni tampoco el término no popular es símbolo de mala calidad, muchas de las veces es todo lo contrario, por estrategias de mercado pasa que los productos selectos hacen énfasis en su alta calidad y los productos populares buscan conquistar las masas perdiendo calidad.

Quizá aquí se presenta un ejemplo de lo mencionado, Play Framework es una tecnología poco conocida en los mercados populares, pero sus bondades son de una considerable condición. Una de sus cualidades es su facilidad de uso lo que le hace un framework muy didáctico al momento de analizar su entorno, es por esta razón que se ha elegido a Play Framework como caso de estudio para analizar cada uno de los componentes que conforman un sistema de información web, a continuación se describe un poco de los antecedentes de este framework. Play Framework es una estructura de trabajo Modelo Vista Controlador de código abierto utilizado para el desarrollo de aplicaciones web, fue escrito en el lenguaje de programación Scala y tiene la disponibilidad de generar sistemas de información con otros lenguajes de programación uno de los más utilizados es Java. Play Framework es una alternativa ágil y simple para el desarrollo de aplicaciones empresariales en Java, se concentra en la productividad y comodidad del desarrollador, buscando ser el complemento ideal para las metodologías de desarrollo ágil. (Play Framework, 2012).

Play Framework es una arquitectura de trabajo moderno que combina los beneficios de los lenguajes de programación de Java y Scala, el enfoque de su estilo de programación está dirigido a ser productivo y de apoyo al desarrollo de un proyecto funcional. Los beneficios para el desarrollador radican en que el manejo del entorno del framework es de ejecución

ágil, con un buen tiempo de respuesta y velocidad en su compilación. Con estas cualidades, el framework lo que busca es la experiencia de un ambiente de desarrollo eficaz y direccionado al levantamiento de proyectos de calidad.

Entre sus beneficios principales tenemos la libertad que brinda al desarrollador al momento de construir aplicaciones, esta libertad y comodidad se alinea con la filosofía del framework que busca ser una tecnología divertida para los equipos de desarrollo. Otro de los beneficios, es que las aplicaciones creadas en este entorno son aplicaciones escalables en el tiempo, es decir que nuevos requerimientos solicitados por parte del cliente pueden ser fácilmente incorporados en la aplicación, esto gracias a la flexibilidad que brinda el entorno de trabajo para rediseñar su lógica de negocio. A continuación, presentamos otros de los beneficios que brinda el framework:

Rápida interacción: La implementación de código puede ser modificable y reutilizable, además la actualización del proyecto en el entorno web una vez que se ha realizado cambios en su código fuente es inmediata, es decir la visualización de los cambios efectuados se la va evidenciando en el mismo momento que se realiza la implementación.

Java y Scala: Lenguajes de programación orientados a objetos cuyas propiedades brindan mayor seguridad a la configuración e información que maneja el proyecto.

OpenSource: Es de código abierto, su uso no depende de una licencia comercial y posee una comunidad que constantemente aporta a la innovación de nuevas funcionalidades para Play Framework.

Flexible: Configurable, multiplataforma, personalizable, estas son algunas de las características que hacen de Play Framework una tecnología que puede interactuar con diferentes entornos de trabajo en producción.

Historia

Play Framework fue creado por el desarrollador Guillaume Bort, desde su aparición este framework ha contado con varias versiones, sin embargo, en la actualidad las versiones más rezagadas ya no las podemos encontrar en línea, porque ya perdieron su vigencia. Es en el año 2008 cuándo el primer código de Play 1.0 se publica por Launchpad, posteriormente en octubre del 2009 se lanza una versión más completa, en el siguiente año 2010, se lanza Play 1.1. En estas versiones se incluye funcionalidades tales como: el servidor de aplicaciones Jboss Netty, el apoyo del lenguaje de programación Scala nativo, interacción con el servidor web GlassFish, transmisión de servicios web, soporte HTTPS y otras características.

En el 2013 se lanza la versión de Play 2.2, cuyas características principales fueron: la gestión de aplicaciones Scala y Java a través de la herramienta de creación de proyectos SBT (Simple Build Tool); el mejor soporte para almacenamiento en buffer. Además, en esta versión es cuando el framework empieza a ofrecer su servicio de multiplataforma, ya que a partir de esta versión brinda soporte a la configuración e instalación de sus funcionalidades en diferentes sistemas operativos, tales como: OS X, Linux y Windows.

En el 2014 se lanza la versión Play 2.3 introduciendo el servicio de gestión de proyectos a través de la estructura de comandos Activator, adicionalmente el framework habilita el soporte para trabajar con la versión de Java 8. En el 2015 se lanza la versión de Play 2.4 su novedad principal es su compilación fuera del contexto del framework, esto con

el fin de permitir el uso de play dentro de otras aplicaciones, así también el módulo de pruebas adquiere un mejor nivel de madurez.

Play 2.5 lanzado en el 2016, presenta como característica principal, el mejoramiento de la administración de los WebSockets es decir se maneja de mejor manera el intercambio de información que se encuentra en formatos binarios, Json y/o XML. La innovación y generación de nuevas versiones de Play Framework aún continua, la fortaleza de la que dispone al momento es que a nivel mundial ha logrado generar una comunidad numerosa, así como también adentrarse en el sector empresarial.

Características de Play Framework

Play Framework en el transcurso de su evolución ha ido fortaleciendo determinadas características que se han convertido en la identidad de su servicio ofrecido, paulatinamente estas características han ido incrementando su nivel de madurez y calidad. Siendo este trabajo continuo, el motivo por el cual esta estructura de trabajo ha alcanzado los méritos suficientes para ser parte de los procesos de negocio del sector empresarial, por la misma razón posee una robusta comunidad que fortalece cada día los servicios que ofrece a sus clientes. Entre sus características más emblemáticas tenemos a las siguientes:

Entornos de prueba: Este framework proporciona ambientes de aseguramiento de la calidad destinados para la realización de pruebas unitarias y funcionales, ya sea para entornos de lenguaje Scala y/o Java. El framework tiene apertura de integración con la aplicación Selenium que es un software que administra un ambiente de pruebas para los sistemas de información web que se desarrollen.

Usabilidad: Otra característica importante de Play Framework es la facilidad de uso que brinda su entorno al desarrollo de aplicaciones, muchas de las veces algunos frameworks por dar mayor relevancia a la

parte técnica olvidan la importancia de ofrecer al usuario una buena usabilidad, en la mayoría de los casos estos sucede porque brindar más atención a las cosas complejas y no a la sencillez.

Estandarizado: Play Framework fue escrito por desarrolladores web para desarrolladores web, esto ha provocado que la arquitectura del framework haya sido diseñada bajo los estándares de la W3C.

Confortable: En Play Framework buena parte de la calidad está en los detalles, individualmente pueden ser pequeñas cosas, pero tomadas en conjunto terminan dando por resultado un entorno de desarrollo más confortable y productivo. La sensación de tranquilidad que se experimenta cuando se trabaja con Play se debe a la ausencia de las frustraciones que usualmente surgen cuando se dedica más tiempo a la configuración del framework que ha desarrollar su aplicación. (Peter Hilton, 2010).

Módulos de Play Framework

Una aplicación elaborada en Play Framework puede estar conformada por múltiples módulos, a través de estos módulos de trabajo se pueden reutilizar componentes prediseñados para el desarrollo de una nueva aplicación. El objetivo de los módulos a más de brindar facilidad al desarrollador al momento de programar también es organización pues de esta manera se puede organizar una aplicación compleja en varias simples.

Así, podemos decir que un módulo es una aplicación estándar de Play Framework que dispone de un servicio que puede ser usado en el desarrollo de una nueva aplicación, este servicio puede ser, por ejemplo: gestión automática de CRUD (Create, Read, Update, Delete), la autenticación de un usuario, la gestión de archivos con extensión “pdf”

y otros. A continuación, se presenta un listado de las características de un módulo de Play Framework:

- Un módulo puede ser configurado en una nueva ruta, para esto hay que configurar manualmente las rutas deseadas.
- Cuando se realiza el llamado a una función en tiempo de ejecución, la búsqueda de ésta primero se la realiza en todos los archivos de la aplicación principal, y luego en los archivos de los módulos configurados.
- Un módulo puede contener código Java distribuido en un archivo de extensión JAR (Java ARchive), este archivo estará incorporado como una librería en la aplicación principal.
- Un módulo puede poseer su propia página de documentación.

Los módulos pueden ser cargados automáticamente o manualmente a la aplicación principal, una vez incorporados estos llegan a formar parte de la aplicación como un directorio más del repositorio de trabajo.

Capítulo III

Configurando Play Framework

En este capítulo se da inicio al desarrollo del caso práctico de la aplicación web tres capas que complementa las bases teóricas de las temáticas planteadas. El caso de estudio que se ha elegido para ser desarrollado ha sido un SACC (Sistema de Administración de Carro de Compras) que son sistemas usados en entornos de retail (comercio de productos) para el registro de las ventas que se realice en pequeñas o grandes tiendas de consumo.

En el transcurso de cada capítulo la aplicación ejemplo irá tomando estructura y de forma práctica se irá clarificando la configuración, funcionalidad y responsabilidad de cada una de las capas que conforma una aplicación web con estructura de tres capas. Aquí se muestra las versiones y configuración de cada uno de los prerrequisitos necesarios para que nuestro ambiente de aplicación web con tres capas, esté listo para ser usado. Es importante mencionar que la didacta de la aplicación ejemplo está focalizada para un sistema operativo windows, en el caso de ambientes unix, determinados pasos de los presentados cambian por otros comandos, pero la base lógica de configuración e implementación es la misma.

Para la didáctica de esta aplicación práctica se ha utilizado la versión de Play Framework 1.2.5, en el transcurso de este caso, las configuraciones y funcionalidades planteadas se las efectúa considerando la versión antes mencionada. Previo al uso del framework es necesario validar determinadas herramientas complemento que son importantes para el correcto funcionamiento del ambiente de desarrollo de software que se desea configurar para la demostración de la aplicación ejemplo planteado.

Prerrequisitos

A continuación, se presenta el detalle de todas las herramientas y validaciones que su ambiente de trabajo debe poseer para poder llevar a cabo cada una de las configuraciones y desarrollos disponibles en los capítulos del presente libro.

Play Framework 1.2.5

Para descargar la versión de framework indicada, una versión anterior o posterior, tiene que dirigirse a la página web oficial www.playframework.com. Una vez que se haya descargado la versión indicada de framework, dispondrá de un archivo de extensión “zip” que contiene una carpeta donde se encuentran todos los archivos necesarios para que el Framework Play sea configurado y puesto en ejecución.

Descomprima el archivo en la unidad C: del disco duro de su máquina como muestra la Figura 3.1. Es necesario que la carpeta se la ubique en un lugar de fácil accesibilidad, ya que ésta dispone en su interior, programas batch con los cuales se tiene que interactuar para la creación de nuevos proyectos, configuración de módulos y otras actividades propias del framework.

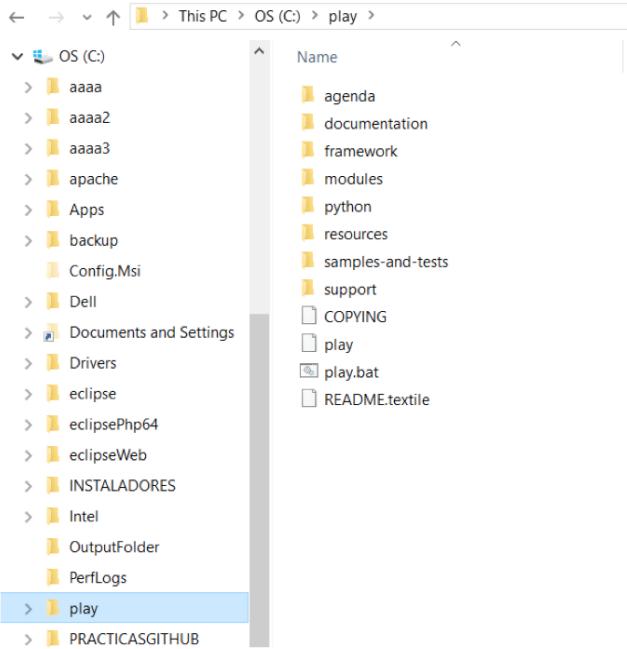


Figura 3.1. Archivos de Play Framework

Java

Para que se ejecute Play Framework y poder usar un IDE (Interface Development Environment) o interfaz de desarrollo, es necesario tener instalado Java, en esta ocasión para trabajar con la versión de Play Framework 1.2.5 y el IDE Eclipse Luna es necesario configurar Java 7. Primero verifique si su máquina posee una versión de Java, ejecutando el comando `java -version` como muestra la Figura 3.2, este comando se ejecuta en el CMD (Command Prompt) de Windows.

 Command Prompt

```
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\acevallo>java -version
java version "1.8.0_111"
Java(TM) SE Runtime Environment (build 1.8.0_111-b14)
Java HotSpot(TM) 64-Bit Server VM (build 25.111-b14, mixed mode)
```

Figura 3.2. Comando para verificar versión de Java

En el caso de no disponer de Java 7, descargar el paquete indicado de la página oficial de Oracle (<http://www.oracle.com/technetwork/java/javase/downloads/>), inicialmente Java nació como una idea innovadora de la compañía Sun MicroSystems pero en la última década fue comprada por Oracle Corporation. Por esta razón la versión indicada y nuevas actualizaciones del JDK (Java Development Kit) se las puede descargar de la página oficial de Oracle, ahí tendrá el listado de JDKs como se muestra en la Figura 3.3, al momento de descargar tomar en cuenta las siguientes consideraciones:

- Sistema Operativo: Están disponibles JDKs para Linux, MacOS, Solaris, Solaris SPARC y Windows.
- Tipo de plataforma: Están disponibles JDKs para plataformas de 32 bits o 64 bits.

Revisar el tipo de sistema operativo y plataforma de su equipo para descargar el JDK que corresponda.

Java SE Development Kit 7u80		
You must accept the Oracle Binary Code License Agreement for Java SE to download this software.		
<input type="radio"/> Accept License Agreement	<input checked="" type="radio"/> Decline License Agreement	
Product / File Description	File Size	Download
Linux x86	130.44 MB	jdk-7u80-linux-i586.rpm
Linux x86	147.68 MB	jdk-7u80-linux-i586.tar.gz
Linux x64	131.69 MB	jdk-7u80-linux-x64.rpm
Linux x64	146.42 MB	jdk-7u80-linux-x64.tar.gz
Mac OS X x64	196.94 MB	jdk-7u80-macosx-x64.dmg
Solaris x86 (SVR4 package)	140.77 MB	jdk-7u80-solaris-i586.tar.Z
Solaris x86	96.41 MB	jdk-7u80-solaris-i586.tar.gz
Solaris x64 (SVR4 package)	24.72 MB	jdk-7u80-solaris-x64.tar.Z
Solaris x64	16.38 MB	jdk-7u80-solaris-x64.tar.gz
Solaris SPARC (SVR4 package)	140.03 MB	jdk-7u80-solaris-sparc.tar.Z
Solaris SPARC	99.47 MB	jdk-7u80-solaris-sparc.tar.gz
Solaris SPARC 64-bit (SVR4 package)	24.05 MB	jdk-7u80-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	18.41 MB	jdk-7u80-solaris-sparcv9.tar.gz
Windows x86	138.35 MB	jdk-7u80-windows-i586.exe
Windows x64	140.09 MB	jdk-7u80-windows-x64.exe

[Back to top](#)

Figura 3.3. Tipos de JDK 7

Instalado Java 7, al ejecutar el comando *java -version* en el cmd de Windows ya le presentará la versión que haya instalado. Otra forma de verificar la instalación es dirigiéndose a la carpeta ProgramFiles o Archivos de Programa de su unidad C: y si la instalación ha sido exitosa, aquí encontrará una carpeta Java que contendrá los archivos de la versión JDK que haya instalado. (Ver Figura 3.4)

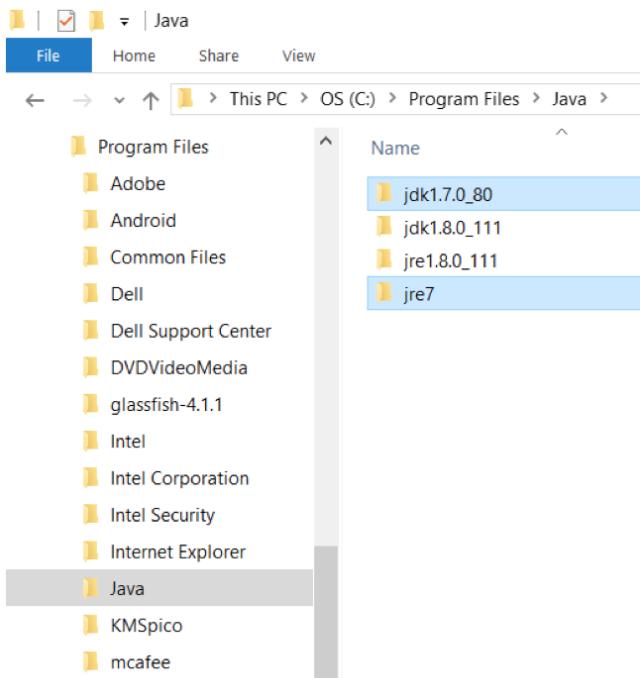


Figura 3.4. Carpetas de instalación Java 7

Variables de ambiente

En el caso del sistema operativo Windows, es importante también configurar las variables de ambiente que trabajan con Java 7. Tres son las variables importantes que hay que validar, éstas son: CLASSPATH, JAVA_HOME y Path, éstas deben poseer la ruta donde se encuentran las carpetas del Java 7. En la Figura 3.5, se muestra la configuración y ruta que debe poseer cada una de las variables de ambiente.

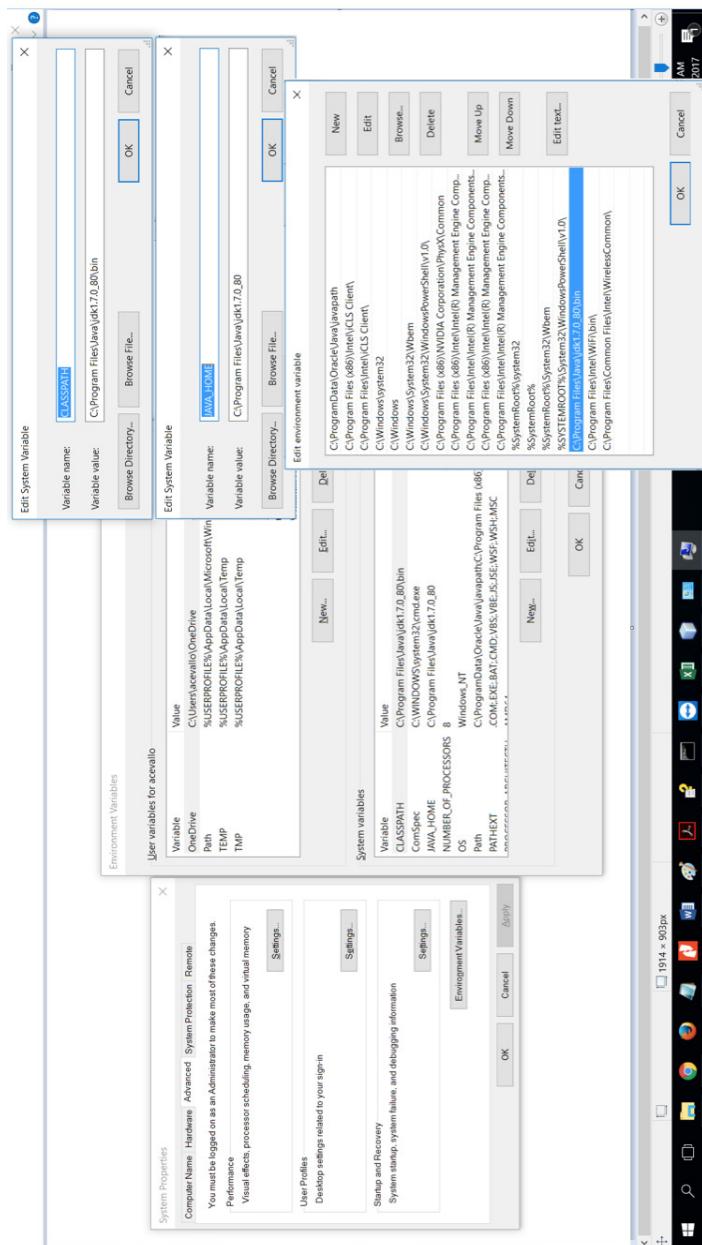


Figura 3.5. Variables de ambiente java

Resumiendo:

CLASSPATH= C:\Program Files\Java\jdk1.7.0_80\bin

JAVA_HOME= C:\Program Files\Java\jdk1.7.0_80

Path= C:\Program Files\Java\jdk1.7.0_80\bin

IDE (Interface Development Environment)

Con Java 7 instalado y configurado, el siguiente paso es instalar y/o configurar el ambiente de desarrollo deseado, entre los IDEs más comunes tenemos a Eclipse y Netbeans, para el presente caso de estudio se trabajará con Eclipse Luna. Para descargar la versión Luna, una versión actualizada o posterior hay que dirigirse a la página oficial de Eclipse (www.eclipse.org/), ahí encontrará una variedad de IDEs para diferentes entornos y lenguajes de software (ver Figura 3.6). Para nuestro caso de estudio requerimos del Eclipse IDE for Java EE Developers, pues es la aplicación que ha sido diseñada para soportar el desarrollo de aplicaciones web con Java. Al momento de descargar el IDE tomar en cuenta las siguientes consideraciones:

- *Versión de Eclipse:* Las versiones de eclipse están emitidas con nombres de planetas, en algunos casos son planetas conocidos y en otros planetas recientemente descubiertos, tales como: Neon, Mars, Luna, Kepler y otros.
- *Tipo de IDE:* Existen IDEs para diferentes tipos de entorno y lenguajes de software, ambientes como: PHP, Android, C, C++ y otros.
- *Tipo de plataforma:* Verificar si su Sistema operativo es de 32 o 64 bits para descargar el que le corresponde.

The screenshot displays the Eclipse IDE download page with five distinct sections, each representing a different variant of the IDE:

- Eclipse IDE for Java EE Developers**: Includes Java EE tools like JPA, JSF, and Mylyn.
- Eclipse IDE for Java Developers**: Essential tools for Java developers including a Java IDE, Git client, XML Editor, Mylyn, Maven, and Gradle integration.
- Eclipse IDE for C/C++ Developers**: An IDE for C/C++ developers with Mylyn integration.
- Eclipse for Android Developers**: An IDE for developers creating Android applications.
- Eclipse for PHP Developers**: Essential tools for any PHP developer, including PHP language support, Git client, Mylyn and editors for JavaScript, HTML, CSS and...

Each section includes a download link, file size, and download count. The Windows version is available in 32-bit and 64-bit formats.

Figura 3.6. Variedad de IDEs de Eclipse

Una vez que haya descargado el Eclipse IDE for Java EE Developers, dispondrá de un archivo con extensión “.zip” este archivo hay que descomprimirlo en la unidad C: de su equipo, como muestra la Figura 3.7. Dentro de la carpeta Eclipse se encuentra el archivo ejecutable “eclipse.exe”, al ejecutarlo, si la instalación y configuración del JDK es correcta, la interfaz del IDE Eclipse Luna se desplegará.

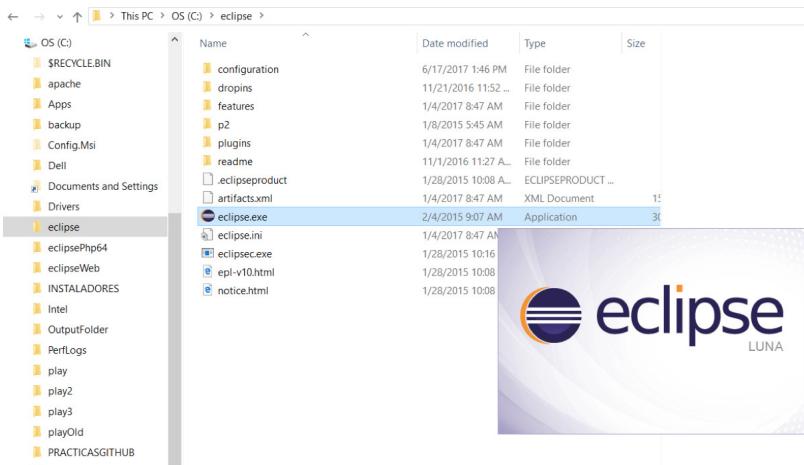


Figura 3.7.Ruta y ejecución del IDE Eclipse

PostgreSql

Otra aplicación que es necesario instalar es el gestor de Base de Datos PostgreSQl, en esta aplicación ejemplo se instala y configura PostgreSql 9.6 (motor de base de datos) y pgAdmin 4 (interfaz de administración). Esta base de datos la puede descargar de la página oficial de PostgreSql (www.postgresql.org/) como muestra la Figura 3.8, usted puede descargar la versión de base de datos que desee, al descargar su base de datos tome en cuenta las siguientes consideraciones:

- Versión de PostgreSql: Las versiones de PostgreSql a la fecha oscilan entre las versiones 8.1 y 10, es importante señalar que las últimas actualizaciones son versiones Beta, por lo tanto su estabilidad al momento de usarla no es garantizada.
- Sistema Operativo: Los motores de base de datos de Postgresql, están disponibles para Linux, MacOS y Windows.

- Tipo de plataforma: Verificar si su Sistema operativo es de 32 o 64 bits para descargar la base de datos que le corresponde.

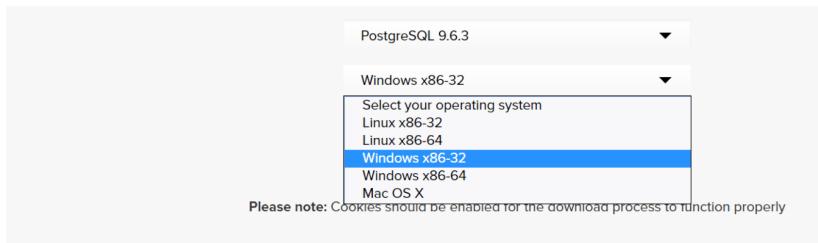


Figura 3.8. Versiones del motor de Base de Datos PostgreSql

Una vez descargado el paquete de instalación de PostgreSQL y dar inicio al wizard de instalación, recuerde dos cosas importantes. La primera es la contraseña de acceso con la que instala la base de datos, si olvida esa contraseña no se podrá acceder al almacén de datos; y la segunda, es el puerto de acceso, por defecto el puerto de configuración es el 5432, si anteriormente ya se instaló otra versión de la base de datos es necesario que configure la nueva instalación en otro puerto, y al igual que la contraseña si olvida el número de puerto de acceso, posteriormente no podrá conectar sus aplicaciones al motor de base de datos PostgreSQL.

Creación de un proyecto en play framework

Una vez que ha instalado y configurado todos los requisitos indicados en la sección 4.1, su ambiente está listo para crear un nuevo proyecto en Play Framework. Para esto, lo primero por hacer, es crear una carpeta donde se va guardar todos los proyectos con los que trabajará en Eclipse, se recomienda que esta carpeta sea de fácil accesibilidad, es decir que no tenga una ruta demasiado grande, la podría crear en el primer nivel de la unidad c:. A esta carpeta de almacén de proyectos, el IDE Eclipse lo denomina workspace o espacio de trabajo y será el lugar donde se guarden todos los archivos que el desarrollador de aplicaciones vaya creando para

su proyecto, pudiendo ser estos archivos de extensión: html, java, css, js y otros. Para esta aplicación ejemplo el workspace se lo ha creado en la siguiente ruta y con el siguiente nombre, c:\workspaceBook\.

Definido su espacio de trabajo, el siguiente paso es crear el nuevo proyecto, para esto tiene que usar comandos de play que pueden ser ejecutados en la consola CMD (command prompt) de Windows, primero verifique si su Play Framework está bien configurado ejecutando el comando `# play` como se muestra en la Figura 3.9a, si el mensaje fue satisfactorio, hay que dirigirse a la carpeta donde se encuentra su workspace en este caso sería c:\workspaceBook, una vez que se encuentra dentro del workspace ejecute el comando `# play new` acompañado del nombre que desea para su proyecto, en este caso usaremos “sacc” como nombre del proyecto (ver Figura 3.9.b)

A

```
C:\Users\acevallo>:play\play
~ play! 1.2.5, http://www.playframework.org
~ Usage: play cmd [app_path] [->options]
~ with, new Create a new application
~ run Run the application in the current shell
~ help Show play help
~
```

B

```
C:\Microsoft Windows [Version 10.0.14393]
(C) 2016 Microsoft Corporation. All rights reserved.

C:\Users\acevallo>cd ..

C:\Users>cd ..

C:\>cd workspaceBook

C:\workspaceBook>:play new sacc
~ play! 1.2.5, http://www.playframework.org
~ The new application will be created in C:\workspaceBook\sacc
~ What is the application name? [sacc]
~ OK, the application is created.
~ Start it with : play run sacc
~ Have fun!
~
```

```
C:\workspaceBook>dir
Volume in drive C is OS
Volume Serial Number is E2EF-554B

Directory of C:\workspaceBook

06/21/2017 11:54 AM <DIR>
06/21/2017 11:54 AM <DIR> ..
06/18/2017 04:29 PM <DIR> .metadata
06/18/2017 04:29 PM <DIR> RemoteSystemsTempFiles
06/21/2017 11:54 AM <DIR> sacc
0 File(s) 0 bytes
5 Dir(s) 805,594,222,592 bytes free

C:\workspaceBook>
```

Figura 3.9. Creación de un proyecto en Play Framework

Es de mucha ayuda agregar como una nueva variable al listado de variables de path la ruta de la carpeta que contiene los archivos de Play Framework como muestra la Figura 3.10a, de esta manera cuando haga uso de los comandos de Play, no tendrá que escribir el comando con toda la ruta donde se encuentra los archivos de play framework, sino, escribirá directamente el comando de play sin indicar su previa ruta (ver Figura 3.10b).

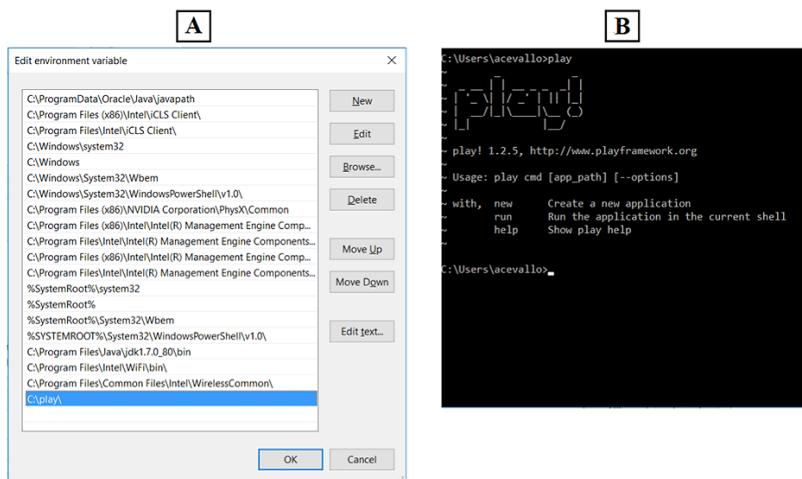


Figura 3.10. Configuración de atajos para uso de comandos

Al crear el nuevo proyecto, dentro del workspace se crea una nueva carpeta con el nombre del proyecto, ingrese a la carpeta del nuevo proyecto y ejecute el comando # **play run**, de esta manera el proyecto se encuentra activo en el puerto por defecto <http://localhost:9000>. En la Figura 3.11. se puede apreciar la ejecución del comando y el proyecto abierto en el browser.

A

```
sc Command Prompt: play run
It's Windows [version 10.0 - 14393]
© Microsoft Corporation. All rights reserved.

A workspacebook
A workspacebook ...
A workspacebook ...

spacebook@spacebook: ~
```

spacebook@spacebook:~\$ play run

1.2.2.5, <http://www.playframework.org>

C to stop

Starting C:\workspacebook\spacebook@spacebook:~

Listening for HTTP on port 9000 (Waiting a first request to start) ...

2.684 INFO - Starting C:\workspacebook\spacebook@spacebook:~

3.438 WARN - You're running play! in DEV mode

3.601 INFO - Listening for HTTP on port 9000 (Waiting a first request to start) ...

3.891 INFO - Application "space" is now started!

B

Your new application is ready!

Congratulations, you've just created a new play application. This page will help you in the few next steps.

Why do you see this page?

The `conf/routes` file defines a route that tell Play to invoke the `Application.index()` action when a browser requests the URL using the `GET` method:

```
* Application home page
  GET / → Application.index()
```

So play has invoked the controller's `Application.index()` method:

```
public static void index() {
    render();
}
```

Using the `render()` call, this action sets play to display a template. By convention play has displayed the `app/views/Application/index.html` template.

```
Play routes: "main.html" → Application.index()
Application.index() → Application.index()
```

Figura 3.11. Comando de ejecución de un proyecto Play Framework

Si todos los prerequisitos se encuentran bien instalados, configurados y los comandos de play han sido ejecutados correctamente, en su explorador podrá apreciar la página de bienvenida de Play Framework. En esta ocasión se ha puesto en ejecución un proyecto de play de forma manual, es decir la compilación y ejecución del proyecto se la ha realizado con el uso de comandos en consola, sin el uso de ningún IDE (para detener la ejecución del proyecto en la consola debe presionar las teclas `ctrl+c`). Una vez que se ha verificado que la creación del proyecto ha sido correcta, se procede a vincular el proyecto creado con su IDE de preferencia, en este caso de estudio se realizará el vínculo con Eclipse.

Para realizar esta vinculación hay que ingresar a la carpeta del proyecto, en la Figura 3.12a se puede ver el listado de carpetas y archivos necesarios para el correcto funcionamiento de la aplicación, cabe señalar que todos estos archivos se generan con la ejecución del comando de play para creación de proyectos. En esta ubicación ejecute el comando `# play eclipsify` o su abreviación `# play ec` este comando como se muestra en la Figura 3.12b crea en el proyecto una carpeta con el nombre `eclipse`, la cual contiene los archivos necesarios para que desde el IDE Eclipse esta aplicación pueda ser importada.

A

```
C:\workspaceBook>cd sacc
C:\workspaceBook\sacc>dir
Volume in drive C is OS
Volume Serial Number is E2EF-5548

Directory of C:\workspaceBook\sacc

06/21/2017 12:00 PM <DIR> .
06/21/2017 12:00 PM <DIR> ..
06/21/2017 11:54 AM <DIR> app
06/21/2017 11:54 AM <DIR> conf
06/21/2017 11:54 AM <DIR> lib
06/21/2017 11:54 AM <DIR> public
06/21/2017 11:54 AM <DIR> test
06/21/2017 11:54 AM <DIR> tmp
06/21/2017 12:00 PM <DIR>
0 File(s) 0 bytes
8 Dir(s) 895,576,156,400 bytes free
```

B

```
C:\workspaceBook\sacc>play ec
~ play! 1.2.5, http://www.playframework.org
~ Ok, the application is ready for eclipse
~ Use File/Import/General/Existing project to import C:\workspaceBook\sacc into eclipse
~ Use eclipsify again when you want to update eclipse configuration files.
~ However, it's often better to delete and re-import the project into your workspace since eclipsify keeps dirty caches...
~
```

```
C:\workspaceBook\sacc>dir
Volume in drive C is OS
Volume Serial Number is E2EF-5548

Directory of C:\workspaceBook\sacc

06/21/2017 12:19 PM <DIR> .
06/21/2017 12:19 PM <DIR> ..
06/21/2017 12:19 PM <DIR> 5,598 ..\classpath
06/21/2017 12:19 PM <DIR> 443 ..\project
11/30/2016 04:38 PM <DIR> ..\settings
06/21/2017 11:54 AM <DIR> ..\app
06/21/2017 11:54 AM <DIR> ..\conf
06/21/2017 12:19 PM <DIR> ..\lib
06/21/2017 11:54 AM <DIR> ..\public
06/21/2017 11:54 AM <DIR> ..\test
06/21/2017 12:00 PM <DIR> ..\tmp
0 File(s) 6,041 bytes
10 Dir(s) 895,568,143,360 bytes free
```

Figura 3.12. Ejecución del comando *Eclipsify*.

Creada en nuestro proyecto la carpeta Eclipse, el siguiente paso es arrancar nuestro IDE haciendo doble click en el archivo “eclipse.exe” de la carpeta c:\eclipse\). Ya en el interfaz principal del IDE, como muestra la Figura 3.13 hay que importar el proyecto que se ha creado, haciendo click derecho sobre el explorador de proyectos, selecciona la opción “*import*”, esta acción le permitirá desplegar una ventana donde puede seleccionar el tipo de proyecto que deseas importar, ahí deberá seleccionar la opción “*Existing Projects into Workspace*”.

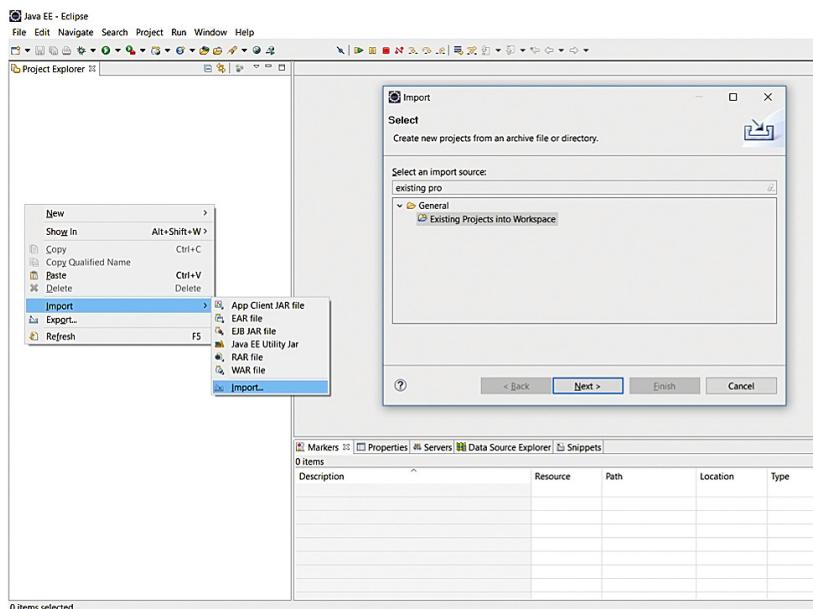


Figura 3.13. Importación de un proyecto Play Framework hacia Eclipse (Parte 1)

Después de hacer click en la opción “next” le aparecerá una ventana (ver Figura 3.14 en la cual haciendo click en la opción “browse” tiene que seleccionar la ruta donde se encuentra su proyecto, para este caso de estudio será la ruta c:\workspaceBook\sacc\).

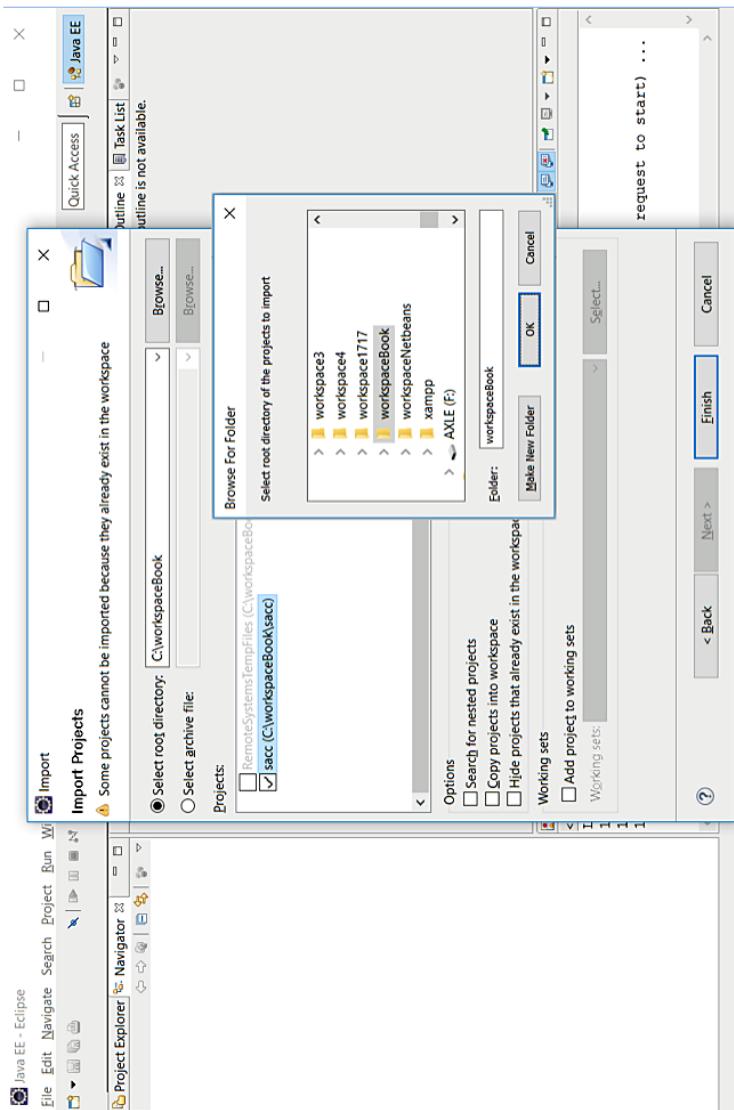


Figura 3.14. Importación de un proyecto Play Framework hacia Eclipse (Parte 2)

Al finalizar la importación del proyecto, en el navegador de proyectos del IDE eclipse, se dispondrá de todos las carpetas y archivos necesarios para el funcionamiento de su aplicación. Sobre la estructura que se muestra en la sección izquierda de la Figura 3.14 es donde se añade los archivos de programación que sean necesarios para la evolución del desarrollo de su aplicación.

De la estructura presentada, cuatro son las carpetas más importantes a la hora de desarrollar su aplicación, a continuación, una breve descripción de éstas:

app: Posee los contenedores para los archivos del entorno MVC (Modelo Vista Controlador) propuesto. La carpeta *models* almacena todas las clases de extensión java que hayan sido diseñadas en el modelo de dominio que soporta las necesidades de su aplicación. *Views*, almacena todos los archivos html en los cuales se haya diseñado el estilo de interfaz que visibilizará el usuario. *Controllers*, almacena clases de extensión java que son clases destinadas a vincular los archivos de interfaz con las clases del modelo de dominio, es decir son intermediador para models y views.

conf: Posee archivos de configuración del sistema, como por ejemplo la configuración de una cadena de conexión a una base de datos, la activación de nuevos módulos de trabajo como el trabajo con sesiones, creación de nuevas rutas de accesibilidad y otros.

eclipse: Posee archivos de compilación, depuración y ejecución del proyecto, es aquí desde donde se arranca el proyecto para que pueda ser visualizado en su navegador a través de la ruta <http://localhost:9000>. Como muestra la Figura 3.15 al dar click derecho en el archivo que tiene por identificación el nombre del proyecto con una extensión “*launch*”, se habilita la opción *Run As* que una vez elegida, le permite seleccionar el nombre de proyecto, para que este pueda ser ejecutado y usado a

través de su navegador por el puerto antes mencionado. Finalmente, para detener el proyecto tiene que presionar en el botón rojo de “*stop*” ubicado en la parte inferior derecha.

public: Posee las carpetas contenedoras de los archivos empleados para el diseño de la interfaz de usuario. La carpeta *images* almacenará todas las *imágenes* de extensión “*jpg*”, “*png*” y otros que han sido usadas en el proyecto. La carpeta *stylesheets* almacenará todos los archivos de estilo de extensión “*css*” que se han creado y usado en el interfaz. La carpeta *javascript* almacenará todos los archivos y librerías de extensión “*js*” que han creado o pugins que se han descargado para brindar mayor dinamismo a la interfaz del proyecto.

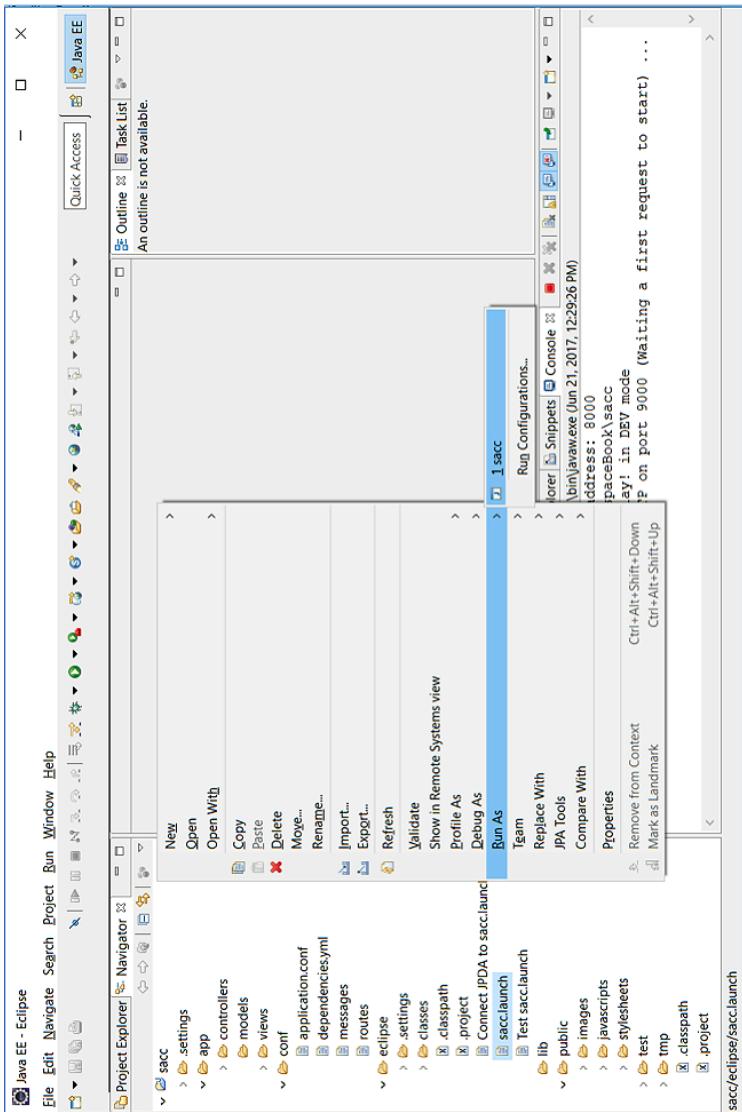


Figura 3.15 Estructura del proyecto Play Framework y sus opciones para ejecutarlo

En resumen, a lo largo de este capítulo se ha indicado los pasos a seguir para instalar y configurar los prerequisitos necesarios para trabajar con Play Framework MVC. También se ha mostrado la forma de usar los comandos básicos de play para crear un nuevo proyecto y poder importarlo al IDE Eclipse. Ya con el proyecto configurado para que pueda ser manipulado por un IDE, nuestro ambiente de trabajo está listo para empezar con el desarrollo de la aplicación.

Sección 2: Primera Capa de Presentación

En esta sección se abordará el detalle de los aspectos teóricos y funcionales de la primera capa de una estructura MVC (Modelo Vista Controlador), generalmente a este segmento de una aplicación web también se la denomina como capa de presentación o vista. Su característica e importancia principal es su contacto con el usuario, a través de esta capa el usuario visualiza la información que posee el sistema, así como también es el medio por el cual interactúa con ella, con el propósito de enviar información para que sea almacenada en el sistema o de extraer información que requiera a su interfaz para la toma de decisiones.

El contacto directo de esta capa con el usuario da lugar a que su presentación sea acorde a las necesidades que solicite el usuario, pudiendo ser éstas, necesidades de presentación, publicidad o transaccionalidad. Por esta razón la primera capa se convierte en un segmento muy importante de la estructura de aplicaciones MVC, ya que la mayor parte del éxito de la aplicación depende de la forma de plasmar el requerimiento del usuario en la interfaz, si se implementa una interfaz que no cubre las necesidades del cliente, la aplicación puede fracasar.

Para fortalecer este acuerdo con el usuario es importante usar las buenas prácticas de la etapa de toma de requerimientos en un proceso de desarrollo de software, esto ayudará a interpretar y abstraer de mejor

manera las necesidades que tiene el usuario, para así ofrecer una solución a medida. Es por esta razón que es de gran significado, mantener una buena y correcta comunicación con el usuario, pues es la base, para que el éxito del diseño de la interfaz de la primera capa se consolide.

Expuesto el preámbulo funcional de la capa de presentación, también esta sección está acompañada de los aspectos teóricos que dan lugar a la construcción de una interfaz de usuario, tales como lenguajes de marcado, estilo, interpretado y dinámicos. Entre estos tenemos a lenguajes como: html (HyperText Markup Language), css (Cascading Style Sheets), javascript, jquery. En la etapa final de la sección sobre la base del ambiente configurado en el Capítulo 4, se da continuidad a la construcción del sistema de información web ejemplo, focalizando la configuración y desarrollo de lo que corresponde a la implementación funcional de la primera capa.

Capítulo IV

Lenguajes de Interfaz

Para desarrollar las interfaces que se encuentran al navegar por la WWW (World Wide Web) es necesario conocer la sintaxis de un lenguaje de interfaz o lenguaje de marcado, al referirse al término marcado se está conceptualizando la forma en que un editor puede registrar manuscritos con estilos y tipografías en un papel. En el caso del mundo digital, el marcado es el término empleado para describir a un conjunto de códigos que se los conoce también con el nombre de etiquetas, que son añadidas al texto electrónico para definir la estructura y el formato en el que se presentar en una interfaz. Puede ser utilizado, para propósitos diferentes como son: la escritura, la impresión, el intercambio de datos, la presentación, la publicidad y otros.

Un lenguaje de marcas es una forma de codificar formatos para un texto, su uso se lo lleva a cabo a través de la incorporación de etiquetas que contienen información y funcionalidades complementarias a la estructura del texto. El término universal para identificar este tipo de lenguaje es “HyperText Markup Language” cuyo significado es Lenguaje de Marcas de Hipertexto, que es un estándar en el desarrollo de software enfocado en la elaboración de páginas web, permite definir la estructura del contenido de una página web.

Inicialmente el lenguaje de marcado en el mundo digital estaba soportado únicamente por SGML, lamentablemente por su complejidad de uso, únicamente personas especializadas podían utilizar este lenguaje, es por esta razón que aparece el lenguaje de marcado HTML el cual resuelve el problema de la complejidad, esto a través de la definición de un listado de etiquetas destinadas a definir una estructura y una semántica para los documentos digitales. Es así como el diseño de documentos digitales con marcas e hipertexto se populariza, creando un espacio disponible para la creatividad e innovación, espacio digital que hasta el día de hoy sigue creciendo.

HTML se deriva del metalenguaje SGML (Standard Generalized Markup Language) y es el formato definido a nivel mundial para escribir los documentos para la (WWW) World Wide Web, el organismo que va desarrollando y estableciendo estos estándares se denomina (W3C) World Wide Web Consortium, esto con el objetivo de normalizar el desarrollo y la expansión de la web. HTML nace como un lenguaje diseñado para el intercambio de documentos de investigación, permitiendo que su uso no represente mayor complejidad y pueda ser utilizado por cualquier persona sin tener la necesidad de que sea un especialista en el tratamiento de este tipo de documentos.

El hipertexto en la actualidad

El hipertexto es una herramienta específica para un solo propósito, es un modo de interactuar con el texto. Actualmente, el término hipertexto se ha expandido para incluir una gran cantidad de aplicaciones computacionales tales como libros interactivos, encyclopedias, índices de referencia en línea y otras formas de lectura y escritura no lineales, así como procedimientos de “meta-análisis”. (Rodríguez et al., 2002)

Suárez *et al*, 2002 nos menciona que la ventaja de este lenguaje cada vez va creciendo ya que tiene muchos propósitos, como por ejemplo incluir aplicaciones como libros interactivos, libros electrónicos todo material que pueda servir para hacer una investigación científica que el estudiante o cualquier usuario lo necesite.

Hipertexto y lectura

El diseño de hipertexto elimina alternativas uniendo textos basados en el mismo tópico y elimina la incertidumbre de presentar información no relacionada. Esto es un punto importante ya que los estudiantes tienen un mayor sentido de control que mantiene el interés en los

textos, esto se debe a que el proceso de búsqueda manual para encontrar los textos de referencia cruzada se elimina, realizándose el enlace en forma automatizada intuitiva aun con los que se encuentran fuera de ese documento como son los de Internet. (Rodríguez *et al.*, 2002)

De acuerdo al criterio anterior, el hipertexto ayuda a mejorar la calidad de la redacción de los textos, es decir presenta la información más destacada sobre el tema que se necesita, ayuda también a que la búsqueda manual se elimine y solo sea mediante la red.

Este tipo de acceso a textos múltiples mejora las habilidades del pensamiento crítico debido a que los lectores pueden elegir el texto que garantice un mejor contenido y presentación. Dicha distribución proporciona además el acceso a diferentes puntos de vista; en contraste, los libros impresos tienden a presentar una visión única. (Rodríguez *et al.*, 2002)

Cualidades del hipertexto

A la vez que el entorno de hipertexto proporciona a los alumnos un proceso de aprendizaje interactivo y nuevas experiencias de lectura, (“todo a la mano”) también puede constituir una barrera para aquellos que no están familiarizados con la tecnología. Los documentos de hipertexto son diferentes de los impresos de tres maneras: primero el hipertexto requiere del uso de las computadoras. Segundo, el hipertexto presenta información en forma de texto verbal y no verbal. La hipertextualidad en forma inevitable incluye un mayor porcentaje de información no verbal que el material impreso. El hipertexto utiliza iconos que no se encuentran en textos impresos, tales como flechas, botones, y barras de desplazamiento que le sirven al estudiante como guías a través del texto. En tercer lugar, presenta información en un abordaje de lectura no lineal. (Rodríguez *et al.*, 2002)

La habilidad del hipertexto para “enlazar” o conectar varios fragmentos de información entre sí, hace posible que el lector seleccione tópicos o secciones de un directorio mediante una palabra clave, o mediante un ícono, y tenga acceso a información adicional. Con este sistema se experimenta el texto como parte de una red de textos “navegables”, en lugar de una secuencia lineal de ideas. (Rodríguez *et al.*, 2002)

El hipertexto tiene una ventaja muy importante que consiste en enlazar íconos que nos permiten tener acceso a cualquier sitio que se desee, ya que puede contener información adicional a la que está implementada en ese sitio, el hipertexto facilita textos navegables, es decir información completa y no solo ideas centrales.

¿Qué es Html?

HTML es una implementación del standard SGML (Standard Generalized Markup Language), estándar internacional para la definición de texto electrónico independiente de dispositivos, sistemas y aplicaciones. Metalenguaje para definir lenguajes de diseño descriptivos; proporciona un medio de codificar documentos hipertexto cuyo destino sea el intercambio directo entre sistemas o aplicaciones. (Rufo, 2001)

“HTML siempre ofreció diferentes formas de construir y organizar la información dentro del cuerpo de un documento. Uno de los primeros elementos provistos para este propósito fue <table>. Las tablas permitían a los diseñadores acomodar datos, texto, imágenes y herramientas dentro de filas y columnas de celdas, incluso sin que hayan sido concebidas para este propósito.” (Gauchat, 2012)

HTML es muy importante hoy en día, es por eso que se debe recordar que, en los primeros días de la web, las tablas fueron una revolución, un gran paso hacia adelante con respecto a la visualización de los documentos y la experiencia ofrecida a los usuarios. Gradualmente,

otros elementos reemplazaron su función, permitiendo lograr lo mismo con menos código, lo que facilitando de este modo la creación, permitiendo portabilidad y ayudando al mantenimiento de los sitios web. (Gauchat, 2012)

El elemento `<div>` comenzó a dominar la escena. Con el surgimiento de webs más interactivas y la integración de HTML, CSS y Javascript, el uso de `<div>` se volvió una práctica común. Pero este elemento, así como `<table>`, no provee demasiada información acerca de las partes del cuerpo que está representando. Desde imágenes a menús, textos, enlaces, códigos, formularios, cualquier cosa puede ir entre las etiquetas de apertura y cierre de un elemento `<div>`. En otras palabras, la palabra clave `<div>` solo especifica una división en el cuerpo, como la celda de una tabla, pero no ofrece indicio alguno sobre qué clase de división es, cuál es su propósito o qué contiene. (Gauchat, 2012)

Etiquetas Html: Una etiqueta HTML es un término rodeado por un signo de menor y otro de mayor; por ejemplo:

****: Esta etiqueta ayuda a destacar un texto con negrita.

****: Esta etiqueta delimita o cierra el uso de negrita en un texto, mayoritariamente algunos elementos requieren esa apertura y cierre de etiqueta.

**
**: Esta etiqueta es un ejemplo de aquellas etiquetas que no requieren de una delimitación o cierre; y su funcionalidad es de indicar al navegador que inserte un salto de línea en un documento HTML.

Otra característica importante de las etiquetas es que pueden declararse de forma anidada, por lo que podemos definir un párrafo (mediante las etiquetas `<p>` y `</p>`) y en su interior incluir una lista de elementos, imágenes, texto destacado, etc. empleando diferentes etiquetas. (Paniagua, 2012)

Elementos Html: Según el autor Eguíluz en el 2007, menciona que además de etiquetas y atributos, HTML define el término “elemento” para referirse a las partes que componen los documentos HTML. Aunque en ocasiones se habla de forma indistinta de “elementos” y “etiquetas”, en realidad un elemento HTML es mucho más que una etiqueta, ya que está formado por:

- Una etiqueta de apertura.
- Cero o más atributos.
- Opcionalmente puede disponer de cualquier tipo de contenido de texto.
- Una etiqueta de cierre.

Por ejemplo, un párrafo con atributos y contenidos se considera un elemento HTML, como se puede ver en la siguiente Figura 4.1. (Eguíluz, 2007)

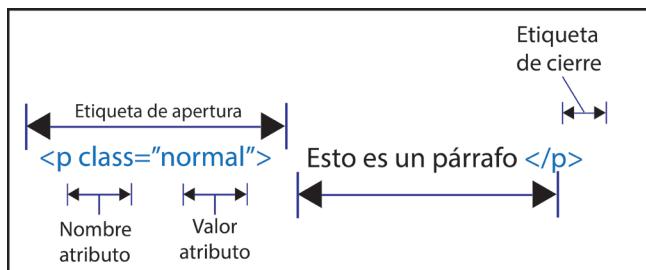


Figura 4.1. Esquema de las partes de un elemento HTML. Adaptado de (Libros Web, 2017)

Forma de escribir una etiqueta

La forma de escribir una etiqueta básica es <ETIQUETA>; además la gran mayoría de estas se deben “cerrar” de esta manera: </ETIQUETA>. La forma de escribir una etiqueta básica es <ETIQUETA>; además la gran mayoría de estas se deben “cerrar” de esta manera: </ETIQUETA>. Para cerrar una etiqueta con atributos se utiliza solo la etiqueta de cierre, esto es sin atributos. (Távara, 2000)

Colores

Según el autor Anibarro en el 2001, menciona que existen dos formas para aplicar colores a una página web:

1. Se especifica el color deseado directamente con el nombre del color en inglés: Ejemplo: blue, green, yellow.
2. Se especifica el color deseado mediante números hexadecimales mediante la siguiente estructura:

#RRVVA

- El color tiene un signo de numeral # antecediendo a los 6 números
- Existen dos números para cada color principal: rojo, verde y azul.
 - Cada uno de los números varía hexadecimalmente {0,1,2,...,9,A,B,...F}. (Anibarro, 2001)

Ubicación, formato y atributos de texto

- **Etiqueta <center> </center>:** Se utiliza para centrar el texto/imagen o datos que se encuentren entre la apertura y el cierre.

- **Etiqueta :** Esta es la etiqueta que nos posibilita un texto con negrillas.
- **Etiqueta <u> </u>:** Etiqueta que posibilita resaltar un texto con subrayado.
- **Etiqueta <i> </i>:** Etiqueta que permite resaltar el texto con inclinación itálica. (Anibarro, 2001)

Hojas de estilo

Las Hojas de Estilo (o CSS, por Cascading StyleSheets) son un mecanismo que permiten aplicar formato a los documentos escritos en HTML ó XHTML (y en otros lenguajes estructurados, como XML) separando el contenido de las páginas de su apariencia. Para el diseñador, esto significa que la información estará contenida en la página HTML, pero este archivo no debe definir cómo será visualizada esa información, sino la hoja de estilos asociada a él. Las indicaciones acerca de la composición visual del documento estarán especificadas en el archivo de la CSS. (Florián, 2007)

CSS es el mecanismo que permite separar los contenidos definidos mediante XHTML y el aspecto de esos contenidos en la pantalla del usuario. Una ventaja añadida de la separación de los contenidos y su presentación es que los documentos XHTML creados son más flexibles (ver Figura 4.2), ya que se adaptan mejor a las diferentes plataformas: pantallas de ordenador, pantallas de móviles, impresoras, dispositivos utilizados por personas discapacitadas, etc.” (Eguíluz, 2007)

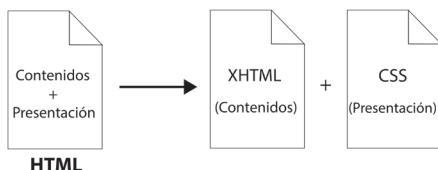


Figura 4.2. Separación de contenidos y presentación. Adaptado de (Libros Web, 2017)

CSS tiene un set predeterminado de propiedades destinados a sobrescribir los estilos provistos por navegadores y obtener la organización deseada. Estas propiedades no son específicas, tienen que ser combinadas para formar reglas que luego serán usadas para agrupar cajas y obtener la correcta disposición en pantalla. La combinación de estas reglas es normalmente llamada modelo o sistema de disposición. Todas estas reglas aplicadas juntas constituyen lo que se llama un modelo de caja, el modelo válido y ampliamente adoptado es el llamado Modelo de Caja Tradicional, el cual ha sido usado desde la primera versión de CSS. (Gauchat, 2012)

Una sección de definición de estilos CSS (tanto en etiqueta style como en archivo externo) permite definir un conjunto de reglas CSS. Las reglas CSS se componen de: ***Un selector** que indica a qué elementos HTML se aplicará la regla, y ***Un conjunto de declaraciones**, indicando los estilos a aplicar a los elementos seleccionados, así como se muestra en la Figura 4.3. (Cabrera, 2012)

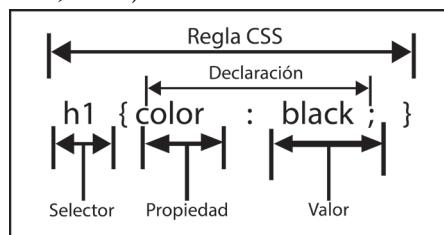


Figura 4.3. Reglas CSS. Adaptado de (Libros Web, 2017)

Según Cabrear, una declaración se compone de una o varias propiedades CSS con sus valores, siendo:

- Propiedad: Característica a modificar.
- Valor: Establece el valor a aplicar a la característica en cuestión.

Los estilos aplicados directamente sobre un elemento HTML incluyen únicamente la sección de declaraciones, ya que el elemento seleccionado es sobre el que estamos actuando con el atributo style. En la especificación actual, CSS 2.1, hay definidas 115 propiedades, si bien en CSS3 ya va por 239, a falta de cerrar la especificación de este estándar. (Cabrera, 2012)

¿Para qué sirve los CSS?

Las hojas de estilo son una parte fundamental es por eso que nos sirve para:

- Permitir elegir el color apropiado a cualquier texto.
- Permite seleccionar el tipo de letra utilizado en el sitio.
- Estable el tamaño, bordes, fondo del texto.

Los CSS permite hacer el diseño de nuestro propio sitio con nuestros propios parámetros, es decir se puede hacer un menú que este a la izquierda, que tenga cualquier tipo de letra, cualquier fondo etc. (Menéndez & Barzanallana)

¿Dónde se escribe el código CSS?

Según los autores Menéndez y Barzanallana, básicamente se puede escribir el código CSS en tres lugares diferentes:

- En un archivo con extensión CSS (este es el método más recomendado).
- En la cabecera <head> del archivo HTML;
- Directamente en las etiquetas del archivo HTML a través de un atributo de estilo (éste es el método menos recomendable).
(Menéndez & Barzanallana)

Para recordar CSS es otro lenguaje que complementa a HTML. Su función consiste en dar formato a las páginas web, a sí mismo cuando un navegador no tiene conocimiento de un formato de instrucción simplemente lo ignora. No olvidarse que se puede escribir el código CSS en varios lugares diferentes, lo más aconsejable es crear un archivo separado con extensión .css. (Menéndez & Barzanallana)

Formas de usar estilos y lenguajes complemento

Utilización de estilos

Para crear una página web se debe empezar desarrollando el documento HTML con todos los contenidos, definiendo la función que ha de tener cada elemento dentro de la página, es decir, si el elemento contiene un párrafo, una lista, una imagen, una tabla de información, etc. Después de tener todos los elementos que se han de mostrar en la página, se procede a asignar el diseño de cada uno de esos elementos mediante la aplicación de estilos CSS. Una hoja de estilos es un conjunto de reglas que los navegadores utilizan para interpretar cómo deben mostrar los diferentes elementos de un documento HTML. (Zafra, 2015)

Según Zafra, las distintas formas de incluir estilos CSS en un documento HTML son:

- Incluir CSS en los elementos HTML.
- Incluir CSS en el mismo documento HTML.
- Definir CSS en un archivo externo.

Se pueden definir los estilos dentro del mismo documento HTML, normalmente dentro de la etiqueta `<head>` mediante la etiqueta `<style type="text/ CSS">`, y aunque no es recomendado, se pueden incluir en cualquier parte del documento. La mejor manera es agrupar todos los

estilos CSS en un solo documento .css, ya que es la forma más organizada y accesible de mantener todas las definiciones. (Zafra, 2015)

Tipos de estilo: incrustados, enlazados, importados, en línea

Gracias a la flexibilidad de este lenguaje, es posible incluir estilos CSS de distintas maneras para realizar una misma tarea. Hay que tener en cuenta cada uno de los métodos, ya que puede facilitar el trabajo o, por el contrario, si no se hace buen uso, puede crear conflictos entre estilos y hacer todo más complicado. (Zafra, 2015)

Descripción de los tipos de estilos

Según el autor (Zafra, 2015) se ha mencionado una forma de incluir estilos CSS incrustados dentro de un documento HTML, mediante el atributo “style” dentro de la etiqueta HTML. Pero existen otros métodos de incluir estilos:

- Enlazar una hoja de estilos externa a un documento HTML.
- Incrustar el estilo en el documento HTML, bien definiendo los estilos dentro de las etiquetas, o declarándolos en alguna parte del documento.
- Importar una hoja de estilos externas a un documento HTML.
- Importar hojas de estilo a través de otro archivo de estilos.

Según Zafra, es importante saber que cada una de estas formas tiene una jerarquía de preferencia para cuando un mismo estilo está definido en varios estamentos:

- El navegador interpretará en primer lugar las propiedades declaradas en una hoja de estilos externa.
- Si en la cabecera del documento, dentro de la etiqueta `<head> </`

- head>, se repite alguna propiedad declarada en la hoja de estilos externa, se aplica el estilo incrustado dentro del documento HTML, obviando la misma propiedad definida en la hoja de estilos.
- Si en un elemento concreto están definidas una o varias propiedades específicas dentro de la etiqueta HTML mediante el atributo “style”, se aplicarán estos estilos ignorando los anteriores. (Zafra, 2015)

Enlazar una hoja de estilo externa a un documento Html.

Para enlazar una hoja de estilos externa en un documento HTML se realiza a través de la etiqueta <link>. Este elemento deberá estar posicionado dentro de <head> con los atributos rel , type , href y title definidos para conseguir una adecuada carga de los estilos. (Zafra, 2015)

Incrustar un estilo dentro de un documento Html.

Para incrustar estilos se hace a través de la etiqueta <style> específica del documento, dentro de la sección <head>. Dentro de la sección <head> se pueden incluir tantos estilos CSS como sean necesarios, incluso toda una hoja de estilos completa sin que esta se almacene en un fichero externo. Siempre que se utilice la etiqueta <style> para referirse a estilos CSS es obligatorio incluir el atributo “type” indicando que es del tipo “text/css”. (Zafra, 2015)

Utilización y optimización de los tipos de estilos.

Según Zafra, la correcta utilización de las hojas de estilo es vital para hacer un desarrollo profesional fácil de editar y que no contenga errores visuales ni código repetido o innecesario, consiguiendo el menor tiempo de carga posible para hacer una experiencia más rápida en la navegación y adquirir mejor posicionamiento en los motores de búsqueda. Las recomendaciones para un uso correcto de estilos tienen mucho que ver

con las reglas de accesibilidad antes definidas, ya que todo está orientado a cumplir los estándares definidos por la W3C:

- Agrupar todos los estilos generales posibles en un único documento .css, y enlazarlo a los documentos HTML mediante la etiqueta <link>
- Evitar en la medida de lo posible los estilos incrustados y los estilos en línea.
- Separar los estilos destinados a distintos dispositivos (o medios) en hojas de estilo alternativas.
- Utilizar unidades relativas y porcentuales.

La optimización de los estilos es muy importante en el desarrollo y mantenimiento de un sitio, algo que a menudo no se tiene en cuenta. Las páginas destinadas a tener alto tráfico deben estar optimizadas, obteniendo archivos más ligeros y disminuyendo la carga del servidor al limitar el número de peticiones, mejorando el posicionamiento web. Como en cualquier lenguaje de programación, es una práctica habitual generar el código de la manera más fácil para el desarrollador, teniendo en cuenta algunas reglas de optimización, y una vez realizado, dedicar un tiempo a optimizarlo del todo. (Zafra, 2015)

Javascript

“JavaScript es un lenguaje de programación, leve, interpretado, orientado a objetos, basado en prototipos y en first-class functions (función de primera clase).” Fue creado para transformar las páginas web estáticas en dinámicas y sean más interactivas para el usuario, es decir páginas que tengan animación, textos e imágenes que aparezcan y desaparezcan, con botones que ejecuten alguna acción al dar clic sobre ellos y ventanas que muestren al usuario mensaje de aviso. Es importante mencionar que pese al nombre de JavaScript no existe ninguna relación con el lenguaje

de programación Java. (Ayoze, 2017)

JavaScript está orientada a objetos, debido a que puede contener datos y métodos que actúan sobre esos datos. Si mencionamos programación orientada a objetos lo relacionamos con la creación de clases y herencias, pero en JavaScript no cuenta con clases, pero tiene constructores que realizan las mismas operaciones que hacen las clases y los prototipos son la herencia. (Garro, 2014)

JavaScript maneja eventos, por ejemplo, el hacer clic en un botón, arrastrar, mover o apuntar el mouse es un evento y este realiza una acción de acuerdo a la programación realizada. Los eventos de JavaScript permiten la interacción entre las aplicaciones JavaScript y los usuarios. Cada vez que se pulsa un botón, se produce un evento, al igual que si pulsamos una tecla, también se produce un evento. Es importante mencionar que no solo realiza un evento cuando el usuario interviene, también cuando se carga una página se ejecuta un evento. Para que los eventos realicen alguna acción hay que otorgar una función que estará incluida en el código HTML, que interactuará con el Modelo de Objetos de la página. (Garro, 2014)

Por lo general el código JavaScript se inserta entre las etiquetas html: <script> </script>. También pueden estar ubicados en ficheros externos usando: <script type="text/javascript" src="micodigo.js"></script>. JavaScript es soportado por la mayoría de los navegadores como Internet Explorer, Opera, Mozilla Firefox, entre otros; y puede incluirse en cualquier documento y es compatible con HTML en el navegador del cliente.

Variables	Etiquetas que se refieren a un valor cambiante.
Operadores	Pueden usarse para calcular o comparar valores. Ejemplo: pueden sumarse dos valores, pueden compararse dos valores....
Expresiones	Cualquier combinación de variables, operadores, y declaraciones que evalúan a algún resultado. Ejemplo: intTotal=100;intTotal> 100
Sentencias	Una sentencia puede incluir cualquier elemento de la gramática de JavaScript. Las sentencias de JavaScript pueden tomar la forma de condicional , bucle, o manipulaciones del objeto. La forma correcta para separarlas es por punto y coma, esto sólo es obligatorio si las declaraciones múltiples residen en la misma línea. Aunque es recomendable que se acostumbre a terminar cada instrucción con un punto y coma, se ahorrará problemas.
Objetos	Estructura “Contenedora” de variables, procedimientos y funciones, cada valor refleja una propiedad individual de ese objeto.
Funciones y Métodos	Una función es un conjunto de elementos que realizan alguna acción. Puede aceptar los valores entrantes (los parámetros que le asignaremos a la función), y puede devolver un valor saliente. Un método simplemente es una función contenida en un objeto.

Tabla 4.1. Elementos principales de JavaScript. Adaptado de (Desarrollo Web, 2001)

JQuery

“jQuery es una biblioteca de JavaScript cross-browser desarrollada para simplificar los scripts client-side que interactúan con el HTML. Esta fue lanzada en enero de 2006 en el BarCamp de Nueva York por John Resig.”. Esta librería es utilizada en programación avanzada de aplicaciones, cuando se utiliza jQuery no es necesario preocuparse por el navegador con que se abrirá la página, debido a que esta librería puede acoplarse a cualquier librería. (Ayoze, 2017)

La librería de jQuery es de código abierto, ya que no es necesario adquirir una licencia para utilizarla. “La biblioteca también ofrece la

posibilidad de la creación de plugins sobre ella. Haciendo uso de tales facilidades, los desarrolladores pueden crear capas de abstracción para la interacción de más bajo nivel, simplificando el desarrollo de aplicaciones web dinámicas de gran complejidad.” (Ayoze, 2017)

La ventaja principal de jQuery es que es mucho más fácil aprender a utilizarlo. Se puede agregar plugins fácilmente, convirtiéndose en un ahorro de tiempo y esfuerzo. La comunidad Open Source de jQuery es amplia y permite que la librería siempre cuente con soporte constante y rápido, publicándose actualizaciones de manera constante. Una de las ventajas más enfatizadas de jQuery es que nos permite realizar efectos especiales para la página web, que son fáciles, rápidas de desarrollar y con pocas líneas de código. Estos efectos hacen que la página web sea más dinámica y lo más importante es atractiva ante el usuario, es de mucho valor que nuestra página web tenga un buen diseño que incluya efectos, ya que la mayor parte de las personas son más observadores en interfaz.

En la sintaxis de jQuery existe dos signos o caracteres que se deben poner de forma obligatoria, debido a que estos tienen responsabilidades importantes que harán que funciones correctamente la acción que se desea efectuar con la librería jQuery. Los caracteres son los siguientes “\$: Este carácter especial es una función de jQuery que busca todos los nodos (elementos) que concuerden con la expresión que escribamos en su interior.”, “#: Este carácter sirve para indicar un id que se encuentre dentro del HTML que estemos utilizando.” Estos dos caracteres son útiles para llamar un elemento preciso, puede ser para obtener su valor o cambiarlo, mediante estos dos signos se busca dentro del código HTML el elemento. (Montes & Carmona, 2015)

Por ejemplo para hacer un envío de información de los formularios se utiliza dos métodos, alguno de ellos se presenta en la Tabla 4.2:

MÉTODO	DESCRIPCIÓN
GET	GET lleva los datos de forma visible, el medio de envío es la URL.
POST	GET lleva los datos de forma oculta, el medio de envío es la URL.
data()	Sirve tanto para guardar un dato en un elemento como para consultarlos.
removeData()	Sirve para eliminar un dato de un elemento y su funcionamiento es tan simple como enviar por parámetro el dato que se quiere eliminar del elemento.

Tabla 4.2. *Métodos para envío de información por formularios. Adaptado de (González, 2009)*

El tipo de evento se define a partir de una función click () o similares. Existen diferentes tipos de funciones que implementan cada uno de los eventos normales, como dblclick(), focus(), keydown(), etc. Como parámetro en la función click() o similares tenemos que enviar una función, con el código que pretendemos ejecutar cuando se produzca el evento en cuestión. La función que enviamos por parámetro con el código del evento, en este caso la función a ejecutar al hacer clic, tiene a su vez otro parámetro que es el propio evento que estamos manejando. En el código anterior tenemos la variable “mevento”, que es el evento que se está ejecutando y a través de esa variable tenemos acceso a varias propiedades y métodos para personalizar aún más nuestros eventos. Como decimos, existen diversos tipos de propiedades y métodos sobre el evento que recibo por parámetro. En este caso utilizamos mevento.preventDefault() para evitar el comportamiento por defecto de un enlace. Como sabemos, al pulsar un enlace el navegador nos lleva al href definido en la etiqueta A correspondiente, algo que evitamos al invocar a preventDefault() sobre nuestro evento. (Alvarez, 2012)

Html, CSS, JavaScript y JQuery

HTML es la base fundamental de una página web, debido a que es el encargado de ordenar el contenido para que sea más entendible y claro para el usuario, también es necesario para marcar el contenido que el cliente visualizará a través de los diferentes navegadores, es decir leerá e indicará al navegador lo que se va a visualizar. CSS es el encargado del diseño de la apariencia, estilos, bordes, colores; es decir se encargará del maquillaje exacto para que la página web sea más atractiva y llame la atención del usuario. JavaScript es el encargado del comportamiento y la interacción con el usuario mediante eventos que efectúan una acción en la página web.

Si se desarrolla un sitio web con Html, CSS, Javascript y JQuery, todas estas herramientas de programación estarían ofreciendo al cliente un producto de calidad y también estarían reduciendo el tiempo de desarrollo en el equipo de trabajo. Teniendo en cuenta que cada lenguaje de programación tiene características particulares que facilitan la construcción de sitios web estáticos y dinámicos, con el objetivo de optimizar la construcción de aplicaciones web eficientes se recomienda la utilización de estos lenguajes de programación.

Para realizar un sitio web completo y bien estructurado es necesaria la utilización de los lenguajes de programación ya mencionados, sin dejar de lado ninguno de ellos ya que cada uno de ellos es complemento del otro, es así como se conseguirá un sitio web con mejor usabilidad. “Resumiendo y organizando las ideas, tenemos que cada lenguaje es responsable de lo siguiente: Contenido (HTML), Presentación (CSS), Comportamiento (JavaScript) y Eficiencia en comportamiento (Jquery).” (Ayoze, 2017)

Capítulo V

Aplicando lenguajes interactivos en ambiente Play Framework

En este capítulo se dará continuidad a la elaboración de nuestro caso de estudio ejemplo, en el Capítulo 4 se trató sobre la configuración de los prerequisitos necesarios para trabajar con un sistema de información web de tres capas. Ahora nos focalizaremos en la configuración, desarrollo e implementación de las necesidades de la primera capa o capa vista, es decir la preparación de una interfaz de usuario que permita gestionar información vinculante al SACC (Sistema de Administración de Carro de Compras). Las necesidades del usuario son:

- Visualizar la lista de productos que dispone una tienda.
- Seleccionar un determinado producto y registrar la cantidad que desea consumir.
- Administrar el carro de compras, que acumulará los productos que haya seleccionado.

Es importante mencionar que en este capítulo únicamente se trabajará por preparar una interfaz que ayude al usuario a realizar las acciones que requiere. Para lograr este objetivo se ha planificado las siguientes etapas, cuyo detalle se irá presentando en el transcurso de este capítulo:

- Boceto de la interfaz deseada
- Creación de la estructura HTML y CSS que soportará la interfaz
- Uso de un Slider con JQUERY para presentación de los productos
- Uso de JavaScript para permitir al usuario la selección de productos
- Uso de JavaScript para permitir al usuario la administración del carro de compras

Boceto de la interfaz deseada

Como primer paso, es de mucha ayuda realizar un boceto de las partes por la que estará compuesta la interfaz de usuario, existen muchas aplicaciones de licencia libre y propietaria que ayudan al diseño de este

tipo de esquemas, tales como: Fireworks, Balsamiq Mockups, Photoshop, Inkscape, Gimpshop y otros. Dos son los factores a tomar en cuenta para definir la realización de una interfaz de alta o baja elaboración, hablando estéticamente: el primero es la habilidad que tenga con el diseño gráfico y sus herramientas vinculantes; el segundo es el requerimiento planteado por el usuario, si busca un diseño llamativo por temas de publicidad u otras necesidades. Si el caso fuera, que se requiere una interfaz altamente elaborada y sus habilidades con el diseño gráfico no son óptimas, es mejor que trabaje en equipo con un diseñador gráfico.

Para nuestro caso de estudio ejemplo, no se busca hacer mayor relevancia en la estética, así que en el boceto de interfaz únicamente nos interesa dar ubicación a las secciones de trabajo que debe poseer la interfaz para soportar las funcionalidades planteadas para el SACC. En la Figura 5.1. se muestra las seis secciones que se han establecido para nuestra interfaz:

- SECCIÓN 1 o SECCIÓN CABECERA: Usada como cabecera de la aplicación, aquí puede ubicar logotipos o títulos de identificación de la página.
- SECCIÓN 2 o SECCIÓN MENÚ: Barra de menú, espacio en el que se puede publicar subtítulos de identificación u opciones de menú, que serían vínculos de accesibilidad a otros servicios de la página.
- SECCIÓN 3 o SECCIÓN CATÁLOGO: Espacio destinado a receptar contenidos para la temática de la página, en este caso será el lugar donde se presente el slider con el catálogo de productos que posea la tienda, para que el usuario pueda seleccionar el producto que deseé.
- SECCIÓN 4 o SECCIÓN DETALLE: Espacio destinado a receptar contenidos para la temática de la página, en este caso será el lugar donde el usuario visualice la información detallada del producto que ha seleccionado, permitiéndole también ingresar la cantidad de

productos que desea añadir al carro de compras.

- SECCIÓN 5 o SECCIÓN COMPRAS: Espacio destinado a receptar contenidos para la temática de la página, en este caso será el lugar donde se administra el carro de compras con los productos que ha seleccionado.
- SECCIÓN 6 o SECCIÓN PIE: Pie de página, espacio destinado para presentar información complementaria de la página, tales como contactos, dirección, horarios de atención y otros.

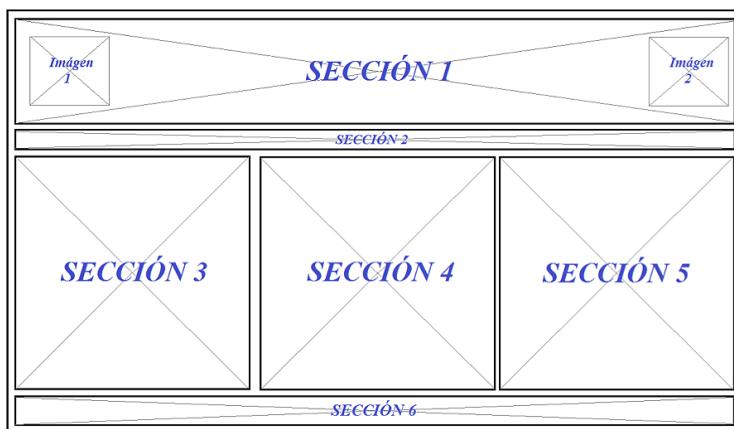
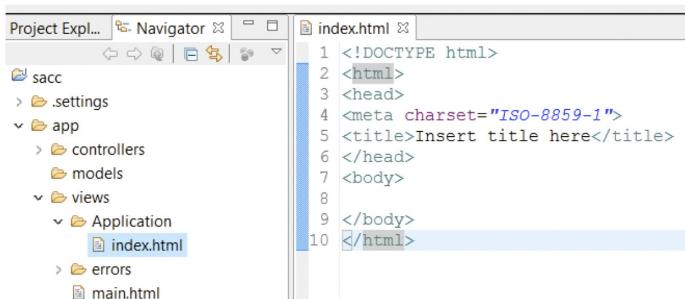


Figura 5.1.Secciones de la aplicación caso de estudio

Html y CSS

Una vez que se ha definido el boceto de la interfaz con el usuario, se procede a diseñar la estructura usando el lenguaje de hipertexto HTML y el lenguaje de estilos CSS, el complemento de ambos permite crear el diseño de la interfaz planteada. Para lograr armar esta estructura, sobre la base del proyecto SACC creado y utilizando el IDE Eclipse se ha seguido los siguientes pasos.

En primera instancia nos dirigimos a la carpeta “*app/views/Application*” del proyecto, en la carpeta *views* es dónde se almacenará todas las vistas o archivos html que tenga el proyecto, dentro de la carpeta Application encontrará el archivo *index.html*, que es la página de bienvenida por defecto que tiene play framework. Como vamos a diseñar nuestra propia interfaz puede cambiar de nombre o borrar este archivo existente para crear su propio *index.html*, al crear su propio archivo, tendrá la estructura básica html (ver Figura 5.2) sobre la cual vamos a trabajar.



The screenshot shows a software interface with a 'Project Explorer' on the left and a code editor on the right. The Project Explorer displays a file tree for a project named 'sacc'. Under the 'app' directory, there are 'controllers', 'models', 'views', and 'errors' sub-directories. Within 'views', there is a 'Application' folder which contains two files: 'index.html' (selected in the list) and 'main.html'. The code editor window on the right shows the content of 'index.html':

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="ISO-8859-1">
5 <title>Insert title here</title>
6 </head>
7 <body>
8
9 </body>
10 </html>
```

Figura 5.2. Estructura básica de un archivo Html

Una vez que tenemos nuestro propio *index.html*, se procede a editarlo de acuerdo a la estructura de diseño de interfaz que buscamos, de acuerdo a lo planificado empezaremos creando la Sección 1, es necesario mencionar que se usa lenguaje html para crear la estructura de la sección y por otro lado se utiliza lenguaje css para dar formato a dicha estructura, es así como por medio de una sinergia entre ambos lenguajes se logra obtener la interfaz de usuario deseada. Recuerde que los tags html que utilice los escribe dentro del **body** y el código css dentro del **head** porque para este caso la implementación de los estilos se lo está haciendo de forma embebida. Además, tomar en cuenta que cada sección está soportada por un *<div></div>*

SECCIÓN 1 o SECCIÓN CABECERA: Para el caso de esta sección, por el hecho de requerir ubicar dos imágenes ha sido necesario el uso de

dos <div> internos más. Uno de ellos alineado a la izquierda y el otro a la derecha, los tres estilos definidos nos ayudan a determinar el fondo de la sección, sus dimensiones, tamaño y color de letra. En la Figura 5.3a tenemos el código html necesario para crear la sección con sus imágenes internas y en la Figura 5.3b tenemos el código css necesario para dar el formato adecuado a dicha sección. Finalmente, el resultado obtenido es una interfaz con una cabecera, acompañada por un título central y dos logotipos.

En el proceso de construcción de esta sección es importante mencionar que las imágenes requeridas tienen que ser cargadas físicamente en nuestro proyecto, en la carpeta de la siguiente ruta “/public/images/”. Recordar que todos los archivos complemento que se vinculen a la interfaz tales como imágenes, estilos o pugins javascripts tienen que ser ubicados en la carpeta del proyecto /public, la creación de nuevas carpetas para una mejor organización de sus archivos es decisión del desarrollador.

A

```

31 <body>
32
33 <div id="seccion1">
34   <div id="img1">
35     
36   </div>
37   <b>UNIVERSIDAD TECNICA DE COPÓPAXI</b>
38
39   <div id="img2">
40     
41   </div>
42   </div>
43 </div>
44 </body>
45
46 </html>
```

B

```

3 <head>
4   <meta charset="ISO-8859-1">
5   <title>Sistema de Administración de un Carro de Compras</title>
6
7 <style>
8   #seccion1 {
9     background-color: black;
10    height: 150px;
11    color: white;
12    text-align: center;
13    padding: 10px;
14    line-height: 110px;
15    font-size: xx-large;
16  }
17
18  #img1 {
19    float: left;
20    padding: 5px;
21  }
22
23  #img2 {
24    float: right;
25    padding: 5px;
26  }
27 </style>
28
29 </head>
```

Figura 5.3. Código HTML y CSS de la Sección 1

SECCIÓN 2 o SECCIÓN MENÚ: En la Figura 5.4a se puede apreciar el código html que permite estructurar un contenido para usarlo como subtítulo de la interfaz, en la Figura 5.4b en cambio tenemos el código css que brinda a la sección el formato adecuado para que sea presentado como una barra de menú, en esta ocasión el área ha sido usada para presentar un subtítulo, pero si se decidiera en el sistema implementar más servicios el área podría ser usada para fijar un menú de opciones.

```
A
<div id="section2">Ingeniería en Informática y Sistemas  
Computacionales</div>
B
#section2 {  
background-color: red;  
color: white;  
font-size: large;  
text-align: center;  
padding: 5px;  
}
```

Figura 5.4. Código HTML y CSS de la Sección 2

SECCIÓN 3 o SECCIÓN CATÁLOGO: La sección tres ofrece una estructura destinada a registrar y presentar contenidos en la zona central de la página con una alineación a la izquierda. En la Figura 5.5a tenemos al código html de la sección, lista para recibir la información que se desee presentar, y en la Figura 5.5b tenemos el código de los estilos que le dan la forma rectangular y el tamaño de la sección.

```
A
<div id="section3"></div>
B
#section3 {  
height: 450px;  
width: 420px;  
float: left;  
padding: 5px;  
border-style: solid;  
border-width: 1px;  
}
```

Figura 5.5. Código HTML y CSS de la Sección 3

SECCIÓN 4 o SECCIÓN DETALLE: La sección cuatro ofrece una estructura destinada a registrar y presentar contenidos en la zona central de la página con una alineación a la mitad. En la Figura 5.6a tenemos al código html de la sección, lista para recibir la información que se deseé presentar, y en la Figura 5.6b tenemos el código de los estilos que le dan la forma rectangular y el tamaño de la sección.

```
<div id="section4"></div>
```

```
#section4 {  
    height: 450px;  
    width: 420px;  
    float: left;  
    padding: 5px;  
    border-style: solid;  
    border-width: 1px;  
}
```

Figura 5.6. Código HTML y CSS de la Sección 4

SECCIÓN 5 o SECCIÓN COMPRAS: La sección cinco ofrece una estructura destinada a registrar y presentar contenidos en la zona central de la página con una alineación a la derecha. En la Figura 5.7a tenemos al código html de la sección, lista para recibir la información que se deseé presentar, y en la Figura 5.7b tenemos el código de los estilos que le dan la forma rectangular y el tamaño de la sección.

```
<div id="section5"></div>
```

```
#section5 {  
    height: 450px;  
    width: 420px;  
    float: left;  
    padding: 5px;  
    border-style: solid;  
    border-width: 1px;  
}
```

Figura 5.7. Código HTML y CSS de la Sección 5

SECCIÓN 6 o SECCIÓN PIE: Finalmente con el código html de la última sección presentado en la Figura 5.8a se estructura el contenido de la información complementaria que se desea presentar en la página, y en la Figura 5.8b se encuentra el código de estilos que dan formato a la sección.

```
A
<div id="section5"></div>

B
#section5 {
    height: 450px;
    width: 420px;
    float: left;
    padding: 5px;
    border-style: solid;
    border-width: 1px;
}
```

Figura 5.8. Código HTML y CSS de la Sección 6

SECCIÓN 6 o SECCIÓN PIE: Finalmente con el código html de la última sección presentado en la Figura 5.8a se estructura el contenido de la información complementaria que se desea presentar en la página, y en la Figura 5.8b se encuentra el código de estilos que dan formato a la sección.

[A]

```

index.html :: 
1 <!DOCTYPE html>
2 <html>
3 <head>
4 |meta charset="ISO-8859-1">
5 <title>Sistema de Administración de un Carro de Compras</title>
6
7 <style>
8 #section1 {
9     background-color: black;
10    height: 150px;
11    color: white;
12    text-align: center;
13    padding: 10px;
14    line-height: 110px;
15    font-size: xx-large;
16 }
17
18 #img1 {
19     float: left;
20     padding: 5px;
21 }
22
23 #img2 {
24     float: right;
25     padding: 5px;
26 }
27
28 #section2 {
29     background-color: red;
30     color: white;
31     font-size: large;
32     text-align: center;
33     padding: 5px;
34 }
35
36 #section3 {
37     height: 450px;
38     width: 420px;
39     float: left;
40     padding: 5px;
41     border-style: solid;
42     border-width: 1px;
43 }
44
45 #section4 {
46     height: 450px;
47     width: 420px;
48     float: left;
49     padding: 5px;
50     border-style: solid;
51     border-width: 1px;
52 }
53
54 #section5 {
55     height: 450px;
56     width: 420px;
57     float: left;
58     padding: 5px;
59     border-style: solid;
60     border-width: 1px;
61 }
62
63 #section6 {
64     background-color: black;
65     color: white;
66     clear: both;
67     text-align: center;
68     padding: 5px;
69     font-size: smaller;
70 }
71 </style>
72
73 </head>
```

Figura 5.8. Código HTML y CSS de la Sección 6

```

74 <body>
75
76     <div id="section1">
77         <div id="img1">
78             
79         </div>
80
81         <b>UNIVERSIDAD TÉCNICA DE COTOPAXI</b>
82
83         <div id="img2">
84             
86         </div>
87
88
89         <div id="section2">Ingeniería en Informática y Sistemas
90             Computacionales</div>
91
92         <div id="section3"></div>
93
94         <div id="section4"></div>
95
96         <div id="section5"></div>
97
98         <div id="section6">Av. Simón Rodríguez s/n Barrio El Ejido Sector
99             San Felipe. Latacunga - Ecuador. Teléfonos: (593) 03 2252205 / 2252307
100            / 2252346. www.utc.edu.ec institucional@utc.edu.ec</div>
101
102
103
104     </body>
105 </html>

```

Figura 5.8. Código HTML y CSS de la Sección 6

El resultado final de esta sinergia de desarrollo entre los lenguajes html y css ha sido la interfaz presentada en la Figura 5.10, se ha diseñado la interfaz de acuerdo a los lineamientos iniciales que se definieron con el usuario. Ahora tenemos listas las seis secciones de nuestro sistema de información web ejemplo que se irán dotando de funcionalidades en el transcurso de los siguientes capítulos.



Figura 5.10. Interfaz de la aplicación de caso de estudio

Jquery

Una vez que tenemos el diseño de la interfaz de nuestro sistema de información web, que se ha estructurado con el uso de lenguaje html y css, vamos a incorporar la sinergia de la sintaxis del lenguaje y librería Jquery. Para el ejemplo de uso de esta biblioteca se trabajará en la SECCIÓN 3 o SECCIÓN CATÁLOGO, el objetivo es usar un slider de imágenes para presentar los productos que posee el catálogo. Para el uso de este slider es necesario incorporar al proyectos cuatro elementos esenciales para el correcto funcionamiento del slider, a continuación se detalla cada uno de estos elementos y su rol a cumplir en el proyecto:

- *Biblioteca de Jquery:* Para poder hacer uso de la sintaxis de Jquery es necesario descargarse la biblioteca de la página oficial de Jquery (www.jquery.com/), constantemente se están publicando nuevas versiones de la biblioteca con mejores funcionalidades. Sin embargo, es importante mencionar que, con la creación de nuestro proyecto de Play Framework, por defecto tenemos una versión de la biblioteca de Jquery que la podemos usar, ésta se encuentra en la siguiente ruta del proyecto “/public/javascripts/”.
- *Librería de Thumbelina:* Es la librería que posee los efectos que dispone el slider, es decir el movimiento que permite los botones para moverse a la derecha y a la izquierda. Otra de sus funciones es tomar las imágenes que se desean presentar e incorporarlas en el slider. Esta librería es gratuita y puede descargarse de varias páginas de la red, una de ellas es el repositorio de software de GitHub (www.github.com/StarPlugins/thumbelina) la denominación de la librería es “thumbelina.js”. Una vez descargado, el archivo de esta librería debe ser ubicado en la siguiente ruta del proyecto “/public/javascripts/”.

- *Archivo de estilos Thumbelina:* Es un archivo “css” que posee todos los formatos de estilo para el slider, tales como el diseño de los botones, la forma de presentación de las imágenes, el tamaño, sus bordes, su posición y otros. Este archivo de estilos es gratuito y lo puede descargar de varias páginas de la red, una de ellas es el repositorio de software de GitHub (www.github.com/StarPlugins/thumbelina) la denominación del archivo de estilos es “thumbelina.css”. Una vez descargado, el archivo de esta librería debe ser ubicado en la siguiente ruta del proyecto “/public/stylesheets/”.
- *Imágenes para Slider:* Corresponde a las imágenes que desea presentar en el Slider, estas imágenes deben ser ubicadas en la carpeta de nuestro proyecto con la siguiente ruta “/public/images/”. Para nuestro ejemplo las imágenes que se incorporarán serán imágenes de bebidas de consumo: Coca Cola, Fanta, Sprite, Fioravanti, Inca Kola, Nestea.

Con las librerías, el archivo de estilos y las imágenes en nuestro proyecto (ver Figura 5.11), estamos preparados para aplicar el código necesario para que en nuestra interfaz podamos apreciar el catálogo de productos en un Slider. Sobre la base de nuestra estructura de interfaz se incorpora nuevo código html, css y ahora Jquery que nos permitirá lograr la funcionalidad que deseamos.

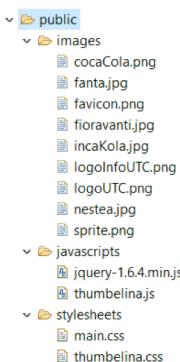
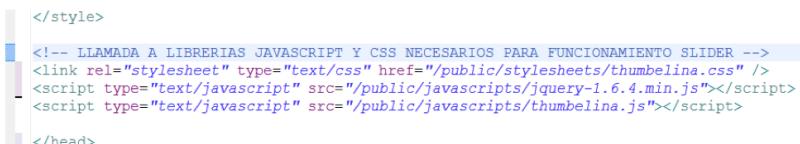


Figura 5.11. Ubicación para los archivos tipo: imagen, javascript y css

Lo primero que hacemos es dirigirnos a la cabecera “`<head>`” de nuestro “index.html” ahí hacemos el llamado a las librerías y archivos de estilos (ver Figura 5.12) que se han incorporado en nuestro proyecto. Esto nos permitirá manipular las funcionalidades y estilos de formato del Slider Thumbelina, así se podrá ubicar el Slider donde desee con la cantidad de productos que deseé.



```

</style>

<link rel="stylesheet" type="text/css" href="/public/stylesheets/thumbelina.css" />
<script type="text/javascript" src="/public/javascripts/jquery-1.6.4.min.js"></script>
<script type="text/javascript" src="/public/javascripts/thumbelina.js"></script>
</head>

```

Figura 5.12. Llamada a las librerías javascript y css

El siguiente paso es escribir el código css que se muestra en la Figura 5.13a que dará formato, tamaño y ubicación deseada al Slider. La ubicación de este código será en la cabecera `<head>` dentro de los estilos `<style>`. En la Figura 5.13b se presenta el código de sintaxis Jquery que hace un llamado a las funcionalidades definidas en la librería “thumbelina.js” para permitir la movilidad hacia la izquierda o derecha del Slider. La ubicación de este código será en la cabecera `<head>` dentro de los tags de scripts `<script>`.



A
B

```

#slider1 {
    position: relative;
    margin-left: 80px;
    width: 256px;
    height: 120px;
    border-top: 1px solid #aaa;
    border-bottom: 1px solid #aaa;
    position: relative;
}
</style>


<script type="text/javascript">
$(function() {
    $('#slider1').Thumbelina({
        $bwdBut : $('#slider1 .left'),
        $fwdBut : $('#slider1 .right')
    });
})
</script>
</head>

```

Figura 5.13. Código CSS y JQuery para Slider

Como último paso, en la Figura 5.14 tenemos la escritura del código que creará el espacio estructural para el Slider, así como también el lugar donde se adecuarán la lista de imágenes que se desea cargar en

el Slider. Cada uno de los tags agregados a esta sección, llamarán a los estilos, clases y archivos que requieran por medio de sus identificadores, nombres de formatos de estilos y ubicación de archivos. Este código se encuentra ubicado en el cuerpo <body> de nuestro “index.html” en la Sección 3 o Sección Catálogo <div id=“section3”>.

```
<div id="section3">
    <h1>Catálogo de Productos</h1>
    <div id="slider1">
        <div class="thumbelina-but horiz left"></div>
        <ul>
            <li></li>
            <li></li>
            <li></li>
            <li></li>
        </ul>
        <div class="thumbelina-but horiz right"></div>
    </div>
</div>
```

Figura 5.14. Código HTML para Slider

Finalmente, el resultado del desarrollo aplicado a la Sección de Catálogo se lo aprecia en la Figura 5.15. Es importante mencionar que al momento se ha logrado la sinergia de la sintaxis de tres lenguajes: html, css y Jquery, éste último ya no es únicamente un lenguaje de marcado e hipertexto, es un lenguaje que ya nos permite concebir funcionalidades con una lógica de programación.



Figura 5.15. Slider de imágenes en la interfaz

JavaScript Sección Detalle

Para la sinergia y uso funcional del lenguaje de programación interpretado JavaScript, se hará uso de la SECCIÓN 4 o SECCIÓN DETALLE. La meta que nos hemos propuesto para esta sección es que haciendo uso de JavaScript se permita lograr la siguiente acción: brindar al usuario interactividad con la interfaz, a través del catálogo de productos el usuario podrá seleccionar el producto que desee a través de un click, una vez que haya seleccionado su producto, en la sección de detalle presentaremos al usuario detalles del producto (descripción, precio).

Para conseguir este objetivo, en el proyecto como primer paso se ha tenido que desarrollar en la sección detalle una interfaz con los componentes que soportarán la información del detalle del producto que el usuario haya seleccionado. Luego de brindarle un formato a estos componentes se declara el llamado a una función JavaScript en el evento onClick del componente imagen de la Sección Catálogo, pues del Slider es de

donde el usuario elige el producto que desea para posteriormente cargar la información del detalle del producto elegido en la Sección Detalle. Finalmente se implementa la acción de la función JavaScript, la misma que posee el código necesario para cargar la imagen e información determinada de acuerdo al producto que haya seleccionado el usuario en el catálogo.

En la Figura 5.16 se aprecia el código necesario para crear todos los componentes donde se presentará la información del detalle del producto. Para esto ha sido necesario el uso una tabla que soporte toda la información y dentro de ella: una imagen, cinco títulos de cabecera, cuatro input text y un input button, a cada uno de estos componentes se les ha fijado una identificación, formato, tamaño y ubicación. Este código se ubica en el cuerpo <body> de nuestro “index.html” en la Sección 4 o Sección Detalle <div id=“section4”>.

```
<div id="section4">
    <h1>Detalle del Producto</h1>
    <table border="1" style="margin: 30px;">
        <tr align="center">
            <td colspan="6"></td>
        </tr>
        <tr align="center">
            <td>Código</td>
            <td>Descripción</td>
            <td>Precio</td>
            <td>Cantidad</td>
            <td>Acción</td>
        </tr>
        <tr align="center">
            <td><input id="code" name="code" type="text" size="4"
                style="background: #eeeeee;"></td>
            <td><input id="desc" name="desc" type="text" size="8"
                style="background: #eeeeee;"></td>
            <td><input id="price" name="price" type="text" size="4"
                style="background: #eeeeee;"></td>
            <td><input id="qty" name="qty" type="text" size="4"></td>
            <td><input id="add" name="add" value="Aregar" type="submit"
                size="4"></td>
        </tr>
    </table>
</div>
```

Figura 5.16. Código HTML de la Sección Detalle del Producto

Posteriormente en la Sección Catálogo, en el evento onClick de cada una de las imágenes de los productos se declara el llamado a la función “*loadImage(this);*” (ver Figura 5.17). Es así como estamos receptando la acción del usuario con la interfaz, cada vez que da click en una de las imágenes del catálogo, nosotros determinamos que producto es el que ha seleccionado para finalmente de acuerdo a su selección cargar el detalle del producto que corresponda en la Sección Detalle. Este código se ubica en el cuerpo <body> de nuestro “*index.html*” en la Sección 3 o Sección Catálogo <div id=“section3”> el llamado a la función se ubica como una propiedad de cada uno de los tags de imagen.

```
<div id="section3">
    <h1>Catálogo de Productos</h1>
    <div id="slider1">
        <div class="thumbelina-but horiz left"><</div>
        <ul>
            <li></li>
            <li></li>
            <li></li>
            <li></li>
        </ul>
        <div class="thumbelina-but horiz right"><</div>
    </div>
</div>
```

Figura 5.17.Llamado a una función JavaScript

Como último paso en la Figura 5.18 se muestra el desarrollo de la acción de la función “*loadImage()*”. La lógica de programación de esta función tiene una sintaxis del lenguaje JavaScript y está focalizada en que cada vez que el usuario hizo click en una de las imágenes del catálogo, la función recibe el identificador (id) de esta imagen y de acuerdo a su identificador carga la imagen que corresponda, ya sea esta: Coca Cola, Fanta, Sprite o Fioravanti. Este código se ubica en la cabecera <head> de nuestro “*index.html*” dentro de los tags de scripts <script>.

```

function loadImage(img) {
    if (img.id == 1) {
        document.getElementById("imageSelected").src = "/public/images/cocaCola.png";
        document.getElementById("code").value = "1";
        document.getElementById("desc").value = "Coca Cola";
        document.getElementById("price").value = "0.75";
    }
    if (img.id == 2) {
        document.getElementById("imageSelected").src = "/public/images/fanta.jpg";
        document.getElementById("code").value = "2";
        document.getElementById("desc").value = "Fanta";
        document.getElementById("price").value = "0.50";
    }
    if (img.id == 3) {
        document.getElementById("imageSelected").src = "/public/images/sprite.png";
        document.getElementById("code").value = "3";
        document.getElementById("desc").value = "Sprite";
        document.getElementById("price").value = "0.60";
    }
    if (img.id == 4) {
        document.getElementById("imageSelected").src = "/public/images/fioravanti.jpg";
        document.getElementById("code").value = "4";
        document.getElementById("desc").value = "Fioravanti";
        document.getElementById("price").value = "0.50";
    }
}
</script>

```

Figura 5.18. Función JavaScript

Finalmente, con las características planteadas inicialmente, se ha conseguido desarrollar la interfaz de la Sección Detalle. En la Figura 5.19 se muestra el detalle del producto Sprite, después de que el usuario a dado click o ha seleccionado del catálogo el producto que deseaba.



Figura 5.19. Sección del detalle del producto en interfaz

Es importante recordar que se ha incrementado al proyecto la sinergia del lenguaje JavaScript, nuestro proyecto al momento posee una estructura y una interfaz ahora interactiva con el usuario, gracias al uso de los lenguajes: html, css, Jquery y Javascript. Los dos últimos, siendo lenguajes de programación han fortalecido la interactividad del usuario con nuestra interfaz, ahora la información presentada ya no solo es estática, sino que a través de acciones que el usuario realice con la interfaz a través del teclado o el mouse, nuestra página recibe nuevos parámetros para poder realizar nuevas acciones. La interactividad y los efectos visuales es la fortaleza de estos lenguajes de programación interpretados (JavaScript y Jquery) conocidos también con el nombre de lenguajes del lado del cliente. En esta sección se ha presentado una pequeña muestra de su potencial y uso.

JavaScript Sección Compras

En la SECCIÓN 5 o SECCIÓN COMPRAS se ha destinado un espacio de contenido para que el sistema de información web presente información referente al carro de compras que el usuario está preparando con su selección de productos. Cada vez que el usuario en la Sección de Detalle registra una cantidad del producto que desea y presiona el botón agregar, este producto se irá añadiendo al carro de compras. Es importante mencionar que esta acción implica el registro de la transacción en una base de datos (contenidos de tercera capa o capa de datos), y al momento nos encontramos presentando contenidos sobre la primera capa o capa de presentación por esta razón la meta de esta sección es únicamente preparar la interfaz necesaria para presentar esta información, posteriormente en los siguientes capítulos se implementará la funcionalidad completa.

Para cumplir con el objetivo del diseño de la interfaz de esta Sección de Compras primero ha sido necesario escribir el código para ubicar

los componentes. Posteriormente se analiza que determinados botones, tales como la eliminación de un ítem o la finalización de la compra, requerirán una confirmación previa antes de ser usados, por esta razón ha sido necesario declarar una función JavaScript para controlar la acción en cada uno de estos botones. Finalmente se escribe el código JavaScript necesario para que estas funciones brinden una alerta de confirmación antes de realizar su acción.

En la Figura 5.20 se muestra el código html y css escrito para crear los componentes necesarios para esta sección. Dentro de la Sección de Compras se ha hecho uso de: una tabla con una fila de títulos de descripción; varias filas con cuatro componentes “input text” para presentar información acerca de la descripción, precio, cantidad y subtotal a pagar del producto; cada fila con dos botones “input button” que permitirán la acción de editar la cantidad deseada del producto y eliminar el ítem del carro de compras. Y en la última fila un componente “input text” para presentar la suma total de todos los ítems seleccionados por el usuario, acompañado por un botón que permitirá la finalización de la compra. Este código se ubica en el cuerpo `<body>` de nuestro “index.html” en la Sección 5 o Sección Compras `<div id=“section5”>`.

```

<div id="section5">
    <h1>Carro de Compras</h1>
    <table border="1" style="margin: 5px;">
        <tr align="center">
            <td>Descripción</td>
            <td>Precio</td>
            <td>Cantidad</td>
            <td>Subtotal</td>
            <td>Acción</td>
        </tr>
        <tr align="center">
            <td><input id="desc1" name="desc1" type="text" size="8"
                style="background: #eeeeee;"></td>
            <td><input id="price1" name="price1" type="text" size="4"
                style="background: #eeeeee;"></td>
            <td><input id="qty1" name="qty1" type="text" size="4"></td>
            <td><input id="subto1" name="subto1" type="text" size="4"
                style="background: #eeeeee;"></td>
            <td><input id="edit1" name="edit1" value="Editar" type="submit"
                size="4"> <input id="delete1" onClick="deleteItem();"
                name="delete1" value="Eliminar" type="submit" size="4"></td>
        </tr>
        <tr align="center">
            <td><input id="desc2" name="desc2" type="text" size="8"
                style="background: #eeeeee;"></td>
            <td><input id="price2" name="price2" type="text" size="4"
                style="background: #eeeeee;"></td>
            <td><input id="qty2" name="qty2" type="text" size="4"></td>
            <td><input id="subto2" name="subto2" type="text" size="4"
                style="background: #eeeeee;"></td>
            <td><input id="edit2" name="edit2" value="Editar" type="submit"
                size="4"> <input id="delete2" onClick="deleteItem();"
                name="delete2" value="Eliminar" type="submit" size="4"></td>
        </tr>
        <tr align="center">
            <td colspan="3" align="right">TOTAL:</td>
            <td><input id="price2" name="price2" type="text" size="4"
                style="background: #eeeeee;"></td>
            <td><input id="edit2" name="edit2" onClick="endSale();"
                value="Finalizar" type="submit" size="4"></td>
        </tr>
    </table>
</div>

```

Figura 5.20. Llamado a la función eliminar de JavaScript

En la Figura 5.20 tenemos secciones marcadas en semicírculos que corresponden al código de declaración de las funciones JavaScript que controlarán las acciones de esos botones. Los botones en los que se requieren dar un aviso de confirmación antes de ejecutar su acción son los de eliminación de ítem y el de confirmación de la compra. Como último en la Figura 5.21 tenemos el código escrito para las funciones

JavaScript declaradas, con dicho código se puede enviar una alerta al usuario antes de que se ejecute la acción de eliminación o confirmación. Esto ayudará a que el usuario analice en una instancia más, si está seguro de su acción. Este código se ubica en la cabecera <head> de nuestro “index.html” dentro de los tags de scripts <script>.

```
function deleteItem() {
    confirm('Está seguro que desea borrar este item?');
}

function endSale() {
    confirm('Está seguro que desea finalizar la compra?');
}
</script>
```

Figura 5.21. Validación de confirmación para la función JavaScript de eliminar.

Finalmente, en la Figura 5.22 se puede apreciar la interfaz que se ha desarrollado con el código escrito, así como también el mensaje de alerta para el usuario, antes de que la acción requerida por los botones críticos sea ejecutada.



Figura 5.22. Sección de carro de compras en interfaz

Es importante mencionar que la funcionalidad completa de la administración del carro de compras requiere un almacén de datos, siendo esta temática parte de los contenidos de los capítulos venideros, por el momento se ha preparado la interfaz necesaria para su uso. Por otro lado, en esta sección no se ha incorporado a la sinergia del proyecto el uso de un nuevo lenguaje, al momento seguimos teniendo cuatro lenguajes de trabajo: html, css, JavaScript y Jquery.

Sección 3: Segunda capa de negocio

En esta sección se abordará el detalle de los aspectos conceptuales y funcionales de la segunda capa de una estructura MVC, generalmente a este segmento de un sistema de información web se le denomina capa de negocio. Su característica e importancia principal es el procesamiento de datos de las peticiones que el usuario ha enviado a través de la interfaz. Una vez que estas peticiones son procesadas, los datos resultado son enviados a la tercera capa para que sean almacenados. Podemos decir entonces que parte del rol de esta capa es ser un intermediario entre la primera y la tercera capa.

El desarrollo de esta capa, se lo realiza de acuerdo con los requerimientos funcionales que el usuario haya definido, generalmente cuando existe una gran cantidad de requerimientos es necesario generar módulos de trabajo, agrupando los requerimientos que tienen correlación. Toda información que ingresa o solicita el usuario debe ser procesada en la capa de negocio para finalmente emitir una información de salida, la cual será la que se almacene o extraiga de la capa de datos. La información solicitada o enviada por el usuario en esta capa de negocio sufre diversos cambios para cumplir con lo planteado en el requerimiento. La calidad de una segunda capa se la puede apreciar en el momento que grandes volúmenes de información son entregados a esta capa para su procesamiento, si la capa posee un buen diseño, ésta responde con

eficiencia y precisión en la obtención de los resultados.

Generalmente la segunda capa es diseñada con modelos orientados a objetos, este tipo de paradigma brinda una mejor flexibilidad al momento de desarrollar los requerimientos que desea el usuario y sus propiedades de encapsulación brindar mayor seguridad a los datos que se procesan. Para trabajar con este paradigma es fundamental realizar un diagrama de clases que soporte todos los requerimientos funcionales que se han definido con el usuario, el diagrama de clases es la base para el desarrollo de toda la capa de negocio del sistema de información web. Expuesto el preámbulo funcional de la capa de negocio, en esta sección también se presenta los aspectos teóricos que dan lugar a la construcción de una capa con lógica de negocio, tales como: las diferentes opciones de lenguajes de programación para la capa de negocio que existen, las ventajas del lenguaje de programación Java y el paradigma orientado a objetos al momento de realizar procesos de negocio. En la etapa final de la sección sobre la base del ambiente práctico desarrollado en el Capítulo 7, se da continuidad a la construcción del sistema de información web ejemplo, focalizando la construcción y desarrollo, de lo que corresponde a los aspectos funcionales de la segunda capa.

Capítulo VI

Lenguajes de capa de negocio (JAVA)

La lógica de negocio es un conjunto de reglas que se siguen en el software para reaccionar ante distintas situaciones. Al utilizar una lógica de negocios en una aplicación, se logrará tener un mejor procesamiento de la información que solicite o envíe el usuario mejorando la comunicación entre el sistema de información y el usuario. (Alvarez, Desarroll Web, 2014).

Otros autores la identifican también como capa de negocio, y se menciona que esta capa es la encargada de procesar y validar las reglas del negocio, actúa como un servidor para la capa de presentación y traslada las solicitudes de ésta a la capa de datos actuando como cliente de la misma. En otras palabras, se puede decir que en esta capa de negocio existen tareas y cada una de ellas está definida como una operación por los requerimientos de la aplicación. Una tarea de negocios es la operación que define el usuario en la aplicación. (Coti, 2003)

Estas reglas de negocio en relación con el tiempo tienden a cambiar con frecuencia de acuerdo a las tareas específicas de negocios a las que dan soporte, es por este motivo que es necesario guardar las reglas de negocio en diferentes componentes. La aplicación debe almacenar la lógica de negocio en un entorno independiente de los demás componentes de la estructura del sistema de información.

Funciones de la capa de Negocio

Recibir la información de la capa de presentación: Es una de las funciones más importantes ya que la lógica de negocio recibe las tareas emitidas por el usuario a través de la capa de presentación. Esta capa es la que interactúa con el usuario y recibe los requerimientos o información que envía el usuario.

Procesar la información en la capa de negocio: Una vez que se ha receptado los datos enviados por la capa de presentación, las reglas de negocio definidas empiezan a actuar, procesando toda la información receptada con el fin de obtener un resultado.

Enviar resultados a la capa de presentación: La lógica de negocio permite la transferencia de los resultados emitidos de las soluciones o problemas que se encontraron en el proceso, esta información llega a la capa de presentación para que sea interpretada por el usuario.

Varios son los lenguajes de programación usados en la capa de negocio para generar reglas de negocio, entre ellos tenemos: Java, C#, PHP, Ruby, Python y otros. Para este caso de estudio se ha hecho uso de Java por sus características OpenSource y su paradigma de lenguaje de programación orientada a objetos. Es por esta razón que a continuación se describirá con mayor detalle las características del lenguaje de programación Java.

Historia de Java

Inicialmente un equipo de trabajo de la empresa SUN MICROSYSTEM se encontraba trabajando en un proyecto denominado Green en este equipo se encontraba el ingeniero Bill Joy, el proyecto consistía en desarrollar aplicaciones para teléfonos y televisores. El grupo tenía amplios conocimientos sobre la programación orientada a objetos, por lo cual decidieron trabajar en C++ ya que en esa época era el mejor lenguaje que demostraba sus capacidades. Pero a medida que se iba desarrollando C++ empezó a mostrar rápidamente muchas limitaciones en cuanto al ámbito de desarrollo, ya que mostraba numerosas incompatibilidades con hardware y los diferentes sistemas operativos que existían en esa época y sobre todo el mayor problema fue la portabilidad.

Por todas las dificultades que mostró C++ en el proceso de desarrollo, decidieron que la mejor opción era crear un nuevo lenguaje de programación que se acoplara a sus necesidades. El equipo de trabajo se puso a trabajar en el nuevo lenguaje donde se desarrolló el lenguaje OAK estaba orientado para la creación de software para la televisión interactiva, este lenguaje era pequeño, firme y una de las características principales era que estaba orientado a objetos. Pero el nuevo lenguaje fracaso y fue entonces cuando desarrollaron una máquina virtual la cual era independiente del hardware y de la plataforma en la cual se trabaje; ya que estaba orientado a trabajar en el desarrollo de aplicaciones para internet y una de las cosas que en este lenguaje apareció fue el fácil manejo para sustituir el código nativo del nuevo lenguaje, lo cual fue de gran ayuda para los programadores.

“La creación de este lenguaje y plataforma se inspiró en las funcionalidades interesantes propuestas por otros lenguajes tales como C++, Eiffel, SmallTalk, Objective C, Cedar/Mesa, Ada, Perl” (Groussand, 2014). El resultado la unión de los lenguajes antes mencionados dio origen a una plataforma y un lenguaje adecuado para el desarrollo de las diferentes aplicaciones que deben ser seguras, fácil de manejar y portables ya que deben trabajarse también por vía Internet. Es importante mencionar que al principio JAVA fue denominado C++--, pero después de un tiempo su nombre fue cambiado a OAK, más tarde decidieron llamarlo JAVA que significa café, debido a que durante todo su proceso de desarrollo el equipo de investigadores que se encontraban a cargo en especial los programadores y diseñadores bebieron grandes cantidades de café y así fue como nació JAVA en 1991.

“Al programar en Java no se parte de cero. Cualquier aplicación que se desarrolle depende (o se apoya, según como se quiera ver) en un gran número de clases preexistentes. Algunas de ellas las ha podido hacer el propio usuario, otras pueden ser comerciales, pero siempre hay un

número muy importante de clases que forman parte del propio lenguaje (el API o Application Programming Interface de Java). Java incorpora en el propio lenguaje muchos aspectos que en cualquier otro lenguaje son extensiones propiedad de empresas de software o fabricantes de ordenadores (threads, ejecución remota, componentes, seguridad, acceso a bases de datos, etc.)” (Castañeda) . Es por este motivo que JAVA es un lenguaje muy utilizado para aprender programas ya que contiene extensiones de otros lenguajes, lo cual hace que sea de fácil comprensión para el desarrollador o persona interesada en aprender a programar.

Qué es JAVA.

“El lenguaje Java es un lenguaje esencial para desarrollar aplicaciones en cualquier sistema de computación relacionado con el internet, por lo que todos los planes de estudio de informática incluyendo uno o más cursos donde se usa el lenguaje Java” (Durán, Gutiérrez, & Pimentel, 2007). El lenguaje de programación Java es considerado como un lenguaje de alto nivel, ya que es de fácil entendimiento para el desarrollador, además se puede manejar en diferentes plataformas por lo cual permite desarrollar aplicaciones participativas las cuales se encuentran colocadas en la red por medio de la vía Web. Java convierte el código fuente en instrucciones las cuales son interpretadas por la máquina virtual Java, lo cual produce un mejor ambiente de trabajo, a diferencia de los lenguajes tradicionales que traducían el código fuente en un lenguaje que solo la maquina podía interpretar pero no el desarrollador. El lenguaje de programación Java está orientado a objetos por lo cual cada programa se crea a partir de clases, es por eso que a Java se puede considerar como una herramienta para dar solución a los diferentes problemas de programación.

Características de Java

En la actualidad Java ha adquirido enorme popularidad, esto se debe a su rápida difusión y utilización en el ámbito de la programación en Internet, esto ha sido posible por sus características. “SUN caracteriza a Java como un lenguaje sencillo, orientado a objetos, distribuido, interpretado, robusto, independiente de las arquitecturas, portable, eficaz, multihilo y dinámico” (Groussand, 2014)

- **Sencillo.**- “Los lenguajes de programación orientados a objetos no son sencillos ni fáciles de utilizar, pero Java es un poco más fácil que el popular C++2, lenguaje de desarrollo de software más popular hasta la implementación de Java” (Joyanes & Fernández, 2002). Java posee una estructura similar a las de C y C++, pero con la diferencia que posee una programación más simplificada; es decir posee una sintaxis que facilita el desarrollo de programas.
- **Orientado a objetos.**- “Java es un lenguaje diseñado partiendo de cero, como resultado de esto se realiza una aproximación limpia, útil y pragmática a los objetos. El modelo de objeto de Java es simple y fácil de ampliar” (Castañeda). El punto de creación de los programas en Java se centra en crear, manipular y construir objetos debido a la flexibilidad y reusabilidad que posee la programación orientada a objetos.
- **Distribuido.**- “Java fue diseñado con extensas capacidades de interconexión TCP/IP. De hecho permite a los programadores acceder a la información a través de la red con tanta facilidad como a los Archivos locales” (Castañeda). Java facilita el desarrollo de aplicaciones en red, ya que se puede acceder al programa mediante la utilización de una dirección la cual accede de forma fácil al servidor donde se encuentra alojado el programa.
- **Interpretado.**- “Los programas se compilán en una Máquina Virtual Java generándose un código intermedio denominado bytecode”

(Joyanes & Fernández, 2002). Al contar Java con una máquina que interpreta el lenguaje de alto nivel en un lenguaje de máquina, hace que sea más fácil la compilación y ejecución del programa,

- **Robusto.-** “Java es fuertemente tipado, por lo que permite comprobar el código en tiempo de compilación” (Castañeda). Es decir Java verifica el código fuente esto incluye lo que es sintaxis, tipos, librerías entre otros en el instante de la ejecución del programa lo cual permite disminuir los errores.
- **Seguro.-** Dentro de Java existe un encargado de gestionar el consumo de memoria de cada uno de los objetos que se encuentran dentro del programa, lo cual verifica que si hay o no modificación del código.
- **Independiente de las arquitecturas.-** “Una de las características más notables de Java es que es de arquitectura neutral, lo que también se define como independiente de la plataforma” (Joyanes & Fernández, 2002). Al programar en el lenguaje Java no importa la arquitectura en la cual se esté desarrollando la aplicación, ya que Java se adapta de manera correcta a la arquitectura.
- **Portable.-** Java puede ser ejecutada en cualquier plataforma ya que los datos son del mismo tamaño, porque las bibliotecas de Java facilitan la escritura del código en plataformas que no posean adaptación.
- **Eficaz.-** “Incluso si un programa Java es interpretado, lo que es más lento que un programa nativo, Java pone en marcha un proceso de optimización de la interpretación del código, llamado Jit (Just In Time) o Hot Spot” (Groussand, 2014). Los compiladores de Java permiten que los programas se ejecuten en plataformas independientes, lo cual se logra el mismo rendimiento que los compiladores de lenguajes tradicionales como C++.
- **Multihilo.-** “Java es uno de los primeros lenguajes que se han diseñado explícitamente para tener la posibilidad de múltiples hilos de ejecución; es decir, Java es multihilo (multithreading)” (Joyanes

& Fernández, 2002). Java permite efectuar varios procesos o tareas a la vez, con el fin de incrementar la velocidad de cada una de las aplicaciones, ya sea esto por medio del CPU o del procesador.

- **Dinámico.**- “En Java, como dijimos, el programador no tiene que editar los vínculos (obligatorio en C y C++). Por lo tanto es posible modificar una o varias clases sin tener que efectuar una actualización de estas modificaciones para el conjunto del programa” (Groussand, 2014). Dentro de Java se puede agregar nuevos métodos a una clase sin necesidad de afectar a las demás clases, este proceso se lo puede realizar en la compilación.

Ventajas de Java

Java permite crear programas en diferentes plataformas, esto se debe a que posee muchas funcionalidades y portabilidad la cual se acopla sin ningún inconveniente. Además, Java se encuentra totalmente gratuito lo cual hace más sencillo el aprendizaje de la programación en este lenguaje, ya que trabaja en red lo cual hace que sea totalmente accesible. Java al ser un lenguaje muy popular cuenta con información suficiente; es decir, si se desea incursionar en la programación Java se puede encontrar el apoyo necesario para conocer este lenguaje y ayuda personal de algunos usuarios activos en el tema.

Los compiladores de Java permiten ejecutar el mismo programa en cualquier plataforma, hay que rescatar que Java posee una máquina virtual, la cual facilita la interpretación del código. Java tiene dos herramientas muy conocidas en el ámbito de desarrollo como son Eclipse y NetBeans que son un claro ejemplo de lo que es el lenguaje Java, los cuales ofrecen un entorno amigable de desarrollo y una capacidad de depuración.

Historia del paradigma orientado a objetos

Tradicionalmente el estilo de desarrollo de aplicaciones se lo realizaba con la programación estructurada la cual consistía en dividir un problema en pequeños problemas y subdivisiones del problema general para poder llegar a acciones muy simples las cuales se convertían en un proceso fácil de entender y codificar, con el pasar del tiempo fueron apareciendo nuevas necesidades por satisfacer, pero este paradigma de programación no lograba cumplir con los objetivos. Por tal motivo la programación orientada a objetos nace con la finalidad de acabar con la complejidad que poseían los programas informáticos de esa época, la programación orientada a objetos es otra manera de dividir los problemas basándose en el mundo real.

“Los lenguajes de programación tradicionales no orientados a objetos, como C, Pascal, BASIC, o Modula-2, basan su funcionamiento en el concepto de procedimiento o función” (Cachero, Ponce de León, & Saquete, 2006). Por tal motivo en estos lenguajes la programación se basa en funciones las cuales contienen un conjunto de instrucciones que trabajan bajo argumentos los cuales posterior a proceso producen resultados, por lo que se dice que solamente es una secuencia de llamada a distintas funciones en el mismo programa.

Los lenguajes orientados a objetos se diferencian del lenguaje tradicional principalmente en poseer un objeto y no basarse en funciones, en este caso el objeto no es más que una representación dentro concreta de algo en un programa, el cual contiene toda la información requerida para realizar la abstracción, en donde se describen los atributos (características) y métodos (operaciones) que puede tener una clase. La programación orientada a objetos es sin duda una innovadora forma de solucionar los problemas que se dan en el mundo real, un claro ejemplo de programación orientada a objetos es el lenguaje Java.

El Paradigma Orientado a Objetos

“El Paradigma Orientado a Objetos (PaOO) es pues una filosofía de desarrollo y empaquetamiento de programas que permite crear unidades funcionales extensibles y genéricas, de forma que el usuario las pueda utilizar según sus necesidades y de acuerdo con las especificaciones del problema a desarrollar” (Amo, Martínez, & Segovia, 2005). Este paradigma en la actualidad es el más utilizado en todo el mundo del ámbito del desarrollo e ingeniería de software, la Programación Orientada a Objetos se encarga de aproximar lo más cercano posible el lenguaje al problema. La idea de este paradigma es desarrollar programas que contengan las características principales del problema o necesidad a solucionar.

En programación orientada a objetos se busca que el programa se controle a sí mismo para que de esta manera la mente el programador se libere y se ocupe en realizar aplicaciones de mayor complejidad (ver Figura 6.1), por lo que cada objeto es una entidad que se controla a sí misma. Esto es posible ya que los objetos son únicos con sus métodos y atributos propios lo cual no permite mezclar datos que no correspondan a los suyos.

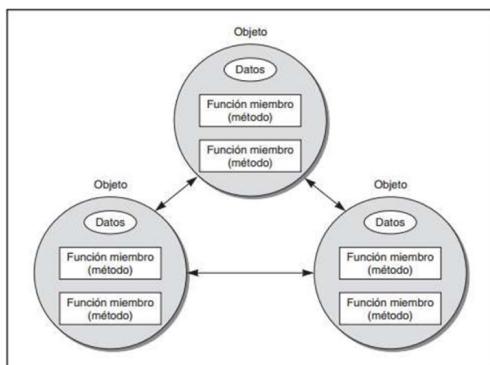


Figura 6.1. Organización de un programa orientado a objetos. Adaptado de (Edukativos, 2013)

Ventajas del Paradigma Orientado a Objetos

Sin duda alguna el paradigma orientado a objetos posee varias ventajas para las partes interesadas en este caso es el programador o desarrollador y el usuario, ya que ambos interactúan con el paradigma por medio de los lenguajes de programación. El paradigma busca un software de calidad que prometa mayor productividad por parte del programador y sobre todo que tenga un costo bajo del mantenimiento.

Una de las ventajas del paradigma orientado a objetos es la herencia, ya que nos evita la redundancia de código y a su vez permite extender el uso de varias clases ya existentes en el programa. Los módulos con los que el paradigma trabaja se comunican entre sí, lo cual ayuda a desarrollar el programa en un corto tiempo. Otra de las ventajas que ofrece este paradigma es la ocultación de datos, esto ayuda mucho al programador porque de esta manera se puede construir programas seguros, esta ventaja es una de las mejores ayudar para el programador debido a que al ocultar ciertos datos estamos evitando que personas ajenas accedan a la información.

Elementos de la P.O.O.

Según Castro, para la correcta definición de un programa basado en el paradigma de la programación orientada a objetos se debe conocer los elementos que lo conforman, siendo estos elementos: Clase, Objeto, Atributo, Método, Mensaje, Estado.

La clase

Según (Pavón, 2008) la clase representan un tipo particular de objetos como:

- Objetos con características y comportamiento similar.
- Categoría de objetos.

Cada clase tiene asociada un código que determina:

- Los atributos que tienen los objetos de la clase.
- Los métodos que pueden ejecutar los objetos de la clase y como lo hacen.



Figura 6.2. Ejemplos Clase.

Es la definición y declaración de datos y operaciones, la que representa cada uno de los objetos que comparten una estructura y comportamientos. Una clase es aquella en la que declaramos nuestro objeto (ver Figura 6.2).

El Objeto

Los objetos representan cosas, pueden ser simples o complejos o a su vez reales o imaginarios. (Pavón, 2008). Algunos ejemplos de objetos se muestran en la Figura 6.3



Figura 6.3. Ejemplos Objetos.

Es una entidad física o abstracta del cual podemos extraer varias características y funciones del mundo real de estos elementos que me ayudarán a definir con mayor claridad mi clase.

Atributo

Según (Pavón, 2008) son “Valores o características de los objetos, que permiten definir el estado del objeto u otras cualidades”. Algunos ejemplos de atributos se muestran en la Figura 6.4.



Figura 6.4. Atributos de los Objetos.

Son las características individuales que diversifican un objeto de otro y fijan su apariencia, estado u otras cualidades. Los atributos se recogen en variables declaradas y cada objeto particular puede tener valores distintos para estas variables.

El Método

La razón de establecer los atributos en vista privada, es poder controlar la modificación del estado del objeto y no permite que este evolucione hacia los estados incoherentes. Pero debemos establecer los mecanismos para poder modificar el objeto y son los métodos, el sistema adecuado. Podemos organizar los métodos en tres tipos, teniendo en cuenta aspectos sintéticos y semánticos. (Bernal, 2012)

Como ya hemos dicho anteriormente los métodos son las acciones que se pueden realizar con los objetos. También se podría definir un método como la implementación de un mensaje, al fin y al cabo, un mensaje es la llamada o invocación de un método de un objeto.

Existen dos métodos especiales que se denominan constructor y destructor.

El Mensaje

Según (Pavón, 2008), los objetos se comunican e interactúan entre sí por medio de mensajes. Si un objeto desea que otro objeto haga algo le envía un mensaje que puede tener información adicional en forma de parámetros. Cuando un objeto recibe un mensaje ejecutara un método u operación (ver Figura 6.5). Componentes de un mensaje:

- Objeto destinatario del mensaje (mi Coche).
- Método que se debe ejecutar como respuesta (cambiar marcha).
- Parámetros necesarios del método (segunda).

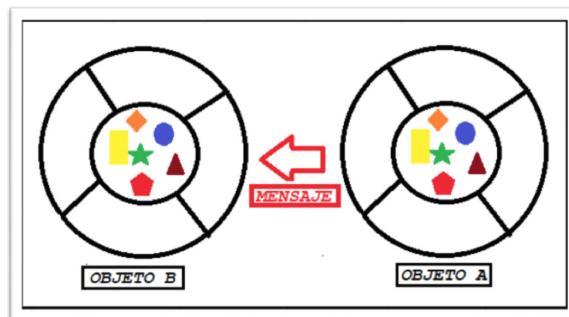


Figura 6.5.Mensaje en los Objetos

Los mensajes son la forma que tienen de comunicarse distintos objetos entre sí. Un objeto por sí sólo no es demasiado útil, sino que se suele utilizar dentro de una aplicación o programa que utiliza otros objetos.

El comportamiento de un objeto está reflejado en los mensajes a los que dicho objeto puede responder. Representan las acciones que un determinado objeto puede realizar.

El Estado

Según (Bernal, 2012), está compuesto de datos, será uno o varios atributos a los que se habrán asignado unos valores concretos (datos). El estado de un objeto es la apariencia que el objeto presenta al usuario, y depende del valor que tenga sus propiedades. Un cambio de estado se logra alterando al menos una de las propiedades del objeto.

El estado de un objeto es una de las posibles condiciones en que el objeto puede existir, normalmente cambia en el transcurso del tiempo. El estado de un objeto es implementado por un conjunto de propiedades (atributos), además de las conexiones que puede tener con otros objetos

Capítulo VII

Aplicando JAVA en el ambiente Play Framework

Luego de conocer los aspectos teóricos de algunas de las partes que conforman esta segunda capa de un sistema de información web, en este capítulo vamos a continuar con nuestro caso de estudio ejemplo. Esta vez nos focalizaremos en la parte técnica necesaria para configurar y desarrollar las funcionalidades de la Segunda Capa o Capa de Negocio de nuestra aplicación. Es aquí donde se diseña y desarrolla una arquitectura de software que soporte y procese cada uno de los requerimientos que tiene el usuario. En un contexto global podemos decir que por medio de esta capa es como el usuario procesa la información (Segunda Capa) que ha enviado a través de la interfaz (Primera Capa) para que finalmente sea guardada en un almacén de datos (Tercera Capa).

Para llevar a cabo en la segunda capa, esta arquitectura de software necesaria para el procesamiento de información, se requiere el uso de un lenguaje de programación de tercer o cuarto nivel, que en el mejor de los casos soporte las propiedades de la Programación Orientada a Objetos. La P.O.O. ofrece algunas propiedades al software tales como encapsulación (seguridad), reusabilidad, refactorización, escalabilidad y otras. Es por estas razones que trabajar en un ambiente de estructura MVC (Modelo Vista Controlador) es considerado de mayor complejidad, pero a la vez más seguro y escalable en el tiempo, es decir este tipo de sistemas pueden trascender en el tiempo acoplándose constantemente a las nuevas necesidades que tenga el negocio del cliente.

Entre lenguajes con licencia propietaria y libre que comúnmente son utilizados en la Segunda capa o Capa de Negocio tenemos a Java, PHP, C#, Ruby and Rails, Scala, Python y otros, la mayoría de los lenguajes de segunda capa trabajan con objetos y soportan las propiedades y fundamentos de la programación orientada a objetos. En la mayoría de los casos cada tecnología posee su propio Framework MVC, incluso existiendo tecnologías que tienen más de un Framework que los apoya, convirtiéndose este ámbito en una batalla de Frameworks, cada uno oferta mejores servicios para brindar mejor competitividad.

Como ya se ha mencionado para este caso de estudio se trabaja con Play Framework un entorno que apoya el desarrollo con los lenguajes de Java y Scala. Por esta razón para mostrar los aspectos técnicos de la Segunda Capa con este Framework, en el transcurso de este capítulo se mostrará el uso del lenguaje de tercera generación y orientado a objetos JAVA. Un lenguaje de programación de licencia libre y actualmente propiedad de uno de los gigantes del software, Oracle Corporation.

Para dar inicio al desarrollo de esta Segunda Capa, en primera instancia lo que ha sido importante determinar son las funcionalidades que tendrá el Sistema de Información Web SACC (Sistema de Administración de Carro de Compras). En el Capítulo 7 se habló de las necesidades del usuario para con el SACC, ahora que nos encontramos en la fase de implementación de la capa negocio es necesario plantear estas necesidades del usuario en forma de funcionalidades del sistema. Analizando dichas necesidades las funcionalidades que se han planteado son las siguientes:

- Cargar productos en un catálogo
- Creación de una nueva venta.
- Agregar nuevos ítems al carro de compras.
- Borrar ítems del carro de compras.
- Calcular el total de la compra.

Es importante mencionar que en este capítulo se diseñará y desarrollará la arquitectura de software necesaria para soportar la transaccionalidad de las funcionalidades planteadas. No se presentará aún, como almacenar en una base de datos los procesos de negocio que se hayan realizado, ni tampoco como presentar esta información procesada en la interfaz del usuario. Este capítulo es netamente transaccional y en lo que respecta al vínculo de esta capa con el almacenamiento y la interfaz serán temáticas de los capítulos venideros.

Ya focalizándonos en nuestro proyecto el desarrollo de esta segunda capa o capa de negocio se lo lleva a cabo mayoritariamente en la carpeta “*/app/models*” y parcialmente en la carpeta “*/app/controllers*”. Para alcanzar la meta planteada y desarrollar las funcionalidades definidas, se ha planificado las siguientes etapas de trabajo, el detalle de cada una de ellas se lo irá presentando en el transcurso del capítulo:

- Diseño UML de la arquitectura del Software.
- Exportación de la arquitectura a clases Java.
- Desarrollo de la estructura de las clases.
- Desarrollo de las funcionalidades

Diseño UML de la arquitectura del Software

Planteadas las funcionalidades, el primer paso es realizar un modelo de clases con la ayuda de una herramienta UML (Unified Modeling Language), puede usar el modelador que más le agrade. Para la didáctica de este ejemplo se usará la herramienta AmaterasUML, que es un plugin para Eclipse que permite manejar la herramienta de modelado dentro de la interfaz de Eclipse. AmaterasUML es una aplicación con licencia libre y se la puede descargar de su página oficial (es.osdn.net/projects/amateras/releases/), al descargar la aplicación, también dispondrá de archivos que le indicarán los pasos a seguir para su instalación y configuración.

Una vez que hemos instalado la herramienta AmaterasUML en nuestro Eclipse, tendremos la disponibilidad de trabajar con diferentes diagramas UML, tales como: actividades, clases, secuencia o casos de uso. Para nuestro ejemplo necesitamos crear un modelo de clases, pues es el modelo que nos ayuda a diseñar una arquitectura de software que soporte nuestra aplicación. Como muestra la Figura 7.1 en el explorador de proyectos hemos creado una carpeta “*/uml*” para guardar los archivos

de nuestro diseño, en esta carpeta se ha creado un nuevo diagrama de clases de la herramienta AmaterasUML.

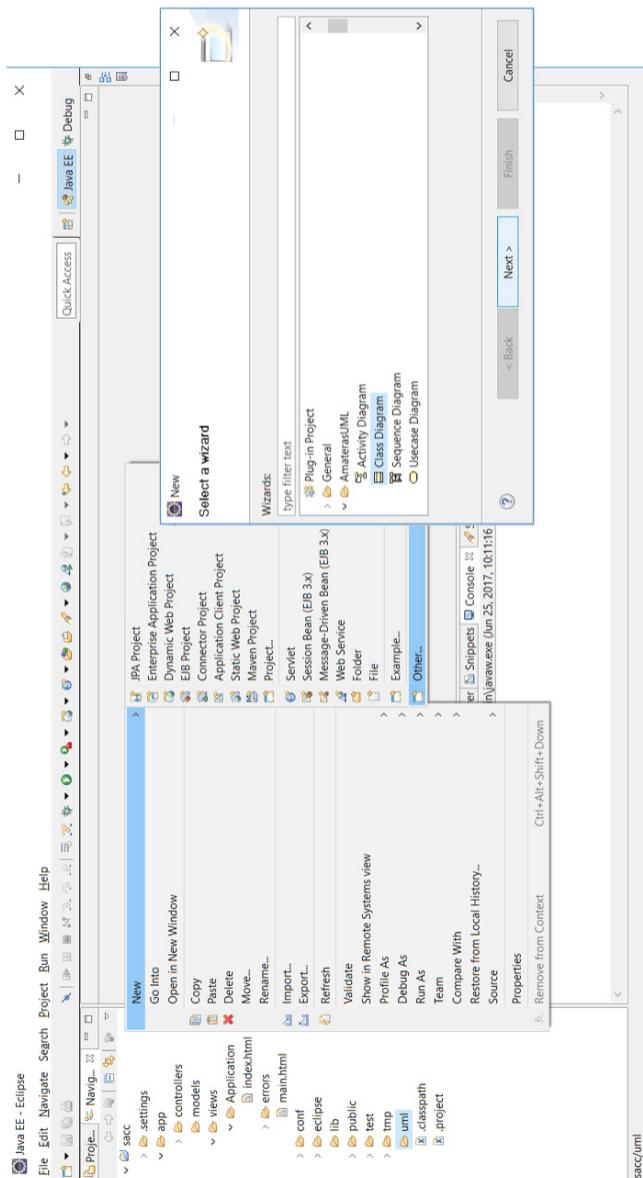


Figura 7.1.Creando diagramas de clases en Eclipse

Creado nuestro diagrama de clases, procedemos a agregar las clases necesarias para satisfacer las funcionalidades del sistema de información. En la Figura 7.2, se puede apreciar que en la parte izquierda de la sección central se posee una barra para agregar las clases y relaciones que se deseé para el diseño. Después de analizar las funcionalidades planteadas a inicio de este capítulo se ha decidido diseñar el siguiente modelo de clases, con sus respectivos atributos y relaciones de composición y asociación que se muestran entre ellas.

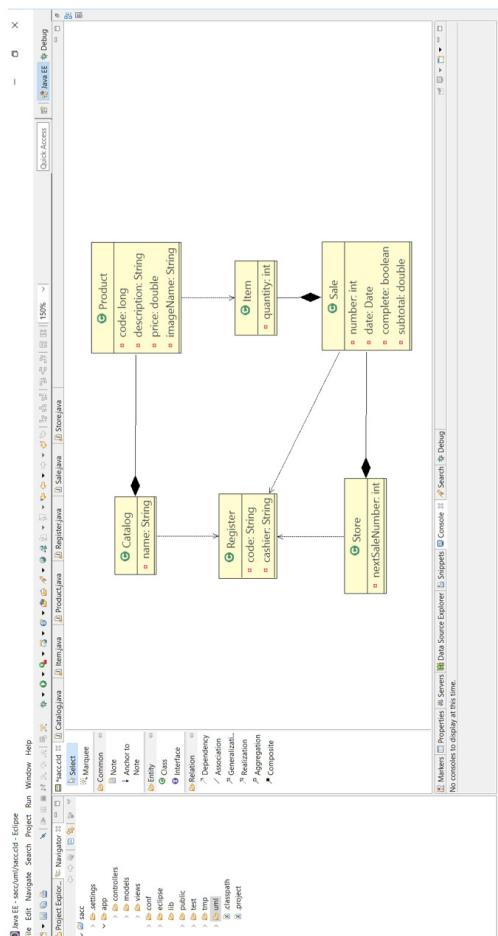


Figura 7.2. Diseñando un diagrama de clases con Amateras UML

Exportación de la arquitectura a clases Java

La gran ventaja de tener una herramienta UML dentro del IDE Eclipse es que en la mayoría de los casos los diagramas de clases pueden ser exportados a clases Java. AmaterasUML es el caso de la aplicación que sí permite exportar el diseño realizado a clases, para esto tenemos que dar click derecho sobre el diagrama y como muestra la Figura 7.3, exportar todas las clases que se han modelado a la carpeta “/app/models” que es el lugar donde almacenaremos toda la arquitectura de nuestro software.

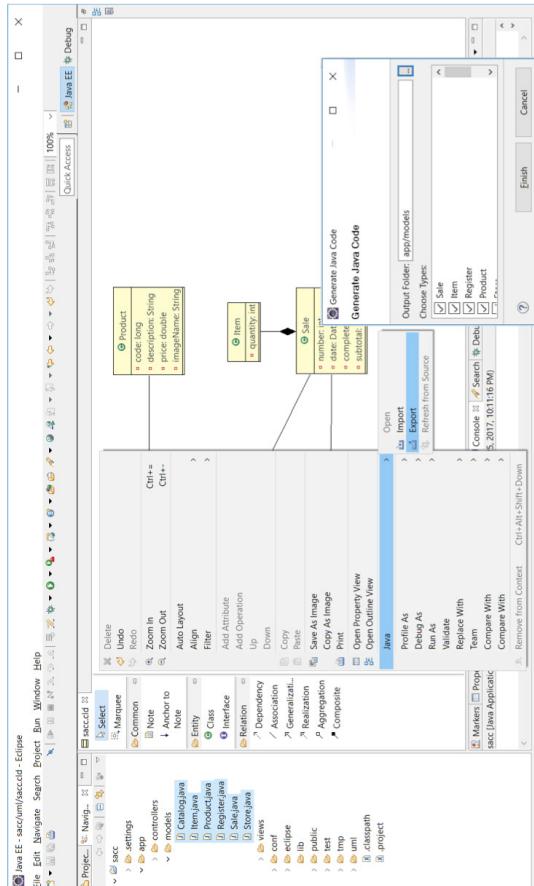


Figura 7.3. Exportación de diagrama de clases a código JAVA

Es importante mencionar que las clases de extensión java exportadas no son perfectas, de hecho, ningún modelador UML las exportará de forma impecable y completa. Por lo tanto, el siguiente paso será corregir las clases, revisar cada una de las partes inconsistentes o faltantes de su estructura, así como también plasmar en código las relaciones que se han diseñado en el modelo. Sin embargo, hay que valorar las líneas de código y tiempo ahorrado con la exportación del formato de las clases desde AmaterasUML.

Desarrollo de la estructura de las clases

Una vez que disponemos de las clases java es necesario revisar su estructura y completar sus partes en caso de ser necesario. Los detalles iniciales que se validarán en cada una de las clases exportadas serán: la correcta declaración del paquete al que pertenecen (models), las librerías necesarias, sus atributos, constructores, métodos de lectura y escritura (getters and setters) y finalmente las relaciones existentes entre las clases. Todas las validaciones mencionadas exceptuando la relación entre clases se las puede corregir o crear automáticamente, para luego adecuarlas a nuestra necesidad. En el caso de las librerías basta por pasar el cursor por la palabra clave marcada en rojo para que liste en pantalla las opciones de librería que pueden ser importadas o también, puede hacer click en la barra de la izquierda dónde se activa un indicador cada vez que tenemos un problema en determinada línea de código. Esta acción no solo la podemos hacer uso en el caso de librerías faltantes sino también cada vez que tengamos alguna inconsistencia en nuestro código, si tenemos palabras clave marcadas en rojo, Eclipse nos propone alternativas de solución cada vez que tenemos errores en el programa fuente.

Para el caso de los constructores y funciones de lectura y escritura (getters and setters), se los puede crear automáticamente haciendo click derecho sobre el nombre de la clase, con esta acción se presentará las ventanas

presentadas en la Figura 7.4, eligiendo source tendremos opción para generar automáticamente constructores, funciones de lectura-escritura y otros. Con respecto a las funciones de lectura y escritura recordar que su número de funciones son dos por cada atributo existente.

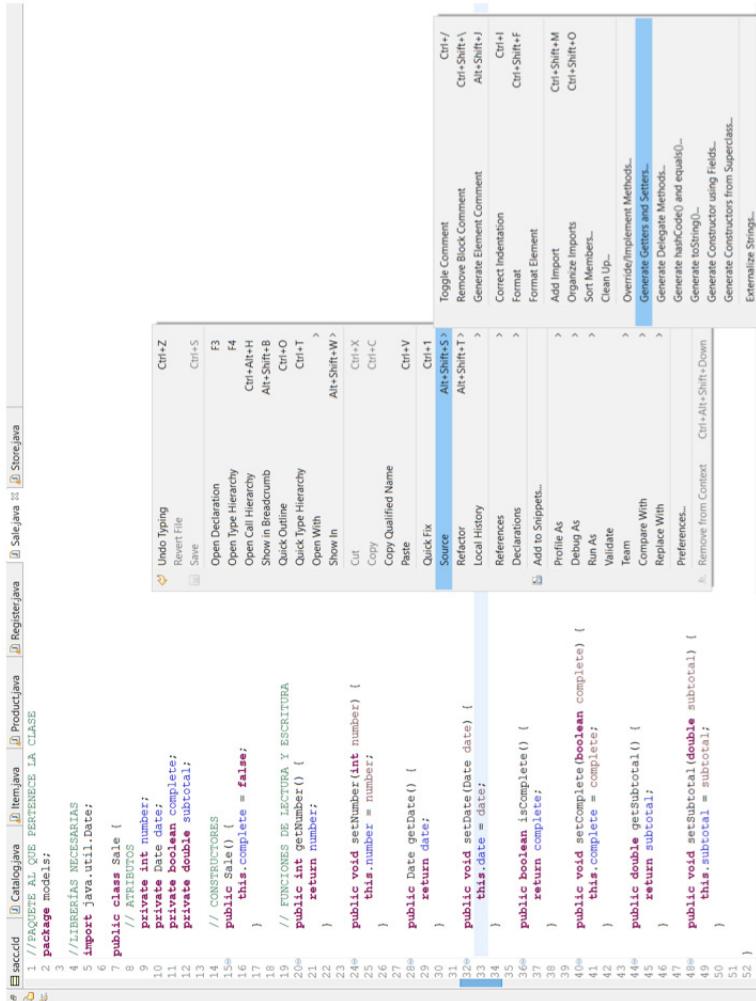


Figura 7.4. Generación de las funciones Getters and Setters en clases Java

La validación de las clases está centrada en cada una de sus partes principales: atributos, constructores y funciones de lectura y escritura. En la Figura 7.4 también puede apreciar cada una de las partes esenciales por la cual está estructurada una clase, estas partes son las que también crearemos en el resto de clases.

Una vez que se ha validado la estructura de todas las clases, el siguiente paso es, de acuerdo al modelo que se ha definido, plasmar en código fuente las relaciones existentes entre las clases. Para esto es importante analizar, el tipo de relación que se ha definido, AmaterasUML posee las relaciones de Programación Orientada a Objetos: dependencia, asociación, generalización, realización, agregación y composición. Pero excluyendo los tipos de relación que corresponden a una Herencia o Interfaz (generalización/realización), de las demás es importante hacer una analogía con los modelos entidad relación e identificar cuáles de ellas corresponden a una relación uno a uno o uno a varios.

En nuestro modelo de clases del sistema de información SACC se ha hecho uso de los tipos de relación: dependencia y composición. Haciendo analogía con el modelo entidad relación se discierne que la Dependencia representa una relación uno a uno y la Composición representa la relación de uno a varios. Es importante esta analogía puesto que al momento de plasmar en código fuente estas relaciones, la conceptualización de “uno a uno” o “uno a varios” nos ayuda a decidir como plantear la relación y en que clase.

Para el caso de una relación uno a uno escribimos en la clase que corresponda, la declaración como atributo simple de la otra clase con la que tiene relación. En la Tabla 7.1 tenemos una comparación de las relaciones diseñadas en UML y el código fuente Java escrito para representar estas relaciones.

Dependencia (uno a uno)	
Relación UML	Relación en Java
<pre> classDiagram class Catalog { name: String } class Register { code: String cashier: String } class Store { nextSaleNumber: int } class Sale { number: int date: Date complete: boolean subtotal: double } Catalog "1" --> "1" Register Register "*" --> "1" Store Register "*" --> "1" Sale </pre>	<pre> 1 //PAQUETE AL QUE PERTENECE LA CLASE 2 package models; 3 4 //DECLARACIÓN DE LA CLASE 5 public class Register { 6 // ATRIBUTOS 7 private String code; 8 private String cashier; 9 private Store store; 10 private Catalog catalog; 11 private Sale currentSale; 12 13 // CONSTRUCTORES 14 public Register(String code, String 15 cashier) { 16 this.code = code; 17 this.cashier = cashier; 18 this.store = store; 19 this.catalog = catalog; 20 this.currentSale = null; 21 } </pre>
<pre> classDiagram class Product { code: long description: String price: double imageName: String } class Item { quantity: int } Product "1" --> "1" Item </pre>	<pre> 1 //PAQUETE AL QUE PERTENECE LA CLASE 2 package models; 3 4 //DECLARACIÓN DE LA CLASE 5 public class Item { 6 // ATRIBUTOS 7 private int quantity; 8 private Product product; 9 10 // CONSTRUCTORES 11 public Item(int quantity, Product p) { 12 this.quantity = quantity; 13 this.product = p; 14 } 15 </pre>

Tabla 7.1. Relaciones de dependencias en la aplicación de caso de estudio

Y en el caso de la relación uno a varios en la clase que corresponda declararemos como atributo un vector (Set, List, Map) que soporte objetos de la otra clase con la que tiene relación. En la Tabla 7.2 tenemos una comparación de las relaciones diseñadas en UML y el código fuente Java escrito para representar estas relaciones. Es importante considerar que

una vez que definimos nuestro vector en cada clase que corresponda, es necesario en su constructor instanciar los espacios de memoria que se requieren para dicho vector.

Composición (uno a varios)	
Relación UML	Relación en Java
<pre> classDiagram class Catalog { <<Catalog>> #name : String } class Product { <<Product>> #code : long #description : String #price : double #imageName : String } Catalog "1" *-- "*" Product </pre>	<pre> 1 //PAQUETE AL QUE PERTENECE LA CLASE 2 package models; 3 4 //LIBRERIAS NECESARIAS 5 import java.util.HashMap; 6 import java.util.Map; 7 8 //DECLARACIÓN DE LA CLASE 9 public class Catalog { 10 // ATRIBUTOS 11 private String name; 12 private Map<Long, Product> products; 13 14 // CONSTRUCTORES 15 public Catalog(String name) { 16 this.name = name; 17 this.products = new HashMap<Long, Product>(); 18 } 19 </pre>
<pre> classDiagram class Sale { <<Sale>> #number : int #date : Date #complete : boolean #subtotal : double } class Store { <<Store>> #nextSaleNumber : int } Sale "1" *-- "*" Store </pre>	<pre> 1 //PAQUETE AL QUE PERTENECE LA CLASE 2 package models; 3 4 //LIBRERIAS NECESARIAS 5 import java.util.HashSet; 6 import java.util.Set; 7 8 //DECLARACIÓN DE LA CLASE 9 public class Store { 10 // ATRIBUTOS 11 private int nextSaleNumber; 12 private Set<Sale> completedSales; 13 14 // CONSTRUCTORES 15 public Store() { 16 this.nextSaleNumber = 1; 17 this.completedSales = new HashSet<Sale>(); 18 } 19 </pre>

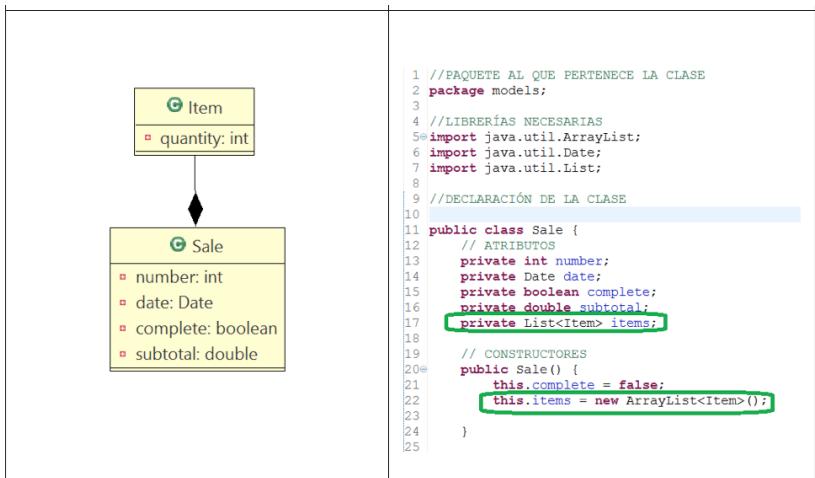


Tabla 7.2. Relaciones de composición en la aplicación de caso de estudio

Haciendo un resumen de las modificaciones realizadas a la estructura de cada una de las clases de nuestro modelo, se ha validado los siguientes aspectos principales de una clase: correcto paquete de ubicación, llamado a las librerías necesarias, creación de atributos de relación (dependencia o composición), generación y adecuación de constructores, generación de funciones de escritura y escritura.

Desarrollo de las funcionalidades

Una vez que hemos logrado consolidar la estructura de las clases java de nuestro modelo, el siguiente paso es desarrollar las funcionalidades que se han planificado al inicio de este capítulo para nuestro sistema. Por cada una de las funcionalidades se irá detallando el código fuente necesario para conseguir el comportamiento deseado para nuestro sistema de información. Este desarrollo únicamente se lo realizará en el contexto de la segunda capa o capa de negocio, las funcionalidades quedarán listas para que, en los capítulos venideros, esta transaccionalidad pueda ser

almacenada en una base de datos y presentada en nuestra interfaz que fue construida en el Capítulo 7.

Centrándonos en nuestro proyecto MVC (Modelo Vista Controlador) play framework creado, al hablar de capa de negocio, hablamos de que el área de trabajo para su desarrollo se lo realiza en las carpetas “*models*” o modelo y “*controllers*” o controlador. Pero es importante mencionar que ambas poseen la frontera que las relaciona tanto con la primera capa (presentación) y la tercera capa (datos), en el caso del modelo posee el vínculo con la base de datos y el controlador posee el vínculo con la interfaz. La forma de interrelación entre capas se abordará posteriormente, por esta razón en este capítulo se ha buscado escribir código que únicamente se desempeñe en la segunda capa, sin embargo, es importante receptar la estructura del contexto de nuestro ambiente de trabajo.

Sobre los roles de trabajo de cada uno: en el modelo se desarrolla todas las funciones o métodos necesarios para cumplir con las funcionalidades planteadas, analizando en primera instancia a que clase le corresponde asignarle cada responsabilidad; en el controlador se desarrolla la orquestación de toda la funcionalidad, haciendo llamados a todas las clases y métodos implicados para solventar el desempeño de la funcionalidad requerida.

Con esta perspectiva, para cada una de las siguientes funcionalidades, en la clase que corresponda escribiremos al final de ellas, los métodos necesarios para que la funcionalidad pueda ser ejecutada, esto en el modelo. Al final realizaremos una orquestación de una funcionalidad para verificar que nuestro esquema funciona correctamente y está trabajando sin errores, esto en el controlador.

- Cargar productos en un catálogo:

Para cubrir con esta funcionalidad en el modelo se planteó la relación de composición entre las clases “*Catalog*” y “*Product*” es decir que un catálogo puede contener varios productos. Esta relación se la establece en *Catalogo* a través de la declaración como atributo de un vector de productos. Por esta razón el método que tenga la acción de agregar productos se la debe escribir en la clase *Catalog*, pues en esta clase se encuentra el vector de productos que será el almacén de todos los objetos de tipo *Product* que se desee crear. La acción que se tiene que realizar es instanciar un nuevo producto de acuerdo a los parámetros que decida el usuario, para que posteriormente éste sea agregado al vector de productos, el método que escribiremos es el siguiente:

```
// FUNCIONES VARIAS
public void addProduct(long code, String description, double price,
    String imageName) {
    Product p = new Product(code, description, price, imageName);
    this.products.put(code, p);
}
```

- Creación de una nueva venta:

En nuestro modelo de clases se definió que una venta (Sale) tendrá relación con la central de registro (Register) y con la tienda (Store). En el caso de la central de registro almacena la venta que se encuentra vigente o abierta y para el caso de la tienda en un vector de ventas irá guardando todas aquellas ventas que fueron finalizadas. Con esta aclaración la creación de la venta es una funcionalidad que le corresponde efectuar a la central registradora pues es ella la que va dando apertura a nuevas ventas. Es así, que en la clase “*Register*” se ha desarrollado el siguiente método:

```
// FUNCIONES VARIAS
public void makeNewSale() {
    this.currentSale = new Sale();
}
```

- Agregar nuevos ítems al carro de compras:

En el caso de la selección de nuevos artículos de compra o ítems al carro de compras (Sale), la petición ingresa por la central registradora, ésta debe registrar el nuevo ítem a agregar. La acción de este método será buscar en el catálogo de productos el nuevo ítem que el usuario desea agregar al carro de compras, si éste existe se procede a crear en la venta el nuevo ítem. Este método se lo escribe en la clase “*Register*”, de la siguiente manera:

```
public void enterItem(long code, int qty) {
    Product p = this.catalog.findProduct(code);
    this.currentSale.makeItem(p, qty);
}
```

Como el método de buscar producto no lo tenemos declarado en el catálogo, es necesario escribir el código de este método en la clase “*Catalog*”, de la siguiente manera:

```
public Product findProduct(long code) {
    return this.products.get(code);
}
```

Tampoco tenemos declarado el método que corresponde a la creación de un nuevo ítem, éste debe permitir la instanciación de un nuevo objeto ítem y su agregación al vector de ítems de la venta. Para esto es necesario en la clase “*Sale*” escribir el siguiente método:

```
// FUNCIONES VARIAS
public void makeItem(Product p, int qty) {
    Item i = new Item(qty, p);
    this.items.add(i);
}
```

- Borrar ítems del carro de compras:

Todos los ítems que el usuario va agregando a su carro de compras se van almacenado en la clase venta, por lo tanto, en el caso de que el usuario desee retirar algún producto que no deseé es en la venta donde debe eliminar dicho ítem. Para realizar esta acción, en el vector de ítems de la venta se busca el ítem que se desea retirar de carro de compras y se lo elimina de la lista. Para esto es necesario en la clase “Sale” escribir el siguiente método:

```
public void deleteItem(long deleteId) {
    this.getItems().remove(Item.findById(deleteId));
}
```

- Calcular el total de la compra

El lugar dónde vamos almacenando la cantidad y el producto que el usuario está recopilando al carro de compras, es en los atributos de cada ítem. Por esta razón es en la clase ítem donde podría calcular un subtotal del costo que representa la cantidad que el usuario lleva de ese ítem, posteriormente se podría sumar todos los subtotales y calcular el total final de la compra. Para conseguir este subtotal en ítem debemos multiplicar la cantidad por el precio del producto, para esto es necesario en la Clase “Item” escribir el siguiente método:

```
// FUNCIONES VARIAS
public double calculateSubTotal() {
    return this.product.getPrice() * this.quantity;
}
```

Finalmente, después de desarrollar en las clases correspondientes los métodos necesarios para cumplir con las funcionalidades de nuestro sistema de información SACC, procedemos a realizar una prueba de uso en nuestro controlador (controllers). Se había mencionado que el

controlador también es parte de la segunda capa o capa de negocio y es el componente donde se puede realizar una orquestación con la estructura de clases, atributos, constructores y métodos que se han desarrollado en el modelo (models). En la Figura 7.5 se aprecia que en el controlador “Application.java” se ha implementado una función de prueba denominada “*testingBusinessTier2*” que instancia un catálogo para agregar en este objeto cuatro productos (bebidas), finalmente se recorres el vector de productos y se imprime en consola. Siendo ésta una prueba de uso de la arquitectura de nuestro sistema de información.

Project Explorer

```

10 public class Application extends Controller {
11     public static void index() {
12         testingBusinessTier2();
13     }
14 }
15
16 render();
17 }
18
19 private static void testingBusinessTier2() {
20     Catalog catalog = new Catalog("Bebidas");
21     catalog.addProduct(1, "Coca-Cola 1.5 lt.", 0.75, "cocaCola.png");
22     catalog.addProduct(2, "Fanta 1.5 lt.", 0.50, "fanta.jpg");
23     catalog.addProduct(3, "Sprite 1.5 lt.", 0.60, "sprite.png");
24     catalog.addProduct(4, "Fioravanti 1.5 lt.", 0.50, "fioravanti.jpg");
25
26     for (Product p : catalog.getProducts().values()) {
27         System.out.println(String.format("%1$14d | %2$-20s | %3$8.2f |",
28             p.getCode(), p.getDescription(), p.getPrice()));
29     }
30 }

```

Navigator

sacc

settings

app

controllers

Application.java

models

Catalog.java

Item.java

Product.java

Register.java

Sale.java

Store.java

views

conf

eclipse

settings

classes

.classpath

.project

Connect JPDAs to sacc.launch

sacc.launch

Test sacc.launch

lib

public

test

tmp

uml

Catalog.java

Item.java

Register.java

Sale.java

Product.java

Sale.java

Store.java

Application.java

Properties

Servers

Data Source

Explorers

Snippets

Console

Search

Debug

1	Coca-Cola 1.5 lt.	0.75	
2	Fanta 1.5 lt.	0.50	
3	Sprite 1.5 lt.	0.60	
4	Fioravanti 1.5 lt.	0.50	

Figura 7.5. Ejecución de la lógica de negocio de la aplicación caso de estudio

Para finalizar este capítulo presentamos el código fuente desarrollado en cada una de las clases de nuestro modelo: Catalog, Product, Register, Store, Sale e Item. El conjunto de estas clases y sus relaciones representa la arquitectura de la lógica de negocio definida para nuestro software.

Catalog: Brinda una categoría de agrupación para los productos existentes en la tienda, pudiendo ser estos productos de bebida, alimentos, electrodomésticos u otros.

```
1 //PAQUETE AL QUE PERTENECE LA CLASE
2 package models;
3
4 //LIBRERÍAS NECESARIAS
5 import java.util.HashMap;
6 import java.util.Map;
7
8 //DECLARACIÓN DE LA CLASE
9 public class Catalog {
10     // ATRIBUTOS
11     private String name;
12     private Map<Long, Product> products;
13
14     // CONSTRUCTORES
15     public Catalog(String name) {
16         this.name = name;
17         this.products = new HashMap<Long, Product>();
18     }
19
20     // FUNCIONES DE LECTURA Y ESCRITURA
21     public String getName() {
22         return name;
23     }
24
25     public void setName(String name) {
26         this.name = name;
27     }
28
29     public Map<Long, Product> getProducts() {
30         return products;
31     }
32
33     public void setProducts(Map<Long, Product> products) {
34         this.products = products;
35     }
36
37     // FUNCIONES VARIAS
38     public void addProduct(long code, String description, double price,
39                           String imageName) {
40         Product p = new Product(code, description, price, imageName);
41         this.products.put(code, p);
42     }
43
44     public Product findProduct(long code) {
45         return this.products.get(code);
46     }
47 }
```

Product: Será un almacén de las características y precios de todos los productos existentes.

```
1 //PAQUETE AL QUE PERTENECE LA CLASE
2 package models;
3
4 //DECLARACIÓN DE LA CLASE
5 public class Product {
6     // ATRIBUTOS
7     private long code;
8     private String description;
9     private double price;
10    private String imageName;
11
12    // CONSTRUCTORES
13    public Product(long code, String description, double price, String imageName) {
14        this.code = code;
15        this.description = description;
16        this.price = price;
17        this.imageName = imageName;
18    }
19
20    // FUNCIONES DE LECTURA Y ESCRITURA
21    public long getCode() {
22        return code;
23    }
24
25    public void setCode(long code) {
26        this.code = code;
27    }
28
29    public String getDescription() {
30        return description;
31    }
32
33    public void setDescription(String description) {
34        this.description = description;
35    }
36
37    public double getPrice() {
38        return price;
39    }
40
41    public void setPrice(double price) {
42        this.price = price;
43    }
44
45    public String getImageName() {
46        return imageName;
47    }
48
49    public void setImageName(String imageName) {
50        this.imageName = imageName;
51    }
52    // FUNCIONES VARIAS
53 }
```

Register: Es la central de registro, una clase que ayuda a la interrelación con las demás clases del modelo, esta interrelación fortalece el proceso de venta que se ha planteado

```

1 //PAQUETE AL QUE PERTENECE LA CLASE
2 package models;
3
4 //DECLARACION DE LA CLASE
5 public class Register {
6     // ATRIBUTOS
7     private String code;
8     private String cashier;
9     private Store store;
10    private Catalog catalog;
11    private Sale currentSale;
12
13    // CONSTRUCTORES
14    public Register(String code, String cashier, Store store, Catalog catalog) {
15        this.code = code;
16        this.cashier = cashier;
17        this.store = store;
18        this.catalog = catalog;
19        this.currentSale = null;
20    }
21
22    // FUNCIONES DE LECTURA Y ESCRITURA
23    public String getCode() {
24        return code;
25    }
26
27    public void setCode(String code) {
28        this.code = code;
29    }
30
31    public String getCashier() {
32        return cashier;
33    }
34
35    public void setCashier(String cashier) {
36        this.cashier = cashier;
37    }
38
39    public Store getStore() {
40        return store;
41    }
42
43    public void setStore(Store store) {
44        this.store = store;
45    }
46
47    public Catalog getCatalog() {
48        return catalog;
49    }
50
51    public void setCatalog(Catalog catalog) {
52        this.catalog = catalog;
53    }
54
55    public Sale getCurrentSale() {
56        return currentSale;
57    }
58
59    public void setCurrentSale(Sale currentSale) {
60        this.currentSale = currentSale;
61    }
62
63    // FUNCIONES VARIAS
64    public void makeNewSale() {
65        this.currentSale = new Sale();
66    }
67
68    public void enterItem(long code, int qty) {
69        Product p = this.catalog.findProduct(code);
70        this.currentSale.makeItem(p, qty);
71    }
72
73 }

```

Store: Corresponde al registro de la tienda o sucursales de la misma, que es la propietaria de los catálogos y productos existentes en ella.

```
1 //PAQUETE AL QUE PERTENECE LA CLASE
2 package models;
3
4 //LIBRERÍAS NECESARIAS
5 import java.util.HashSet;
6 import java.util.Set;
7
8 //DECLARACIÓN DE LA CLASE
9 public class Store {
10     // ATRIBUTOS
11     private int nextSaleNumber;
12     private Set<Sale> completedSales;
13
14     // CONSTRUCTORES
15     public Store() {
16         this.nextSaleNumber = 1;
17         this.completedSales = new HashSet<Sale>();
18     }
19
20     // FUNCIONES DE LECTURA Y ESCRITURA
21     public int getNextSaleNumber() {
22         return nextSaleNumber;
23     }
24
25     public void setNextSaleNumber(int nextSaleNumber) {
26         this.nextSaleNumber = nextSaleNumber;
27     }
28
29     public Set<Sale> getCompletedSales() {
30         return completedSales;
31     }
32
33     public void setCompletedSales(Set<Sale> completedSales) {
34         this.completedSales = completedSales;
35     }
36
37     // FUNCIONES VARIAS
38
39 }
```

Sale: Es la venta que el usuario da apertura para anexar o retirar los productos que desea comprar.

```

1 //PAQUETE AL QUE PERTENECE LA CLASE
2 package models;
3
4 //LIBRERÍAS NECESARIAS
5 import java.util.ArrayList;
6 import java.util.Date;
7 import java.util.List;
8
9 //DECLARACIÓN DE LA CLASE
10 public class Sale {
11     // ATRIBUTOS
12     private int number;
13     private Date date;
14     private boolean complete;
15     private double subtotal;
16     private List<Item> items;
17
18     // CONSTRUCTORES
19     public Sale() {
20         this.complete = false;
21         this.items = new ArrayList<Item>();
22     }
23
24
25     // FUNCIONES DE LECTURA Y ESCRITURA
26     public int getNumber() {
27         return number;
28     }
29
30     public void setNumber(int number) {
31         this.number = number;
32     }
33
34     public Date getDate() {
35         return date;
36     }
37
38     public void setDate(Date date) {
39         this.date = date;
40     }
41
42     public boolean isComplete() {
43         return complete;
44     }
45
46     public void setComplete(boolean complete) {
47         this.complete = complete;
48     }
49
50     public double getSubtotal() {
51         return subtotal;
52     }
53
54     public void setSubtotal(double subtotal) {
55         this.subtotal = subtotal;
56     }
57
58     public List<Item> getItems() {
59         return items;
60     }
61
62     public void setItems(List<Item> items) {
63         this.items = items;
64     }
65
66     // FUNCIONES VARIAS
67     public void makeItem(Product p, int qty) {
68         Item i = new Item(qty, p);
69         this.items.add(i);
70     }
71
72     public void deleteItem(long deleteId) {
73         this.getItems().remove(Item.findById(deleteId));
74     }
75
76 }

```

Item: Posee el producto y la cantidad que el usuario consumidor desea comprar de ese producto. Varios de estos ítems pueden ser anexados o retirados de una venta.

```
1 //PAQUETE AL QUE PERTENECE LA CLASE
2 package models;
3
4 //DECLARACIÓN DE LA CLASE
5 public class Item {
6     // ATRIBUTOS
7     private int quantity;
8     private Product product;
9
10    // CONSTRUCTORES
11    public Item(int quantity, Product p) {
12        this.quantity = quantity;
13        this.product = p;
14    }
15
16    // FUNCIONES DE LECTURA Y ESCRITURA
17    public int getQuantity() {
18        return quantity;
19    }
20
21    public void setQuantity(int quantity) {
22        this.quantity = quantity;
23    }
24
25    public Product getProduct() {
26        return product;
27    }
28
29    public void setProduct(Product product) {
30        this.product = product;
31    }
32
33    public static Object findById(long deleteId) {
34        // TODO Auto-generated method stub
35        return null;
36    }
37
38    // FUNCIONES VARIAS
39    public double calculateSubTotal() {
40        return this.product.getPrice() * this.quantity;
41    }
42
43 }
```

Sección 4: Tercera Capa de Datos

En esta sección se abordará el detalle de los aspectos teóricos y funcionales de la tercera capa de una estructura MVC, generalmente a este segmento del sistema de información también se lo denomina capa de datos o capa de persistencia. Su característica e importancia principal es ser un canal de comunicación para el almacenamiento de la información que ha emitido la capa de negocio como resultado de su procesamiento. Esta capa brinda la posibilidad de almacenar esta información en objetos que son instanciados de acuerdo al modelo diseñado. Así también, de acuerdo a la configuración realizada en esta capa se puede conseguir que estos objetos persistan su información en una base de datos de cualquier índole, por ejemplo: PostgreSQL, MySQL, SQLServer y otras.

Es importante la correcta configuración de los aspectos funcionales de esta capa, pues de ésta depende la eficiencia y seguridad que se ofrezca a los datos recopilados en el sistema de información web. En el caso de grandes volúmenes de información una correcta configuración de esta capa puede permitir una mejor eficiencia al momento de gestionar la información. En lo que respecta a la seguridad, el hecho de persistir todos los objetos con información recopilados en la segunda capa, da lugar a que exista una convertibilidad de la información desde un modelo de clases y objetos hacia un modelo entidad relación con tablas y registros. Esta convertibilidad es un nivel de seguridad adicional en este ambiente de trabajo web, para un intruso sería una nueva barrera que tendría que superar para acceder a la información.

Es importante también para esta capa un correcto diseño del modelo de clases que soporta el sistema de información web, ya que sobre este modelo se realiza la configuración del ORM (Object Relation Mapping) que es la técnica con la cual se efectúa la convertibilidad de un modelo de clases a un modelo entidad relación. Existe una variedad de ORMs

en el mercado muchos de ellos comerciales y otros no comerciales, cada uno con una cualidad que los caracteriza, pero todos con la finalidad de transformar los objetos de una clase en registros de una tabla.

Expuesto el preámbulo funcional de la capa de datos, en esta sección también se describe los aspectos conceptuales que permiten el almacenamiento de la información procesada en una base de datos. También se describe las funcionalidades técnicas de un ORM y su relación con las bases de datos. En la etapa final de la sección sobre la base del caso de estudio práctico desarrollado hasta el capítulo 10, se da continuidad al desarrollo del sistema de información web planteado, en este caso se presenta los pasos a seguir para la creación de una base de datos en PostgreSQL y la configuración del ORM de Play Framework denominado JPA (Java Persistence API).

Capítulo VIII

**Los ORMs (Object
Relation Mapping) y las
Bases de Datos**

La tecnología en la actualidad ha ido sobresaliendo de una excelente manera y es por ello que brinda grandes alternativas a los desarrolladores de software, entre ellos se encuentran los ORM (mapeo objeto-relacional) desarrollados en plataforma Open Source, logrando ser una alternativa de trabajo eficiente y sin ataduras comerciales. Por otra parte, las Bases de Datos relacionales también son un complemento que ayuda a los desarrolladores de software a simplificar su trabajo y desarrollar aplicativos de calidad.

Es una técnica de programación que es conocido popularmente como mapeo de objetos relacional o también como Object-Relation Mappin (ORM) y que consiste en vincular los objetos utilizados en nuestro código con una base de datos relacional (Alvial, Saavedra, & Valenzuela, 2011). Es decir, prácticamente surge esta técnica a raíz que todos los sistemas informáticos están orientados hacia la programación orientada a objetos (POO) que conjuntamente con las bases de datos son de tipo relacional, generando que exista cierto conflicto a la hora de almacenar datos ya que admiten solo datos de tipo (char, varchar, int etc.) y por ende no se puede almacenar directamente los objetos del código dentro de las tablas, sino que, primero se debía transformar en registros (logrando que afecte a otras tablas). Es en ese instante que los ORM logran obtener gran importancia dentro del campo del desarrollo de software debido a que convierte de forma automática los objetos en registros y así recíprocamente.

Generalmente los programadores optan por utilizar esta técnica ya que posee ciertas características, tales como:

- **Rapidez de desarrollo.-** es totalmente automático ya que trabaja velozmente creando un esquema de tablas y relaciones.
- **Abstracción del motor de la base de datos.-** es decir, al momento que se emplea ORM y se convierte automáticamente los objetos en

registros y viceversa. Y este a la vez tiene que acoplarse a los diversos proveedores, tales como (MySql, Oracle, PostreSQL, etc.).

- **Lenguaje propio para consultas a la base de datos.**- Esta característica se podría decir que es la más importante. Ya que permite a los desarrolladores extraer datos (filtros, ordenaciones, agrupaciones) es entonces donde los desarrolladores se olvidan de la sintaxis que suele tener SQL Server, y proceden a utilizar el propio lenguaje de la herramienta.
- **Interfaz usable.**- su manejo se vuelve simple ya que está diseñado para el manejo de objetos por medio de su mismo lenguaje de consulta.

Ventajas del Mapeo de Objetos Relacional.

- **Reutilización.** Utiliza los métodos de un objeto desde distintas partes de la aplicación, dependiendo de la necesidad del desarrollador.
- **Encapsulación.** Almacena la lógica de los objetos (encapsulación) logrando de tal manera modificar a la aplicación con tan solo modificar una función.
- **Portabilidad.** Permite adaptarse a cualquier otro tipo de base de datos son ningún tipo de inconvenientes.
- **Seguridad.** Facilita la seguridad a datos contra los ataques.
- **Mantenimiento del código.** El código generado es ordenado, modificable y por lo tanto es mejorable según su evolución.

Desventajas del Mapeo de Objetos Relacional.

- **Involucra tiempo.**- debido a su complejidad se requiere de tiempo de capacitación.
- **Curva de aprendizaje.**- la temática es altamente extensa debido a que posee una gran variedad de librerías que presenta ORM.
- **Menor rendimiento cuando se desarrolla un mapeador.**- se puede

dar cuando en la aplicación se suscite cierta cantidad de abundancia de contenido entre la capa del código y la base de datos relacional, generando que el rendimiento de la herramienta sea completamente bajo.

Como ya se explicó anteriormente el Mapeo de objetos relacional es una técnica muy práctica de programación, ya que permite la interacción entre el lenguaje de programación orientada a objetos y la base de datos relacional. Es decir; transforma los datos codificados las líneas de código y los transforma a datos que son almacenados en una base de datos relacional. Esta práctica que se plantea viene de la mano con ciertas características propias de la Programación Orientada a Objetos (POO), tales como la herencia y polimorfismo.

- **Herencia.**- proceso en el que se pueden generar más clases a raíz de una ya existente.
- **Polimorfismo.**- proceso que se trata de integrar varias clases que se derivan de una clase base, en donde cada una de las clases pueden utilizar los mismos métodos de otra de una manera distinta.

No obstante, el Mapeo de objeto-relacional consta además de ciertos patrones de diseño, tales como:

- **Patrón Repository.**- este tipo de patrón de diseño es aquel que utiliza un repositorio para ser un mediador entre el controlador y la base datos ya que el repositorio se encarga de crear como una especie de puente para desarrollar consultas y mapear los resultados a la entidad de negocio. Si embargo, el patrón repository ayuda a mantener un cierto orden en el desarrollo de la aplicación, ya que posee una arquitectura de tres capas y puede mejorar la usabilidad del sistema informático.

- **Patrón Active Record.**- este tipo de patrón de diseño está compuesto por tablas (filas y columnas), además consta de la lógica de negocio propuesta en la aplicación en el que permite acceder uniformemente a datos. El active Record proviene de un Domain Model esto quiere decir que cada clase está muy cerca a la representación de las bases de datos en el que permite realizar un conjunto de propiedades, tales como el CRUD, validaciones entre otros. Cabe recalcar que cada active record es responsable de sí mismo.

A continuación, en la Figura 8.1 se muestra el esquema de los patrones de diseño.

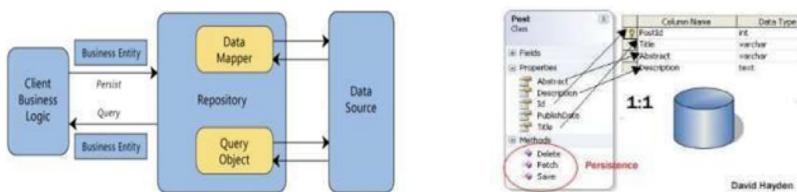


Figura 8.1. Esquema del patrón repository. Adaptado de (Duarte, 2014)

Tipos de ORMs

Actualmente la tecnología constantemente va evolucionando con el pasar de los años y se van desarrollando varios artefactos que ayudan de alguna manera a los seres humanos a simplificar algunas cosas, y en especial en el campo de la INFORMÁTICA ya que se han ido desarrollando diversos tipos de software tanto libres o propietarios, es entonces que el Mapeo de Objeto Relacional obtiene mayor relevancia dentro del campo de desarrollo de software, declarándose de tipo OPEN SPURSE y de tal manera brindando a todo el mundo el libre acceso a esta herramienta, para lo cual se procederá a mencionar los ORMs más utilizados.

- **HIBERNATE.**- Esta herramienta es libre con una licencia “LGPL (Lesser GNU Public Licence)” y trabaja con la plataforma Java, pues consiste en mapear los objetos de las clases de la aplicación y transformarlos a datos, pero esta herramienta permite modificar los datos obtenidos según la necesidad de los desarrolladores. Estos datos son almacenados de tipo “XML (Lenguajes de marcas extensibles)”.

Además, implementa la función de metadatos que consiste en emplear palabras claves que permiten agrupar grandes cantidades de información. Cabe recalcar que tras el uso de este Framework utiliza menos líneas de código para brindar una mejor interpretación, como también permite adaptarse a otros tipos de arquitecturas como: J2EE, JNDI, JTA, ayudando a que su configuración sea más optima en otros servidores. Sin embargo, al momento que se utiliza las clases se genera más sobrecarga que las consultas que se realizan en SQL Server. Además, cada una de las tablas generadas deben tener obligatoriamente una clave primaria para lograr identificar los registros ingresados, finalmente requiere que las computadoras tengan mayor rendimiento para trabajar sin inconvenientes.

- **DOCTRINE.**- Esta herramienta es libre y consiste en mapear los objetos de las clases de la aplicación y transformar a datos, lo que caracteriza a este framework es que, este contiene su “propio lenguaje SQL denominado DQL (Doctrine Query Languaje). Por otro lado, su sistema de archivo es YAML”. Además, hoy en la actualidad es un gran ORM que facilita la manipulación de su código gracias a su sistema de archivo, es decir; permite a los desarrolladores ejecutar varias operaciones para la búsqueda de información según crea conveniente el programador y permite crear manual o automáticamente la base de datos. (Amador, 2013)

- **HIBERNATE.-** Esta herramienta es la conversión de HIBERNATE para trabajar en la plataforma .NET, ya que este ORM hoy en la actualidad es el más utilizado para facilitar el mapeo de atributos entre la base de datos relacional y el modelo de objetos pues funciona con un sistema de archivo XML que permite establecer relaciones, además es fácil y adaptable a cualquier otro servidor, tales como: MySQL, PostgreSQL, Oracle, MS SQL Server, etc. (Yanes & Gracia, 2011)
- **PROPEL.-** esta herramienta es libre ya que ofrece múltiples funciones, herramientas y clases que ayudan a facilitar el desarrollo de los sistemas informáticos, puesto que se encuentra integrado por Symfony. Finalmente, su sistema de archivo es XML y trabaja únicamente en la plataforma PHP 5 logrando diseñar aplicativos nuevos, eficientes y usables.
- **SYMFONY.-** esta herramienta se diferencia gracias a la utilización del patrón de diseño MVC (Modelo, Vista, Controlador) y a la utilización de la programación orientada a objetos (POO). Por otro lado, su sistema de archivos es XML, es adaptable a cualquier otro servidor pero que trabaja con la plataforma PHP 5.2.
- **LINQ.-** Lenguaje Integrated Query esta herramienta es libre ya que permite la ejecución de consultas integradas, esto se llega a lograr debido a que tiene integrado extensiones del NET, pero, por otro lado, resulta fácil la interpretación de la funcionalidad de este framework.

Bases De Datos Relacionales.

Este tipo de base de datos no es más que un repositorio compartido de datos en el que hace disponible la relación entre datos. Esto quiere decir que por medio de estas conexiones permite que exista cierta relación entre ambas tablas dentro de una base de datos, brindando integridad, seguridad y confiabilidad a los usuarios que se dedican a

trabajar exclusivamente con bases de datos relacionales. Sin embargo, una base de datos relacional cumple con el **Modelo Relacional** ya que es el modelo más manipulado hoy en día.

Entre las principales características que cumple la base de datos relacional es que se compone de diversas tablas (filas y columnas) o relaciones, cada tabla jamás debe poseer dos tablas con el mismo nombre ni registros. Para un mejor desenvolvimiento es necesario utilizar claves primarias, ajenas (o foráneas) para distinguir a una tabla de la otra.

Elementos de una Base De Datos Relacional.

- **Relaciones base y derivadas.-** consiste en que todos los datos ingresados son almacenados y en el que se puede acceder a ellos por medio de relaciones, las relaciones que son a las almacenadas se convierten en: “Relaciones base”. En cambio, otras relaciones no almacenan datos y pasan a ser: “Relaciones Derivadas “. Esta relación es importante ya que expresa información de diversas relaciones como si fuese uno solo.
- **Restricciones.-** en otros términos, no es más que una condición que se debe cumplir ya que algunas pueden ser puestas por el usuario y otras suelen ser por la base de datos misma, como por ejemplo en una tabla utilizar campos solo numéricos, etc. Por otra parte, las restricciones no forman parte del modelo relacional, pero si juegan un papel muy eficiente ya que brinda una organización de datos.
- **Dominio.-** prácticamente se trata de que, si un atributo tiene ciertos elementos, los demás datos deben ser los mismos. Los dominios pueden ser: numéricos, alfa numéricos, fechas, texto, etc.
- **Clave primaria.-** no es más que una clave principal y única que se le denomina a un solo atributo de la tabla para lograr distinguir a una tabla de las demás. No obstante, ningún campo de dicha clave principal debe poseer valores de tipo NULL.

- **Clave foránea.**- es aquella en la que determina la relación que posee ambas tablas, esto quiere decir; es una migración del código principal de una tabla a otra.
- **Clave índice.**- es aquella en la que son desarrolladas por los programadores mismos, creando ciertas combinaciones de campos de una tabla para tener acceso rápido a los datos de dicha tabla. Sin embargo, la clave índice no forma parte de la base de datos, pero si son un complemento agregado que ayuda a acceder directamente a datos.

Manejo De Bases De Datos Relacionales

Existen diversos softwares que manejan bases de datos relaciones, a este se le domina como “**SGBD**” (Sistema de Gestión de Bases de Datos relacional) o **RDBMS** (Relational Database Management System) “ (SuárezMS, 2008). Para lo cual unos de los más populares son: PostgreSQL, Oracle, Microsoft SQL Server DB2, Informix, MySQL, FireBird, Interbase.

- **POSTGRES SQL** - es un software libre de gran alcance que consiste en el gestionamiento de bases de datos relacional, además está en el mercado por 15 años de desarrollo siendo un software altamente fiable, gracias a su arquitectura. Postgress trabaja con cualquier sistema operativo, tales como: Linux, Unix (en todas sus versiones) Y Windows en varios idiomas, entre sus principales características es que es fácil de manipular, por su estabilidad, potencia, robustez, trabaja mediante un sistema MVCC, soporta distintos tipos de datos, como: numéricos, enteros, Booleano, intervalo, timestamp, compatible con el almacenamiento de objetos como imágenes, sonidos o también videos y finalmente porta con una buena documentación.
- **ORACLE.**- es un software libre desarrollado por la empresa misma de Oracle Corporation que trata sobre el desarrollo de bases de datos

según fuentes bibliográficas expresan que: " ocupa el primer lugar en la categoría de las bases de datos y el séptimo lugar a nivel mundial de las compañías de tecnologías de la información." (Ecured, 2016) Ya que es declarado como un gestor de bases de datos muy potente a nivel mundial por su rendimiento a gran escala, finalmente trabaja en base a una herramienta cliente-servidor.

- **Microsoft SQL Server.**- es un software libre que consiste en el manejo de las bases de datos del modelo relacional, en el que fue desarrollado por la empresa misma de Microsoft. Una de las principales características de este software es que puede soportar múltiples transacciones, tales como: (CRUD, cancelar, almacenar, etc.) además administra información de otros servidores de datos, es capaz de guardar grandes cantidades de información, está compuesto por T-SQL el cual permite desarrollar operaciones que son claves dentro de SQL Server.
- **MySQL.**- es un software libre ya que fue: "desarrollado bajo licencia dual GPL/Licencia comercial por Oracle Corporation" (Quizlet, 2017) y que consiste en el gestionamiento de bases de datos puesto que fue desarrollado por la empresa fundada por David Axmark Larsson y Michael Widenius llamada MySQL AB considerándose el software más popular de todos los tiempos. Entre una de las principales características que lo distinguen es que está disponible para cualquier tipo de plataformas y sistemas, además su conectividad es completamente segura, muy usable ya que es completamente fácil de manipularlo, posee un sistema de memoria de reserva basado en threads entre otros.
- **Interbase.**- es un software libre que consiste en el gestionamiento de la base de datos, pues actualmente no es el software más implementado ya que corre solo con servidores como Linux, Windows y Solaris. Por otra parte, lo que le caracteriza es que no necesita muchos recursos para ser instalado y finalmente su arquitectura es multi-generacional (es relativo a varias generaciones).

- **FireBrid.**- es un software libre que consiste en el gestionamiento de bases de datos tales como: ejecutar consultas, diseño de tablas, aplicar CRUD en registros, generar scripts, analizar, monitorear entre otras más funcionales que proporciona este software. Entre las principales características es que es multi-plataforma y es adaptable a cualquier sistema operativo, brinda buena seguridad en usuario y roles, además tiene la capacidad de almacenar elementos tipo blob (o también de texto denominado bloque de texto que puede abarcar hasta 65535 caracteres), alto rendimiento, entre otros. Actualmente se ha liberado la versión 22 portando mayores funcionalidades eficientes.
- Cada una de las bases de datos relacionales son las más populares y utilizadas frecuentemente por múltiples desarrolladores, cada uno cumple una función específica y especialmente es adaptada cada una de ellas a las necesidades requeridas por los desarrolladores. Presentan una interfaz completamente entendible, es lo que les hace muy usable.

Capítulo IX

Configurando ORM con un almacén de datos PostgreSQL

Luego de conocer los aspectos conceptuales de la Tercera Capa o Capa de Datos, en este capítulo vamos a configurar y preparar el ambiente de trabajo en lo que respecta al almacén de datos de nuestro sistema de información ejemplo. Es importante mencionar que aparte de la configuración del entorno de la tercera capa, posteriormente será necesario una configuración adicional para su interrelación con la segunda capa o capa de negocio. La temática de interrelación entre segunda y tercera capa será tratada en un capítulo posterior, al momento nos focalizaremos en preparar las herramientas necesarias para que nuestra tercera capa soporte la información que le envíen.

Dos factores son importantes considerar para la preparación de nuestro ambiente de trabajo de la tercera capa. La primera, es el motor de base de datos elegido como gestor de almacén de información, para nuestro caso de estudio se ha utilizado la herramienta PostgreSQL 9.6 y su administrador gráfico PgAdmin4. La Segunda, es la creación y configuración de una puerta de enlace que permita que todo este ambiente de almacenamiento tenga accesibilidad desde nuestro sistema de información, específicamente desde la capa de negocio.

En cuanto la puerta de enlace es importante considerar la forma de interacción que tendrá la segunda capa y tercera capa, por defecto Play Framework tiene habilitado como método de comunicación a JPA (Java Persistence API) que es un ORM con especificaciones Java que definen un estándar API y que será el intérprete en la comunicación de nuestra capa de datos y negocio.

Una vez que se ha definido las características de las herramientas de software a ser usadas en esta tercera capa, ha sido necesario definir las siguientes etapas de trabajo para cumplir con la meta propuesta:

- Creación de un almacén de datos en PostgreSQL
- Configuración de la puerta de acceso a la tercera capa

Creación de un almacén de datos en PostgreSQL

Antes de crear el almacén de datos en PostgreSQL, es importante crear un usuario en PostgreSQL con el cual nuestra aplicación SACC se conecte a la base de datos. La creación del usuario es por temas de seguridad, ya que no es conveniente para un administrador de base de datos compartir a terceros el usuario y password del administrador con todos los privilegios que este posee. Lo que el administrador de base de datos debe hacer, es crear un usuario con los privilegios necesarios para acceder al almacén de datos que la aplicación implicada lo demande.

Para nuestro caso de estudio, como se muestra en la Figura 9.1 se ha creado el usuario de base de datos “sacc” usando como password el mismo nombre del usuario. Para la creación de usuario damos click derecho sobre la opción “Login/Group/ Roles”.

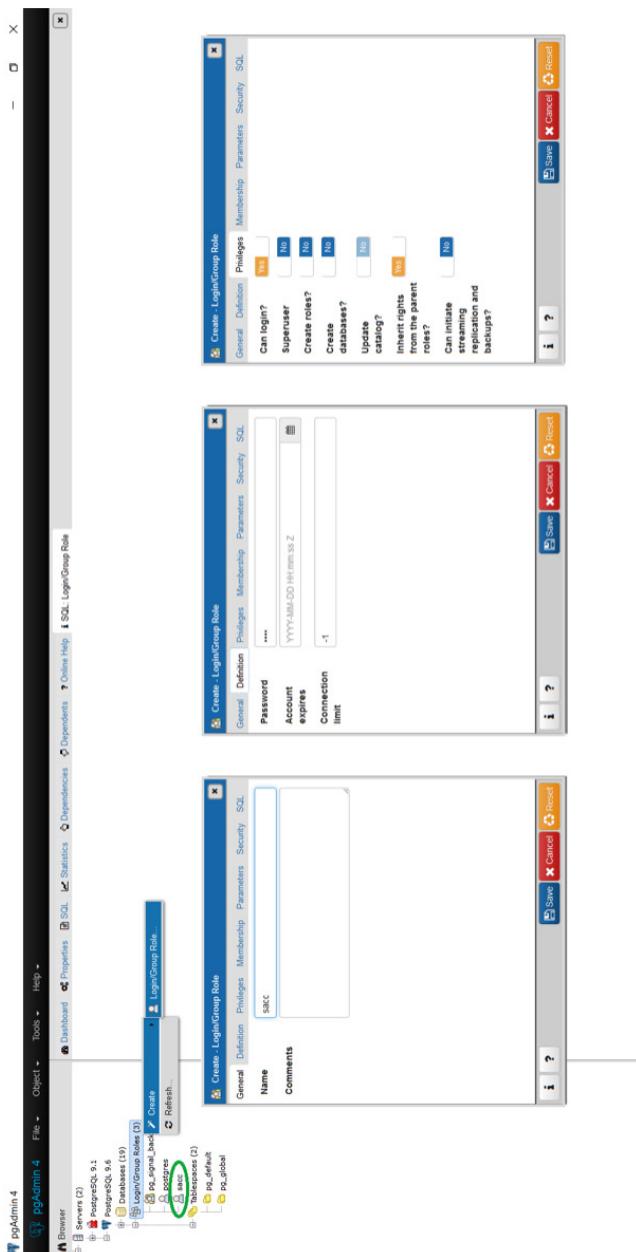


Figura 9.1. Creación de usuarios en PostgreSQL

Una vez que el usuario ha sido creado, procedemos a crear la base de datos, es importante vincular el nuevo usuario en el almacén de datos que creemos. En la Figura 9.2 se presenta las acciones para crear la base de datos, como primer paso damos click derecho sobre “Databases”. Recuerde que en “Tables” se almacenará toda la información que reciba esta tercera capa.

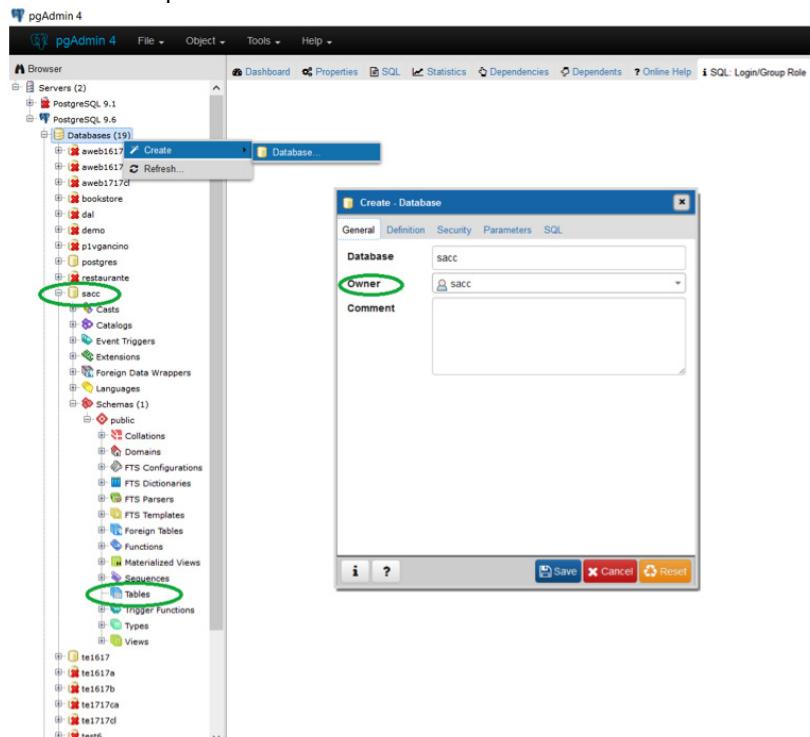


Figura 9.2. Creación de una base de datos en PostgreSQL

Configuración de la puerta de acceso a la tercera capa

Una vez que tenemos nuestra base de datos creada con su respectivo usuario de trabajo, el siguiente paso es crear una puerta de acceso desde nuestro proyecto de play framework hacia el almacén de datos. Es decir,

esta puerta será el canal de comunicación entre el segunda capa o capa de negocio y la tercera capa o capa de datos. Para nuestro caso de estudio el enlace que se establecerá es la comunicación entre Java y la base de datos PostgreSQL.

Esta puerta se la configura a través de una cadena de conexión, la misma que tiene que ser aplicada en un archivo de configuración de nuestro proyecto. Recuerde que los archivos de configuración de nuestro proyecto se encuentran dentro de la carpeta “*%conf*”. Y para este caso la cadena de conexión la podemos configurar bajo la línea 91 del archivo “*application.conf*”. En la Figura 9.3 se muestra la cadena conexión necesaria, esta cadena es una copia de la línea 91, con las siguientes modificaciones: la primera es la edición de los datos de usuario, clave, nombre del servidor y nombre de la base de datos, en nuestro caso se ha creado una base de datos y usuario con el nombre de “*sacc*” y el nombre del servidor es “*localhost*”; la segunda es eliminar el carácter “#” que significa que dicha línea de configuración está o no está comentada.

```

Project Explorer Navigator
sacc
  settings
    app
      controllers
        Application.java
      models
        Catalog.java
        Item.java
        Product.java
        Register.java
        Sale.java
        Store.java
      views
    conf
      application.conf
      dependencies.yml
      messages
      routes
  eclipse
    settings
    classes
      classpath
      project
      Connect JPA to saccLaunch
      saccLaunch
      Test saccLaunch
  lib
  public
  test
  tmp
  uml
  .classpath
  .project
application.conf
# application configuration
# application.log.path=/log4j.properties
# application.log.system.out=off
#
# Database configuration
# -----
# Enable a database engine if needed.
# -----
# To quickly set up a development database, use either:
# - mem : for a transient in memory database (H2 in memory)
# - fs : for a simple file written database (H2 file stored)
# db=mem
#
# To connect to a local MySQL5 database, use:
# db=mysql://user:pwd@host/database
# -----
# To connect to a local PostgreSQL9 database, use:
# db=postgres://user:pwd@host/database
# db=postgres://sacc:sacc@localhost/sacc
#
# If you need a full JDBC configuration use the following :
# db.url=jdbc:postgresql:database_name
# db.driver=org.postgresql.Driver
# db.user=root
# db.pass=secret
#
# Connections pool configuration :
# db.pool.timeout=1000
# db.pool.maxSize=30
# db.pool.minSize=10
#
# Specify the custom JPA dialect to use here (default to guess):
# jpa.dialect=org.hibernate.dialect.PostgreSQLDialect
#
# Specify the ddl generation pattern to use. Set to none to disable it
# (default to update in DEV mode, and none in PROD mode):
# jpa.ddl=update
#
# Debug SQL statements (logged using DEBUG level):
# jpa.debugSQL=true
#
# You can even specify additional hibernate properties here:
# hibernate.use_sql_comments=true
#
# -----
# test.module.cobertura=${play.path}/modules/cobertura
# test.application.mode=dev
# test.db.url=jdbc:h2:mem:play;MODE=MYSQL;LOCK_MODE=0
# test.jpa.ddl=create
# test.mail.smtp=mock
#
# -----
# 213 #test.module.cobertura=${play.path}/modules/cobertura
# 214 #test.application.mode=dev
# 215 #test.db.url=jdbc:h2:mem:play;MODE=MYSQL;LOCK_MODE=0
# 216 #test.jpa.ddl=create
# 217 #test.mail.smtp=mock
# 218 #

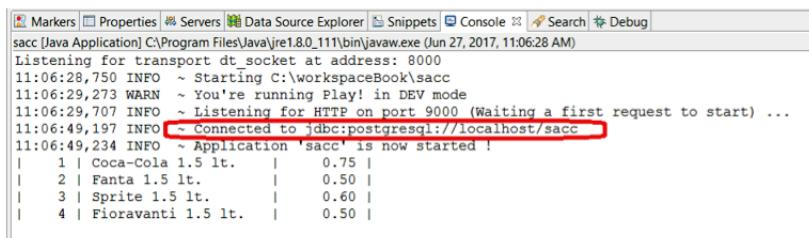
```

Figura 9.3. Configuración de la cadena de conexión de Play a PostgreSQL

En el mismo archivo “*application.conf*” se puede modificar aspectos relacionados a la labor del ORM de play framework que es el JPA, el mismo que será el lenguaje de comunicación entre nuestro modelo de clases y la gestor de base de datos. Por defecto viene configurado y habilitado el JPA, pero en el caso de requerir diferentes comportamientos de su uso, puede alterar su configuración entre las líneas 116 y 129 o también entre las líneas 213 y 217 (ver Figura 9.3).

Finalmente, para verificar la conexión con la base de datos, ejecutamos el proyecto y refrescamos la página principal de la interfaz, si la

configuración está correcta y hay enlace con el gestor de datos en la consola le desplegará el siguiente mensaje (ver Figura 9.4).



The screenshot shows a Java application window with the title bar "sacc [Java Application] C:\Program Files\Java\jre1.8.0_111\bin\javaw.exe (Jun 27, 2017, 11:06:28 AM)". The console tab is active, displaying log messages. One message is highlighted with a red box: "Connected to jdbc:postgresql://localhost/sacc".

```
sacc [Java Application] C:\Program Files\Java\jre1.8.0_111\bin\javaw.exe (Jun 27, 2017, 11:06:28 AM)
Listening for transport dt_socket at address: 8000
11:06:28,750 INFO ~ Starting C:\workspaceBook\sacc
11:06:29,273 WARN ~ You're running Play! in DEV mode
11:06:29,707 INFO ~ Listening for HTTP on port 9000 (Waiting a first request to start) ...
11:06:49,197 INFO ~ Connected to jdbc:postgresql://localhost/sacc
11:06:49,234 INFO ~ Application 'sacc' is now started!
| 1 | Coca-Cola 1.5 lt. | 0.75 |
| 2 | Fanta 1.5 lt. | 0.50 |
| 3 | Sprite 1.5 lt. | 0.60 |
| 4 | Fioravanti 1.5 lt. | 0.50 |
```

Figura 9.4. Ejeccción de conexión a PostgreSQL exitosa

Sección 5: Interacción entre capas

En las secciones dos, tres y cuatro se ha realizado una descripción detallada de los aspectos conceptuales, funcionales y prácticos de cada una de las capas (presentación, negocio y datos) que integran la estructura de un sistema de información web. Pero a más de conocer las responsabilidades que tienen cada una de las capas en su entorno estructural, es necesario también conocer las estrategias de interacción que tienen entre ellas.

Como la segunda capa o capa de negocio es la intermediaria entre las tres capas, es necesario conocer la forma de iteración entre la segunda-tercera capa y la segunda-primera capa. En esta sección se presenta las palabras clave que generalmente son usadas para el intercambio de información y comunicación entre las capas implicadas. Es necesario primero revisar y configurar la iteración entre la capa dos (negocio) y tres (datos) puesto que con la relación de estas capas se logra cargar la información inicial necesaria para el desempeño del sistema de información web.

Una vez que la información inicial ha sido cargada, ya se puede proceder a relacionar la capa uno (presentación) y dos (negocio), de esta manera

se cumple con el ciclo de comunicación completo de la estructura de aplicación planteada. Es así como termina el desarrollo del caso de estudio práctico, todas las capas han sido configuradas y desarrolladas, para finalmente habilitar la correlación entre ellas, de esta manera el usuario puede extraer información desde la tercera capa hacia la primera capa y viceversa, puede enviar información desde la primera capa hacia la tercera capa.

Capítulo X

Tags para interacción entre segunda y tercera capa

Luego de haber tratado de forma independiente los fundamentos, configuraciones y desarrollos necesarios para la segunda capa (capa de negocio) y tercera capa (capa de datos), es momento de conocer la forma en que ambas capas pueden interrelacionarse. La necesidad principal de que las dos capas se comuniquen es a razón de que toda la información que se ha recopilado y procesado a forma de clases y objetos en la capa de negocio tiene que transformarse a tablas con filas de contenidos. Esto a razón de que la cantidad de datos en los sistemas de información mayoritariamente siempre tienden a incrementarse y no es conveniente que los datos estén almacenados a manera de objetos en memoria (como lo hace la capa de negocio con su lenguaje de programación orientada objetos) sino que por la cantidad estos puedan ser almacenados en un gestor de base de datos, creado específicamente para soportar grandes cantidades de información.

Tecnológicamente existen varias maneras de comunicar estas dos capas, una de ellas es el uso del ORM (Object Relation Mapping) y dentro de este tipo de tecnología existen varias herramientas, entre las más conocidas tenemos a Hibernate, JPA (Java Persistence API), Django ORM, PonyORM, SQLObject y otros. Para nuestro caso de estudio usamos JPA cuya configuración en Play Framework viene habilitada por defecto, pero para su uso se requiere de determinados tags o palabras claves en cada una de las clases de nuestro modelo de dominio establecido en la capa de negocio.

Para cumplir con la meta de este capítulo (la interrelación entre la segunda y tercera capa) se ha definido las siguientes etapas de trabajo, en el transcurso se irá presentando el detalle de cada una de estas etapas:

- Asignar los tags o palabras claves a las clases de nuestro modelo
- Asignar los tags o palabras claves de las relaciones entre clases
- Ejecución de prueba para almacenamiento de datos

Asignar los tags o palabras claves a las clases de nuestro modelo

Para que las clases sean transformadas a tablas, tres son las adecuaciones que debemos hacer a nuestras clases de la capa de negocio: la primera es agregar el tag `@Entity` sobre la declaración de la clase; la segunda es extender de la clase Model, `extends Model`, y la tercera es importar las librería necesarias para el uso de estos tags o palabras clave. Efectuadas estas tres adecuaciones la clase quedaría con la siguiente estructura:

```
7 import javax.persistence.Entity;
8 import play.db.jpa.Model;
9
10
11 //DECLARACIÓN DE LA CLASE
12 @Entity
13 public class Store extends Model {
14     // ATRIBUTOS
15     private int nextSaleNumber;
16     private Set<Sale> completedSales;
```

El extender a la clase de Model, nos permite que la clase pueda ser interpretada por el JPA, para que, por medio de su lenguaje de comunicación con el gestor de datos, se pueda almacenar la información de la clase. Y con respecto al uso de la palabra clave “Entity” su función es ser un identificador que comunica al JPA que dicha clase deberá ser tratada como una tabla.

Estas adecuaciones hay que hacerlo con todas las clases de nuestro modelo de negocio, es decir con: Catalog, Item, Product, Register, Sale y Store. La definición de estas palabras claves nos permite que las clases persistan (convertir objeto de memoria en dato almacenado) en nuestra base de datos a manera de tablas, esto a través de la actuación o intermediación del JPA de play framework.

Asignar los tags o palabras claves de las relaciones entre clases

En la Tabla 10.1 se registró las relaciones existentes en nuestro modelo y también se planteó, la forma en que convertíamos la relación UML en relación Java. En dicha tabla la relación Java definida, únicamente efectuaba el vínculo a manera de clases y objetos guardados en memoria; ahora como tenemos configurado nuestro JPA que se comunica con una base de datos, estas clases se convertirán en tablas, por lo tanto, es necesario comunicar al JPA través de palabras clave, que existen relaciones en las clases que requieren ser tratadas en la base de datos como relaciones de entidad-relación.

De la misma forma como en el capítulo 10 se listó las relaciones existentes en nuestro modelo y su convertibilidad de relación UML a relación Java, ahora vamos a listar las relaciones existentes mostrando las adecuaciones necesarias para que las relaciones persistan en la base de datos. En la Tabla 10.1 se muestra las palabras clave que han sido necesarias ubicar sobre el código de las relaciones, para que el JPA de play framework las interprete y las convierta en relaciones entre tablas en nuestra base de datos PostgreSql.

En la Tabla 10.1 se lista las relaciones de dependencia de nuestro modelo, las cuales para que persistan con nuestra base de datos requiere de las palabras claves `@OneToOne`. El término “`cascade = CascadeType.ALL`” que se encuentra entre paréntesis significa que la persistencia se propagará para todas las operaciones (borrar, refrescar, combinar) de relaciones entre entidades.

Dependencia (uno a uno)	
Relación Java	Relación Java conz persistencia JPA
<pre> 1 //PAQUETE AL QUE PERTENECE LA CLASE 2 package models; 3 4 //DECLARACIÓN DE LA CLASE 5 public class Register { 6 // ATRIBUTOS 7 private String code; 8 private String cashier; 9 private Store store; 10 private Catalog catalog; 11 private Sale currentSale; 12 13 // CONSTRUCTORES 14@ public Register(String code, String 15 this.code = code; 16 this.cashier = cashier; 17 this.store = store; 18 this.catalog = catalog; 19 this.currentSale = null; 20 } </pre>	<pre> 1 //PAQUETE AL QUE PERTENECE LA CLASE 2 package models; 3 4@import javax.persistence.CascadeType; 5 import javax.persistence.Entity; 6 import javax.persistence.OneToOne; 7 import play.db.jpa.Model; 8 9 //DECLARACIÓN DE LA CLASE 10 @Entity 11 public class Register extends Model { 12 // ATRIBUTOS 13 private String code; 14 private String cashier; 15@ @OneToOne(cascade = CascadeType.ALL) 16 private Store store; 17@ @OneToOne(cascade = CascadeType.ALL) 18 private Catalog catalog; 19@ @OneToOne(cascade = CascadeType.ALL) 20 private Sale currentSale; 21 22 // CONSTRUCTORES 23@ public Register(String code, String cashie 24 this.code = code; 25 this.cashier = cashier; 26 this.store = store; 27 this.catalog = catalog; 28 this.currentSale = null; 29 } ..</pre>
<pre> 1 //PAQUETE AL QUE PERTENECE LA CLASE 2 package models; 3 4 //DECLARACIÓN DE LA CLASE 5 public class Item { 6 // ATRIBUTOS 7 private int quantity; 8 private Product product; 9 10 // CONSTRUCTORES 11@ public Item(int quantity, Product p 12 this.quantity = quantity; 13 this.product = p; 14 } 15 </pre>	<pre> 1 //PAQUETE AL QUE PERTENECE LA CLASE 2 package models; 3 4@import javax.persistence.CascadeType; 5 import javax.persistence.Entity; 6 import javax.persistence.OneToOne; 7 import play.db.jpa.Model; 8 9 //DECLARACIÓN DE LA CLASE 10 @Entity 11 public class Item extends Model { 12 // ATRIBUTOS 13 private int quantity; 14@ @OneToOne(cascade = CascadeType.ALL) 15 private Product product; 16 17 // CONSTRUCTORES 18@ public Item(int quantity, Product p) { 19 this.quantity = quantity; 20 this.product = p; 21 } 22 </pre>

Tabla 10.1. Relaciones de dependencia de la aplicación caso de estudio en modo persistencia.

<

En la Tabla 10.2 se lista las relaciones de Composición de nuestro modelo, las cuales para que persistan con nuestra base de datos requiere de las palabras claves “`@OneToMany`”. Aquí también utilizamos “`cascade = CascadeType.ALL`” para que la persistencia se propague para todas las operaciones (borrar, refrescar, combinar) de relaciones entre entidades.

Composición (uno a varios)	
Relación Java	Relación Java con persistencia JPA
<pre> 1 //PAQUETE AL QUE PERTENECE LA CLASE 2 package models; 3 4 //LIBRERÍAS NECESARIAS 5 import java.util.HashMap; 6 import java.util.Map; 7 8 //DECLARACIÓN DE LA CLASE 9 public class Catalog { 10 // ATRIBUTOS 11 private String name; 12 private Map<Long, Product> products; 13 14 // CONSTRUCTORES 15 public Catalog(String name) { 16 this.name = name; 17 this.products = new HashMap<Long, Product>(); 18 } 19 </pre>	<pre> 1 //PAQUETE AL QUE PERTENECE LA CLASE 2 package models; 3 4 //LIBRERÍAS NECESARIAS 5 import java.util.HashMap; 6 import java.util.Map; 7 8 import javax.persistence.CascadeType; 9 import javax.persistence.Entity; 10 import javax.persistence.MapKey; 11 import javax.persistence.OneToMany; 12 13 import play.db.jpa.Model; 14 15 //DECLARACIÓN DE LA CLASE 16 @Entity 17 public class Catalog extends Model { 18 // ATRIBUTOS 19 private String name; 20 @OneToMany(cascade = CascadeType.ALL) 21 @MapKey(name = "code") 22 private Map<Long, Product> products; 23 24 // CONSTRUCTORES 25 public Catalog(String name) { 26 this.name = name; 27 this.products = new HashMap<Long, Product>(); 28 } </pre>
<pre> 1 //PAQUETE AL QUE PERTENECE LA CLASE 2 package models; 3 4 //LIBRERÍAS NECESARIAS 5 import java.util.HashSet; 6 import java.util.Set; 7 8 //DECLARACIÓN DE LA CLASE 9 public class Store { 10 // ATRIBUTOS 11 private int nextSaleNumber; 12 private Set<Sale> completedSales; 13 14 // CONSTRUCTORES 15 public Store() { 16 this.nextSaleNumber = 1; 17 this.completedSales = new HashSet<Sale>(); 18 } 19 </pre>	<pre> 1 //PAQUETE AL QUE PERTENECE LA CLASE 2 package models; 3 4 //LIBRERÍAS NECESARIAS 5 import java.util.HashSet; 6 import java.util.Set; 7 import javax.persistence.CascadeType; 8 import javax.persistence.Entity; 9 import javax.persistence.OneToMany; 10 import play.db.jpa.Model; 11 12 //DECLARACIÓN DE LA CLASE 13 @Entity 14 public class Store extends Model { 15 // ATRIBUTOS 16 private int nextSaleNumber; 17 @OneToMany(cascade = CascadeType.ALL) 18 private Set<Sale> completedSales; 19 20 // CONSTRUCTORES 21 public Store() { 22 this.nextSaleNumber = 1; 23 this.completedSales = new HashSet<Sale>(); 24 } </pre>

```

1 //PAQUETE AL QUE PERTENECE LA CLASE
2 package models;
3
4 //LIBRERIAS NECESARIAS
5 import java.util.ArrayList;
6 import java.util.Date;
7 import java.util.List;
8
9 //DECLARACIÓN DE LA CLASE
10
11 public class Sale {
12     // ATRIBUTOS
13     private int number;
14     private Date date;
15     private boolean complete;
16     private double subtotal;
17     private List<Item> items;
18
19     // CONSTRUCTORES
20     public Sale() {
21         this.complete = false;
22         this.items = new ArrayList<Item>();
23     }
24
25 }
26
27
28
29
1 //PAQUETE AL QUE PERTENECE LA CLASE
2 package models;
3
4 //LIBRERIAS NECESARIAS
5 import java.util.ArrayList;
6 import java.util.Date;
7 import java.util.List;
8 import javax.persistence.CascadeType;
9 import javax.persistence.Entity;
10 import javax.persistence.OneToMany;
11 import play.db.jpa.Model;
12
13 //DECLARACIÓN DE LA CLASE
14 @Entity
15 public class Sale extends Model {
16     // ATRIBUTOS
17     private int number;
18     private Date date;
19     private boolean complete;
20     private double subtotal;
21     @OneToMany(cascade = CascadeType.ALL)
22     private List<Item> items;
23
24     // CONSTRUCTORES
25     public Sale() {
26         this.complete = false;
27         this.items = new ArrayList<Item>();
28     }
29 }

```

Tabla 10.2. Relaciones de composición de la aplicación caso de estudio en modo persistencia.

Una vez que se ha fijado las palabras clave necesarias en cada una de las relaciones, nuestro modelo está listo para poder ser interpretado por el JPA de play framework, una vez interpretado, este modelo se convertirá en tablas con relaciones al estilo de un modelo entidad-relación en nuestra base de datos. Es importante mencionar que los tags o palabras clave utilizadas en este capítulo son únicamente los más elementales en JPA, existen una serie de palabras clave más, que ayudan a mejorar la comunicación entre nuestro modelo de clases y nuestro modelo entidad-relación a través de JPA.

Ejecución de prueba para almacenamiento de datos

Finalmente, para verificar que nuestra capa de negocio este correctamente configurada y revisa la persistencia con nuestra capa de datos, procedemos a ejecutar nuestra función de prueba desarrollada en el controlador “Application.java”. Realizamos una pequeña modificación a la función,

simplemente después de instanciar los objetos necesarios y agregar al vector de productos del catálogo las bebidas requeridas, incluimos la siguiente sentencia “`catalog.save();`” como muestra la Figura 10.1.

```
1 package controllers;
2
3 import play.*;
4
5
6 public class Application extends Controller {
7
8     public static void index() {
9
10         testingBusinessTier2();
11
12         render();
13     }
14
15     private static void testingBusinessTier2() {
16         Catalog catalog = new Catalog("Bebidas");
17         catalog.addProduct(1, "Coca-Cola 1.5 lt.", 0.75, "cocaCola.png");
18         catalog.addProduct(2, "Fanta 1.5 lt.", 0.50, "fanta.jpg");
19         catalog.addProduct(3, "Sprite 1.5 lt.", 0.60, "sprite.png");
20         catalog.addProduct(4, "Fioravanti 1.5 lt.", 0.50, "fioravanti.jpg");
21         catalog.save();
22
23         for (Product p : catalog.getProducts().values()) {
24             System.out.println(String.format("| %1$4d | %2$-20s | %3$8.2f |",
25                                 p.getCode(), p.getDescription(), p.getPrice()));
26         }
27     }
28 }
29 }
```

Figura 10.1. Sentencia para ejecutar persistencia en el almacén de datos

Con esta sentencia lo que estamos haciendo es que se confirme la ejecución de la persistencia con nuestra base de datos. Es decir cada vez que a una clase le asignemos un `save()` el JPA de play framework buscará todos los objetos vinculados a esa clase que hasta ese momento se hayan instanciado en memoria y los transformará en información que pueda ser almacenada en tablas de nuestra base de datos. Es la sincronización de los objetos con las tablas de PostgreSQL.

Si toda la configuración está en orden al correr nuevamente el proyecto y actualizar la página de la interfaz, el JPA habrá convertido todas las clases en tablas físicas en nuestra base de datos. Nos dirigimos a nuestro PgAdmin de PostgreSQL y en la sección de tablas del esquema, como

se presenta en la Figura 10.2, verificamos que automáticamente todas las tablas a un estilo de modelo entidad-relación han sido creadas en nuestra base de datos. Al revisar los datos de las tablas Catalog, Product y su relación se puede apreciar que todos los objetos que instanciamos en el controlador han sido almacenados.

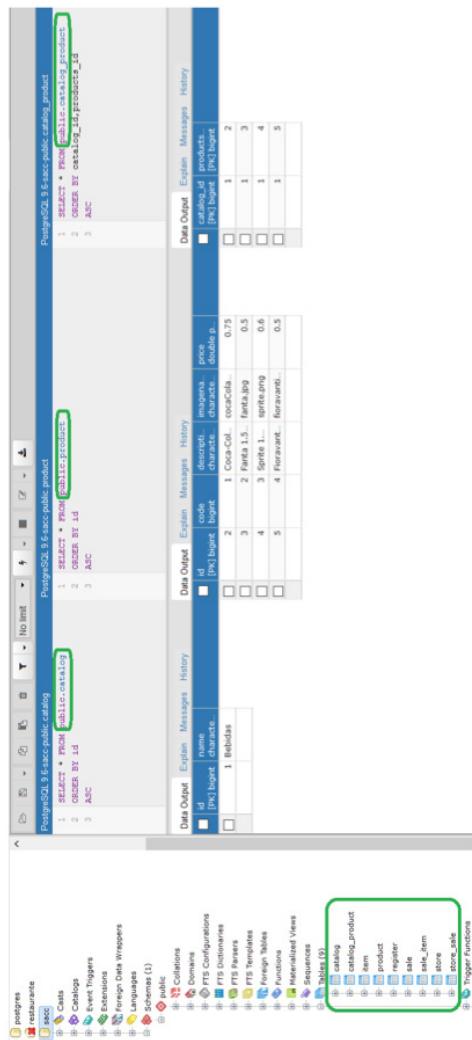


Figura 10.2. Tablas creadas en la Base de datos por el ORM de Play Framework

Capítulo XI

Tags para interacción entre primera y segunda capa

Luego de haber tratado de forma independiente los fundamentos, configuraciones y desarrollos necesarios para la segunda capa (capa de negocio) y tercera capa (capa de datos), es momento de conocer la forma en que ambas capas pueden interrelacionarse. La necesidad principal de que las dos capas se comuniquen es a razón de que toda la información que se ha recopilado y procesado a forma de clases y objetos en la capa de negocio tiene que transformarse a tablas con filas de contenidos. Esto a razón de que la cantidad de datos en los sistemas de información mayoritariamente siempre tienden a incrementarse y no es conveniente que los datos estén almacenados a manera de objetos en memoria (como lo hace la capa de negocio con su lenguaje de programación orientada objetos) sino que por la cantidad estos puedan ser almacenados en un gestor de base de datos, creado específicamente para soportar grandes cantidades de información.

Tecnológicamente existen varias maneras de comunicar estas dos capas, una de ellas es el uso del ORM (Object Relation Mapping) y dentro de este tipo de tecnología existen varias herramientas, entre las más conocidas tenemos a Hibernate, JPA (Java Persistence API), Django ORM, PonyORM, SQLObject y otros. Para nuestro caso de estudio usamos JPA cuya configuración en Play Framework viene habilitada por defecto, pero para su uso se requiere de determinados tags o palabras claves en cada una de las clases de nuestro modelo de dominio establecido en la capa de negocio.

Para cumplir con la meta de este capítulo (la interrelación entre la segunda y tercera capa) se ha definido las siguientes etapas de trabajo, en el transcurso se irá presentando el detalle de cada una de estas etapas:

- Asignar los tags o palabras claves a las clases de nuestro modelo
- Asignar los tags o palabras claves de las relaciones entre clases
- Ejecución de prueba para almacenamiento de datos

Asignar los tags o palabras claves a las clases de nuestro modelo

Para que las clases sean transformadas a tablas, tres son las adecuaciones que debemos hacer a nuestras clases de la capa de negocio: la primera es agregar el tag `@Entity` sobre la declaración de la clase; la segunda es extender de la clase Model, `extends Model`, y la tercera es importar las librería necesarias para el uso de estos tags o palabras clave. Efectuadas estas tres adecuaciones la clase quedaría con la siguiente estructura:

```
7  
8 import javax.persistence.Entity;  
9 import play.db.jpa.Model;  
10  
11 //DECLARACIÓN DE LA CLASE  
12 @Entity  
13 public class Store extends Model {  
14     // ATRIBUTOS  
15     private int nextSaleNumber;  
16     private Set<Sale> completedSales;
```

El extender a la clase de Model, nos permite que la clase pueda ser interpretada por el JPA, para que, por medio de su lenguaje de comunicación con el gestor de datos, se pueda almacenar la información de la clase. Y con respecto al uso de la palabra clave “Entity” su función es ser un identificador que comunica al JPA que dicha clase deberá ser tratada como una tabla.

Estas adecuaciones hay que hacerlo con todas las clases de nuestro modelo de negocio, es decir con: Catalog, Item, Product, Register, Sale y Store. La definición de estas palabras claves nos permite que las clases persistan (convertir objeto de memoria en dato almacenado) en nuestra base de datos a manera de tablas, esto a través de la actuación o intermediación del JPA de play framework.

Asignar los tags o palabras claves de las relaciones entre clases

En la Tabla 10.1 se registró las relaciones existentes en nuestro modelo y también se planteó, la forma en que convertíamos la relación UML en relación Java. En dicha tabla la relación Java definida, únicamente efectuaba el vínculo a manera de clases y objetos guardados en memoria; ahora como tenemos configurado nuestro JPA que se comunica con una base de datos, estas clases se convertirán en tablas, por lo tanto, es necesario comunicar al JPA través de palabras clave, que existen relaciones en las clases que requieren ser tratadas en la base de datos como relaciones de entidad-relación.

De la misma forma como en el capítulo 10 se listó las relaciones existentes en nuestro modelo y su convertibilidad de relación UML a relación Java, ahora vamos a listar las relaciones existentes mostrando las adecuaciones necesarias para que las relaciones persistan en la base de datos. En la Tabla 10.1 se muestra las palabras clave que han sido necesarias ubicar sobre el código de las relaciones, para que el JPA de play framework las interprete y las convierta en relaciones entre tablas en nuestra base de datos PostgreSQL.

En la Tabla 10.1 se lista las relaciones de dependencia de nuestro modelo, las cuales para que persistan con nuestra base de datos requiere de las palabras claves @OneToOne. El término “cascade = CascadeType.ALL” que se encuentra entre paréntesis significa que la persistencia se propagará para todas las operaciones (borrar, refrescar, combinar) de relaciones entre entidades.

Dependencia (uno a uno)	
Relación Java	Relación Java con persistencia JPA
<pre> 1 //PAQUETE AL QUE PERTENECE LA CLASE 2 package models; 3 4 //DECLARACIÓN DE LA CLASE 5 public class Register { 6 // ATRIBUTOS 7 private String code; 8 private String cashier; 9 private Store store; 10 private Catalog catalog; 11 private Sale currentSale; 12 13 // CONSTRUCTORES 14 public Register(String code, String 15 this.code = code; 16 this.cashier = cashier; 17 this.store = store; 18 this.catalog = catalog; 19 this.currentSale = null; 20 } </pre>	<pre> 1 //PAQUETE AL QUE PERTENECE LA CLASE 2 package models; 3 4 import javax.persistence.CascadeType; 5 import javax.persistence.Entity; 6 import javax.persistence.OneToOne; 7 import play.db.jpa.Model; 8 9 //DECLARACIÓN DE LA CLASE 10 @Entity 11 public class Register extends Model { 12 // ATRIBUTOS 13 private String code; 14 private String cashier; 15 @OneToOne(cascade = CascadeType.ALL) 16 private Store store; 17 @OneToOne(cascade = CascadeType.ALL) 18 private Catalog catalog; 19 @OneToOne(cascade = CascadeType.ALL) 20 private Sale currentSale; 21 22 // CONSTRUCTORES 23 public Register(String code, String cashier, 24 this.code = code; 25 this.cashier = cashier; 26 this.store = store; 27 this.catalog = catalog; 28 this.currentSale = null; 29 } </pre>
<pre> 1 //PAQUETE AL QUE PERTENECE LA CLASE 2 package models; 3 4 //DECLARACIÓN DE LA CLASE 5 public class Item { 6 // ATRIBUTOS 7 private int quantity; 8 private Product product; 9 10 // CONSTRUCTORES 11 public Item(int quantity, Product p) 12 this.quantity = quantity; 13 this.product = p; 14 } </pre>	<pre> 1 //PAQUETE AL QUE PERTENECE LA CLASE 2 package models; 3 4 import javax.persistence.CascadeType; 5 import javax.persistence.Entity; 6 import javax.persistence.OneToOne; 7 import play.db.jpa.Model; 8 9 //DECLARACIÓN DE LA CLASE 10 @Entity 11 public class Item extends Model { 12 // ATRIBUTOS 13 private int quantity; 14 @OneToOne(cascade = CascadeType.ALL) 15 private Product product; 16 17 // CONSTRUCTORES 18 public Item(int quantity, Product p) { 19 this.quantity = quantity; 20 this.product = p; 21 } </pre>

Tabla 10.1. Relaciones de dependencia de la aplicación caso de estudio en modo persistencia.

En la Tabla 10.2 se lista las relaciones de Composición de nuestro modelo, las cuales para que persistan con nuestra base de datos requiere de las palabras claves **cascade = CascadeType.ALL**. Aquí también utilizamos **mappedBy**.

= `CascadeType.ALL`" para que la persistencia se propague para todas las operaciones (borrar, refrescar, combinar) de relaciones entre entidades.

Composición (uno a varios)	
Relación Java	Relación Java con persistencia JPA
<pre> 1 //PAQUETE AL QUE PERTENECE LA CLASE 2 package models; 3 4 //LIBRERÍAS NECESARIAS 5 import java.util.HashMap; 6 import java.util.Map; 7 8 //DECLARACIÓN DE LA CLASE 9 public class Catalog { 10 // ATRIBUTOS 11 private String name; 12 private Map<Long, Product> products; 13 14 // CONSTRUCTORES 15 public Catalog(String name) { 16 this.name = name; 17 this.products = new HashMap<Long, Product>(); 18 } 19 </pre>	<pre> 1 //PAQUETE AL QUE PERTENECE LA CLASE 2 package models; 3 4 //LIBRERÍAS NECESARIAS 5 import java.util.HashMap; 6 import java.util.Map; 7 8 import javax.persistence.CascadeType; 9 import javax.persistence.Entity; 10 import javax.persistence.MapKey; 11 import javax.persistence.OneToMany; 12 13 import play.db.jpa.Model; 14 15 //DECLARACIÓN DE LA CLASE 16 @Entity 17 public class Catalog extends Model { 18 // ATRIBUTOS 19 private String name; 20 @OneToMany(cascade = CascadeType.ALL) 21 @MapKey(name = "code") 22 private Map<Long, Product> products; 23 24 // CONSTRUCTORES 25 public Catalog(String name) { 26 this.name = name; 27 this.products = new HashMap<Long, Product>(); 28 } </pre>
<pre> 1 //PAQUETE AL QUE PERTENECE LA CLASE 2 package models; 3 4 //LIBRERÍAS NECESARIAS 5 import java.util.HashSet; 6 import java.util.Set; 7 8 //DECLARACIÓN DE LA CLASE 9 public class Store { 10 // ATRIBUTOS 11 private int nextSaleNumber; 12 private Set<Sale> completedSales; 13 14 // CONSTRUCTORES 15 public Store() { 16 this.nextSaleNumber = 1; 17 this.completedSales = new HashSet<Sale>(); 18 } 19 </pre>	<pre> 1 //PAQUETE AL QUE PERTENECE LA CLASE 2 package models; 3 4 //LIBRERÍAS NECESARIAS 5 import java.util.HashSet; 6 import java.util.Set; 7 8 import javax.persistence.CascadeType; 9 import javax.persistence.Entity; 10 import javax.persistence.OneToMany; 11 import play.db.jpa.Model; 12 13 //DECLARACIÓN DE LA CLASE 14 @Entity 15 public class Store extends Model { 16 // ATRIBUTOS 17 private int nextSaleNumber; 18 @OneToMany(cascade = CascadeType.ALL) 19 private Set<Sale> completedSales; 20 21 // CONSTRUCTORES 22 public Store() { 23 this.nextSaleNumber = 1; 24 this.completedSales = new HashSet<Sale>(); 25 } </pre>

<pre> 1 //PAQUETE AL QUE PERTENECE LA CLASE 2 package models; 3 4 //LIBRERIAS NECESARIAS 5 import java.util.ArrayList; 6 import java.util.Date; 7 import java.util.List; 8 9 //DECLARACIÓN DE LA CLASE 10 11 public class Sale { 12 // ATRIBUTOS 13 private int number; 14 private Date date; 15 private boolean complete; 16 private double subtotal; 17 private List<Item> items; 18 19 // CONSTRUCTORES 20 public Sale() { 21 this.complete = false; 22 this.items = new ArrayList<Item>(); 23 } 24 } 25 </pre>	<pre> 1 //PAQUETE AL QUE PERTENECE LA CLASE 2 package models; 3 4 //LIBRERIAS NECESARIAS 5 import java.util.ArrayList; 6 import java.util.Date; 7 import java.util.List; 8 import javax.persistence.CascadeType; 9 import javax.persistence.Entity; 10 import javax.persistence.OneToMany; 11 import play.db.jpa.Model; 12 13 //DECLARACIÓN DE LA CLASE 14 @Entity 15 public class Sale extends Model { 16 // ATRIBUTOS 17 private int number; 18 private Date date; 19 private boolean complete; 20 private double subtotal; 21 @OneToMany(cascade = CascadeType.ALL) 22 private List<Item> items; 23 24 // CONSTRUCTORES 25 public Sale() { 26 this.complete = false; 27 this.items = new ArrayList<Item>(); 28 } 29 } </pre>
--	---

Tabla 10.2. Relaciones de composición de la aplicación caso de estudio en modo persistencia.

Una vez que se ha fijado las palabras clave necesarias en cada una de las relaciones, nuestro modelo está listo para poder ser interpretado por el JPA de play framework, una vez interpretado, este modelo se convertirá en tablas con relaciones al estilo de un modelo entidad-relación en nuestra base de datos. Es importante mencionar que los tags o palabras clave utilizadas en este capítulo son únicamente los más elementales en JPA, existen una serie de palabras clave más, que ayudan a mejorar la comunicación entre nuestro modelo de clases y nuestro modelo entidad-relación a través de JPA.

Ejecución de prueba para almacenamiento de datos

Finalmente, para verificar que nuestra capa de negocio este correctamente configurada y revisa la persistencia con nuestra capa de datos, procedemos a ejecutar nuestra función de prueba desarrollada en el controlador “Application.java”. Realizamos una pequeña modificación a la función,

simplemente después de instanciar los objetos necesarios y agregar al vector de productos del catálogo las bebidas requeridas, incluimos la siguiente sentencia “catalog.save();” como muestra la Figura 10.1.

```
1 package controllers;
2
3* import play.*;
4
5 public class Application extends Controller {
6
7     public static void index() {
8
9         testingBusinessTier2();
10
11         render();
12     }
13
14     private static void testingBusinessTier2() {
15         Catalog catalog = new Catalog("Bebidas");
16         catalog.addProduct(1, "Coca-Cola 1.5 lt.", 0.75, "cocaCola.png");
17         catalog.addProduct(2, "Fanta 1.5 lt.", 0.50, "fanta.jpg");
18         catalog.addProduct(3, "Sprite 1.5 lt.", 0.60, "sprite.png");
19         catalog.addProduct(4, "Fioravanti 1.5 lt.", 0.50, "fioravanti.jpg");
20         catalog.save();
21
22         for (Product p : catalog.getProducts().values()) {
23             System.out.println(String.format("| %1$4d | %2$-20s | %3$8.2f |",
24                                         p.getCode(), p.getDescription(), p.getPrice()));
25         }
26     }
27 }
28 }
```

Figura 10.1. Sentencia para ejecutar persistencia en el almacén de datos

Con esta sentencia lo que estamos haciendo es que se confirme la ejecución de la persistencia con nuestra base de datos. Es decir cada vez que a una clase le asignemos un *save()* el JPA de play framework buscará todos los objetos vinculados a esa clase que hasta ese momento se hayan instanciado en memoria y los transformará en información que pueda ser almacenada en tablas de nuestra base de datos. Es la sincronización de los objetos con las tablas de PostgreSQL.

Si toda la configuración está en orden al correr nuevamente el proyecto y actualizar la página de la interfaz, el JPA habrá convertido todas las clases en tablas físicas en nuestra base de datos. Nos dirigimos a nuestro PgAdmin de PostgreSQL y en la sección de tablas del esquema, como

se presenta en la Figura 10.2, verificamos que automáticamente todas las tablas a un estilo de modelo entidad-relación han sido creadas en nuestra base de datos. Al revisar los datos de las tablas Catalog, Product y su relación se puede apreciar que todos los objetos que instanciamos en el controlador han sido almacenados.

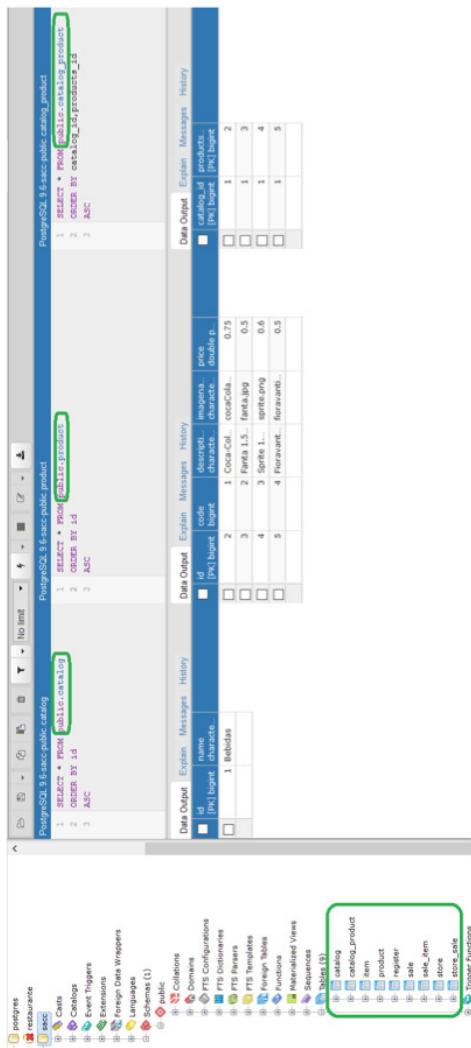


Figura 10.2 Tablas creadas en la Base de datos por el ORM de Play Framework

Capítulo XI

Tags para interacción entre primera y segunda capa

Hasta el momento se ha analizado y configurado las tres capas de trabajo de un ambiente MVC (Modelo Vista Controlador), hemos conocido cual es la responsabilidad individual de cada de ellas en un sistema de información web. Adicional a esto, también se ha mostrado la forma de interrelacionar la segunda capa y la tercera capa, lo único que nos hace falté conocer para completar el aprendizaje del contexto global de nuestro ambiente MVC es saber la forma de interrelacionar la primera capa (capa de presentación) con la segunda capa (capa de negocio).

La importancia que tiene esta interrelación es que a través de la primera capa (interfaz) el usuario puede extraer o enviar la información que requiera, extraerla desde la tercera capa (base de datos) o enviarla hasta la tercera capa. En el capítulo 13 se ha visto la forma como se comunica la segunda con la tercera capa. Para realizar la extracción y envío de datos mencionados desde la interfaz hasta la base de datos, nos hace falta conocer la forma de comunicación entre la primera y segunda capa, una vez que se logre esta comunicación, para llegar a la tercera capa se hará uso del enlace que ya se ha planteado en el anterior capítulo, entre la segunda y tercera capa.

Es decir, la segunda capa es como un intermediario entre la comunicación del usuario con la base de datos, este intermediario se encarga de brindar seguridad y procesamiento al contenido de la información que se quiere transmitir. Este intermediario (segunda capa) generalmente está desarrollado con Programación Orientada a Objetos y su propiedad de encapsulación es uno de los factores que ayudan a fortalecer la seguridad de la información, pero, así como podemos incrementar la seguridad de la aplicación, se incrementa también la complejidad de la estructura y uso del sistema de información web.

Analizado el contexto general y focalizándonos en la comunicación entre la capa de presentación con la capa de negocio, el medio de comunicación

que usa la interfaz para la recepción o envío de información del usuario es a través de los componentes de html, entre los más importantes tenemos input, select, textArea, radio button, check list, button y otros. Estos componentes presentan la información del usuario y por medio de tags o palabras clave propias de play framework, podemos cargar o enviar información desde o hacia la capa de negocio, para que la información sea procesada.

En cambio, por el lado de la capa de negocio son los controladores (controllers) el medio de comunicación con la interfaz, a través de los parámetros de las funciones de los controladores la lógica de negocio puede receptar información. Mientras que para enviar información a la interfaz, se lo hace a través de la palabra clave `render()`, la cual permite el envío de uno o varios objetos de Java, para que puedan ser manipulados en interfaz.

En resumen, la interrelación de estas dos capas consistirá en enviar los datos que se recopilen en modo de lenguaje Html (primera capa) hacia el controlador que tratará la información en modo de lenguaje Java (segunda capa). O viceversa enviar la información que se posea en modo de lenguaje Java (segunda capa) hacia la interfaz que tratará la información en modo de lenguaje Html (primera capa).

En el capítulo 7 se estableció las necesidades que tiene el usuario a nivel de interfaz (primera capa), las cuales fueron las siguientes:

- Visualizar la lista de productos que dispone una tienda.
- Seleccionar un determinado producto y registrar la cantidad que desea consumir.
- Administrar el carro de compras, que acumulará los productos que haya seleccionado.

Mientras que en el capítulo 10, se transformó las necesidades del usuario en funcionalidades del proyecto (segunda capa), definiéndose las siguientes:

- Cargar productos en un catálogo
- Creación de una nueva venta.
- Agregar nuevos ítems al carro de compras.
- Borrar ítems del carro de compras.
- Calcular el total de la compra.

Entonces en este capítulo mostraremos, como interactuar la interfaz que se diseñó para satisfacer las necesidades del usuario con el modelo de negocio que se desarrolló para soportar las funcionalidades del sistema. Es decir, que para cumplir con la meta de este capítulo seguiremos las mismas etapas de trabajo planteadas en el capítulo 7, con la diferencia de que todas las acciones que se realizaron a nivel de interfaz, ahora en este capítulo tendrán comunicación con la capa de negocio para que los datos de interfaz puedan ser procesados y almacenados en base de datos. Así lograremos migrar de una interfaz estática a una interfaz dinámica. Las etapas de trabajo definidas son las siguientes:

- *Visualizar la lista de productos que dispone una tienda:* Se mostrará la lista de productos que tenga el sistema llamando a las funciones que correspondan de la lógica de negocio que extraigan la información desde nuestra base de datos.
- *Seleccionar un determinado producto y registrar la cantidad que desea consumir:* Se creará a nivel de lógica de negocio una nueva venta, para que en ella se pueda agregar los productos que el usuario requiera en el carro de compras, esta información será almacenada en base de datos.
- *Administrar el carro de compras, que acumulará los productos que haya seleccionado:* Permitirá la eliminación desde base de datos de aquellos

productos que el usuario ya no deseé adquirir, así como también el desarrollo de una función que calcule el valor a pagar de la venta de acuerdo a los productos que el usuario haya seleccionado.

Visualizar la lista de productos extrayéndolos desde la capa de negocio

Para el efecto de este requerimiento, tres son los pasos a cumplir para conseguir que nuestro Slider de productos se cargue en interfaz con imágenes e información registrada en la base de datos. La ventaja de que el Slider de productos se cargue con información extraída desde base de datos, radica en que posteriormente con la ayuda de una sección de administración se podría ingresar más productos al almacén de datos del sistema, como el Slider puede leer los productos desde la base de datos podrá cargar todos los productos nuevos que se hayan añadido. De la misma forma en el caso de que en la sección de administración se decida modificar los precios u otras características del producto, automáticamente en interfaz también se podrá apreciar las modificaciones que se hayan hecho a las características del producto.

El primer paso ha sido desarrollar en el controlador “*application.java*” el código necesario para instanciar una nueva tienda (Store) y un nuevo catálogo (Catalog), en el caso del catálogo se requiere agregar la cantidad de productos que se deseé, los mismos que estarán disponibles en la interfaz para su venta. Dentro del controlador en la función “*index*” se tenía una función de prueba que se la realizó para verificar si la configuración de la segunda capa era correcta, como ya no es necesaria la comentamos y escribimos el código que se muestra a continuación:

```

10 public class Application extends Controller {
11
12@   public static void index() {
13
14     // testingBusinessTier2();
15
16     Store store = null;
17     if (!Store.findAll().isEmpty())
18         store = (Store) Store.findAll().get(Store.findAll().size() - 1);
19     else {
20         store = new Store();
21         store.save();
22     }
23
24     Catalog catalog = null;
25     if (!Catalog.findAll().isEmpty())
26         catalog = (Catalog) Catalog.findAll().get(
27             Catalog.findAll().size() - 1);
28     else {
29         catalog = new Catalog("bebida");
30         catalog.addProduct(1, "Coca-Cola 1.5 lt.", 0.75, "cocaCola.png");
31         catalog.addProduct(2, "Fanta 1.5 lt.", 0.50, "fanta.jpg");
32         catalog.addProduct(3, "Sprite 1.5 lt.", 0.60, "sprite.png");
33         catalog.addProduct(4, "Fioravanti 1.5 lt.", 0.50, "fioravanti.jpg");
34
35         catalog.save();
36     }
37     render(catalog, store);
38 }
```

Las dos primeras cajas de texto resaltadas sirven para validar la existencia de una tienda y catálogo ya existente en la base de datos, esto con el fin de que se repitan los datos cada vez que se inicie el sistema. Y en la última caja de texto resaltada tenemos `render(catalog, store)` es la función que está dando la orden de buscar un *html* con el nombre de la función en la capa de presentación. Es decir, en este caso como nuestra función se llama *index*, el render realiza la búsqueda de un archivo “*index.html*” en la vista y envía a la interfaz los objetos “*catalog*” y “*store*” que tiene como parámetros, con todos sus datos.

El segundo paso es desarrollar en interfaz un lazo repetitivo con tags o palabras claves de play framework, que nos permita recorrer todos los productos que se han enviado en el objeto catalogo (catalog). Ya no se tendrá que imprimir en interfaz imagen por imagen como lo hicimos en el Capítulo 7, ahora el lazo repetitivo extraerá todos los productos que existan en la base de datos y los imprimirá en pantalla. Para esto, en

la sección 3 de nuestro index.html, adecuamos el código de la siguiente manera:

```
143@    <div id="section3">
144        <h1>Catálogo de Productos</h1>
145@        <div id="slider1">
146            <div class="thumbelina-but horiz left"></div>
147@        <ul>
148            #{list items:catalog.products.values(), as:'product'}
149@            <li>
150                
153            </li>
154            #!(list)
155        </ul>
156        <div class="thumbelina-but horiz right"></div>
157    </div>
158</div>
```

Los cuadros marcados, superior e inferior son palabras clave propias de play framework que permiten la ejecución de un lazo repetitivo, el vector a recorrer son los productos que se los está extrayendo de catálogo, objeto que fue enviado a la interfaz a través del *render()*. Los cuadros intermedios marcados, también son palabras clave que permiten retornar el valor que contienen los atributos del objeto implicado, en este caso del objeto producto.

Y como tercer paso, después de recordar que la función JavaScript “*loadImage(this)*” nos ayuda a cargar en la sección 4 los detalles del producto que el usuario haya seleccionado (ver Capítulo 7), ahora vamos adecuar esta función para que también trabaje con información extraída desde la capa de negocio. Para esto, es necesario usar el mismo lazo repetitivo que ayudará a validar la selección de los productos del catálogo. A continuación, las adecuaciones necesarias para la función “*loadImage(this)*”:

```

104     function loadImage(img) {
105         #{list items:catalog.products.values(), as:'product'}
106         if(${product.code}==${img.id}){
107             document.getElementById("imageSelected").src="/public/images/${product.imageName}";
108             document.getElementById("code") .value="${product.code}";
109             document.getElementById("desc") .value="${product.description}";
110             document.getElementById("price") .value="${product.price}";
111         }
112     }
113 }

```

De la misma manera que el anterior código, los cuadros marcados representan la declaración del lazo repetitivo que recorre el vector de productos que tiene el catálogo. Y los textos subrayados representan la llamada a los valores de los atributos del objeto producto, los mismos que son presentados en los componentes html que se han definido para presentar la información del detalle del producto.

Para revisar que nuestro código está funcionando de manera correcta, ejecutamos la aplicación y apreciaremos en la interfaz que al momento de cargar el detalle de un producto la información presentada será la misma que se cargó en la capa de negocio, tal como lo muestra la Tabla 11.1

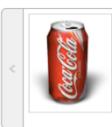
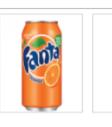
Catálogo de Productos			Detalle del Producto													
  			 <table border="1"> <thead> <tr> <th>Código</th><th>Descripción</th><th>Precio</th><th>Cantidad</th><th>Acción</th></tr> </thead> <tbody> <tr> <td>3</td><td>Sprite 1.5 lt.</td><td>0.6</td><td></td><td>Agregar</td></tr> </tbody> </table>				Código	Descripción	Precio	Cantidad	Acción	3	Sprite 1.5 lt.	0.6		Agregar
Código	Descripción	Precio	Cantidad	Acción												
3	Sprite 1.5 lt.	0.6		Agregar												
<pre> Catalog catalog = new Catalog("Bebidas"); catalog.addProduct(1, "Coca-Cola 1.5 lt.", 0.75, "cocaCola.png"); catalog.addProduct(2, "Fanta 1.5 lt.", 0.50, "fanta.jpg"); catalog.addProduct(3, "Sprite 1.5 lt.", 0.60, "sprite.png"); catalog.addProduct(4, "Fioravanti 1.5 lt.", 0.50, "fioravanti.jpg"); catalog.save(); </pre>																

Tabla 11.1. Información de la capa de negocio reflejada en la capa de presentación

En el cuadro superior marcado se presenta los datos de un producto que ha sido desplegado en interfaz (primera capa), mientras que en el cuadro inferior se presenta el momento en que el producto es instanciado (segunda capa). Si nos fijamos en los valores de los productos ambos coinciden, lo que quiere decir que el producto que se instanció en la capa de negocio se lo presentó en la capa de presentación, siendo esta la prueba que muestra la interrelación entre la primera capa y la segunda capa.

Registrar el producto seleccionado y su cantidad en un carro de compras

Después de que en la sección 4 se presenta el detalle del producto seleccionado, el usuario tiene un campo en blanco para ingresar la cantidad que desea adquirir de dicho producto, ingresado este valor el usuario puede hacer click en el botón agregar para registrar este producto en su carro de compras.

Para desarrollar esta funcionalidad primero se tienen que escribir en el controlador “Application.java” el código necesario para crear una nueva venta (Sale), puesto que los ítems que el usuario elija tienen que ser vinculados a una venta. Para esto se añadió en la función *index()* del controlador la creación de la central de registros de ventas y en ella se añadió una venta, también se creó en el controlador una nueva función denominada “*addItem()*”. Esta función es la que recibe la acción del botón agregar y recibe a través de sus parámetros, todos los datos del producto que el usuario desea ingresar como ítem en su carro de compras. Para desarrollar estas funcionalidades adecuamos el código de nuestro controlador *Application.java* de la siguiente manera:

```

27         Catalog.findAll().size() - 1);
28     else {
29         catalog = new Catalog("bebida");
30         catalog.addProduct(1, "Coca-Cola 1.5 lt.", 0.75, "cocaCola.png");
31         catalog.addProduct(2, "Fanta 1.5 lt.", 0.50, "fanta.jpg");
32         catalog.addProduct(3, "Sprite 1.5 lt.", 0.60, "sprite.png");
33         catalog.addProduct(4, "Fioravanti 1.5 lt.", 0.50, "fioravanti.jpg");
34         catalog.save();
35     }
36     // PARA AGREGAR PRODUCTO EN CARRO DE COMPRAS
37     Register register = null;
38     if (!Register.findAll().isEmpty())
39         register = (Register) Register.findAll().get(
40             Register.findAll().size() - 1);
41     else {
42         register = new Register("Fulano D'Tal", "SFU3773", store, catalog);
43         register.makeNewSale();
44         register.save();
45     }
46     render(catalog, store, register);
47 }
48
51@ public static void addItem(long code, int qty, long catalog_, long store_,
52    long register_) {
53
54     Catalog catalog = Catalog.findById(catalog_);
55     Store store = Store.findById(store_);
56     Register register = Register.findById(register_);
57
58     register.enterItem(code, qty);
59     register.save();
60
61     index();
62 }

```

El cuadro superior marcado, corresponde al código necesario para crear una nueva central de registros o recuperar la almacenada en caso de existir, también posee la creación de la venta, dónde se almacenará los ítems que el usuario seleccione. En el cuadro inferior se tiene la lógica de programación para que un nuevo ítem sea agregado a la venta o carro de compras, finalizada la lógica se retorna a la interfaz del *index()* principal.

Luego de preparar en el controlador la lógica de negocio necesaria para que se procese la información de estas funcionalidades planteadas, nos dirigimos al *index.html*. En la sección 4 (detalle del producto) debemos preparar un formulario para que a través del “*submit*” del botón agregar, la aplicación pueda dirigirse al controlador llevando consigo toda la

información de los componentes *html* existentes entre el formulario. Adicional es necesario agregar “*inputs*” (cajas de texto) de tipo “*hidden*” (oculto), que guarden la información de los objetos: catalog, store y register que han sido traídos a la interfaz y ahora necesitan ser enviados al controlador a nuestra nueva función *additem()*. Para esto, en el archivo *index.html* sección 4, realice las siguientes adecuaciones de código:

```

160<div id="section4">
161    <h1>Detalle del Producto</h1>
162    <form id="Form" action:@Application.addItem(),method:'POST', enctype:'multipart/form-data'>
163        <table border="1" style="margin: 10px;">
164            <tr align="center">
165                <td colspan="6"></td>
167            </tr>
168            <tr align="center">
169                <td>Código</td>
170                <td>Descripción</td>
171                <td>Precio</td>
172                <td>Cantidad</td>
173                <td>Acción</td>
174            </tr>
175            <tr align="center">
176                <td><input id="code" name="code" type="text" size="4"
177                    style="background: #eeeeee;"></td>
178                <td><input id="desc" name="desc" type="text" size="8"
179                    style="background: #eeeeee;"></td>
180                <td><input id="price" name="price" type="text" size="4"
181                    style="background: #eeeeee;"></td>
182                <td><input id="qty" name="qty" type="text" size="4"></td>
183                <td>
184                    <input id="add" name="add" value="Agregar" type="submit" size="4">
185                    <input id="catalog_" name="catalog_" type="hidden" size="2" value="$catalog.id">
186                    <input id="store_" name="store_" type="hidden" size="2" value="$store.id">
187                    <input id="register_" name="register_" type="hidden" size="2" value="$register.id">
188                </td>
189            </tr>
190        </table>
191    </form>
192</div>
```

Los cuadros superior e inferior son tags o palabras clave de play framework que nos ayudan a declarar un formulario, el mismo que dispara su acción en cuanto un *submit* es ejecutado. En este caso cuando el botón *submit* agregar es presionado nuestro formulario se dirigirá a la función *addItem()* de nuestro controlador, llevando consigo los valores que contengan los componentes *html* de nuestra interfaz. El cuadro interno posee los *inputs hidden* que guardan los *ids* de los objetos de *catalog, store y register*, valores necesarios para que en el controlador se realice las operaciones necesarias para agregar un nuevo ítem a la venta.

Como último paso en la sección 5 de nuestro archivo *index.html* necesitamos declarar un lazo repetitivo para recorrer el vector de ítems que tiene nuestra venta. Puesto que cada vez que el usuario ingrese un ítem, la sección 5 debe registrar y presentar los ítems que se están guardando en nuestra venta (Sale). Para esto adecuamos nuestra sección 5, de acuerdo al siguiente código:

```

194@ <div id="section5">
195    <h1>Carro de Compras</h1>
196    <table border="1" style="margin: 5px;">
197        <tr align="center">
198            <td>Descripción</td>
199            <td>Precio</td>
200            <td>Cantidad</td>
201            <td>Subtotal</td>
202            <td>Acción</td>
203        </tr>
204
205        #{list items:register.currentSale.items, as:'item'}  

206        <tr align="center">
207            <td><input id="desc1" name="desc1" type="text" size="8"  

208                style="background: #eeeeee;" value="${item.product.description}"></td>
209            <td><input id="price1" name="price1" type="text" size="4"  

210                style="background: #eeeeee;" value="${item.product.price}"></td>
211            <td><input id="qty1" name="qty1" type="text" size="4" value="${item.quantity}"></td>
212            <td><input id="subtol" name="subtol" type="text" size="4"  

213                style="background: #eeeeee;"></td>
214            <td><input id="delete1" onClick="deleteItem();"  

215                name="delete1" value="Eliminar" type="submit" size="4"></td>
216        </tr>
217        #({/list})
218        <tr align="center">
219            <td colspan="3" align="right">TOTAL:</td>
220            <td><input id="price2" name="price2" type="text" size="4"  

221                style="background: #eeeeee;"></td>
222            <td><input id="edit2" name="edit2" onClick="endSale();"  

223                value="TOTAL COMPRA" type="submit" size="4"></td>
224        </tr>
225    </table>
226 </div>
```

En los cuadros superior e inferior tenemos las palabras clave para declarar nuestro lazo repetitivo que recorre los ítems que tiene la venta, este lazo repetitivo nos permite repetir la fila entre la que se encuentra, tantas veces como ítems existan. Los textos subrayados son el llamado a los valores de los atributos de los objetos ítem y producto que se han registrado en el carro de compras.

Eliminar ítems del carro de compras y cálculo del total de la compra

Luego de que el usuario ya puede agregar ítems a su compra, es necesario como parte de su administración del carro de compras, que también pueda eliminar ítems que no deseé. Además se busca que de acuerdo a los ítems que va adquiriendo el usuario en su carro de compras, se pueda calcular el subtotal que representa cada uno de ellos y el total a pagar en toda la compra.

Lo primero a realizar es preparar el controlador y las clases del modelo que sean pertinentes, con los métodos necesarios para que la capa de negocio soporte estas funcionalidades. En el controlador *Application.java* es necesario adecuar la siguiente función:

```
64@    public static void deleteItem(long register_, long deleteId) {  
65@  
66@        Register register = Register.findById(register_);  
67@        register.getCurrentSale().deleteItem(deleteId);  
68@        register.save();  
69@  
70@        index();  
71@    }
```

Esta nueva función del controlador receptará a través de sus parámetros el identificador de la central de registros y el ítem que desea eliminar el usuario. Posteriormente se busca en la central de registros la venta abierta por el usuario y se llama a la función de eliminación del ítem. El detalle de la función de eliminación se encuentra en la función creada en la clase *Sale.java*, en esta clase es necesario adecuar el siguiente código que eliminará el ítem no deseado por el usuario en la venta:

```
78@    public void deleteItemFromSale(long deleteId) {  
79@        this.getItems().remove(Item.findById(deleteId));  
80@    }
```

Y para calcular el subtotal que representa el ítem elegido por el usuario de acuerdo a la cantidad ingresada del mismo, es necesario crear el cálculo del subtotal que será la multiplicación entre el precio del producto y la cantidad. La adecuación de este método se lo tendrá que hacer en la clase Item.java, de la siguiente manera:

```
46     // FUNCIONES VARIAS
47     public double calculateSubTotal() {
48         return this.product.getPrice() * this.quantity;
49     }
```

Después de que hemos creado los métodos necesarios en nuestra lógica de negocio para soportar las funcionalidades propuestas, en nuestro html de la interfaz es necesario crear un formulario cuya acción se direccione a la función *deleteItem()* de nuestro controlador. Este formulario debe ser ubicado en la sección 5 que es la poseedora de los botones eliminar y los identificadores de los ítems. Hay que recordar que la acción del formulario se dispara con un *submit* de un botón y en este caso nos corresponde asignarle esta característica al botón eliminar.

En el botón eliminar también se ha fijado la declaración de una función JavaScript que nos servirá para obtener el identificador del ítem que el usuario desea borrar, así como también le pedirá al usuario una confirmación antes de borrar el ítem. Adicional mencionamos que en inputs (cajas de texto) hidden (oculto) se envían los identificadores de los objetos de la central de registros (*register_*) y el ítem (*deleteId*) a eliminar. Para efectuar estas funcionalidades, en la sección 5 de nuestro archivo index.html adecuar el siguiente código:

```

203<div id="section5">
204<h1>Carro de Compras</h1>
205<#form id: Form, action:@application.deleteItem(), method:'Post', enctype:'multipart/form-data'>
206<table border="1" style="margin: 5px;">
207<tr align="center">
208<td>Descripción</td>
209<td>Precio</td>
210<td>Cantidad</td>
211<td>Subtotal</td>
212<td>Acción</td>
213</tr>
214<#list items:register.currentSale.items as: item'>
215<tr align="center">
216<td><input id="desci" name="desci" type="text" size="8" style="background: #eeeeee; value=$item.product.description!></td>
217<td><input id="price" name="price" type="text" size="4" style="background: #eeeeee; value=$item.product.price!></td>
218<td><input id="qty" name="qty" type="text" size="4" style="background: #eeeeee; value=$item.quantity!></td>
219<td><input id="subtot" name="subtot" type="text" size="4" value=$item.calculatesubtotal!></td>
220<td><input id="delete1" name="delete1" type="button" value="Delete" style="background: #eeeeee; size=4;"></td>
221<td><input id="edit1" name="edit1" type="button" value="Edit" style="background: #eeeeee; size=4;"></td>
222<td><input id="register_id" name="register_id" type="hidden" value=$register.id!></td>
223<td><input id="saleTotal" name="saleTotal" type="text" size="4" style="background: #eeeeee; border: 1px solid black; width: 100px;"></td>
224<td align="right" colspan="3" style="background: #eeeeee; text-align: right; padding-right: 10px; vertical-align: bottom;">
225<#list items:register.currentSale.items as: item'>
226<td align="center" style="background: #eeeeee; text-align: center; width: 100px; vertical-align: bottom;">
227<#list items:register.currentSale.items as: item'>
228<td align="center" style="background: #eeeeee; text-align: center; width: 100px; vertical-align: bottom;">
229<td align="right" style="background: #eeeeee; text-align: right; vertical-align: bottom;">
230<td align="right" colspan="3" style="background: #eeeeee; text-align: right; vertical-align: bottom; padding-right: 10px;">
231<input id="deleteId" name="deleteId" type="hidden" size="21" value="1"/>
232<input id="register_" name="register_" type="hidden" size="2" value="1"/>
233<input id="saleTotal" name="saleTotal" type="text" size="4" style="background: #eeeeee; border: 1px solid black; width: 100px;">
234<input id="edit2" name="edit2" type="button" value="Edit" style="background: #eeeeee; border: 1px solid black; width: 100px;">
235<input id="registerCon" name="registerCon" type="button" value="Contra" style="background: #eeeeee; border: 1px solid black; width: 100px;">
236</td>
237<td><input id="registerCon" name="registerCon" type="button" value="Contra" style="background: #eeeeee; border: 1px solid black; width: 100px;"></td>
238<td><input id="edit2" name="edit2" type="button" value="Edit" style="background: #eeeeee; border: 1px solid black; width: 100px;"></td>
239<td><input id="registerCon" name="registerCon" type="button" value="Contra" style="background: #eeeeee; border: 1px solid black; width: 100px;"></td>
240</td>
241</tr>
242</table>
243</div>

```

Con respecto al cálculo del subtotal de los ítems, en la línea 223 se hace un llamado a la función del ítem (capa de neogio) *calcularSubtotal()* que es la encargada de efectuar la operación necesaria y retornar el valor del resultado. Para el cálculo del valor de la compra total, se ha declarado una función Javascript *endSale()* que se encargará de leer todos los subtotales, sumarlos y presentar su total en el input (caja de texto) del total de la compra.

Finalmente se desarrolla las funciones JavaScript *deleteItem()* y *endSale()*, la primera fijará en una caja de texto oculta, el identificador del ítem que el usuario desea eliminar, para que este sea enviado a la función del controlador y el ítem que corresponda sea eliminado del carro de compras. Previa a esta eliminación el sistema emitirá una confirmación para que el usuario corrobore la eliminación. Dentro de la segunda función de JavaScript, se usa palabras clave de play framework para declarar un lazo repetitivo y recorrer los ítems de una venta, esto para acumular en la variable *acuTotal* todos los subtotales de los ítems y así conseguir el total a pagar de la compra. En la cabecera de nuestro archivo index.html adecuamos las siguientes funciones JavaScript.

```
function deleteItem(deleteId) {
    document.getElementById("deleteId").value=deleteId.id;
    confirm('Está seguro que desea borrar este ítem?');
}

function endSale() {
    acuTotal=0;
    #{list items:register.currentSale.items, as:'item'}
        subtotal=${item.quantity}*${item.product.price};
        acuTotal=acuTotal + subtotal;
    #{/list}
    document.getElementById("saleTotal").value=acuTotal;
}
```

En la función *endSale()* se puede apreciar que la sinergia entre el lenguaje JavaScript y Play Framework es factible. Haciendo un recuento, es importante señalar que en nuestra interfaz o primera capa hemos conseguido la sinergia de cinco lenguajes diferentes, estos han sido:

Html, CSS, JavaScript, Jquery y Play Framework. Cada uno con su rol particular, que aporta a un más eficiente y mejor funcionamiento de nuestra página web.

Para verificar la correcta comunicación entre las capas de nuestro sistema de información web, en la Tabla 11.2 podemos apreciar en la interfaz de nuestra aplicación (primera capa), un carro de compras con un listado de tres ítems con sus características respectivas. Y en la base de datos (tercera capa) podemos apreciar que la tabla que almacena los ítems tiene el mismo número de elementos que nuestro carro de compras, si desea ver el detalle de las características siga las relaciones que existe con las otras tablas.

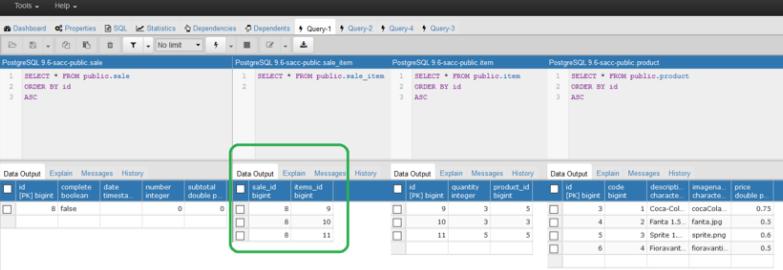
Datos en Primera Capa o Capa de Presentación																																
Ingeniería en Informática y Sistemas Computacionales																																
Catálogo de Productos 	Detalle del Producto  <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>Código</th> <th>Descripción</th> <th>Precio</th> <th>Cantidad</th> <th>Acción</th> </tr> </thead> <tbody> <tr> <td>3</td> <td>Sprite 1.5 lt.</td> <td>0.6</td> <td></td> <td>Agregar</td> </tr> </tbody> </table>	Código	Descripción	Precio	Cantidad	Acción	3	Sprite 1.5 lt.	0.6		Agregar	Carro de Compras <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>Descripción</th> <th>Precio</th> <th>Cantidad</th> <th>Subtotal</th> <th>Acción</th> </tr> </thead> <tbody> <tr> <td>Sprite 1.5 lt.</td> <td>0.6</td> <td>3</td> <td>1.799999</td> <td>Eliminar</td> </tr> <tr> <td>Coca-Cola 1 l.</td> <td>0.75</td> <td>3</td> <td>2.25</td> <td>Eliminar</td> </tr> <tr> <td>Sprite 1.5 lt.</td> <td>0.6</td> <td>5</td> <td>3.0</td> <td>Eliminar</td> </tr> </tbody> </table> TOTAL: 7.05 TOTAL COMPRAS	Descripción	Precio	Cantidad	Subtotal	Acción	Sprite 1.5 lt.	0.6	3	1.799999	Eliminar	Coca-Cola 1 l.	0.75	3	2.25	Eliminar	Sprite 1.5 lt.	0.6	5	3.0	Eliminar
Código	Descripción	Precio	Cantidad	Acción																												
3	Sprite 1.5 lt.	0.6		Agregar																												
Descripción	Precio	Cantidad	Subtotal	Acción																												
Sprite 1.5 lt.	0.6	3	1.799999	Eliminar																												
Coca-Cola 1 l.	0.75	3	2.25	Eliminar																												
Sprite 1.5 lt.	0.6	5	3.0	Eliminar																												
Datos en Tercera Capa o Capa de Datos																																
																																

Tabla 11.2. Correlación de datos entre primera y tercera capa

Es decir, los datos que tenemos en interfaz son los mismos datos que tenemos en la base de datos, de esta manera demostramos la correlación que existe entre todas las capas que conforman nuestro sistema de información web. Es así como llegamos al final de nuestro caso de estudio ejemplo, que puede ser usado como una base académica que fortalezca su aprendizaje y lo encamine al desarrollo de sistemas de información web más robustos.

Sección 6: Casos de éxito de la Universidad Técnica de Cotopaxi

En los últimos años el estudio de la estructura de trabajo de Play Framework se lo ha estado fomentando en las aulas de la Universidad Técnica de Cotopaxi, buscando aprovechar sus cualidades de ser una herramienta con licencia libre en el desarrollo de software ágil. Se diferencia de otros frameworks empresariales por su baja complejidad al momento de configurarlos, es decir brinda sencillez y flexibilidad al momento de usarlo. Por estas razones se ha considerado su uso pedagógico en las aulas, ya que permite esclarecer de forma práctica varios de los conceptos de un sistema de información web en tres capas apoyado en un patrón de arquitectura de software MVC (Modelo Vista Controlador). Estos conceptos de la estructura de una aplicación web serán necesarios en la vida profesional de los estudiantes en el momento que tengan que trabajar con frameworks empresariales más robustos.

Sin embargo, es importante mencionar que el uso de Play Framework no únicamente ha sido pedagógico, sino que también a nivel mundial esta herramienta de desarrollo ágil es usada para generar software comercial. Instituciones públicas y privadas hacen uso de esta estructura de software web para realizar aplicaciones con alta transaccionalidad y manejo de grandes volúmenes de información, muchas de estas aplicaciones se han convertido en sistemas emblemáticos de ciertas instituciones y han sido usados como base de sus procesos de negocio.

En la Universidad Técnica de Cotopaxi el uso de Play Framework, también ha permitido obtener algunos logros, en la presente sección se describe dos casos de éxito que se han presenciado en el contexto de la Universidad y que han sido fruto del trabajo realizado en las aulas. Con el uso de este framework la Universidad gana un concurso nacional de software en la ESPOCH (Escuela Superior Politécnica de Chimborazo), así también varios estudiantes innovan aplicaciones para incorporarlas en el mercado y ofertar servicios a instituciones públicas y privadas.

Capítulo XII

**Primer lugar en
concurso nacional de
software**

En los proyectos nuevos de software, es frecuente encontrar que un proyecto “dócil” se pueda convertir en un “verdadero monstruo” que todo lo “devore” y que infunda el mismo temor cual criatura monstruosa. En este sentido, Fred Brooks (1995, p. 180-181) hizo el comparativo de un proyecto con el “hombre lobo” de la leyenda, para el cual, se anhela encontrar la mágica “bala de plata” (silver bullet, en inglés) que finalmente lo someta, lo mande a descansar. Poco a poco, los parches del software (patches) terminan convirtiendo a un sistema relativamente “pulcro” (que en realidad es raro), en un verdadero spaghetti de elementos tecnológicos, en un auténtico y único “Frankenstein”, como suele referirse irónicamente a ellos. A ese verdadero “monstruo”, ya ni sus creadores “conocen a detalle” pues sus secretos no se encuentran en los manuales o documentos, si es que alguna vez estuvieron ahí, detalles que son indispensables para “seguir manteniéndolo con vida”.

Recientemente estuve navegando por internet, en ocasiones se hace tediosa la cosecha adecuada de información, la infoxicación gana cada día más terreno, se hace evidente un colapso de información, eventualmente el desarrollo de habilidades informacionales se hace necesario, pero bien, en este caso los modestos conocimientos sobre como acarrear información me llevó a la URL <https://www.propiedadintelectual.gob.ec>. Fue fantástico encontrar un título que me llenó de orgullo y satisfacción este se denominaba así: “Ecuador tiene talentos para crear Software Libre”. (ver Figura 12.1)



Figura 12.1. Estudiantes participantes del concurso DevsuCodeJam 2014.

Ciertamente nuestros jóvenes y otros que ya no somos tan jóvenes podemos ser capaces de crear, esa palabra maravillosa que refleja el poder de ser creativo, demuestra que los ecuatorianos tenemos potencial para crear muchas cosas y dentro de ese universo, el desarrollo de software es un eslabón fundamental de la cadena del progreso de las Tecnologías en nuestro país.

Esta noticia recogía aspectos relacionados con el Concurso Nacional de Programación (DevsuCodeJam 2014) lanzado por la empresa Devsu y la Escuela Politécnica Nacional (EPN). La premiación fue dirigida por Andrés Larco, Coordinador Proyecto Imagen Sistemas FIS-EPN y por César Salazar, Gerente de Devsu. Al evento asistieron los concursantes provenientes de diferentes provincias del país. Por su parte, la plataforma MINKA del IEPI fue uno de los auspiciantes del concurso en calidad de plataforma de promoción de software libre en Ecuador.

Aquí se muestra la verdadera importancia de poder manejar herramientas en la cual no haya que realizar un pago por concepto de licencias, debemos estar consciente que software libre no quiere decir software *free* o gratuito, hay que entender que ambas categorías tienen un significado que las diferencian a cada cual.

La participación en el concurso estuvo conformada por estudiantes de los últimos niveles y recién egresados de las carreras de Ingeniería de Sistemas, Informática, Computación y afines. El día del concurso los participantes se dieron cita en las instalaciones de la EPN, donde durante toda una mañana se dedicaron a resolver problemas de programación.

Es hermoso poder leer noticias de este tipo, así mismo muchas universidades celebran de forma local e interna, muchos concursos, proyectos orientados a desarrollar software, sistemas de todo tipo, orientados a informatizar procesos en diversas entidades públicas y privadas, la alternativa de software libre brindan la posibilidad de tener la tecnología que se necesita al alcance de un clic, y nuestros jóvenes estudiantes saben aprovechar esta alternativa.

Otro caso que podemos exponer está relacionado con la participación de los estudiantes de la carrera de Ingeniería en Informática y Sistemas Computacionales de la Universidad Técnica de Cotopaxi en el BARCAMP 2015. Los estudiantes participaron de la socialización de dos eventos organizados por la Pontificia Universidad Católica del Ecuador (PUCE) campus Ambato. El viernes 27 de Noviembre 2015 se realizó el BARCAMP 2015, este encuentro fue gratuito y consistió en la difusión de proyectos y charlas magistrales, enfocadas al desarrollo de software y al intercambio de experiencias sobre programación.

Otro de los eventos organizados por la Universidad Católica es el Taller de Tecnología e Informática, que se llevó a cabo el sábado 28 de Noviembre del 2015, y tuvo un valor de 50 dólares. El curso estuvo dividido en cuatro talleres de 10 horas cada uno; el evento fue de gran importancia porque nutrió a los participantes del que hacer investigativo en la rama del almacenamiento en internet, Introducción al lenguaje de programación KINET 2.0, Redes ópticas y Desarrollo de Aulas virtuales.

Estos ejes temáticos contribuyen al desarrollo de nuevas habilidades para los estudiantes, en cuanto a programación, diseño y administración de base de datos.

Este tipo de seminarios permiten fortalecer el perfil profesional, ampliar su campo de acción y mejorar las oportunidades laborales de los futuros ingenieros en sistemas.

Otra de las ramas en la cual los estudiantes de la carrera de Ingeniería Informática y Sistemas Computacionales ha incursionado es en el desarrollo de sistemas aplicando tecnologías arduino (hardware y software libre), ha realizado concursos internos de preselección de vehículos denominados seguidores en línea, para participar en un evento nacional llamado CER 2016 (Concurso Ecuatoriano de Robótica), actividad que se llevó a cabo en la Universidad Técnica Equinoccial (ver Figura 12.2).



Figura 12.2. Competencia de carros seguidores de línea.

Es importante reconocer que la Universidad Técnica de Cotopaxi conjuntamente con la carrera de Ingeniería Informática y Sistemas Computacionales, han desarrollado importantes aportes en relación al diseño y construcción de vehículos seguidores en línea y robots de

combate que entraron a una preselección para calificar a las mejores maquinas, que participarían en el concurso Nacional CER 2016.

Los méritos obtenidos por el Alma Mater de Cotopaxi son muchos, en las IX JORNADAS DE INGENIERÍA DE SOFTWARE Y III CONCURSO DE DESARROLLO DE SOFTWARE DEL ECUADOR ESPPOCH 2015, los estudiantes de Ingeniería en Sistemas ganaron el concurso Nacional de Desarrollo de Software en la provincia de Chimborazo, obteniendo así el primer lugar, los estudiantes con la excelencia que los caracterizó pusieron en alto la calidad de su formación. El 22 de junio del 2015 los estudiantes Diego Patricio Mosquera Velasquez, Luís René Quisaguano Collaguazo y Edgar Wladimir Acurio Chimba, todos estudiantes del noveno ciclo de la carrera de Ingeniería en Informática y Sistemas Computacionales en ese entonces, participaron en el concurso nacional de desarrollo de software en la Escuela Superior Politécnica de Chimborazo.

La participación de varias universidades de la región y el país y sus excelentes estudiantes no fue una barrera que obstaculizara el buen desempeño de los participantes por la UTC, siendo las 10H00 entraron a demostrar sus conocimientos y habilidades en la resolución de un problema planteado por el comité organizador del concurso. Los minutos pasaron y el desespero por conocer los resultados se hacía evidente por el colectivo que acompañó a los protagonistas, luego de dos horas en angustia y nerviosismo culminan con su labor, cabe destacar que los evaluadores fueron investigadores de la rama de varios países como Colombia, España y Ecuador.

Tal fue la sorpresa cuando evidentemente iniciaron las premiaciones, dando los resultados del tercer y segundo lugar, muy nerviosos y angustiados por no escuchar el nombre de la UTC entre esos lugares, pasaron unos segundo cuando efectivamente el nombre de nuestra

prestigiosa UTC retumbó, al mencionarse “...y el primer lugar es para la UTC...”.

Ciertamente cuando se pone empeño, dedicación y amor a toda una labor se pueden obtener innumerables resultados, así ha sido en el caso de la carrera de Ingeniería en Informática y Sistemas Computacionales de la universidad Técnica de Cotopaxi. Los estudiantes con la dirección y asesoramiento de sus docentes pueden llegar lejos.

Hoy en día esos mismos estudiantes que han ganado numerosos premios en diferentes concursos, son emprendedores que han logrado escalar en la vida profesional, uno de ellos es un excelente programador de una de las empresas de desarrollo de software de Cotopaxi denominada Fenix, otro labora en CNT y el otro creó la empresa Rebian, dedicada al desarrollo de software de todo tipo, ese es el resultado de la esperanza y del conocimiento consolidado de la labor profesional de nuestros docentes en nuestra querida Alma Mater.

Los resultados son numerosos y en distintas disciplinas, otro ejemplo fue evidenciado en el hall del bloque A de la Universidad, se presenció nuevamente el desarrollo de tecnologías en la construcción de insectos de velocidad, son proyectos elaborados por los estudiantes en cuanto a lo que se relaciona a robótica. El jueves 21 de julio del 2016 a las 11:00 de la mañana se inició un evento con la participación de alrededor de 120 estudiantes de la carrera de Informática Sistemas Computacionales. Esto permite demostrar a través de práctica los conocimientos que han adquirido durante su carrera, estos trabajos tienen como finalidad salir a los concursos a nivel nacional.

Jorge Iguamba uno de los participantes y ganador de una de las competencias, manifestó que diseñó un insecto que simula a una araña, tardó un mes en construirla, para ello utilizó material reciclado

como: palillos de helado, placa arduino y un control, la expectativa fue ganar, su reto lo orientaría a mejorar su trabajo para participar con otras instituciones.

En la Universidad Técnica de Cotopaxi, la carrera de Ingeniería en Sistemas, realiza capacitaciones a trabajadores con discapacidad de la empresa de Provefrut, cuyo objetivo es fortalecer el conocimiento en sistemas informáticos, básica media y avanzada (ver Figura 12.3).



Figura 12.3. Estudiante de Ingeniería en Informática y Sistemas Computacionales capacitando.

Estas capacitaciones fueron dirigidas por estudiantes de la carrera de Ingeniería en Sistemas, cumpliendo las horas de vinculación, donde los mismos imparten sus conocimientos adquiridos.

El objetivo de estas capacitaciones es trasladar la tecnología hacia los sectores sociales, incrementando sus conocimientos informáticos, mismos que son útiles y esenciales en la vida diaria.

El Servicio de Integración Laboral el (SIL) para personas con discapacidad tiene convenio con la universidad, cuyo propósito

es aportar los conocimientos de los estudiantes con la ciudadanía, plasmando el desarrollo profesional y académico de los estudiantes a través de la enseñanza, la capacitación de sistemas informáticos a trabajadores de Provefrut, personas que desean superarse y que muestran que la discapacidad no es impedimento para aprender.

Conocer estos servicios informáticos, es importante en la vida diaria por que genera oportunidades ya sea en lo laboral o personal brindando la oportunidad de obtener un ascenso ya que la discapacidad no es impedimento de superación.

Para formar profesionales de calidad, es necesario que los estudiantes inicien su participación desde sus aulas, compartiendo sus conocimientos con la ciudadanía.

Para terminar y citando al mismo autor del primer párrafo de este epígrafe y sin exagerar, puedo afirmar que toda organización moderna tiene su propio “monstruo” similar al descrito, a veces, “prendido con alfileres” y que ante cualquier menor cambio, siempre habrá la incertidumbre de que “el monstruo despierte” y “se vuelva contra sus profanadores”. Día tras día, los proyectos adquieren sus propias dinámicas organizacionales y cuando se atrasa, por lo general, lo hace sin control.

El escenario que se ha descrito es muy bien conocido por todos aquellos que de una manera u otra conocen sobre la cotidianidad de la labor académica – investigativa de la UTC y específicamente la carrera de Informática y Sistemas Computacionales, por todos los docentes y estudiantes que en algún momento podrán formar parte de la industria del software y otras ramas derivadas de esta hermosa especialidad con sus proyectos.

Existen muchos más ejemplos que pueden ser abordados en este acápite, de otras carreras inclusive, lo importante de todo lo expuesto es entender que el poder de la osadía lleva ímpetu y sacrificios; la magia del conocimiento es haber soñado que puedes hacer algo; la virtud está en ser osado y hacerlo, pues en ello está la capacidad de crear; es ahí donde se refleja tu naturaleza inteligente.

Capítulo XIII

Empresa de desarrollo de software Rebian Sistemas

En los últimos años los emprendimientos de han venido desarrollando de manera acelerada, la falta de espacios para los nuevos profesionales con remuneraciones acordes a su grado académico son limitadas lo cual conlleva a los jóvenes con deseos de superación a emprender sus propios negocios.

Algunos casos de éxito que han nacido de ideas de jóvenes que hoy en día se han convertido en empresas líderes a nivel mundial como lo es el caso de google quien empezó sus operaciones en 1988 con un motor de búsqueda que revolucionaría el internet llegando hoy en día a ser el buscador número uno en la web. Otro de los casos de éxito es el famoso Facebook cuando Mark Zuckerberg con solo 19 años de edad empezó una red social que cambiaría radicalmente a los jóvenes y su manera de interactuar entre ellos.

A nivel nacional la industria de las tecnologías de la información y Comunicación está empezando a desarrollarse de manera paulatina según un estudio elaborado por la Escuela de Negocios (Espae), Ecuador aún reporta el segundo valor más bajo en lo que se refiere a exportaciones de servicios de TIC (incluyendo software), un resultado que contrasta con el dominio que en la región tienen Brasil, Argentina y Costa Rica. Esta desventaja, según se explica, está dada por los obstáculos que continúa afrontando esta industria, aun cuando la importancia de las TIC consta en la planificación pública.

Entre las debilidades que se apuntan al sector están la mínima experiencia de cooperación y asociación con las empresas, universidades y con el Gobierno, además de la escasez de centros de transferencia tecnológica y el poco conocimiento de las necesidades y tendencias del mercado internacional. (EKOS, 2017)

Orígenes

Rebian Sistemas es una empresa joven que implementa soluciones informáticas accesibles para cualquier tipo de cliente. Para ello se centran en brindar soluciones empresariales a nivel de software a medida.



Figura 13.1. Logo de Rebian Sistemas. Adaptado de (Facebook, 2016)

En Rebian Sistemas se encuentran convencidos de que las aplicaciones o sistemas informáticos se han convertido en una poderosa herramienta de apoyo para las tareas que se realiza diariamente en cualquier empresa o negocio sin importar la actividad a la que se dediquen. Para ello su compromiso es brindar la mejor alternativa informática para que pueda automatizar las tareas más complicadas y de este modo incremente su productividad y sobre todo le saque el máximo provecho al equipamiento informático que disponga. Es así que su filosofía se resume en brindar “Soluciones Inteligentes” accesibles a cualquier tipo de cliente.

Una empresa joven que nació del emprendimiento de alumnos de la carrera de Ingeniería En Informática Y Sistemas Computacionales de la Universidad Técnica de Cotopaxi rompiendo esquemas lanzaron su empresa de desarrollo de software enfocados en un nicho de mercado de pequeñas y medianas empresas en la zona central del ecuador, sus oficinas ubicadas en la ciudad de macachí provincia de Pichincha.

Sus fundadores Rene Quisaguano, Fabian López y David Criollo son los encargados de implementar estrategias y metodologías de desarrollo de software y su correcta aplicación en los proyectos con la finalidad de que sus colaboradores y pasantes logren dar un resultado de calidad siguiendo estándares internacionales para lograr con la completa satisfacción del cliente.

Servicios

Su arriesgada decisión después de algunos meses de esfuerzo y trabajo empezó a dar sus frutos logran desarrollar software a medida a varias empresas de la zona. Hoy en día seis personas trabajan en Rebian Sistemas. Este equipo es dirigido por fundadores está conformado por diseñadores, especialistas en marketing y desarrolladores de software. Atiende a una cartera de clientes de más de 10 referencias con firmas de diversos sectores productores de la zona.

Los servicios que ofertan son los de Desarrollo de Software, Asesoría y Consultoría Informática, Diseño y Desarrollo de Aplicaciones Web, basando sus proyectos en herramientas que les permite tener una mayor productividad y optimizar recursos y tiempo de desarrollo como son:

- Play framework para el desarrollo web basado en java.
- Bootstrap framework para el desarrollo web basado en php.
- AndroidStudio para el desarrollo de aplicaciones para dispositivos móviles Android.

Rebian Sistemas una empresa joven que nació de la visión e iniciativa de un grupo de estudiantes de la Universidad Técnica de Cotopaxi que utilizando sus conocimientos adquiridos durante su vida universitaria y en base a su esfuerzo y dedicación quieren llegar a consolidarse como una empresa referente en el desarrollo de software a nivel nacional e internacional.

BIBLIOGRAFÍA

Alvarez, M. (2012). Manual de JQuery. España.

Alvarez, M. (2014). Desarroll Web. Retrieved from Desarroll Web:
<https://desarrolloweb.com/articulos/que-es-mvc.html>

Alvial, R., Saavedra, B., & Valenzuela, V. (2011). Informática Viernes. Retrieved from Informática Viernes: https://informaticaviernes2011.files.wordpress.com/2011/07/orm_victor_raquel_berni.pdf

Amador, P. (2013). Estudio Comparativo de Sistemas de Mapeo Objeto Relacional desarrollados en plataforma Open Source. Retrieved from UNISS.

Amo, F., Martínez, L., & Segovia, F. (2005). Introducción a la ingeniería del software. Zaragoza: Delta.

Anibarro, C. (2001). Manual Básico de Html. Bolivia.

Ayoze, A. (2017). Curso de Programación Web: JavaScript, Ajax y jQuery. IT Campus Academy.

Beatriz, F. (2007). Manual rápido de CSS.

Bernal, J. (2012). Universidad Politécnica de Madrid. Retrieved from Universidad Politécnica de Madrid: https://www.etsisi.upm.es/sites/default/files/curso_2013_14/MASTER/MIW.JEE.POOG.pdf

Cabrera, L. (2012). Introducción a CSS. Sevilla.

Cachero, C., Ponce de León, P., & Saquete, E. (2006). Introducción a la programación orientada a objetos. Alicante: Publicaciones de la Universidad de Alicante.

CakePHP. (2012). Retrieved from CakePHP: <https://book.cakephp.org/1.3/es/The-Manual/Beginning-With-CakePHP/Understanding-Model-View-Controller.html>

Castañeda, E. (n.d.). Geocities. Retrieved from Geocities: http://www.geocities.ws/emezas78/matcursos/J_BAS_G02.html

Castro, M. (2002, 01 12). Tecnología Multimedia. Retrieved from Tecnología Multimedia: https://ocw.innova.uned.es/mm2/tm/contenidos/pdf/tema2/lenguajes_marcado.pdf

Castro, R. (n.d.). Elementos de la POO. Ciudad de México.

Ceresola, J. (2012). Material didáctico para la enseñanza de SPRING. Valencia.

Coti, B. (2003). Reglas del negocio en arquitectura tres capas. Guatemala.

DesarrolloWeb. (2001). Retrieved from DesarrolloWeb: <https://desarrolloweb.com/articulos/introduccion-javascript.html>

Duarte, R. (2014). Geeks. Retrieved from Geeks: <https://geeks.ms/rduarte/2014/01/17/implementando-repository-pattern-en-asp-net-mvc-introduccin/>

Durán, F., Gutiérrez, F., & Pimentel, E. (2007). Programación Orientada a Objetos con Java. Madrid: Thomson.

Ecured. (2016). Retrieved from <https://www.upoli.edu.ni/noticias/verNoticia/articulo:524-oracle-academy-en-alianza-con-la-upoli>

Edukativos. (2013). Retrieved from Edukativos: <http://www.edukativos.com/apuntes/archives/3722>

Eguílez, J. (2007). Introducción a XHTML. España: Creative Commons.

EKOS. (2017, 03 21). Retrieved 07 20, 2017, from <http://ekosnegocios.com/negocios/verArticuloContenido.aspx?idArt=8855>

Facebook. (2016). Retrieved from Facebook: <https://www.facebook.com/rbnsistemas/photos/a.1724432517799486.1073741827.1717758125133592/1724433291132742/?type=1&theater>

Florián, B. (2007). Manual rápido de CSS.

Garro, A. (2014). JavaScript. Amsterdam: Creative Commons.

Gauchat, J. (2012). El gran libro de HTML 5, CSS3 y Javascript. Barcelona: Marcombo.

González, E. (2009). Tutorial básico del programador web: HTML desde cero.

Groussand, T. (2014). Java 8: Los fundamentos del lenguaje Java. Barcelona: ENI.

helloacm.com. (2017, 01 24). Retrieved from helloacm.com: [https://helloacm.com/model-view-controller-explained-in-c/amp/](https://helloacm.com/model-view-controller-explained-in-c/)

Joyanes, L., & Fernández, M. (2002). C #: Manual de Programación. McGraw-Hill Interamericana.

Lamarca Lapuente, M. J. (2013, 12 08). Hipertexto: El nuevo concepto de documento en la cultura de la imagen. Retrieved from Hipertexto: El nuevo concepto de documento en la cultura de la imagen.: <http://www.hipertexto.info/documentos/html.htm>

Learn Rails. (2017). Retrieved from Learn Rails: <https://www.tutorialspoint.com/ruby-on-rails/index.htm>

Libros Web. (2017). Retrieved from Libros Web: http://librosweb.es/libro/xhtml/capitulo_2/elementos_html.html

Martínez, A. C. (2013). Estudio del estado del arte sobre Frameworks MVC para el desarrollo de aplicaciones web cliente, caso de estudio ember.js. Madrid.

Menéndez, R., & Barzanallana, A. (n.d.). Desarrollo Aplicaciones Web. Murcia.

Montes, J., & Carmona, S. (2015). Invocación de un servicio web REST desde un cliente web y jQuery. Cádiz.

Paniagua, A. (2012). HTML 5 en la educación. Madrid: Nipo.

- Pavón, J. (2008). Universidad Complutense. Retrieved from Universidad Complutense: <https://www.fdi.ucm.es/profesor/jpavon/poo/1.1.objetos%20y%20clases.pdf>
- Peter Hilton. (2010, Agosto 21). Play Framework. Retrieved from Manuales, tutoriales & referencias: <http://playdocs.appspot.com/documentation/1.2.4/usability>
- Play Framework. (2012, Junio 13). Retrieved from Play Framework: <http://playdocs.appspot.com/documentation/1.2.4/overview>
- Quizlet. (2017). Retrieved from <https://quizlet.com/157520934/bases-de-datos-flash-cards/>
- Reenskaug, T., & Coplien, J. (2009). The DCI Architecture: A New Vision of Object-Oriented Programming. Artima Developer. Retrieved from http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/sanchez_r_ma/capitulo2.pdf
- Rodríguez, J., Fajardo, G., Perdigón, G., Mazón, J., & De la Paz, J. (2002). Aplicación del hipertexto en el aprendizaje asistido por computadora. Revista UNAM.
- Rufo, P. (2001). Html.
- Sarah Clatterbuck, M. P. (2016, 12 02). Definición. Retrieved from Play Framework: <https://engineering.linkedin.com/blog/2016/12/pemberly-at-linkedin>
- Támara, I. (2000). Manual de Html Css. Boston: Free Software Foundation.

Yanes, O., & Gracia, H. (2011). Mapeo Objeto/Relacional (ORM). Telemática.

Yeniseidy Fernández Romero, Y. D. (2012, 04). REVISTA DIGITAL DE LAS TECNOLOGIAS DE LA INFORMACIÓN Y LAS COMUNICACIONES. Retrieved from <http://revistatelematica.cujae.edu.cu/index.php/tele/article/view/15/10>

Zafra, J. (2015). Elaboración de hojas de estilo. Málaga: IC.

Zenexity, B. b. (2012, Junio 13). Play/Modelo de Seguridad. Retrieved from Manuales, tutoriales y referencias: <http://playdoce.appspot.com/documentation/1.2.4/secure>



ISBN: 978-9978-395-48-6

A standard linear barcode representing the ISBN number 978-9978-395-48-6.

9 789978 395486

2017