ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA DE SISTEMAS

DESARROLLO DE UN PROTOTIPO DE UN GENERADOR DE CÓDIGO PARA APLICACIONES JEE6 PARA LA EMPRESA CLEAR MINDS CONSULTORES CIA LTDA.

PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SISTEMAS INFORMÁTICOS Y DE COMPUTACIÓN

JAIRO SANTIAGO MARTÍNEZ UNAUCHO.
san_thiago92@hotmail.com

JORGE LUIS RODRÍGUEZ CHALÁ
jorgerodriguezchala@hotmail.com

DIRECTOR: Ing. CARLOS ESTALESMITMONTENEGRO ARMAS

carlos.montenegro@epn.edu.ec

Quito, Abril 2014

DECLARACIÓN

Nosotros, Jairo Santiago Martínez Unaucho y Jorge Luis Rodríguez Chalá, declaramos bajo juramento que el trabajo aquí descrito es de nuestra autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que hemos consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedemos nuestros derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

_____

Jairo Santiago Martínez Unaucho

_____

Jorge Luis Rodríguez Chalá

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por Jairo Santiago Martínez Unauchoy Jorge Luis Rodríguez Chalá, bajo mi supervisión.

_____

Ing. Montenegro Armas Carlos Estalesmit

DIRECTOR DEL PROYECTO

AGRADECIMIENTOS

En primera instancia queremos agradecer a Dios por darnos el regalo maravilloso que es la vida, la fuerza y la capacidad para cumplir una meta más en nuestro camino hacia la excelencia personal y profesional.

También queremos agradecer al Ing. Carlos Montenegro por brindarnos su ayuda y conocimiento en la dirección del presente trabajo, a la empresa Clearminds Consultores Cía. Ltda, la cual nos abrió las puertas y nos ofreció su colaboración para la conclusión de este trabajo de titulación.

Este agradecimiento va dedicado de igual manera a todos nuestros amigos cultivados en el transcurso de la carrera, quienes con su apoyo incondicional nos supieron dar aliento en los momentos más difíciles y arduos.

Gracias totales.

DEDICATORIA

Quiero dedicar este trabajo a toda mi familia, en especial a mis padres Lourdes y Carlos quienes han sabido inculcar en mí, valores que me han ayudado a crecer como persona, los cuales los llevo presentes día a día y los atesorare eternamente.

A mi hermanita querida, Ruby, quien desde que vio la luz, cambio mi vida totalmente, ella es el motor de mi vida y con una sonrisa me da la fuerza para seguir adelante y no decaer jamás.

A mis queridos abuelitos que en mis primeros años supieron ser mis segundos padres y compartirme su sabiduría, aquella que no se obtiene ni se compara con la académica, es aquella sabiduría que da el amor, los años y sobretodo la sociedad más importante que es la familia.

A mis mejores amig@s y a todos quienes me han brindado su cariño y me han ayudado con consejos sabios, los cuales me han ayudado a superar varios obstáculos que la vida nos pone, para fraguar nuestro espíritu de lucha y dedicación por alcanzar el éxito y la virtud.

Santy.

DEDICATORIA

Quiero dedicar este trabajo especialmente a mis padres, que desde mi niñez me inculcaron valores como la honestidad, el respeto y la humildad, los mismos que hasta el día de han sido grandes pilares en mi vida. Por enseñarme que la mejor manera de salir adelante es luchando hasta llegar a cada objetivo planteado, que el trabajo honrado da frutos, que por más difícil que sea la situación, nunca debemos rendirnos. Por el apoyo que me brindaron en momentos de debilidad, incertidumbre o tristeza, pero sobre todo, por su amor incondicional.

A mis hermanos por haber estado ahí, porque siempre encontré en ellos una palabra de aliento, para poder salir adelante. Porque en cada momento difícil estuvieron ahí, porque su sola presencia me dio ánimos para seguir adelante lograr mis objetivos.

A mis amigos, por su apoyo, su compañía, porque sin duda se han convertido en una segunda familia para mí, porque han estado ahí en cada momento fácil o difícil que atravesé en mi vida tanto profesional como personal, porque en ellos encontré consejo, confianza, cariño, pero sobre todo una muy sincera amistad, que para mí, va mucho más allá que un título académico, su valor no se puede comparar con nada.

A mi entrenador, por haberme enseñando entre muchas otras cosas, algo que para mí tiene gran importancia, que los límites de una persona no existen más que en su cabeza, que podemos llegar hasta donde nos propongamos, además de que el esfuerzo y el trabajo superan a cualquier talento. Y sobre todo, que siempre podemos dar un poco más.

Jorge.

LISTA DE TABLAS

LISTA DE FIGURAS

LISTA DE ANEXOS

RESUMEN

En este proyecto de titulación se realiza el desarrollo de un prototipo de un generador de código para aplicaciones Web (CODGEN), cuyafuncionalidad está dedicada al desarrollo del módulo de gestión de un sistema base. Se emplea la metodología Scrum.

Se inicia el proyecto con la identificación y selección de las herramientas a ser utilizadas para el desarrollo del prototipo. Seguidamente, se establece la descripción del problema, la justificación y la descripción de la metodología utilizada.

A continuación se realiza el levantamiento de requerimientos y la depuración de los mismos, para definirlos con historias de usuario. A partir de ahí se desarrolla la planificación, diseño y ejecución de los sprints. Además se realizan las pruebas pertinentes en cada caso.

Una vez terminada la construcción del prototipo, se corre una simulación del mismo para varios casos tipo; y, se analizan los resultados.

Finalmente,se establecen conclusiones y recomendaciones acerca del desarrollo del proyecto.

TABLA DE CONTENIDO

CAPÍTULO 1

1 PLANTEAMIENTO DEL PROBLEMA

1.1 DESCRIPCIÓN DEL PROBLEMA

La empresa Clearminds Consultores Cía. Ltda., es una empresa que se dedica al desarrollo de aplicaciones web, utilizando el lenguaje Java, bajo la arquitectura JEE6 (Java Platform, Enterprise Edition). Esta arquitectura está basada en JSE (Java Standard Edition), pero con la particularidad de que se añaden librerías y servicios, que dan soporte a ciertas características necesarias para las aplicaciones web, tales como son: la escalabilidad, seguridad, integridad, accesibilidad, transaccionalidad, entre otras, las cuales son muy importantes en cuanto al desarrollo de nivel empresarial se refiere. Esta arquitectura está basada en tres capas las cuales se describen a continuación:

La capa de acceso a datos o capa de persistencia

Esta es la capa que permite mapear el contenido de la base de datos en código Java, admite entre otras características, la traducción de los tipos de datos que se tiene en la base, a tipos de datos Java y por medio de anotaciones permite realizar también las relaciones que existen entre las tablas de la base de datos. En una aplicación orientada a objetos, la persistencia permite que el estado de un objeto sea almacenado y después de un tiempo, este pueda ser recreado al estado en que fue almacenado. Para efectuar la persistencia se utilizará Hibernate, la cual es una tecnología que permite realizar el mapeo de las tablas en entidades con código Java, cada entidad representa una tabla de la base de datos [1].

La capa de negocio

Para la lógica del negocio se utilizan los llamados Session Beans, los cuales permiten realizar básicamente las consultas desde la base de datos. Estas consultas se realizarán por medio del lenguaje JPQL (Java Persistence Query Languaje).

JPQL es una herramienta que consiente la interacción con una base de datos relacional, esta permite manipular la información almacenada en la base de datos. JPQL es un lenguaje muy similar al conocido SQL (Structure Query Languaje), pero la diferencia fundamental radica en que SQL trabaja directamente con las tablas y el lenguaje JPQL trabaja con las entidades mapeadas, lo que hace que las consultas se simplifiquen en la mayoría de los casos.

La capa de Presentación

Esta es la capa que se encarga de la interacción con el usuario, está conformada por los Managed Beans, que se definen como las clases que permiten la comunicación, entre el lenguaje Java y el lenguaje XML (eXtensible Markup Language),el cualse empleará para el desarrollo de las páginas. En esta capa no se maneja ningún tipo de relación con la base de datos, haciendo así que el código sea más manejable y organizado.

El código de esta capa no es reutilizable, es decir, que cada página debe tener su propio Managed Bean (Controlador). Esta capa maneja la apariencia de la aplicación, tema que los programadores menosprecian, pero a los ojos de los usuarios es uno de los temas más importantes en lo que a su aplicación se refiere[2].

Figura 1 Arquitectura JEE6 Empleada por Clearminds Consultores Cía. Ltda[1]

La empresa actualmente realiza el desarrollo de las aplicaciones desde cero, es decir,se parte desde el mapeo de las entidades, luego se realiza el desarrollo de servicios, controladores y páginas de gestión, manualmente. Entiéndase por páginas de gestión, a las páginas que realizan las acciones básicas sobre la base de datos, esto es, insertar, eliminar y actualizar los datos de las tablas.

El desarrollo de estas pantallas llevan un tiempo considerable, dependiendo del número de tablas que existan en la base de datos y más aún, tomando en cuenta que la empresa trabaja con el marco de trabajo Scrum, se producen cambios muy a menudo, lo que provoca la modificación de la base de datos y por lo tanto también significa que el mapeo debe realizarse nuevamente.

Por este motivo, surge la necesidad de automatizar este proceso, para agilizar el desarrollo de aplicaciones web por medio de la generación de entidades, servicios, controladores y páginas de gestión de manera automática, bajo la arquitectura JEE6, la misma que es empleada por la empresa actualmente.

La aplicación permitirá también realizar la validación de campos que no puedan ser nulos, además de la personalización de nombres y tipos de datos.

[1]Gráfico elaborado por los autores.

El promedio de tiempo que toma la elaboración de estos elementos en cada una de las capas, se presenta a continuación en la **Tabla 1**.

Proyecto	Número de Tablas	Capas	Promedio de Líneas de código por Tabla	Promedio de líneas de código (Interfaces de Gestión)	Tiempo promedio por línea de Código (segundos)	Tiempo (horas)	Tiempo Semana Laboral (40 horas)
Proyecto 1	21	Acceso Datos	Autogenerado			0,08	4,12
		Negocio	65	1365	50	18,96	
		Presentación	500	10500		145,83	
Proyecto 2	31	Acceso Datos	Autogenerado			0,08	5,78
		Negocio	57	1767	50	24,54	
		Presentación	480	14880		206,67	
Proyecto 3	11	Acceso Datos	Autogenerado			0,08	1,9
		Negocio	60	660	50	9,17	
		Presentación	445	4895		67,99	
Proyecto 4	34	Acceso Datos	Autogenerado			0,08	6,52
		Negocio	62	2108	50	29,28	
		Presentación	490	16660		231,39	
Promedio	**24,25**						**4,59**

Tabla 1 Estimación de tiempos de los proyectos realizados por Clearminds Consultores Cia. Ltda.[2]

Como se puede apreciar, en la elaboración de 24 pantallas de gestión se emplea un promedio de tiempo de 4,6 semanas aproximadamente. Ese tiempo bien podría ser utilizado en la elaboración de la lógica del negocio, es decir, la funcionalidad más importante del sistema y no invertir tanto en la gestión básica del mismo, lo que sin duda hará a la empresa más productiva, pues habrá una reducción significativa de tiempo y de recursos.

[2] Tabla elaborada por los autores en base a información de los proyecto realizados por Clearminds Consultores Cia. Ltda.

1.2 JUSTIFICACIÓN DE LA METODOLOGÍA DE DESARROLLO

Uno de los pilares para la elaboración exitosa de un proyecto de desarrollo, es adaptarse a las necesidades del cliente y uno de los requerimientos de la empresa Clearminds Consultores Cía. Ltda., es que se utilice una metodología ágil.

Tomando en cuenta que, definir todos los requerimientos de un sistema en su fase inicial es prácticamente imposible, se debe emplear una metodología que permita la flexibilidad suficiente para aumentar o disminuir características al sistema, según se vayan presentando durante el proceso de desarrollo.

Existe también la necesidad de que el cliente se involucre en el proceso y que cada avance que se realice sea supervisado por el mismo, con la finalidad de monitorear que el proyecto vaya por la vía correcta y que el cliente estesatisfecho con lo realizado. De esta manera,se asegura que el progreso del sistema sea consistente con lo que el cliente espera como resultado final. Esto se logra mediante las entregas frecuentes de avances del proyecto, lo que hace que el cliente aprecie que el desarrollo realmente está avanzando y le dé la tranquilidad de que su sistema va por buen camino.

Por este motivo y tomando en cuenta que la empresa utiliza el mismo marco de trabajo, se eligió utilizar Scrum para el desarrollo de este proyecto.

1.2.1 SCRUM

Figura 2 Logo Scrum

Scrum es un marco de trabajo ágil, que puede ser utilizado en cualquier aspecto, pero se ha popularizado en el desarrollo de software. Scrum es altamente recomendado para proyectos en los cuales existan cambios muy a menudo o que los requerimientos tengan una alta probabilidad de sufrir cambios durante el desarrollo del sistema.

Scrum se caracteriza por ser un proceso iterativo e incremental. Estas iteraciones se realizan a través de sprints, cuya duración puede variar entre una y cuatro semanas, es decir, que cada avance debe ser presentado en un máximo de tiempo de un mes.

En la actualidad, hay varios puntos de vista de lo que es Scrum, existen quienes lo ven como una metodología, pero es más acertado verlo como un marco de trabajo que proporciona lineamientos para llevar a cabo la gestión de un proyecto[3].

Una de las características más representativas de Scrum, es dar plena confianza al equipo de cómo hacer el desarrollo, ya que sus miembros son los que conocen la mejor manera de llevar a cabo el desarrollo del sistema. En una reunión de Scrum, no se especifican criterios de validación, criterios de entrada o salida, ni las tareas a realizarse por el equipo,como sucede en algunas metodologías, sino más bien, estas reuniones son para especificar, cuáles serán los resultados esperados una vez finalizado el incremento y es el equipo el que decide en qué forma se va realizar lo antes especificado.

Dentro de la teoría de Scrum se utiliza mucho la palabra equipo, ya que su significado es uno de los pilares de este marco de trabajo. Scrum tiene la característica de tener plena confianza en el equipo de trabajo que está en el proyecto y una de sus curiosidades es que el equipo carece de un líder, las decisiones se toman entre todo el equipo, así que las decisiones individuales quedan descartadas. Además el equipo es poli funcional, es decir, que todos los miembros del equipo deben ser capaces de hacerse cargo de una tarea y realizarla de manera exitosa.

Dentro de este marco de trabajo, se tienen tres roles específicos y muy importantes dentro del equipo:

El **Scrum Master**, el cual tiene la responsabilidad de guiar al equipo, se lo conoce también como facilitador. Actúa como un moderador dentro de las reuniones del equipo y su función no es la de dar soluciones, sino, la de guiar al equipo de la mejor manera para que por sí mismo encuentre las soluciones más adecuadas. Conjuntamente, ayuda al equipo quitándole la presión externa, para que sus

elementos puedan trabajar concentrados en lo que les concierne, que es el desarrollo del sistema como tal. Cabe recalcar que el facilitador no se encarga de asignar responsabilidades, solo es moderador en las reuniones que se realizan.

El **Product Owner**, representa a la empresa cliente, o al grupo de interesados en el producto. Es parte del equipo de desarrollo, participa en las reuniones al final de cada sprint, siendo él quien aprueba los incrementos presentados, en el caso de que los haya, realiza los cambios necesarios en los requerimientos o en el incremento presentado. Siendo un marco de trabajo ágil, el cliente es una parte vital del equipo.

El**Sprint,** es el trabajo que se va realizar en un tiempo determinado para lograr completar una iteración del sistema.Al inicio del sprint se debe realizar una reunión, en la cual todos los integrantes del equipo hacen una consideración de cuantas tareas van a poder realizar a lo largo del sprint y la duración de cada una de esas tareas.

La duración de las tareas está dada por **Story Points**, que son unidades básicas de tiempo definidas por el equipo, que puede ser por ejemplo, el tiempo promedio que se invierte en desarrollar una pantalla básica. El Scrum Master, es el encargado de dirigir estas reuniones y es quien ayuda en la asignación de tareas, mas no las asigna, debido a que está abierta la opción de que los miembros del equipo soliciten tareas de manera voluntaria. Al final de cada día del sprint, los miembros del equipo deben asistir a una reunión diaria llamada **Daily Stand up**.

En el Daily Stand up, todos los asistentes están de pie y discuten acerca de los avances realizados, los problemas que tuvieron y lo que piensa hacer el siguiente día. Estas reuniones son de corta duración y no deberían sobrepasar los 15 minutos. De esta formase mantiene un control de lo que se ha venido haciendo y es mucho más fácil detectar posibles problemas en el desarrollo del sistema, para poderles dar una solución rápida.

Al final de cada sprint, se realiza una revisión, en la cual el equipo demuestra la funcionalidad del incremento realizado, en presencia del **Product Owner** o cualquier **Stakeholder**. El objetivo de estas reuniones es probar el avance,

además de obtener unaretroalimentación, que permita detectar inconvenientes y aplicar los correctivos necesarios.

De igual manera, se realiza una retrospectiva del sprint, en la cual los miembros del equipo comparten las experiencias que tuvieron durante el desarrollo, lo que les va a permitir conocer mejor a sus compañeros, al sistema y en conclusión encontrar opciones para mejorar.

1.2.1.1 Elementos principales de Scrum

Dentro de Scrum se tienen varios elementos, el primer elemento es el producto en sí, el cual representa el norte para el equipo, es decir hasta dónde el equipo debe llegar.

Otro de los elementos importantes, es el **Product Backlog**. Este es una lista completa de funcionalidades que deben ser añadidas al producto. Esta lista debe estar hecha en base a prioridades, por tanto es el Product Owner quien tiene mayor participación en la elaboración de este listado. El equipo siempre debe trabajar en base a esta lista de prioridades, realizando primero las tareas más representativas.

Además el Product Backlog, se crea en base a las llamadas historias de usuario, las cuales son descripciones cortas de la funcionalidad deseada, desde el punto de vista del cliente o de un usuario del sistema.

El **Sprint Backlog**, es algo similar, pero se hace al inicio de cada sprint. Contiene la lista de las tareas a realizarse en el sprint y deben mostrarse también como historias de usuario[4].

1.3 SELECCIÓN Y JUSTIFICACIÓN DE LAS HERRAMIENTAS DE DESARROLLO.

1.3.1 ENTORNO DE DESARROLLO INTEGRADO

En la siguiente tabla, se muestra una comparación entre 2 de los entornos de desarrollo integrado más utilizados, en cuanto al desarrollo de aplicaciones Java se refiere. Se han tomado en cuenta varias características, tales como: el ámbito, el control de versiones, la configuración, la experiencia, etc. Estas características permitirán discernir que IDE resulta ser más conveniente para el desarrollo del proyecto.

Entorno de Desarrollo Integrado / Características	Eclipse	NetBeans
Aplicaciones Web	Complejas - Simples	Complejas-Simples
Ámbito	Soporta el desarrollo de todo tipo de aplicaciones java.	Sobresale en el Desarrollo de aplicaciones de Escritorio, así como cualquier tipo de aplicación java.
Control de Versiones	Si	Si
Editor	Buen editor de código, que incluye • Plegado de código • Completado de código • Navegación	Buen editor de código, que a más de las características del editor de eclipse incluye • Macros

	• Marcado de ocurrencias • Coloreado de sintaxis • Plantillas de código, entre otros	
Configuración	Totalmente configurable	Configuración Limitada
Instalación	No necesaria (Portable)	Necesaria
Experiencia	Alta	Baja
Licencia	Eclipse Public License	• CDDL (Common Development and Distribution License) • GPL (General Public License)

Tabla 2 Tabla Comparativa entre Eclipse y NetBeans[3]
Fuentes: [5][6][7][8]

Figura 3 Logo de Eclipse

Luego de analizar los pros y los contras de la **Tabla 2**, se ha seleccionado Eclipse para el proyecto. Eclipse es un software, el cual tiene entre varias características, la de ser multiplataforma y multilenguaje.

Este software fue pensado para trabajar con el lenguaje de programación Java. Lenguaje empleado para el desarrollo de CODGEN, debido a que se tiene el conocimiento suficiente del mismo, además se ha tomado en cuenta que la empresa cliente trabaja con este lenguaje de programación y con el software anteriormente mencionado. Eclipse además es muy útil pues se presta para la inclusión de plugins, los mismos que son complementos que permiten añadir funcionalidad al software.

Adicionalmente Eclipse trabaja con XML, lenguaje que conjuntamente con Java serán utilizados para el desarrollo del proyecto, asimismo el código generado como producto final de este proyecto estará escrito en dichos lenguajes [8].

[3] Tabla realizada por los autores en base a información de las fuentes.

1.3.2 BASE DE DATOS

A continuación se presenta una tabla comparativa entre MySQL y PostgreSQL, 2 de las bases de datos de licencia libre, más empleadas en el desarrollo de aplicaciones web. Para esta comparativa, se han tomado en cuenta algunas de las características más importantes de cada una de ellas.

Base de Datos / Características	PostgreSQL	MySQL
Lenguaje de Consulta	SQL (Ajustado a la ISO)	SQL
Integridad Referencial	Si	No
Concurrencia en Aplicaciones Web	Alta	Baja
Aplicaciones Web	Complejas	Sencillas
Escalabilidad	Alta	Baja
Consumo de Recursos	Normal	Bajo
Experiencia	Alta	Normal
Licencia	BSD (Berkeley Software Distribution)	GPL (General Public License)

Tabla 3. Tabla comparativa entre PostgreSQL y MySQL[4]
Fuentes: [9][10][11][12]

Figura 4 Logo de PostgreSQL

Luego de analizar varias características de la **Tabla 3**,se seleccionó a PostgreSQL como la base de datos para el desarrollo del proyecto, tomando en cuenta los siguientes detalles:

[4] Tabla realizada por los autores en base a información de las fuentes.

PostgreSQL es un sistema de gestión de base de datos, de distribución libre, que permite el manejo de base de datos de una manera muy eficiente.

Este software es desarrollado por una comunidad de desarrolladores, sin fines de lucro, lo que le ha llevado a alcanzar un gran número de adeptos, por su gran rendimiento y alta confiabilidad.

Entre sus características, se puede destacar que permite una alta concurrencia, es decir que no bloquea el acceso a los usuarios a la base de datos, en el caso en que más de uno intente conectarse a la vez. Además, existen muchos tipos de datos nativos, que permiten la creación de base de datos con los campos que sean más adecuados, según sean las necesidades del usuario. Asimismo, permite hacer todo tipo de transacciones,tales como la elaboración de triggers, conexiones remotas, bases de datos distribuidas, etc.

Cabe mencionar que existen interfaces como pgAdmin, que permiten una administración sencilla de las bases, debido a que cuenta con una interfaz gráfica.

Tomando en cuenta estas importantes características y considerando que la empresa para la que se realizará el producto trabaja con este software, se pensó que sería adecuada la utilización de esta base de datos [3] [7].

1.3.3 SERVIDOR DE APLICACIONES JAVA

En la tabla subsecuente se realiza una comparación, a partir de las características de 2 servidores de aplicaciones. Estos servidores son JBoss y Tomcat, los mismos que son de licencia libre y cada uno tiene algunas características particulares, que sea analizarán a continuación.

Java Aplication Server / Características	JBoss	Tomcat
Lenguaje	Java	Java
Compatibilidad	Cualquier Sistema Operativo con JVM	Cualquier Sistema Operativo con JVM

Acceso a la funcionalidad de JEE	Completo	Limitado
Archivos de Instalación.	Gran Magnitud	Ligera Magnitud
Enterprise Java Beans	Funcionalidad nativa	No acoge de Forma Nativa
Aplicaciones Web	Empresariales, Complejas	Sencillas, no necesitan un Stack completo de JEE
Experiencia	Alta	Normal
Licencia	GPL (General Public License)	ASF (Apache Software Foundation)

Tabla 4. Tabla comparativa entre Jboss y Tomcat[5]
Fuentes:[13][14][15]

Figura 5 Logo de JBoss

Basándose en características de la **Tabla 4,** se seleccionó a Jboss como el servidor que alojará a la aplicación.

Jboss es un servidor de aplicaciones de código abierto, que permite manejar aplicaciones Java. Al ser de código abierto, brinda la facilidad de utilizarlo sin necesidad de preocuparse por licencias o derechos, lo que permite enfocarse más en el desarrollo.

Este servidor de aplicaciones, proporciona entre otras funciones, la de conectarse a la base de datos de una manera sencilla y rápida. Además trabaja de forma muy eficiente con Hibernate.

Maneja la concurrencia en la aplicación, creando hilos de ejecución en el caso de que sea necesario, haciendo que la aplicación funcione de la manera más eficiente.

Además provee librerías que ayudan a mejorar la funcionalidad de las aplicaciones, en cuanto al manejo de Beans se refiere y respectivamente a funciones como la inyección de código.

[5] Tabla realizada por los autores en base a información de las fuentes.

1.3.4 IMPLEMENTACIÓN DE JSF

Basándose en las características que ofrecen tanto Primefaces y Richfaces, 2 de las implementaciones JSF de más demanda, se presenta la siguiente tabla comparativa, con la finalidad de destacar sus diferencias y semejanzas y así poder seleccionar la más apropiada para el proyecto.

Implementación de JSF / Características	PrimeFaces	RichFaces
Permite Trabajar con JSF 2	Si	Si
Permite Trabajar con Ajax	Si	Si
Diseño Web	Innovador	Estándar
Experiencia	Alta	Baja
Licencia	Apache License V2	GNU Lesser General Public License

Tabla 5. Tabla comparativa entre PrimeFaces y RichFaces[6]
Fuentes:[16][17][18]

Figura 6 Logo de PrimeFaces

Luego de hacer un estudio de la **Tabla 5,** se llegó a concluir, que PrimeFaces será la implementación de JSF (Java Server Faces) que se empleará en el proyecto, pues actualmente resulta ser la más innovadora en cuanto a diseño web se refiere.

PrimeFaces permite el uso de componentes visualmente enriquecidos, que mejoran significativamente la capa de presentación de las aplicaciones web. Es de código abierto, desarrollada por Prime Tekchnology. Una de las ventajas de utilizar esta implementación, es la facilidad de integración con otras implementaciones tales como RichFaces o IceFaces.

La continua actualización de las versiones de PrimeFaces asegura que los errores que se presentan en una versión anterior sean corregidos de una manera rápida

[6] Tabla realizada por los autores en base a información de las fuentes.

con la siguiente versión. Actualmente se trabaja con la versión 4.0 liberada aproximadamente hace 3 meses.

CAPÍTULO 2

2 DESARROLLO DEL SISTEMA

2.1 PRODUCT BACKLOG

Como ya se analizó en el anterior capítulo, el primer paso para realizar el desarrollo del sistema es recolectar los requerimientos del usuario.

Después de tener una reunión con el cliente, se elaboró una tabla, en la que se detallan las historias de usuario del cliente y los criterios de aceptación del sistema. Cabe recalcar que el usuario especifica lo que se debe hacer, más no el cómo.

A continuación se detalla esta información en la tabla 6:

©				Criterios de Aceptación			
Rol	Característica / Funcionalidad	Razón / Resultado	Escenario	Criterio de Aceptación (Título)	Contexto	Evento	Resultado / Comportamiento esperado
Como un Cliente/ Usuario	Necesito conectarme a la base de datos Postgres.	Obtener las tablas y sus atributos, en conjunto con la demás información necesaria para el mapeo (Restricciones, Secuencias, Tipos de dato, Claves).	1	Presentar las tablas que serán mapeadas al usuario.	En caso de que la base de datos tenga tablas.	Cuando se presione el botón para realizar la conexión.	Se desplegará un diálogo en el cual se listarán los nombres de las tablas existentes en la base de datos.
			2	Conexión exitosa, pero no se presenta ningún dato.	En caso de que la base de datos tenga tablas, pero no se logran extraer.	Cuando se presione el botón para realizar la conexión.	Se desplegará un diálogo, en el cual se mostrarán los nombres de las tablas que se hayan logrado extraer, o en el peor de los casos, no se mostrará el nombre de ninguna tabla.
			3	Conexión fallida.	En caso de que los parámetros de conexión sean erróneos.	Cuando se presione el botón para realizar la conexión.	Se desplegará un mensaje, en el que se indicará que hubo un error en la conexión. El texto será el siguiente: “Conexión fallida, por favor revise los parámetros de conexión.”

Como un Cliente/ Usuario	Necesito editar la información de los campos que conforman las tablas de la base de datos.	Realizar un mapeo personalizado, con la finalidad de adaptarlo a las necesidades del usuario, por ejemplo renombrado de atributos, tipos de dato, restricciones.	1	Presentar todos los atributos, y que los campos que sean editables estén disponibles para su modificación.	En el caso de que las tablas tengan atributos.	Cuando se selecciona una entidad.	Se desplegará una tabla con todos los campos, tanto editables como no editables.
			2	No se presentan los atributos de las tablas.	En el caso de las tablas tengan atributos, pero no se lograron recuperar.	Cuando se selecciona una entidad.	Se desplegará una tabla con información incompleta, y en el peor de los casos una tabla vacía.
			3	Se guardan las modificaciones realizadas.	En el caso de que se hayan realizado modificaciones.	Cuando se presiona el botón guardar cambios.	Se desplegará un mensaje cuyo texto será: “Cambios guardados correctamente.”
			4	No se lograron guardar las modificaciones realizadas.	En el caso de que se realicen modificaciones y no se pudieron guardar.	Cuando se presiona el botón guardar cambios.	Se desplegará un mensaje cuyo texto será: “No se pudieron guardar los cambios.”
Como un Cliente/ Usuario	Deseo generar las entidades, las mismas que forman parte de la	Generar el mapeo entre la base de datos y la capa de acceso a datos.	1	Se obtienen los archivos con extensión .java, con el código correspondiente.	En el caso de que se encuentre un directorio válido.	Cuando se presiona el botón generar entidades.	Se desplegará un mensaje que indique que las entidades se crearon de manera exitosa.

	capa de acceso a datos, conjuntamente con el archivo de persistencia.		2	No se pudieron crear los archivos, debido a que no se encontró un directorio válido.	En el caso de que no se haya podido crear el directorio de destino.	Cuando se presiona el botón generar servicios.	Se desplegará un mensaje que indique que el directorio no es válido.
Como un Cliente/ Usuario	Deseo generar los servicios, los mismos que forman parte de la capa de lógica del negocio.	Generar el código que permita la interacción con la base de datos, con acciones tales como consultar, insertar, eliminar y actualizar.	1	Se obtienen los archivos con extensión .java, con el código correspondiente.	En el caso de que se encuentre un directorio válido.	Cuando se presiona el botón generar entidades.	Se desplegará un mensaje que indique que las entidades se crearon de manera exitosa.
			2	No se pudieron crear los archivos, debido a que no se encontró un directorio válido.	En el caso de que no se haya podido crear el directorio de destino.	Cuando se presiona el botón generar servicios.	Se desplegará un mensaje que indique que el directorio no es válido.
Como un Cliente/ Usuario	Deseo generar los controladores, los mismos que forman parte de la capa de presentación.	Generar el código que permita tener el control sobre la página web, y a la vez admita la comunicación de las entidades y los servicios, con la página.	1	Se obtienen los archivos con extensión .java, con el código correspondiente.	En el caso de que se encuentre un directorio válido.	Cuando se presiona el botón generar controladores.	Se desplegará un mensaje que indique que los controladores se crearon de manera exitosa.
			2	No se pudieron crear los archivos, debido a que no se encontró un directorio válido.	En el caso de que no se haya podido crear el directorio de destino.	Cuando se presiona el botón generar controladores.	Se desplegará un mensaje que indique que el directorio no es válido.
	Deseo	Generar el	1	Se obtienen los	En el caso de	Cuando se	Se desplegará un

Como un Cliente/ Usuario	generar las páginas web, las mismas que forman parte de la capa de presentación .	código XHTML, que permitirá mostrar al usuario la interfaz web.		archivos con extensión .xhtml, con el código correspondiente.	que se encuentre un directorio válido.	presiona el botón generar páginas.	mensaje que indique que las páginas se crearon de manera exitosa.
			2	No se pudieron crear los archivos, debido a que no se encontró un directorio válido.	En el caso de que no se haya podido crear el directorio de destino.	Cuando se presiona el botón generar páginas.	Se desplegará un mensaje que indique que el directorio no es válido.
Como un Cliente/ Usuario	Deseo que todo el código generado, esté disponible para descarga.	Con el objetivo de almacenar los archivos generados en el ordenador del usuario, y tomando en cuenta que al ser una aplicación web, no se puede tener acceso al servidor.	1	Se descargan los archivos comprimidos en un archivo .Zip.	En el caso de que se hayan generado todos los archivos para todas las capas, y además estoy hayan sido correctamente comprimidos.	Cuando se presiona el botón descargar.	Se realiza la descarga al equipo del usuario.

Tabla 6Historias de Usuario del Cliente y Criterios de Aceptación[7]

[7] Tabla realizada por los autores, en base a información provista por el cliente.

2.1.1 HISTORIAS DE USUARIO TÉCNICAS

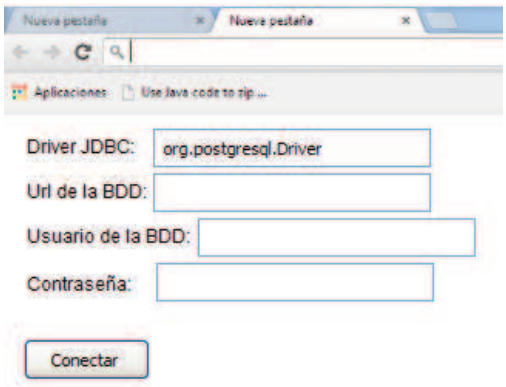
Después de analizar las historias de usuario del cliente, a continuación se procede a realizar las historias de usuario técnicas.

2.1.1.1 HU1

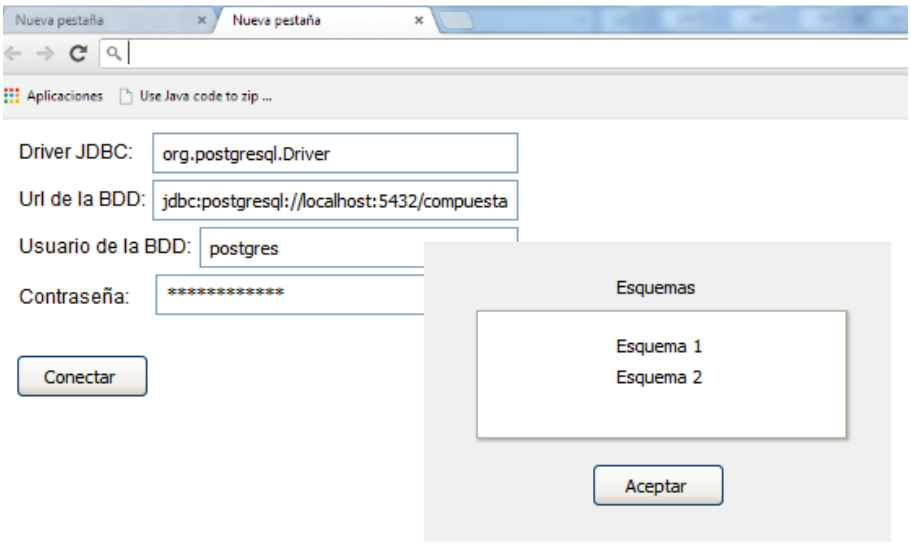
<table>
<tr><td colspan="2">Historia de Usuario</td></tr>
<tr><td>Número: 1</td><td>Nombre: Realizar la conexión a la base de datos.</td></tr>
<tr><td colspan="2">Usuario: Usuario/Cliente</td></tr>
<tr><td>Prioridad en el negocio (alta, media, baja): alta</td><td>Riesgos en el desarrollo(alto, medio, bajo): alto</td></tr>
<tr><td colspan="2">Responsable: Martínez Jairo, Rodríguez Jorge</td></tr>
<tr><td colspan="2">Descripción:

El usuario podrá realizar la conexión a la base de datos, una vez que se conecte se obtendrá la información de la misma. Entre la información que se obtiene tenemos nombre de las tablas, de las columnas, claves primarias, claves foráneas, restricciones de campos no nulos, tipos de dato y secuencias.</td></tr>
<tr><td colspan="2">Requerimientos de Software:

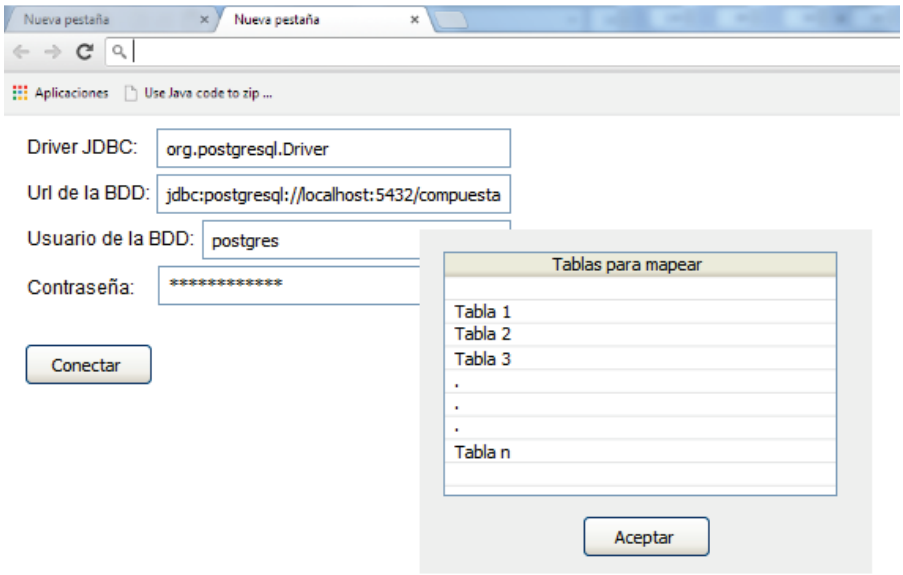
1.- Diseño de interfaz para ingresar los parámetros necesarios para la conexión a la base de datos.

 
</td></tr>
</table>

2.- Se despliegan los esquemas disponibles.



3.- Se despliega la lista con todas las tablas.



Observación: Los esquemas que se muestren dependerán de la constitución de la base de datos del usuario.

El usuario debe tener conexión con el servidor, además de tener activadas las conexiones remotas hacia su base de datos.

2.1.1.2 HU2

<table>
<tr><td colspan="2">Historia de Usuario</td></tr>
<tr><td>Número: 2</td><td>Nombre: Editar los atributos de las tablas que se obtengan de la base de datos.</td></tr>
<tr><td colspan="2">Usuario: Usuario/Cliente</td></tr>
<tr><td>Prioridad en el negocio (alta, media, baja): medio</td><td>Riesgos en el desarrollo(alto, medio, bajo): alto</td></tr>
<tr><td colspan="2">Responsable: Martínez Jairo, Rodríguez Jorge</td></tr>
<tr><td colspan="2">Descripción:

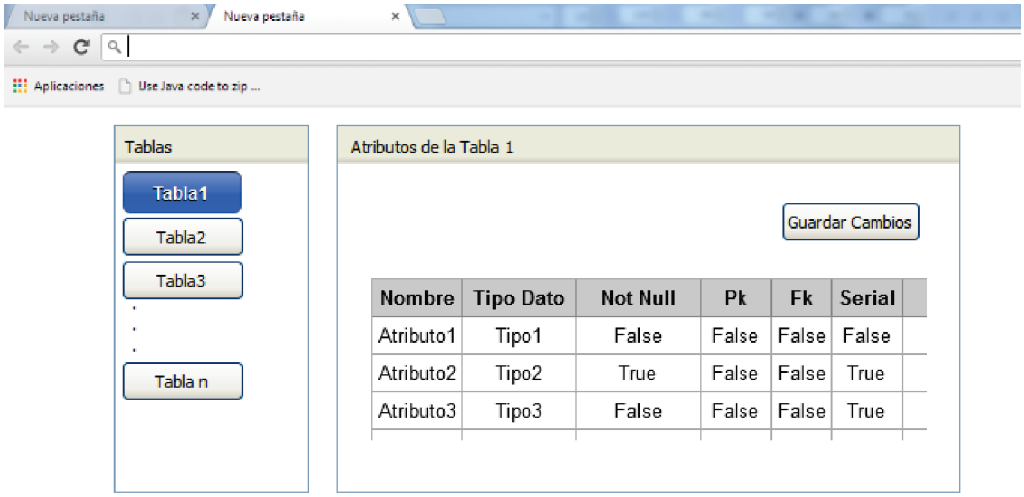
El usuario podrá editar los campos de la base de datos, para así obtener un mapeo más personalizado de la misma. Cabe mencionar que no todos los campos van a ser editables, sino solo los que no alteren la estructura de la base de datos.</td></tr>
<tr><td colspan="2">Requerimientos de Software:

1.- Diseño de interfaz para la edición de los campos de la base de datos.

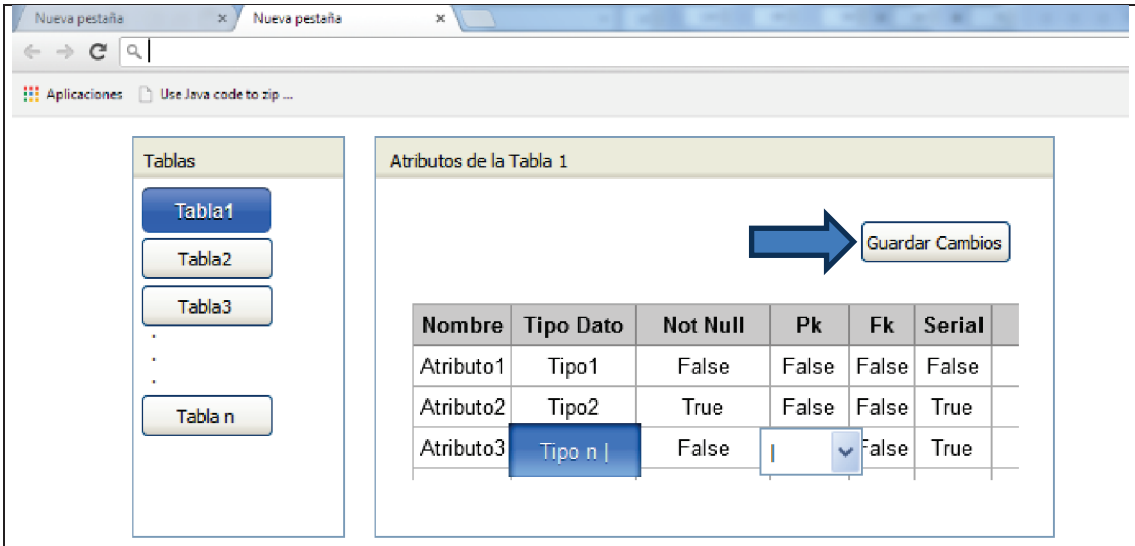
 

2.- Se despliega una lista con todas las tablas que forman parte de la base de datos.</td></tr>
</table>

3.- Seleccionar la tabla a editar. Una vez que se selecciona se despliega una lista con todos los atributos que forman parte de la tabla.



5.- Ubicarse en la celda que se desee editar, y realizar los cambios deseados. Una vez que se termine de editar, los cambios pueden ser guardados.



Observación: Se mostrarán únicamente los campos que hayan sido extraídos de la base de datos.

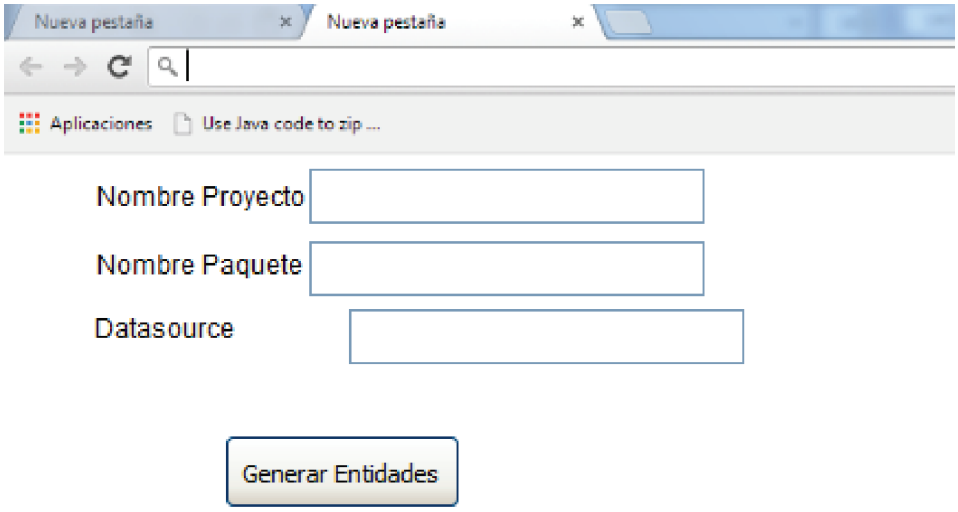
No todos los campos son editables, únicamente se permitirá editar el tipo de dato, el nombre del atributo, restricción de campos no nulos, ya que los demás alterarían la estructura de la base de datos y provocarían conflictos.

2.1.1.3 HU3

Historia de Usuario	
Número: 3	**Nombre:** Generar entidades.
Usuario: Usuario/Cliente	
Prioridad en el negocio (alta, media, baja): alta	**Riesgos en el desarrollo(alto, medio, bajo):** alto
Responsable: Martínez Jairo, Rodríguez Jorge	
Descripción: Se realizará la generación de entidades, de acuerdo a lo indicado por el usuario.	

Requerimientos de Software:

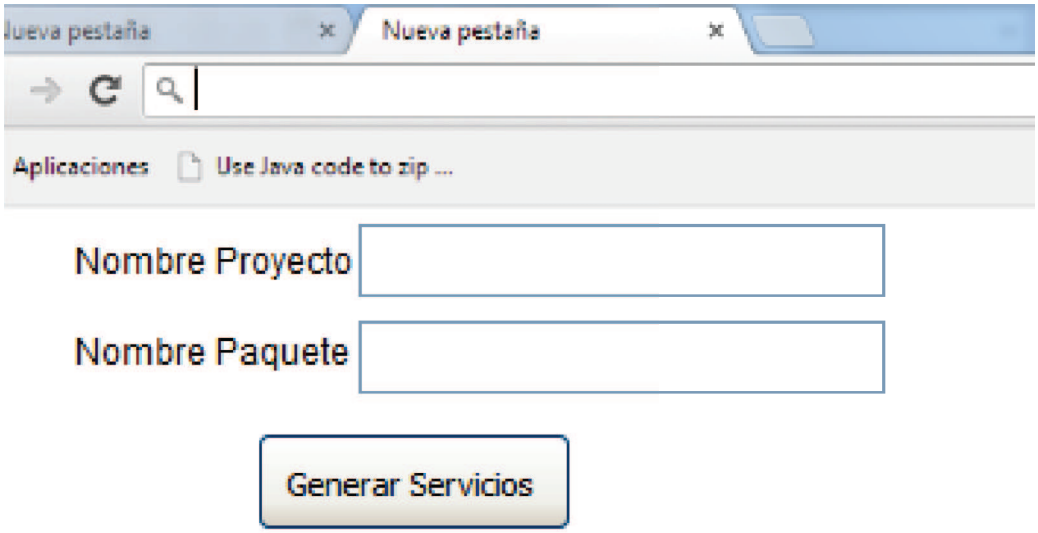
1.- Diseño de interfaz para la generación de entidades.



Observación: Los archivos generados se almacenarán dentro del directorio bin del servidor de aplicaciones. Dentro de este directorio se creará la carpeta con el nombre del proyecto, y dentro de esta se creará la jerarquía de carpetas indicadas en el nombre del paquete raíz.

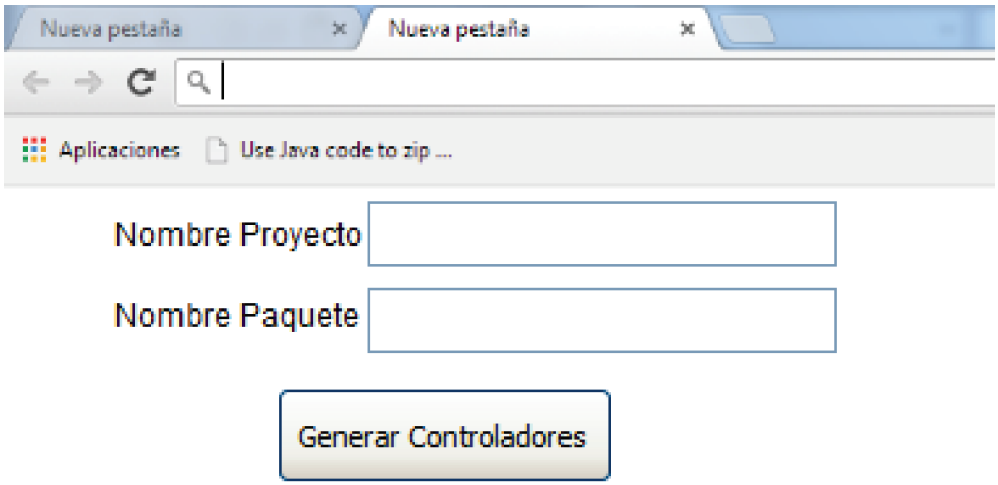
2.1.1.4 HU4

Historia de Usuario	
Número: 4	**Nombre:** Generar servicios.
Usuario: Usuario/Cliente	
Prioridad en el negocio (alta, media, baja): alta	**Riesgos en el desarrollo(alto, medio, bajo):** alto
Responsable: Martínez Jairo, Rodríguez Jorge	
Descripción: Se realizará la generación de servicios, los mismos que contienen la lógica del negocio.	

Requerimientos de Software: **1.-** Diseño de interfaz para la generación de servicios.   
Observación: Los archivos generados se almacenarán dentro del directorio bin del servidor de aplicaciones. Dentro de este directorio se creará la carpeta con el nombre del proyecto, y dentro de esta se creará la jerarquía de carpetas indicadas en el nombre del paquete raíz. Los servicios se generarán a partir de un servicio base, el cual contiene la elaboración genérica de los métodos, haciendo uso así de uno de los principios de la programación orientada a objetos, la herencia.

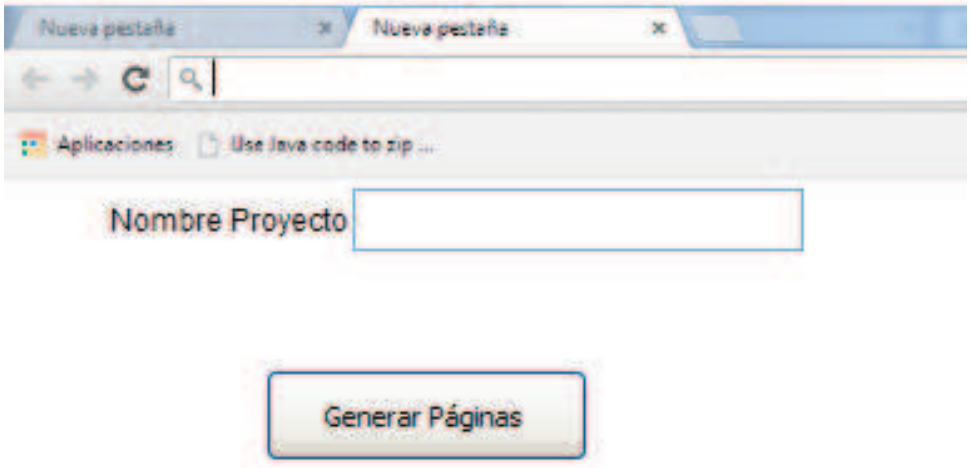
2.1.1.5 HU5

Historia de Usuario	
Número: 5	**Nombre:**Generar controladores.
Usuario: Usuario/Cliente	
Prioridad en el negocio (alta, media, baja): alta	**Riesgos en el desarrollo(alto, medio, bajo):** alto

Responsable: Martínez Jairo, Rodríguez Jorge
Descripción: Se realizará la generación de los controladores, es decir la parte Java de la capa de presentación.
Requerimientos de Software: **1.-** Diseño de interfaz para la generación de controladores.   
Observación: Los archivos generados se almacenarán dentro del directorio bin del servidor de aplicaciones. Dentro de este directorio se creará la carpeta con el nombre del proyecto, y dentro de esta se creará la jerarquía de carpetas indicadas en el nombre del paquete raíz.

2.1.1.6 HU6

Historia de Usuario	
Número: 6	**Nombre:** Generar páginas.
Usuario: Usuario/Cliente	

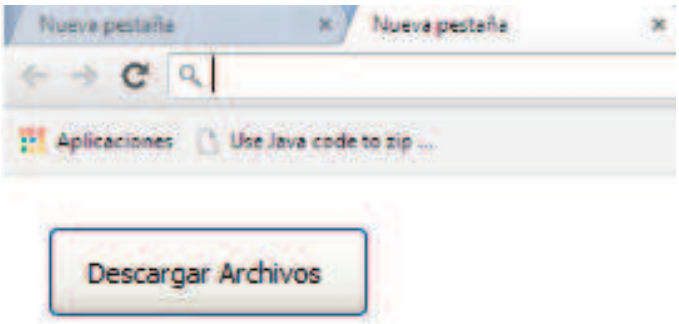
Prioridad en el negocio (alta, media, baja): alta	**Riesgos en el desarrollo(alto, medio, bajo):** alto
Responsable: Martínez Jairo, Rodríguez Jorge	
Descripción: Se realizará la generación de las páginas, las cuales representan la parte gráfica de la aplicación, es decir la parte de más interacción con el usuario.	
Requerimientos de Software: **1.-** Diseño de interfaz para la generación de páginas.  	
Observación: Los archivos generados se almacenarán dentro del directorio bin del servidor de aplicaciones. Dentro de este directorio se creará la carpeta con el nombre del proyecto, y dentro de esta se creará la jerarquía de carpetas indicadas en el nombre del paquete raíz.	

2.1.1.7 HU7

Historia de Usuario	
Número: 7	**Nombre:** Descargar el código generado.

<table>
<tr><td colspan="2">Usuario: Usuario/Cliente</td></tr>
<tr><td>Prioridad en el negocio (alta, media, baja): alta</td><td>Riesgos en el desarrollo(alto, medio, bajo): alto</td></tr>
<tr><td colspan="2">Responsable: Martínez Jairo, Rodríguez Jorge</td></tr>
<tr><td colspan="2">Descripción:

Debido a que el código se genera en el servidor, y el usuario no puede tener acceso directo a él, se permitirá descargar todos los archivos generados en el ordenador del cliente.</td></tr>
<tr><td colspan="2">Requerimientos de Software:

1.- Diseño de interfaz para la descarga de contenido.
 
</td></tr>
<tr><td colspan="2">Observación: Se descargará un archivo con extensión .zip. La descarga se realizará en el directorio destinado para las descargas en el ordenador cliente.</td></tr>
</table>

2.1.2 PRODUCT BACKLOG INICIAL

El Product Backlog es una lista conformada por los requerimientos funcionales del sistema que se va a desarrollar.

Se debe mencionar que esta lista es dinámica, es decir irá cambiando conforme avance el desarrollo del sistema, acorde a las observaciones que se realicen

entre los miembros del equipo y el Product Owner. El orden de la lista está dado por la prioridad que tendría cada requerimiento en el desarrollo del sistema.

A continuación se muestra la lista de requerimientos obtenidos por parte del Product Owner:

Orden	Requerimiento	Descripción
1	Realizar la conexión a la base de datos.	HU1
2	Generar Entidades	HU3
3	Generar Servicios	HU4
4	Generar Controladores	HU5
5	Generar Páginas	HU6
6	Descargar el código generado.	HU7
7	Editar los atributos de las tablas que se obtengan de la base de datos.	HU2

Tabla 7 Product Backlog[8]

2.1.3 PRODUCT BACKLOG REFINADO

El Product Backlog refinado es realizado por el equipo de desarrollo, en base al Product Backlog inicial, ya que en este se definen los requerimientos del usuario de manera general. Además la elaboración del Product Back Log Refinado, ayuda en la preparación de los sprints que se realizarán a futuro.

En la siguiente tabla, se detallan las tareas a realizarse para cada requerimiento:

Orden	Requerimiento	Tareas
1	Diseño de la Arquitectura y funcionalidad de	1.1.-Establecer la arquitectura y funcionalidad de la aplicación.

[8] Tabla realizada por los autores.

	la aplicación	1.2.-Modelo conceptual de BDD 1.3.-Modelo físico de BDD 1.4.-Generar código SQL
2	Conexión a la Base de Datos del Cliente.	2.1.- Establecer los parámetros de conexión a la BDD del Cliente. 2.2.- Realizar la conexión a la BDD del cliente.
3	Extracción de la Metadata de la Base de Datos del Cliente.	3.1.- Extraer el nombre de esquemas y tablas. 3.2.- Extraer el nombre de los atributos de cada tabla. 3.3.- Extraer el tipo de dato de cada uno de los atributos. 3.4.- Establecer el tipo de datos con que serán mapeados los atributos de la tabla (Tipo de datos Java). 3.5.- Extraer las restricciones del tipo NOT NULL para cada uno de los atributos. 3.6.- Extraer las claves primarias de cada una de las Tablas (Pk). 3.7.- Extraer las claves foráneas de cada una de las Tablas (Fk). 3.8.- Extraer los atributos de tipo Autogenerado (Serial). 3.9.- Guardar toda la información obtenida en la Base de Datos de la Aplicación.
4	Generación de Entidades.	4.1.- Consultar la información de cada una de las tablas y sus atributos, desde la base de datos de la aplicación. 4.2.- Crear el archivo de extensión .java. 4.3.-Escribir en el archivo el código

		correspondiente, de acuerdo a la información consultada.
5	Generación de Servicios.	5.1.-Crear el archivo de extensión ServicioBase.java. 5.2.-Leer y Copiar el contenido del archivo ServicioBase.txt, en el archivo ServicioBase.java. 5.3.- Consultar la información de cada una de las tablas y sus atributos, desde la base de datos de la aplicación. 5.4.-Crear el archivo de extensión .java. 5.5.-Escribir en el archivo el código correspondiente, de acuerdo a la información consultada.
6	Generación de Controladores.	6.1.- Consultar la información de cada una de las tablas y sus atributos, desde la base de datos de la aplicación. 6.2.- Crear el archivo de extensión .java. 6.3.-Escribir en el archivo el código correspondiente, de acuerdo a la información consultada.
7	Generación de Páginas.	7.1.- Consultar la información de cada una de las tablas y sus atributos, desde la base de datos de la aplicación. 7.2.- Crear el archivo de extensión .xhtml. 7.3.-Escribir en el archivo el código

		correspondiente, de acuerdo a la información consultada.
8	Descarga de los archivos generados.	8.1.- Comprimir todo los archivos, con su respectiva jerarquía de directorios, en un archivo de extensión .zip. 8.2.- Generar el link de descarga del archivo zip.
9	Edición de los atributos de las Tablas.	9.1.-Consultar la información de cada una de las tablas y sus atributos, desde la base de datos de la aplicación. 9.2.- Mostrar los atributos consultados, en una tabla que permita la edición de los atributos permitidos, como son: nombre, tipo de dato y restricción NOT NULL. 9.3.- Guardar los cambios, en la base de datos de la aplicación.

Tabla 8 Product Backlog Refinado[9]

2.1.4 RESTRICCIONES

CODGEN, al ser un prototipo, tiene restricciones de funcionamiento, las mismas que se han obtenido de los requerimientos y se detallan a continuación:

- Las relaciones existentes en las bases de datos que serán soportadas por la aplicación son las siguientes:
  - Uno a uno
  - Uno a muchos

[9] Tabla realizada por los autores.

- Muchos a muchos
- Relaciones recursivas
- Tablas no relacionadas

- En el caso de las relaciones de uno a muchos, no se permitirá la eliminación de datos de los cuales haya dependencia. La actualización se permitirá únicamente para los campos no relacionados, es decir la clave primaria no se podrá actualizar si se da la situación explicada anteriormente.

- Quedará a responsabilidad del usuario los nombres de los campos que ponga en las respectivas tablas y columnas. Si se pusieran palabras reservadas, el sistema actualmente no las valida y por tanto, producirán errores en el código generado.

- El sistema genera código actualmente compatible con jboss 7.1.1. final, PrimeFaces 4.0 , JDK 7, se podrá ejecutar el código generado en un navegador actual, es decir, que no tenga más de 6 meses sin ser actualizado, tomando en cuenta la fecha en que se ejecute la aplicación.

- Actualmente la aplicación funciona para las bases de datos que estén diseñadas en Postgres.

- La aplicación solo va a mapear tipos de datos específicos.

- Si la base de datos contiene relaciones que tengan fallas de diseño, no se asegurará que el código generado sea correcto.

- La aplicación está destinada para el uso de desarrolladores, es decir, usuarios que tengan un adecuado conocimiento del diseño y manejo de base de datos, además de conocimiento de lenguaje Java.

2.1.5 PRERREQUISITOS

- La aplicación debe estar levantada en un servidor Jboss 7.1.1 Final.
- La conexión remota debe estar permitida en la base de datos del cliente.
- Las bases de datos que se vayan a utilizardeberán estar creadas.

2.2 SPRINTS

2.2.1 ESPECIFICACIÓN DE LOS SPRINTS

2.2.1.1 Primer Sprint

En el primer Sprint, se realizaran las tareas indicadas en el requerimiento número 1, que consta en el Product Backlog Refinado (Tabla 8).

En este Sprint las tareas están dirigidas al diseño de la aplicación CODGEN.

Las tareas a realizarse, el tiempo estimado que tomará su desarrollo y el/los responsable(s) de cada una de las tareas se especifican a continuación en la siguiente tabla:

Nº	Tarea	Tiempo Estimado
1	Establecer la arquitectura y funcionalidad de la aplicación.	16 horas
2	Modelo conceptual de BDD.	8 horas
3	Modelo físico de BDD.	4 horas
4	Generar código SQL.	4 horas
5	Pruebas Unitarias	4 horas
6	Control de Calidad	4 horas
		40 horas

Tabla 9Primer Sprint CODGEN.[10]

[10]Tabla realizada por los autores.

2.2.1.1.1 *Planificación del Sprint*

Para la administración de los Sprints que se realizarán a lo largo del desarrollo de CODGEN se va a utilizar una tabla, la cual se detalla a continuación:

Proyecto			
CODGEN			
Nº de sprint	**Inicio**	**Días**	**Jornada**
1	30-sep-2013	10	4 horas/día
TAREAS		**EQUIPO**	**FESTIVOS**
TIPOS	**ESTADOS**		
Análisis	Pendiente	Ing. Carlos Montenegro	
Codificación	En curso	Jairo Martínez	
Prototipado	Terminada	Jorge Rodríguez	
Pruebas	Eliminada		
Reunión			

Tabla 10 Configuración del Primer Sprint[11].
Fuentes:[19]

La siguiente ilustración muestra el esfuerzo realizado en el desarrollo del primer sprint.

SPRINT	INICIO	DURACIÓN
1	30-sep.-13	10

	L 30-sep.	M 1-oct.	X 2-oct.	J 3-oct.	V 4-oct.	L 7-oct.	M 8-oct.	X 9-oct.	J 10-oct.	L 14-oct.
Tareas pendientes	6	6	6	6	5	5	4	3	2	1
Horas de trabajo pendientes	4	4	4	4	4	4	4	4	4	4

PILA DEL SPRINT					ESFUERZO										
Backlog	Tarea	Tipo	Estado	Responsable											
	Establecer la arquitectura y funcionali	Análisis	Terminada	Jairo/Jorge	4	4	4	4							
	Modelo conceptual de BDD	Análisis	Terminada	Jairo/Jorge					4	4					
	Modelo físico de BDD	Análisis	Terminada	Jairo/Jorge							4				
	Generar código SQL	Codificación	Terminada	Jairo/Jorge								4			
	Pruebas Unitarias	Pruebas	Terminada	Jairo/Jorge									4		
	Control de Calidad	Pruebas	Terminada	Jairo/Jorge										4	

Figura 7 Esfuerzo Realizado en el Primer Sprint[12].
Fuente: [19]

[11] Plantilla elaborada en base a la fuente.
[12] Plantilla elaborada en base a la fuente.

La siguiente ilustración es conocida como el gráfico Burn Down Chart, y en esta se muestra la carga horaria de trabajo que se hace día a día. En el eje Y se pueden ver las horas de trabajo pendientes, y en el eje X se muestran los días. Como se puede ver, la carga horaria fue correctamente estimada, y se cumple con el trabajo estimado, dentro del tiempo previsto.

Figura 8 Burn Down Chart Primer Sprint[13].
Fuente: [19]

En el siguiente gráfico se muestra el número de tareas pendientes vs los días en los que se está previsto realizar el trabajo. Aquí se puede ver claramente, que la duración de las tareas es diferente, pero que la estimación de tiempo es correcta.

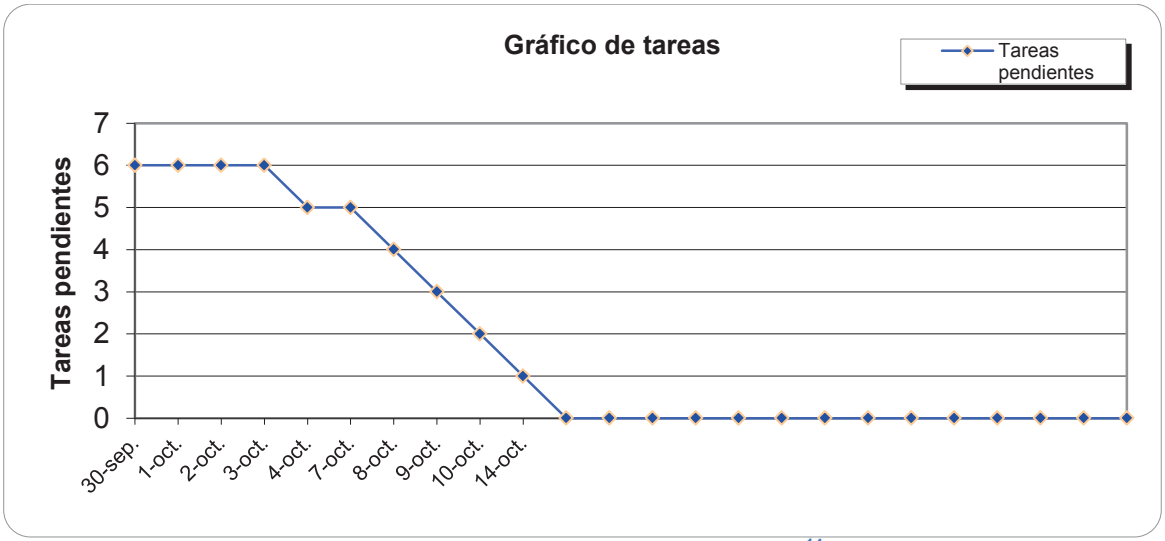


Figura 9Avance de tareas Primer Sprint[14]
Fuente: [19]

[13] Plantilla elaborada en base a la fuente.
[14] Plantilla elaborada en base a la fuente.

2.2.1.2 Segundo Sprint

En este segundo sprint, se realizarán las tareas relacionas al requerimiento 2, las cuales se detallan en el Product Backlog Refinado (Tabla 8).

Las tareas que se llevarán a cabo en este sprint están orientadas, a definir los parámetros que permitirán establecer la conexión con la base de datos del cliente.

Estas tareas tendrán un tiempo estimado, dependiendo de su complejidad. Esta información se detalla en la siguiente tabla:

Nº	Tarea	Tiempo Estimado
1	Establecer los parámetros de conexión a la BDD del Cliente.	8 horas
2	Realizar la conexión a la BDD del cliente.	10 horas
3	Aseguramiento de calidad.	4 horas
		20 horas

Tabla 11 Segundo Sprint CODGEN.[15]

2.2.1.2.1 *Planificación del Sprint*

Para la dirección de este sprintse empleara una tabla siguiendo el formato del sprint anterior.

Proyecto			
CODGEN			
Nº de sprint	**Inicio**	**Días**	**Jornada**
2	15-oct-2013	5	4 horas/día
TAREAS		**EQUIPO**	**FESTIVOS**

[15] Tabla realizada por los autores.

TIPOS	ESTADOS		
Análisis	Pendiente	Ing. Carlos Montenegro	
Codificación	En curso	Jairo Martínez	
Prototipado	Terminada	Jorge Rodríguez	
Pruebas	Eliminada		
Reunión			

Tabla 12 Configuración del Segundo Sprint[16].
Fuentes:[19]

La siguiente figura permite apreciar el esfuerzo realizado en el segundo sprint, además se especifica el estado de las tareas, así como la persona del equipo que ha sido asignada como responsable.

SPRINT	INICIO	DURACIÓN
2	15-oct.-13	5

	M	X	J	V	L
	15-oct.	16-oct.	17-oct.	18-oct.	21-oct.
Tareas pendientes	3	2	2	2	1
Horas de trabajo pendientes	4	4	4	4	4

Tarea	Tipo	Estado	Responsable	ESFUERZO
Establecer los parámetros de conexión a la BDD del Cliente.	Análisis	Terminada	Jorge	4 2
Realizar la conexión a la BDD del cliente.	Codificación	Terminada	Jairo	2 4 4
Aseguramiento de calidad.	Pruebas	Terminada	Jorge	4

Figura 10Esfuerzo Realizado en el Segundo Sprint[17].
Fuentes:[19]

En las figuras 11 y 12 se puede observar con claridad el esfuerzo y el progreso con las tareas correspondientes al segundo sprint.

[16] Plantilla elaborada en base a la fuente.
[17] Plantilla elaborada en base a la fuente.

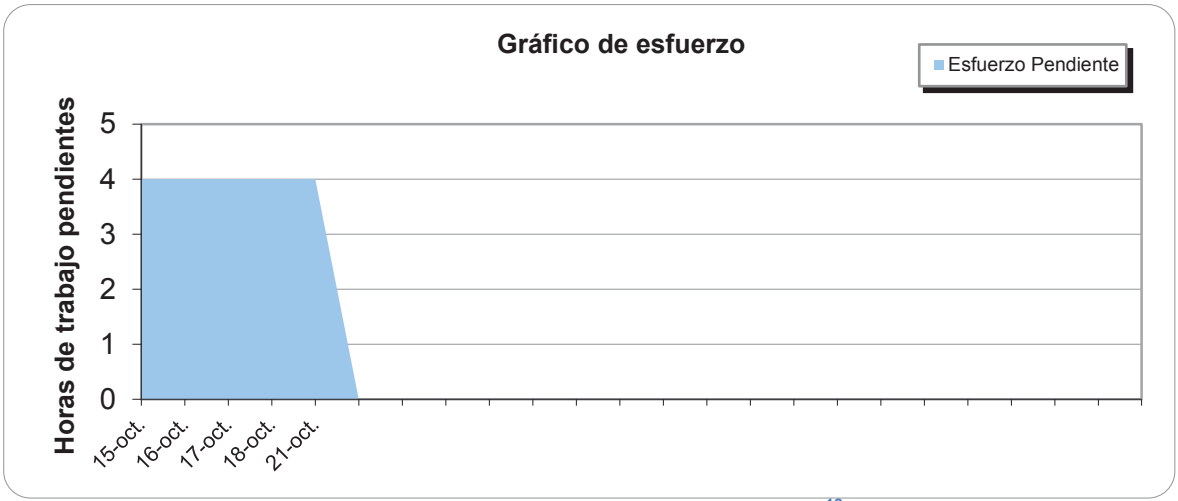


Figura 11 Burn Down Chart Segundo Sprint[18].
Fuentes:[19]

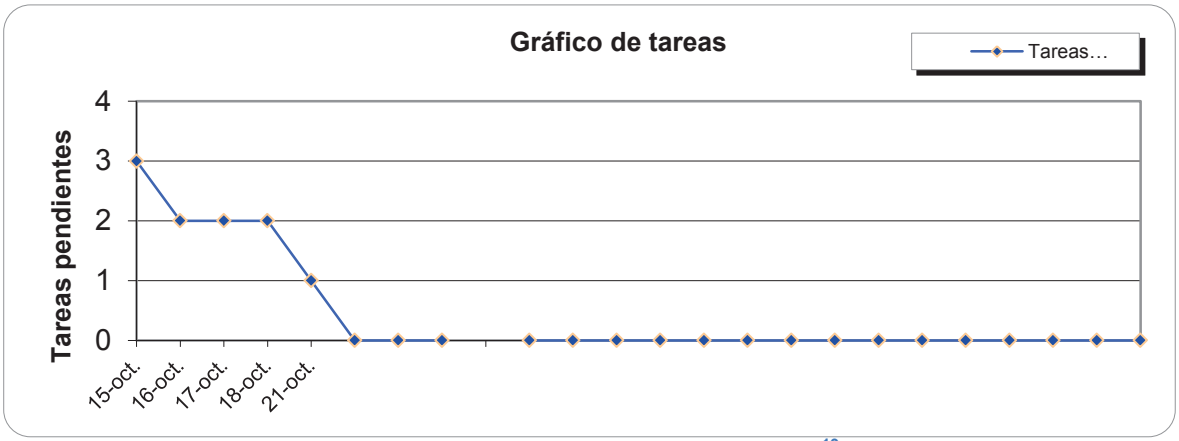


Figura 12Avance de tareas Segundo Sprint[19].
Fuentes:[19]

2.2.1.3 Tercer Sprint

En este sprint, se realizarán las tareas correspondientes al requerimiento 3, detalladas en el Product Backlog Refinado.

Las tareas de este tercer sprint tiene el objetivo de, extraer la metadata de la base de datos del cliente y posteriormente filtrar los datos que sean suficientes y necesarios para que el mapeo se pueda realizar correctamente. Estos datos

[18] Plantilla elaborada en base a la fuente.
[19] Plantilla elaborada en base a la fuente.

serán almacenados en la base de datos empleada CODGEN, para facilitar la edición de los mismos.

Dichas tareas tendrán un tiempo estimado para ser realizadas. Este tiempo se muestra en la siguiente tabla.

Nº	Tarea	Tiempo Estimado
1	Extraer el nombre de esquemas y tablas.	4 horas
2	Extraer el nombre de los atributos de cada tabla.	4 horas
3	Extraer el tipo de dato de cada uno de los atributos.	4 horas
4	Establecer el tipo de datos con que serán mapeados los atributos de la tabla (Tipo de datos Java).	2 horas
5	Extraer las restricciones del tipo NOT NULL para cada uno de los atributos.	4 horas
6	Extraer las claves primarias de cada una de las Tablas (Pk).	5 horas
7	Extraer las claves foráneas de cada una de las Tablas (Fk).	5 horas
8	Extraer los atributos de tipo Autogenerado (Serial).	4 horas
9	Guardar toda la información obtenida en la Base de Datos de la Aplicación.	4 horas
10	Aseguramiento de la Calidad.	4 horas
		40 horas

Tabla 13 Tercer Sprint CODGEN.[20]

[20] Tabla realizada por los autores.

2.2.1.3.1 *Planificación del Sprint*

Para la dirección de este sprintse empleara la siguiente tabla, manteniendo el formato del sprint anterior.

Proyecto			
CODGEN			
Nº de sprint	**Inicio**	**Días**	**Jornada**
3	22-oct-2013	10	4 horas/día
TAREAS		**EQUIPO**	**FESTIVOS**
TIPOS	**ESTADOS**		
Análisis	Pendiente	Ing. Carlos Montenegro	
Codificación	En curso	Jairo Martínez	
Prototipado	Terminada	Jorge Rodríguez	
Pruebas	Eliminada		
Reunión			

Tabla 14 Configuración del Tercer Sprint[21].
Fuentes:[19]

La siguiente figura muestra el esfuerzo realizado con respecto al sprint backlog del tercer sprint. De manera similar a los sprints anteriores, se puede apreciar los responsables y el estado de cada tarea.

[21] Plantilla elaborada en base a la fuente.

SPRINT	INICIO	DURACIÓN
3	22-oct.-13	10

	M 22-oct.	X 23-oct.	J 24-oct.	V 25-oct.	L 28-oct.	M 29-oct.	X 30-oct.	J 31-oct.	V 1-nov.	L 4-nov.
Tareas pendientes	10	9	8	7	6	5	4	3	2	1
Horas de trabajo pendientes	4	4	4	4	4	4	4	4	4	4

Tarea	Tipo	Estado	Responsable									
Extraer el nombre de catálogos y tablas.	Codificación	Terminada	Jorge	4								
Extraer el nombre de los atributos de cada tabla.	Codificación	Terminada	Jairo		4							
Extraer el tipo de dato de cada uno de los atributos.	Codificación	Terminada	Jorge			4						
Establecer el tipo de datos con que serán mapeados los atributos de la tabla (Tipo de datos Java).	Análisis	Terminada	Jairo				2					
Extraer las restricciones del tipo NOT NULL para cada uno de los atributos.	Codificación	Terminada	Jairo				2	2				
Extraer las claves primarias de cada una de las Tablas (Pk).	Codificación	Terminada	Jairo					2	2	1		
Extraer las claves foráneas de cada una de las Tablas (Fk).	Codificación	Terminada	Jorge						2	2	1	
Extraer los atributos de tipo Autogenerado (Serial).	Codificación	Terminada	Jairo							1	3	
Guardar toda la información obtenida en la Base de Datos de la Aplicación.	Codificación	Terminada	Jorge									4
Aseguramiento de la Calidad	Pruebas	Terminada	Jorge									4

Figura 13Esfuerzo Realizado en el Tercer Sprint[22].
Fuentes:[19]

En las figuras 14 y 15 se puede visualizar con claridad el esfuerzo y el progreso con las tareas respectivamente.

Figura 14 Burn Down Chart Tercer Sprint[23].
Fuentes:[19]

[22] Plantilla elaborada en base a la fuente.
[23] Plantilla elaborada en base a la fuente.

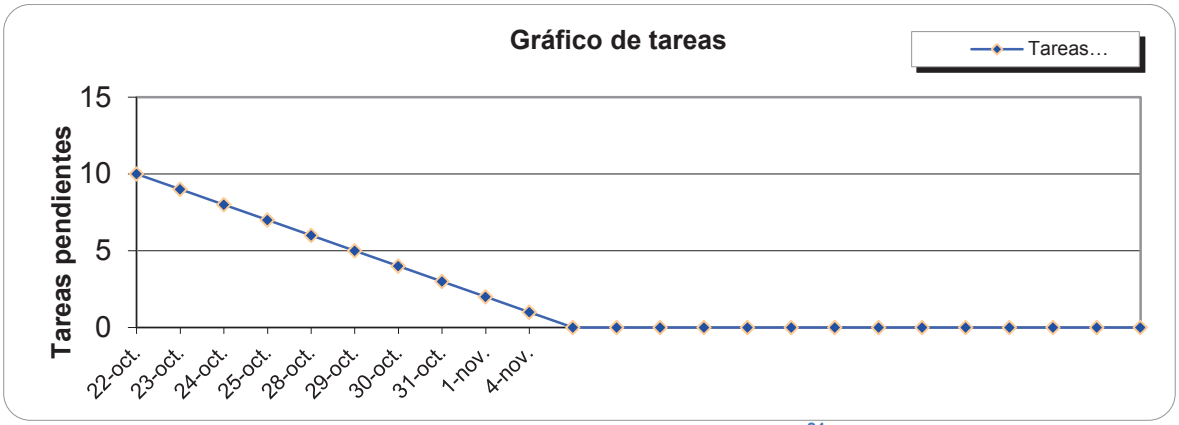


Figura 15Avance de tareas Tercer Sprint[24].
Fuentes:[19]

2.2.1.4 Cuarto Sprint

Las tareas que se realizarán en el cuarto sprint, son las tareas que están indicadas en el cuarto requerimiento del Product Backlog refinado.

Las tareas a realizarse en este sprint, están dirigidas hacia la generación de entidades, las mismas que forman parte de la capa de acceso a datos.

Las tareas a realizarse, el tiempo estimado que tomará su desarrollo y el/los responsable(s) de cada una de las tareas se especifican a continuación en la siguiente tabla:

Nº	Tarea	Tiempo Estimado
1	Consultar información de cada una de las tablas y sus atributos, desde la base de datos de la aplicación.	4 horas
2	Crear el archivo con extensión .java	4 horas
3	Escribir en el archivo el código correspondiente, de acuerdo a la información consultada.	24 horas

[24] Plantilla elaborada en base a la fuente.

4	Aseguramiento de Calidad	8 horas
		40 horas

Tabla 15 Cuarto Sprint CODGEN.[25]

2.2.1.4.1 *Planificación del Sprint*

A continuación se muestra la tabla de configuración del cuarto sprint.

Proyecto			
CODGEN			
Nº de sprint	**Inicio**	**Días**	**Jornada**
4	5-nov-2013	10	4 horas/día
TAREAS		**EQUIPO**	**FESTIVOS**
TIPOS	**ESTADOS**		
Análisis	Pendiente	Jorge Rodríguez	
Codificación	En curso	Jairo Martínez	
Prototipado	Terminada		
Pruebas	Eliminada		
Reunión			

Tabla 16 Configuración del Cuarto Sprint[26].
Fuentes:[19]

La siguiente ilustración muestra el esfuerzo realizado en el desarrollo del cuarto sprint.

[25] Tabla realizada por los autores.
[26] Plantilla elaborada en base a la fuente.

SPRINT	INICIO	DURACIÓN
4	5-nov.-13	10

	M	X	J	V	L	M	X	J	V	L
	5-nov.	6-nov.	7-nov.	8-nov.	11-nov.	12-nov.	13-nov.	14-nov.	15-nov.	18-nov.
Tareas pendientes	5	5	5	5	4	4	3	2	2	1
Horas de trabajo pendientes	4	4	4	4	4	4	4	4	4	4

PILA DEL SPRINT

Tarea	Tipo	Estado	Responsable	M 5-nov.	X 6-nov.	J 7-nov.	V 8-nov.	L 11-nov.	M 12-nov.	X 13-nov.	J 14-nov.	V 15-nov.	L 18-nov.
Consultar la información de cada una de las tablas y sus atributos, desde la base de datos de la aplicación.	Codificación	Terminada	Jorge	4									
Crear el archivo de extensión .java.	Codificación	Terminada	Jairo		4								
Escribir en el archivo el código correspondiente, de acuerdo a la información consultada.	Codificación	Terminada	Jorge			4	4	4	4	4	4		
Aseguramiento de la Calidad	Pruebas	Terminada	Jairo									4	4

Figura 16Esfuerzo Realizado en el Cuarto Sprint[27].
Fuentes:[19]

En el siguiente gráfico se muestra el Burn Down Chart, el mismo que muestra la información sobre la carga de trabajo pendiente diariamente.

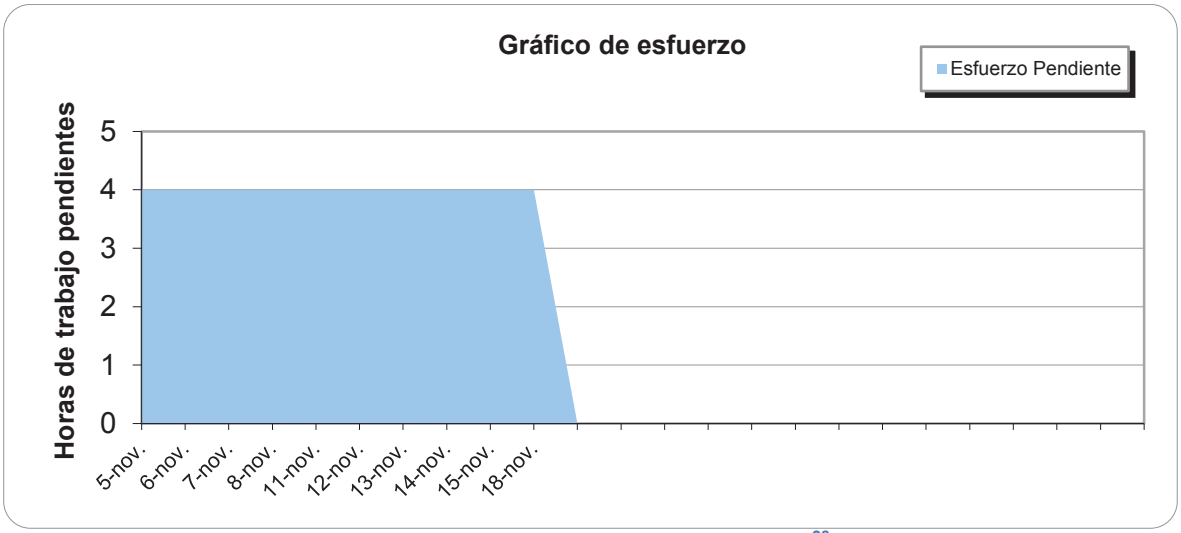


Figura 17 Burn Down Chart Cuarto Sprint[28].
Fuentes:[19]

El siguiente gráfico muestra el número de tareas pendientes para cada día, dentro del sprint.

[27] Plantilla elaborada en base a la fuente.
[28] Plantilla elaborada en base a la fuente.

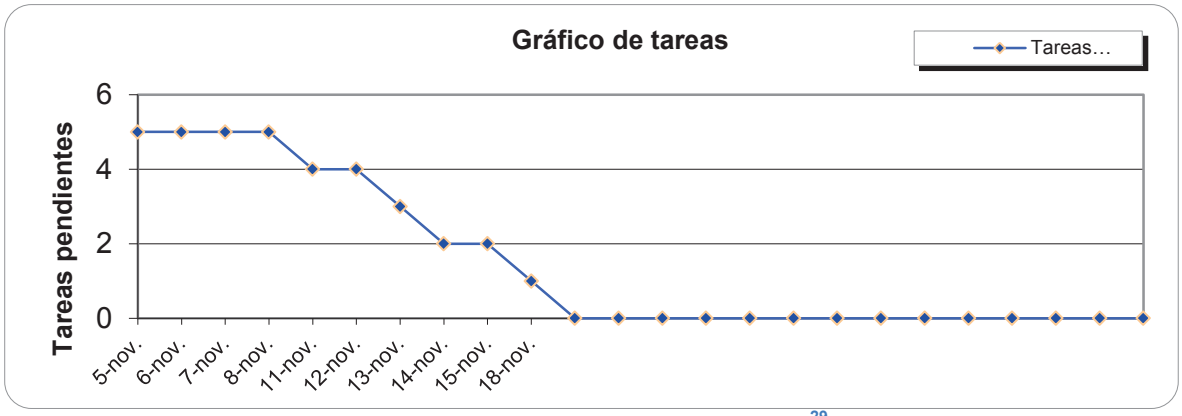


Figura 18Avance de tareas Cuarto Sprint[29].
Fuentes:[19]

2.2.1.5 Quinto Sprint

Las tareas que se realizarán en el cuarto sprint, son las tareas que están indicadas en el quinto requerimiento del Product Backlog refinado.

En esta parte del desarrollo, se efectuará el desarrollo de los servicios, los cuales forman parte de la capa de la lógica del negocio.

Las tareas a realizarse, el tiempo estimado que tomará su desarrollo y el/los responsable(s) de cada una de las tareas se especifican a continuación en la siguiente tabla.

Nº	Tarea	Tiempo Estimado
1	Crear el archivo de nombre ServicioBase.java.	4 horas
2	Leer y Copiar el contenido del archivo ServicioBase.txt, en el archivo ServicioBase.java.	6 horas
3	Consultar la información de cada una de las tablas y sus atributos, desde la base de	4 horas

[29] Plantilla elaborada en base a la fuente.

	datos de la aplicación.	
4	Crear el archivo de extensión .java.	4 horas
5	Escribir en el archivo el código correspondiente, de acuerdo a la información consultada	8 horas
6	Aseguramiento de la Calidad	4 horas
		30 horas

Tabla 17Quinto Sprint CODGEN.[30]

2.2.1.5.1 *Planificación del Sprint*

A continuación se muestra la tabla de configuración del quinto Sprint.

Proyecto			
CODGEN			
Nº de sprint	**Inicio**	**Días**	**Jornada**
5	19-nov-2013	8	4 horas/día
TAREAS		**EQUIPO**	**FESTIVOS**
TIPOS	**ESTADOS**		
Análisis	Pendiente	Jorge Rodríguez	
Codificación	En curso	Jairo Martínez	
Prototipado	Terminada		
Pruebas	Eliminada		
Reunión			

Tabla 18 Configuración del Quinto Sprint[31].
Fuentes:[19]

[30] Tabla realizada por los autores.
[31] Plantilla elaborada en base a la fuente.

La siguiente imagen muestra el esfuerzo realizado en el quinto sprint, en cada uno de los días asignados.

SPRINT	INICIO	DURACIÓN
5	19-nov.-13	8

	M 19-nov.	X 20-nov.	J 21-nov.	V 22-nov.	L 25-nov.	M 26-nov.	X 27-nov.	J 28-nov.
Tareas pendientes	6	5	5	4	3	2	2	1
Horas de trabajo pendientes	4	4	4	4	4	4	4	2

PILA DEL SPRINT

Tarea	Tipo	Estado	Responsable							
Crear el archivo de nombre ServicioBase.java	Codificación	Terminada	Jorge	4								
Leer y copiar el contenido del archivo ServicioBase.txt, en el archivo ServicioBase.java	Codificación	Terminada	Jairo		4	2						
Consultar la información de cada una de las tablas y sus atributos, desde la base de datos de la aplicación.	Codificación	Terminada	Jorge				4					
Crear el archivo de extensión .java.	Codificación	Terminada	Jairo			2		2				
Escribir en el archivo el código correspondiente, de acuerdo a la información consultada.	Codificación	Terminada	Jorge						4	4		
Aseguramiento de la Calidad	Pruebas	Terminada	Jairo					2			2	

Figura 19 Esfuerzo realizado en el Quinto Sprint[32].
Fuentes:[19]

A continuación se puede ver el Burn Down Chart, que permite visualizar la carga de trabajo diaria.

Figura 20 Burn Down Chart Quinto Sprint[33].
Fuentes:[19]

[32] Plantilla elaborada en base a la fuente.
[33] Plantilla elaborada en base a la fuente.

En el siguiente gráfico, se muestra información acerca de las tareas pendientes para cada día.

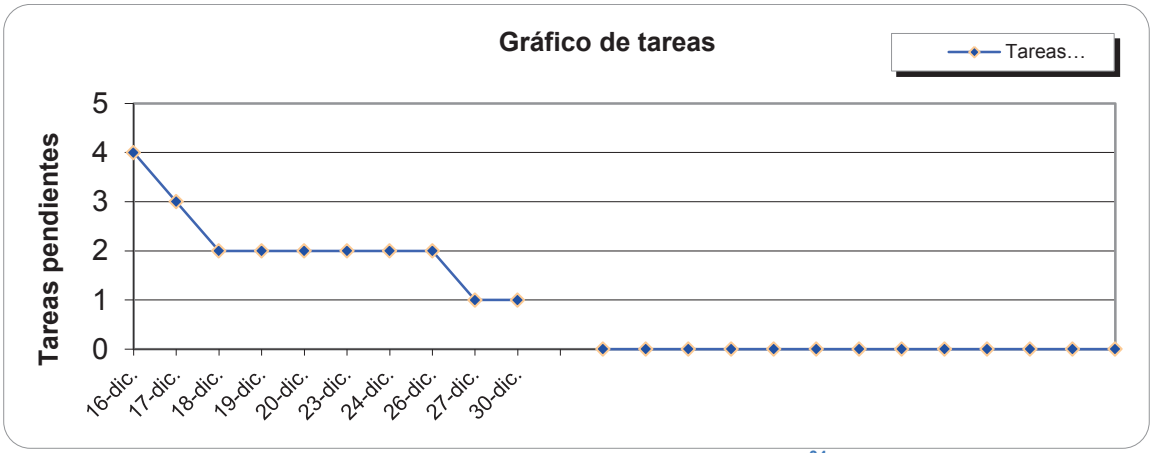


Figura 21Avance de tareas Quinto Sprint[34].
Fuentes:[19]

2.2.1.6 Sexto Sprint

En el presente sprint, se van a llevar a cabo actividades cuyo desarrollo está orientado a la generación del código de una parte de la capa de presentación, es decir los controladores.

Las tareas a realizarse, el tiempo estimado que tomará su desarrollo y el/los responsable(s) de cada una de las tareas se especifican a continuación en la siguiente tabla.

Nº	Tarea	Tiempo Estimado
1	Consultar la información de cada una de las tablas y sus atributos, desde la base de datos de la aplicación.	4 horas
2	Crear el archivo de extensión	4 horas

[34] Plantilla elaborada en base a la fuente.

	.java.	
3	Escribir en el archivo el código correspondiente, de acuerdo a la información consultada.	24 horas
4	Aseguramiento de la Calidad	8 horas
		40 horas

Tabla 19 Sexto Sprint CODGEN.[35]

2.2.1.6.1 *Planificación del Sprint*

A continuación se muestra la tabla de configuración del sexto sprint.

Proyecto			
CODGEN			
Nº de sprint	**Inicio**	**Días**	**Jornada**
6	29-nov-2013	10	4 horas/día
TAREAS		**EQUIPO**	**FESTIVOS**
TIPOS	**ESTADOS**		
Análisis	Pendiente	Jorge Rodríguez	6-dic-2013
Codificación	En curso	Jairo Martínez	
Prototipado	Terminada		
Pruebas	Eliminada		
Reunión			

Tabla 20 Configuración del Sexto Sprint.
Fuentes:[19]

A continuación se muestra una imagen que permite visualizar el esfuerzo a realizarse en cada tarea del sprint.

[35] Tabla realizada por los autores.

SPRINT	INICIO	DURACIÓN
6	29-nov.-13	11

	V	L	M	X	J	L	M	X	J	V
	29-nov.	2-dic.	3-dic.	4-dic.	5-dic.	9-dic.	10-dic.	11-dic.	12-dic.	13-dic.
Tareas pendientes	4	3	2	2	2	2	2	2	1	1
Horas de trabajo pendientes	4	4	4	4	4	4	4	4	4	4

PILA DEL SPRINT

Tarea	Tipo	Estado	Responsable										
Consultar la información de cada una de las tablas y sus atributos, desde la base de datos de la aplicación.	Codificación	Terminada	Jorge	4									
Crear el archivo de extensión .java.	Codificación	Terminada	Jairo		4								
Escribir en el archivo el código correspondiente, de acuerdo a la información consultada.	Codificación	Terminada	Jairo/Jorge			4	4	4	4	4	4		
Aseguramiento de la Calidad	Pruebas	Terminada	Jairo/Jorge									4	4

Figura 22Esfuerzo realizado en el Sexto Sprint[36].
Fuentes:[19]

A continuación se muestra el Burn Down Chart para el presente Sprint.

Figura 23 Burn Down Chart Sexto Sprint[37].
Fuentes:[19]

El siguiente gráfico muestra las tareas pendientes que se realizarán en cada día del Sprint.

[36] Plantilla elaborada en base a la fuente.
[37] Plantilla elaborada en base a la fuente.

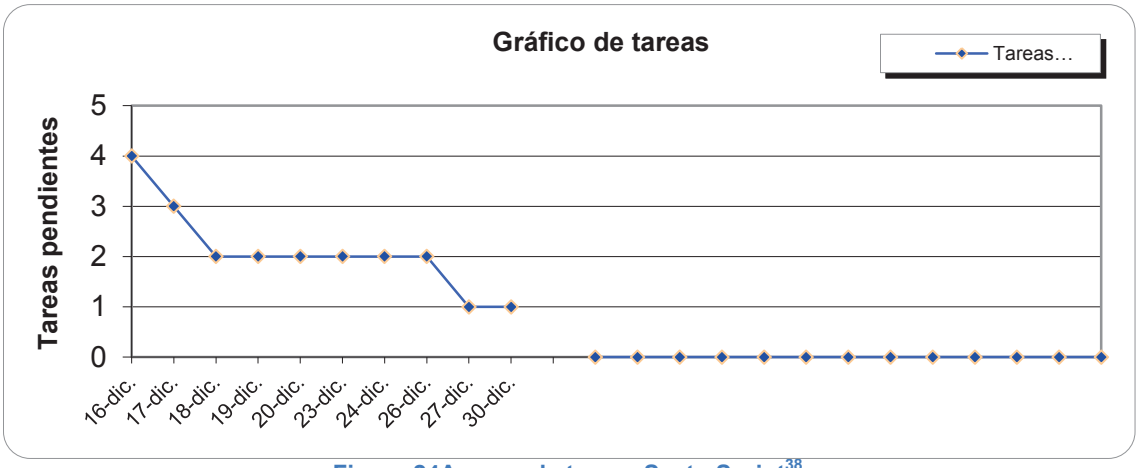


Figura 24Avance de tareas Sexto Sprint[38].
Fuentes:[19]

2.2.1.7 Séptimo Sprint

En el presente Sprint, se realizará la creación de las páginas, mediante algunas actividades que se detallan a continuación.

Nº	Tarea	Tiempo Estimado
1	Consultar la información de cada una de las tablas y sus atributos, desde la base de datos de la aplicación.	4 horas
2	Crear el archivo de extensión .xhtml.	4 horas
3	Escribir en el archivo el código correspondiente, de acuerdo a la información consultada.	24 horas
4	Aseguramiento de la Calidad	8 horas
		40 horas

Tabla 21Séptimo Sprint CODGEN.[39]

[38] Plantilla elaborada en base a la fuente.
[39] Tabla realizada por los autores.

2.2.1.7.1 *Planificación del Sprint*

A continuación se muestra la tabla de configuración del séptimo sprint.

Proyecto			
CODGEN			
Nº de sprint	**Inicio**	**Días**	**Jornada**
7	16-dic-2013	10	4 horas/día
TAREAS		**EQUIPO**	**FESTIVOS**
TIPOS	**ESTADOS**		
Análisis	Pendiente	Jorge Rodríguez	25-dic-2013
Codificación	En curso	Jairo Martínez	
Prototipado	Terminada		
Pruebas	Eliminada		
Reunión			

Tabla 22 Configuración del Séptimo Sprint[40].
Fuentes:[19]

La siguiente imagen muestra la planificación de las actividades, es decir su carga horaria.

[40] Plantilla elaborada en base a la fuente.

SPRINT	INICIO	DURACIÓN
7	16-dic.-13	10

	L 16-dic.	M 17-dic.	X 18-dic.	J 19-dic.	V 20-dic.	L 23-dic.	M 24-dic.	J 26-dic.	V 27-dic.	L 30-dic.
Tareas pendientes	4	3	2	2	2	2	2	2	1	1
Horas de trabajo pendientes	4	4	4	4	4	4	4	4	4	4

PILA DEL SPRINT

Tarea	Tipo	Estado	Responsable									ESFU
Consultar la información de cada una de las tablas y sus atributos, desde la base de datos de la aplicación.	Codificación	Terminada	Jorge	4								
Crear el archivo de extensión .xhtml.	Codificación	Terminada	Jairo		4							
Escribir en el archivo el código correspondiente, de acuerdo a la información consultada.	Codificación	Terminada	Jairo/Jorge			4	4	4	4	4	4	
Aseguramiento de la Calidad	Pruebas	Terminada	Jairo/Jorge								4	4

Figura 25 Esfuerzo realizado en el Séptimo Sprint[41].
Fuentes:[19]

En la siguiente gráfica, se puede ver el esfuerzo pendiente por cada día, conocido como Burn Down Chart.

Figura 26 Burn Down Chart Séptimo Sprint[42].
Fuentes:[19]

La siguiente gráfica muestra las tareas pendientes para cada día del Sprint.

[41] Plantilla elaborada en base a la fuente.
[42] Plantilla elaborada en base a la fuente.

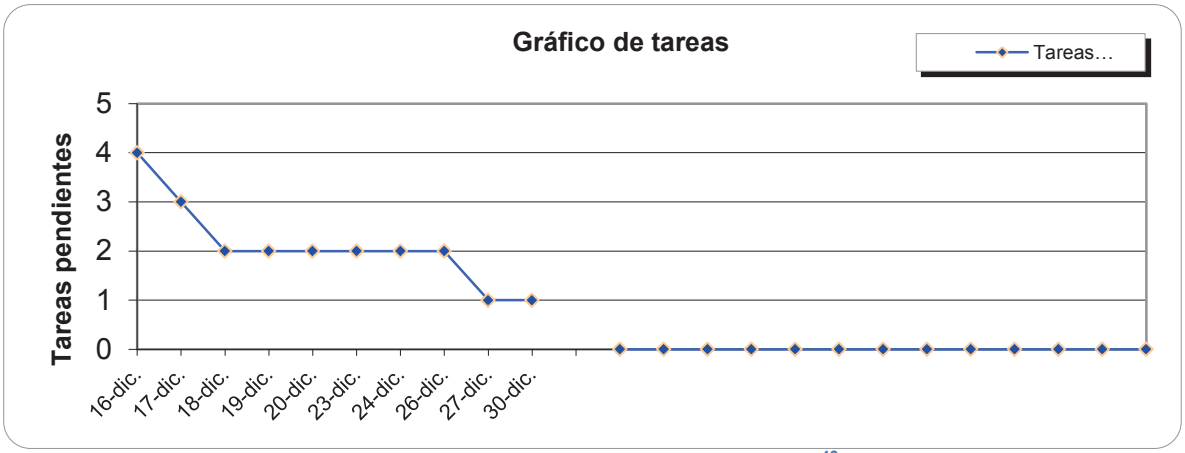


Figura 27 Avance de tareas Séptimo Sprint[43].
Fuentes:[19]

2.2.1.8 Octavo Sprint

En este sprint, se realizarán tareas referentes al 8 requerimiento del product backlog refinado. Estas tareas corresponden a comprimir todos los archivos generados y permitir la descarga de los mismos en un solo enlace.

A continuación se detallan dichas tareas.

Nº	Tarea	Tiempo Estimado
1	Comprimir todos los archivos, con su respectiva jerarquía de directorios, en un archivo de extensión .zip.	10 horas
2	Generar el link de descarga del archivo zip.	6 horas
3	Aseguramiento de la Calidad.	4 horas
		20 horas

Tabla 23 Octavo Sprint CODGEN.[44]

[43] Plantilla elaborada en base a la fuente.
[44] Tabla realizada por los autores.

2.2.1.8.1 *Planificación del Sprint*

En la siguiente tabla se presenta la configuración del octavo sprint.

Proyecto			
CODGEN			
Nº de sprint	**Inicio**	**Días**	**Jornada**
8	31-dic-2013	5	4 horas/día
TAREAS		**EQUIPO**	**FESTIVOS**
TIPOS	**ESTADOS**		
Análisis	Pendiente	Jorge Rodríguez	31-dic-2013
Codificación	En curso	Jairo Martínez	04-ene-2014
Prototipado	Terminada		
Pruebas	Eliminada		
Reunión			

Tabla 24 Configuración del Octavo Sprint.[45]
Fuentes:[19]

En la siguiente figura se presenta la planificación y esfuerzo realizado para cada una de las tareas del octavo sprint.

[45] Plantilla elaborada en base a la fuente.

SPRINT	INICIO	DURACIÓN
8	31-dic.-13	5

	M 31-dic.	J 2-ene.	V 3-ene.	L 6-ene.	M 7-ene.
Tareas pendientes	3	3	2	2	1
Horas de trabajo pendientes	4	4	4	4	4

Tarea	Tipo	Estado	Responsable						
Comprimir todo los archivos, con su respectiva jerarquía de directorios, en un archivo de extensión .zip.	Codificación	Terminada	Jairo	4	4	2			
Generar el link de descarga del archivo zip.	Codificación	Terminada	Jorge			2	4		
Aseguramiento de la Calidad	Pruebas	Terminada	Jorge					4	

Figura 28Esfuerzo realizado en el Octavo Sprint.[46]
Fuentes:[19]

A continuación se presentan gráficas relativas al esfuerzo y avance de las tareas, respectivamente.

Figura 29 Burn Down Chart Octavo Sprint.[47]
Fuentes:[19]

[46] Plantilla elaborada en base a la fuente.
[47] Plantilla elaborada en base a la fuente.

Figura 30Avance de tareas Octavo Sprint.
Fuentes:[19]

2.2.1.9 Noveno Sprint

En este sprint, se llevarán a cabo tareas relacionadas a la edición de atributos, lo que le permitirá al usuario cambiar nombres, tipos de datos, etc. Dichos cambios deberán ser reflejados en la base de datos de la aplicación.

Las tareas del noveno sprint y su respectivo tiempo estimado, se muestran en la tabla siguiente.

Nº	Tarea	Tiempo Estimado
1	Consultar la información de cada una de las tablas y sus atributos, desde la base de datos de la aplicación.	4 horas
2	Mostrar los atributos consultados, en una tabla que permita la edición de los atributos permitidos, como son: nombre, tipo de dato y restricción NOT NULL.	6 horas
3	Guardar los cambios, en la base de datos de la aplicación.	6 horas

4	Aseguramiento de la Calidad.	4 horas
		20 horas

Tabla 25 Noveno Sprint CODGEN.[48]

2.2.1.9.1 *Planificación del Sprint*

A continuación se muestra una tabla correspondiente a la configuración del noveno sprint.

Proyecto			
CODGEN			
Nº de sprint	**Inicio**	**Días**	**Jornada**
9	08-ene-2014	5	4 horas/día
TAREAS		**EQUIPO**	**FESTIVOS**
TIPOS	**ESTADOS**		
Análisis	Pendiente	Jorge Rodríguez	
Codificación	En curso	Jairo Martínez	
Prototipado	Terminada		
Pruebas	Eliminada		
Reunión			

Tabla 26 Configuración del Noveno Sprint.[49]
Fuentes:[19]

En la siguiente figura se muestra la planificación y esfuerzo realizado para cada una de las tareas del noveno sprint.

[48] Tabla realizada por los autores.
[49] Plantilla elaborada en base a la fuente.

SPRINT	INICIO	DURACIÓN
9	8-ene.-14	5

	X	J	V	L	M	
	8-ene.	9-ene.	10-ene.	13-ene.	14-ene.	
Tareas pendientes	4	3	3	2	1	
Horas de trabajo pendientes	4	4	4	4	4	

Tarea	Tipo	Estado	Responsable						
Consultar la información de cada una de las tablas y sus atributos, desde la base de datos de la aplicación.	Codificación	Terminada	Jairo	4					
Mostrar los atributos consultados, en una tabla que permita la edición de los atributos permitidos, como son: nombre, tipo de dato y restricción NOT NULL.	Codificación	Terminada	Jorge		4	2			
Guardar los cambios, en la base de datos de la aplicación.	Codificación	Terminada	Jairo			2	4		
Aseguramiento de la Calidad	Pruebas	Terminada	Jorge					4	

Figura 31 Esfuerzo realizado en el Noveno Sprint.[50]
Fuentes:[19]

En la siguiente gráfica, conocida como Burn Down Chart, se puede apreciar el esfuerzo realizado y el esfuerzo pendiente.

Figura 32 Burn Down Chart Noveno Sprint.[51]
Fuentes:[19]

[50] Plantilla elaborada en base a la fuente.
[51] Plantilla elaborada en base a la fuente.

La gráfica de a continuación muestra el avance diario de las tareas concernientes al noveno sprint.

Figura 33 Avance de tareas Noveno Sprint.[52]
Fuentes:[19]

2.2.2 DISEÑO DE LA APLICACIÓN

Después de haber realizado el análisis correspondiente y la planificación de cada uno de los sprints, se procede a detallar el diseño.

2.2.2.1 Primer Sprint

2.2.2.1.1 *Arquitectura y Funcionalidad de la Aplicación.*

En esta etapa, se procederá a realizar la definición de la arquitectura bajo la cual será diseñada la aplicación. Se utilizará la arquitectura JEE6 para el desarrollo de la aplicación CODGEN (véase 1.1). Además se detalla el funcionamiento que tendrá la aplicación.

[52] Plantilla elaborada en base a la fuente.

Figura 34 Arquitectura y Funcionamiento de la Aplicación CODGEN[53]

Se puede observar que la aplicación se comunicará con la base de datos del servidor, guardando y consultando información de la misma. El cliente se comunicará con la aplicación por medio de un navegador web. Una parte importante del funcionamiento, es que la aplicación debe acceder a la base de datos del cliente para obtener la información de las tablas.

2.2.2.1.2 *Modelo Conceptual de la Base de Datos.*

En este sprint también se definió el modelo Entidad-Relación, el cual será utilizado para generar la base de datos, la cual será empleada por la aplicación CODGEN.

[53]Gráfico realizado por los autores.

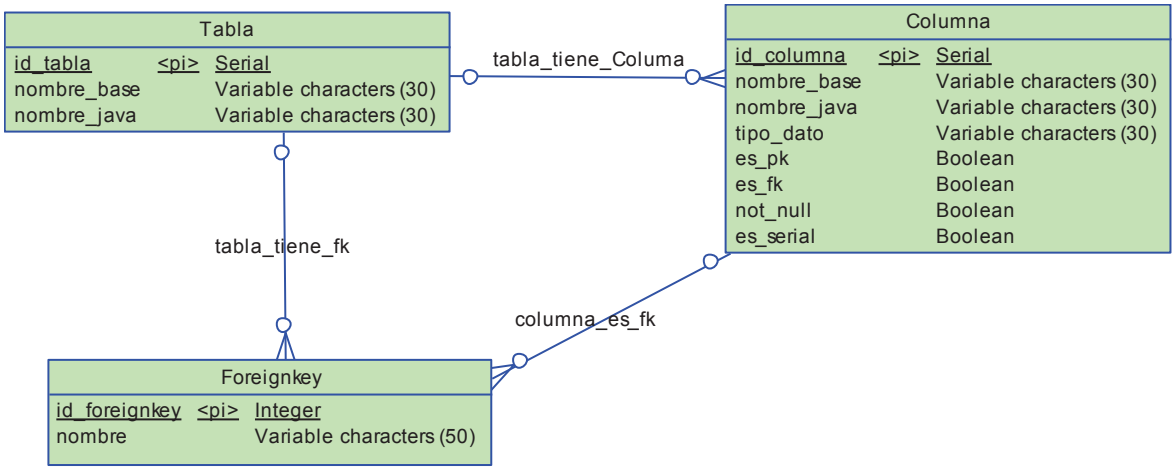


Figura 35Modelo E-R de la Aplicación CODGEN.[54]

Justificación

Los campos considerados para la elaboración de la base de datos, son aquellos que después de hacer un análisis, se llegó a la conclusión de que estos proporcionan el soporte necesario para el funcionamiento adecuado de la aplicación.

La aplicación estará diseñada para mapear relaciones entre tablas, únicamente si las relaciones están indicadas explícitamente en la base de datos. Esta restricción se considera porque en algunos casos, las bases de datos se crean con tablas sueltas, y las relaciones se controlan mediante código.

Los campos que se van a almacenar en la tabla “TABLA”, son necesarios para almacenar el nombre con el que se va a realizar el mapeo, y el nombre que tiene la tabla en la base de datos.

En la tabla “COLUMNA” se tiene de igual manera almacenado un campo para el nombre que tiene la columna en la base de datos, otro campo para almacenar el nombre que tendrá la columna al ser mapeada en código Java. Además se tiene campos para saber si la columna es clave primaria o clave foránea. Se tiene también un campo para almacenar información sobre la restricción “NOT NULL”, este campo se incluye porque hay columnas que tienen esta restricción, pero no son claves foráneas, entonces esta información resulta fundamental para poder

[54]Gráfico realizado por los autores.

mapear de manera correcta. Además también es necesaria la información de si el campo es serial (autogenerado) o no. Esto con el motivo de que si no se mapea de manera correcta, se tendrán conflictos al momento de la inserción en la base de datos con el código generado. Cabe recalcar que además los campos a mapearse se consideraron suficientes para la versión inicial del prototipo.

En la tabla “FOREIGNKEY” se almacena información sobre la clave foránea, con la finalidad de saber si la columna es clave foránea, de que tabla proviene esta clave, que también es un aspecto necesario y muy importante para realizar el mapeo. Se almacenará también el nombre de la clave foránea, que cabe mencionar, no es el mismo nombre de la columna.

2.2.2.1.3 *Modelo Físico de la Base de Datos.*

En base al modelo conceptual de la BDD se ha generado el modelo físico, el mismo que permitirá esclarecer las relaciones, mostrando las claves foráneas existentes y en el caso que sea necesario, mostrando las tablas de rompimiento.

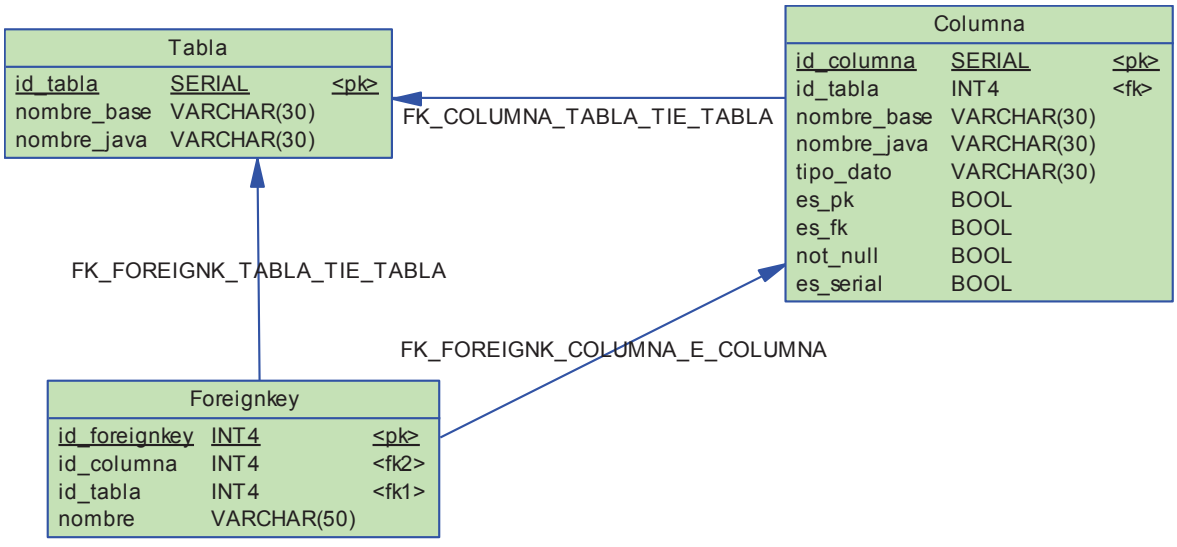


Figura 36Modelo Físico de la BDD de la Aplicación CODGEN.[55]

[55]Gráfico realizado por los autores.

Para tener claro la definición de la capa de datos, se realizara una descripción de la estructura, de cada una de las entidades del modelo E-R de la figura. Para lo cual se usaran las siguientes tablas.

2.2.2.1.4 *Estructura de Tabla.*

Nombre: Tabla

Objetivo: Permitir almacenar el nombre de las tablas de la BDD del cliente, para la generación del código correspondiente.

Campo	Tipo de Dato	Longitud	Descripción
id_tabla	Serial		Clave primaria de la tabla.
nombre_base	Varchar	30	Nombre que tiene la tabla en la BDD del cliente.
nombre_java	Varchar	30	Nombre que tendrá la tabla para el código Java (Entidad).

2.2.2.1.5 *Estructura de Columna.*

Nombre: Columna

Objetivo: Objetivo: Permitir almacenar el nombre, tipo de dato, restricciones de tipo NOT NULL, claves primarias y foráneas, de los atributos de cada una de las tablas de la BDD del cliente, para la generación del código correspondiente.

Campo	Tipo de Dato	Longitud	Descripción
id_columna	Serial		Clave primaria de la tabla.
nombre_base	Varchar	30	Nombre que tiene la columna de la tabla en la BDD del cliente.
nombre_java	Varchar	30	Nombre que tendrá la columna para el código Java (Atributo).
tipo_dato	Varchar	30	Tipo de dato de la columna.
es_pk	Bool		Verdadero, si la columna es clave primaria.
es_fk	Bool		Verdadero, si la columna es

			clave foránea.
not_null	Bool		Verdadero, si la columna tiene la restricción de no ser nula.
es_serial	Bool		Verdadero, si la columna es de tipo serial (Autoincremental).

2.2.2.1.6 *Estructura de ForeignKey.*

Nombre: ForeignKey			
Objetivo: Permitir almacenar el código de la columna que es clave foránea y el código de la tabla de la cual proviene dicha clave foránea, para facilitar las consultas referentes a foreign keys.			
Campo	Tipo de Dato	Longitud	Descripción
id_foreignKey	Serial		Clave primaria de la tabla.
nombre	Varchar	50	Nombre de la Clave Foránea

2.2.2.2 Segundo Sprint

En este sprint se procede a establecer los parámetros necesarios para poder establecer conexión con la base de datos del cliente y seguidamente diseñar una interfaz para solicitar dichos parámetros. Es importante mencionar que esta etapa es esencial dentro del funcionamiento de CODGEN.

2.2.2.2.1 *Parámetros de conexión.*

Para poder establecer estos parámetros, se emplearáel estándar de conexión empleado por el API JDBC, Java DataBase Conectivity más conocido por sus siglas como JDBC. Este estandar será empleado para establecer la conectividad entre el lenguaje de programación JAVA y varias bases de datos (SQL database).

De manera general el API JDBC permite:

- Establecer una conexión a cualquier base de datos o a cualquier fuente de datos tabulares.

- Enviar sentencias SQL.
- Procesar los resultados.

Además, una de las características principales y de gran importancia para la aplicación CODGEN es, el acceso total a los metadatos[20].

Para poder establecer una conexión empleando JDBC, es necesario utilizar el método getConnection(); de la clase DriverManager. Del mencionado método, se tiene que, los parámetros necesarios para establecer la conexión son los siguientes[21]:

Parámetro	Descripción
Driver	Es el controlador de la base de datos, cumple la función de traductor de sentencias Java a lenguaje SQL. (Propio de cada base de datos).
URL	Localizador Uniforme de Recursos, para localizar la base de datos. Debe estar en la forma *Jdbc:subprotocol:subname* donde: **Subprotocolo:** es el nombre del driver o controlador de la base de datos. **Subname**: una manera de identificar la base de datos.
User	Usuario de la base de datos, con cuyo nombre se realiza la conexión.
Password	Contraseña del Usuario.

Tabla 27 Parámetros para establecer conexión a la BDD empleando JDBC[56].
Fuentes:[22][23]

Con estos parámetros suficientes y necesarios para establecer la conexión se puede mostrar el siguiente ejemplo.

[56] Tabla elaborada por los autores en base a las fuentes.

Driver: org.postgresql.Driver

URL: jdbc:postgresql://localhost:5432/nombasedatos

User: postgres

Password: *******

Figura 37 Ejemplo de conexión con una BDD PostgreSQL.

2.2.2.2.2 *Interfaz de conexión – Principal.*

Luego de haber establecido los parámetros de conexión, es necesario diseñar una interfaz, que permitirá solicitar al usuario que ingrese dichos parámetros de manera correcta y así poder establecer una conexión exitosa con su base de datos.

El diseño de dicha interfaz se muestra en la siguiente figura.

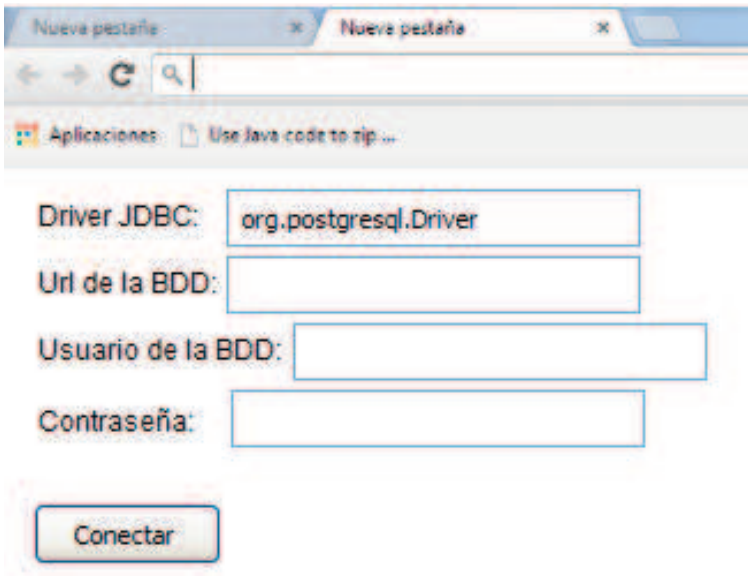


Figura 38 Diseño de la Interfaz de Conexión.

Como se aprecia en la Figura 20 se tiene 4 campos, para solicitar al usuario que ingrese información que a continuación se detalla:

- **Driver JDBC**

El usuario deberá ingresar el nombre del driver o controlador de su base de datos.

- **URL de la BDD**
  El usuario deberá ingresar la localización de su base de datos, de la forma especificada.

- **Usuario de la BDD**
  El usuario deberá ingresar el nombre de usuario mediante el cual podrá conectarse a su base de datos.

- **Contraseña**
  El usuario deberá ingresar la contraseña que fue establecida para el nombre de usuario especificado en el campo anterior.

Debido a que todos y cada uno de estos campos son necesarios para establecer una conexión, ninguno de ellos podrá quedar vacío.

También se puede apreciar un botón cuyo nombre es Conectar, el mismo que captura el texto ingresado en los 4 campos y lo envía a un servlet (controlador) de modo que ejecute un código que permitirá establecer la conexión con la base de datos.

En el caso que se presente una conexión fallida, se mostrara un mensaje que informe al usuario dicho acontecimiento y se le volverá a solicitar el ingreso de los parámetros de conexión.

2.2.2.3 Tercer Sprint

Este sprint tiene lugar luego de haber establecido la conexión;mediante la característica de JDBC que se refiere al acceso total a los metadatos, se hará la extracción de los mismos.

Los metadatos son datos que sirven para describir otros datos, en este caso, el conjunto de datos sirve para describir la estructura de base de datos.

Una vez recolectados los metadatos, se realizará un filtro de los mismos, con la finalidad de tener solamente los datos suficientes para poder realizar un mapeo correcto.

2.2.2.3.1 *Interfaz de conexión – Esquemas.*

Desde los metadatos se realiza la extracción de los nombres de los esquemas, en los cuales constan los objetos de la base de datos tales como: tablas, vistas, índices, usuarios, etc.

Estos esquemas serán presentados al usuario en una lista, con la finalidad de que el usuario pueda seleccionar de cuál de ellos se obtendrán los datos.

A continuación se presenta un diseño del diálogo que será añadido a la interfaz de conexión.

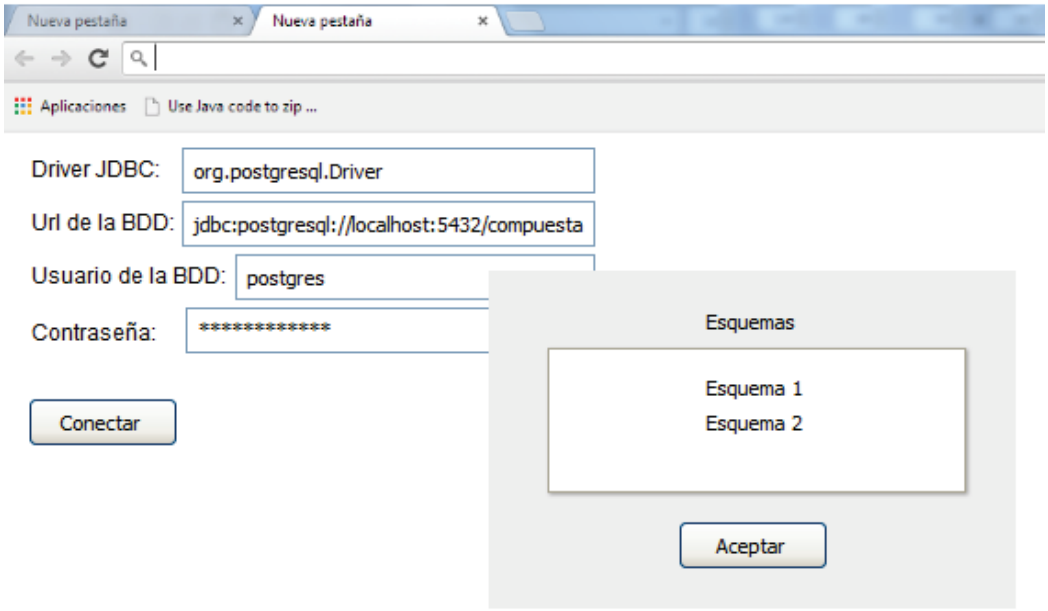


Figura 39Diseño de la Interfaz de Conexión (Esquemas).

El botón Aceptar que se muestra en el diálogo, disparará un método que enlistará todas las tablas que forme parte de la base de datos y del esquema seleccionado por el usuario.

2.2.2.3.2 *Interfaz de conexión – Tablas.*

Una vez seleccionado el esquema del cual se extraerán los objetos, se enlistarán los nombres de todas las tablas de la base de datos, en un diálogo, con la finalidad de que el usuario tenga conocimiento de las tablas que van a ser mapeadas.

En la siguiente figura se presenta el diseño del diálogo que será añadido a la interfaz de conexión.

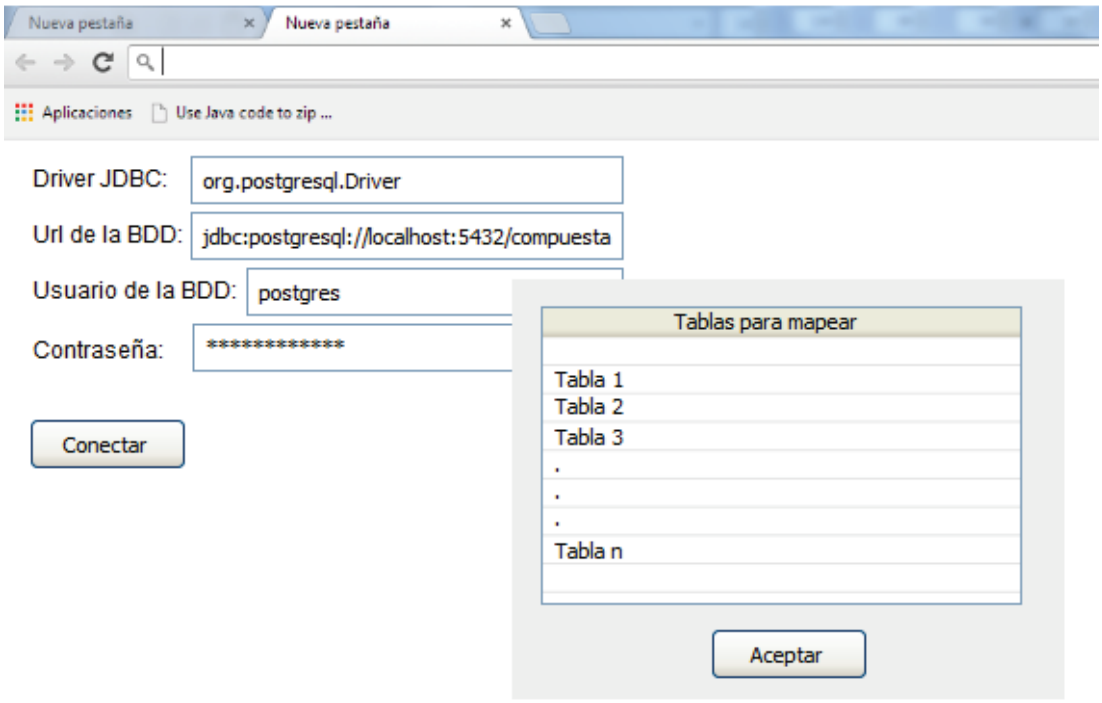


Figura 40Diseño de la Interfaz de Conexión (Tablas).

El botón Aceptar que se muestra en la figura 22, desencadenará la ejecución de una serie de métodos, los cuales permitirán extraer el resto de datos tales como:

- Nombre de las columnas.
- Tipos de datos.
- Restricciones de NOT NULL.
- Claves Primarias.
- Claves Foráneas.

2.2.2.3.3 *Tipos de Datos de BDD a Java.*

La base de datos maneja un conjunto de tipos de datos, los mismos que optimizan el diseño de la estructura de la misma. Tal es el caso de Java que requiere un lenguaje específico para el manejo de los tipos de datos, por lo cual, se requiere contrastar los tipos de datos de la BDD con Java.

Tipo de Dato BDD	Tipo de Dato Java
TINYINT	Short
SMALLINT	Short
INTEGER	Integer
BIGINT	Long
FLOAT	Float
REAL	Double
DOUBLE	Double
DECIMAL	Double
CHAR	String
VARCHAR	String
LONGVARCHAR	String
DATE	Date
TIME	Time
TIMESTAMP	Timestamp
BIT	Boolean
NUMERIC	BigDecimal

Tabla 28 Tipos de datos de BDD a Java[57].

En la tabla 16, se ha puesto a consideración algunos de los tipos de datos con su respectivo equivalente en lenguaje Java. Estos tipos datos son suficientes para el correcto funcionamiento del prototipo CODGEN, debido a que los mismos son empleados con mayor frecuencia en el diseño de base de datos.

[57] Tabla realizada por los autores.

Una vez que se haya realizado la conversión, los metadatos se almacenarán en la base de datos de CODGEN, para que puedan ser utilizados más adelante.

2.2.2.4 Cuarto Sprint

2.2.2.4.1 *Interfaz de Generación de Entidades.*

Para el cuarto sprint, será necesario el diseño de una pantalla, el diseño aproximado se muestra a continuación

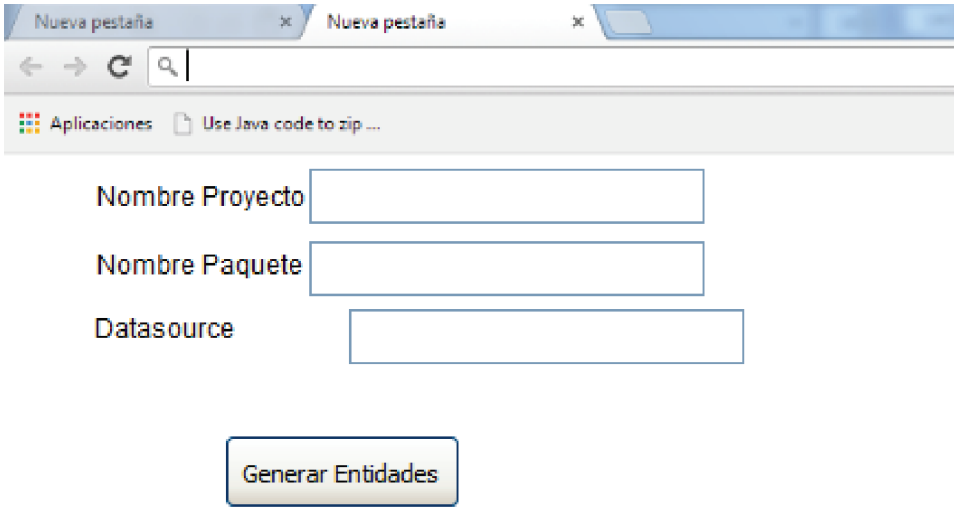


Figura 41Diseño de la Interfaz de Generación de Entidades.

- **Nombre Proyecto**

  El campo nombre de proyecto, es necesario para saber el nombre de la carpeta que alojará los archivos generados por la aplicación.

- **Nombre Paquete**

  Es un campo necesario para que el usuario ingrese el nombre del paquete con el que se generarán las entidades. Además este campo permitirá crear la jerarquía de carpetas que el usuario requiera y de esta manera, cuando haga uso del contenido generado, no tenga ningún inconveniente.

- **Datasource**

  Es necesario para que el archivo persistence.xml pueda ser generado. Este archivo se genera como parte de las entidades, y es un archivo necesario

para poder conectarse a la base de datos de manera correcta. Sin este archivo, el mapeo quedaría incompleto, y no se puede tener acceso a la base de datos.

2.2.2.5 Quinto Sprint

2.2.2.5.1 *Interfaz de Generación de Servicios.*

El diseño aproximado de la interfaz para la generación de servicios, se muestra a continuación:

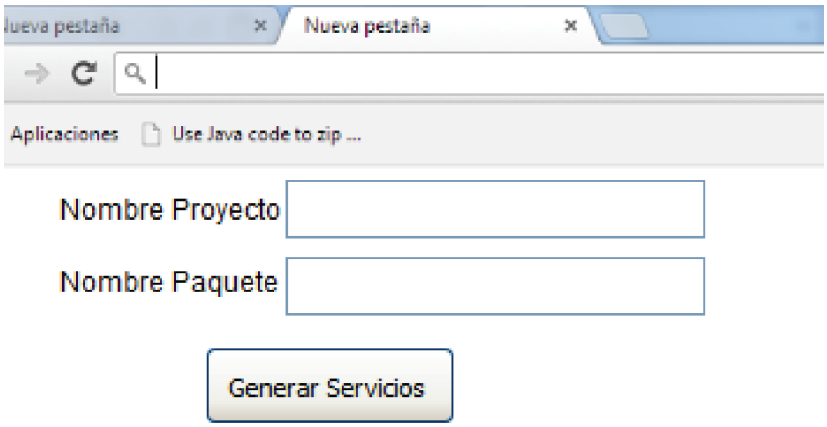


Figura 42 Diseño de la Interfaz de Generación de Servicios.

- **Nombre Proyecto**
  Se refiere al nombre de la carpeta en la cual se crearán los archivos generados por CODGEN.

- **Nombre Paquete**
  Este campo es necesario para que los servicios se generen con el paquete correcto al que pertenecerán una vez que se añadan al proyecto. Además permite crear la jerarquía de carpetas.

2.2.2.6 Sexto Sprint

2.2.2.6.1 *Interfaz de Generación de Controladores.*

La interfaz para la generación de controladores, tendrá aproximadamente el aspecto que se muestra en la figura siguiente.

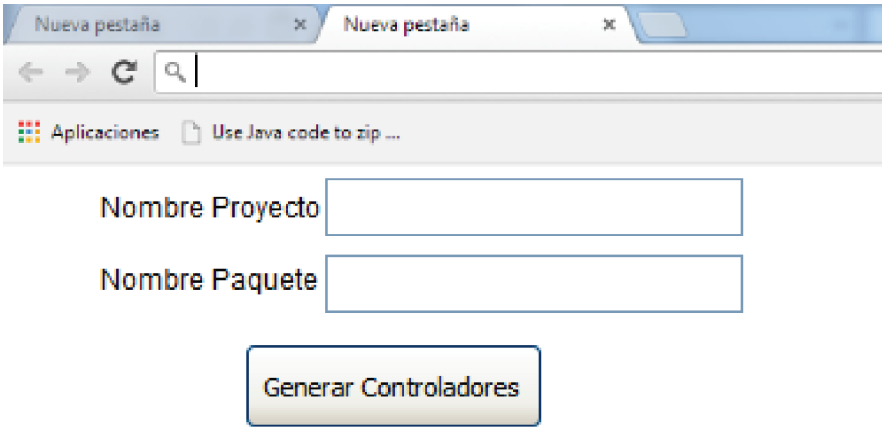


Figura 43 Diseño de la Interfaz de Generación de Controladores.

- **Nombre Proyecto**
  Se refiere al nombre de la carpeta en la cual se crearán los controladores que se van a generar.

- **Nombre Paquete**
  Lo que se ingrese en este campo, permitirá generar la jerarquía de carpetas para los controladores, además que permite establecer el paquete correcto en cada uno de los controladores.

2.2.2.7 Séptimo Sprint

2.2.2.7.1 *Interfaz de Generación de Páginas.*

La interfaz para la generación de páginas, tendrá aproximadamente el aspecto que se muestra en la figura siguiente.

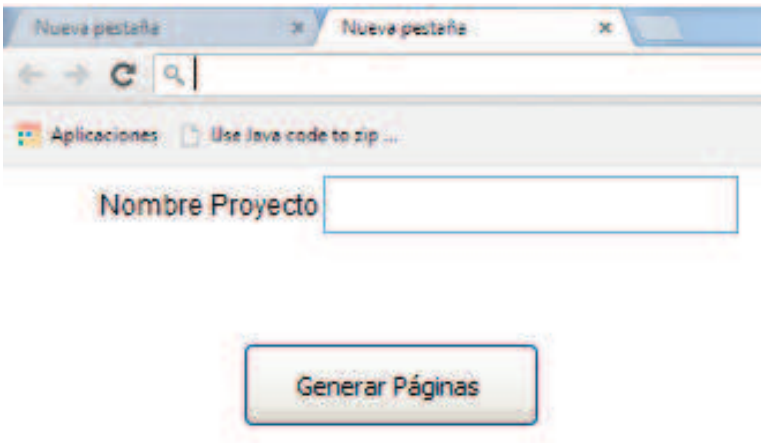


Figura 44Diseño de la Interfaz de Generación de Páginas.

2.2.2.8 Octavo Sprint

2.2.2.8.1 *Interfaz de Resultados.*

La interfaz que permitirá al usuario descargar los archivos, tendrá el diseño de la siguiente figura.

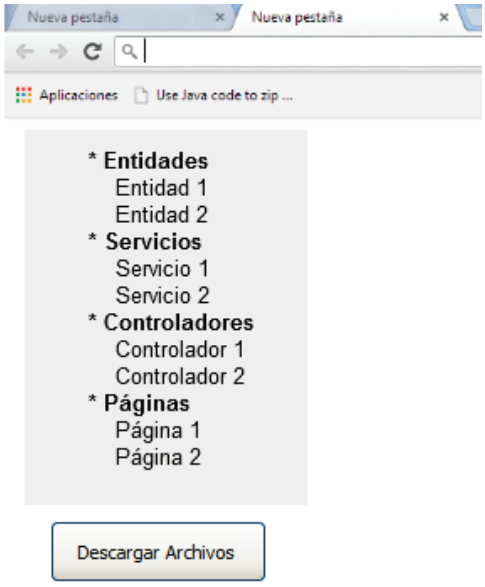


Figura 45 Diseño de la Interfaz de Resultados.

En esta interfaz se presentará al usuario un listado de todos los archivos que han sido generados y mediante el botón de descarga, el usuario podrá guardar estos archivos comprimidos, localmente.

2.2.2.9 Noveno Sprint

2.2.2.9.1 *Interfaz de Personalización.*

Esta interfaz permitirá al usuario editar campos específicos de la tabla que contiene los atributos, tales como, nombre, tipo de dato, etc.

Una vez seleccionada la tabla a ser editada, el usuario podrá seleccionar el campo que desee cambiar. Este campo se habilitará siempre y cuando el atributo no altere la estructura de la base.

Los campos editables serán:

- Nombre
- Tipo de dato
- Not Null

Para que los cambios sean almacenados, el usuario deberá oprimir el botón Guardar Cambios.

En la siguiente gráfica se presenta el diseño de la interfaz de personalización.

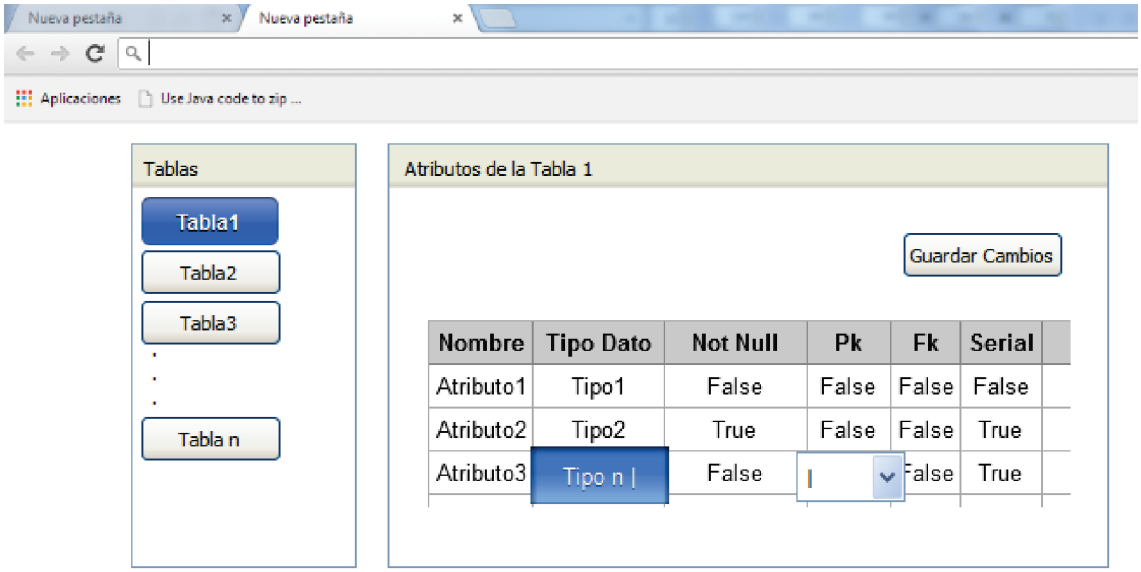


Figura 46 Diseño de la Interfaz de Personalización.

2.2.3 CONSTRUCIÓN DE LA APLICACIÓN

2.2.3.1 Primer Sprint

Al finalizar el primer sprint, se ha obtenido como resultado la generación de un script, el mismo que al ser ejecutado en el DBMS PostgreSQL creará la base de datos para la aplicación. Esta incluye las tablas, índices, restricciones y secuencias, necesarias para almacenar la información que posteriormente será empleada para la generación de código.

A continuación se muestra un extracto representativo del script en lenguaje SQL, generado específicamente para el DBMS PostgreSQL.

```
/*==============================================================*/
/* Table: COLUMNA                                               */
/*==============================================================*/
createtable COLUMNA (
   ID_COLUMNA           SERIAL               notnull,
   ID_TABLA             INT4                 null,
   NOMBRE_BASE          VARCHAR(30)          notnull,
   NOMBRE_JAVA          VARCHAR(30)          notnull,
   TIPO_DATO            VARCHAR(30)          notnull,
   ES_PK                BOOL                 notnull,
   ES_FK                BOOL                 notnull,
   NOT_NULL             BOOL                 notnull,
   ES_SERIAL            BOOL                 notnull,
constraint PK_COLUMNA primarykey (ID_COLUMNA)
);
```

```
/*==============================================================*/
/* Table: FOREIGNKEY                                            */
/*==============================================================*/
createtable FOREIGNKEY (
   ID_FOREIGNKEY        INT4                 notnull,
   ID_COLUMNA           INT4                 null,
   ID_TABLA             INT4                 null,
   NOMBRE               VARCHAR(50)          null,
constraint PK_FOREIGNKEY primarykey (ID_FOREIGNKEY)
);
/*==============================================================*/
/* Table: TABLA                                                 */
/*==============================================================*/
createtable TABLA (
   ID_TABLA             SERIAL               notnull,
   NOMBRE_BASE          VARCHAR(30)          notnull,
   NOMBRE_JAVA          VARCHAR(30)          notnull,
constraint PK_TABLA primarykey (ID_TABLA)
);

altertable COLUMNA
addconstraint FK_COLUMNA_TABLA_TIE_TABLA foreignkey (ID_TABLA)
references TABLA (ID_TABLA)
ondeleterestrictonupdaterestrict;

altertable FOREIGNKEY
addconstraint FK_FOREIGNK_COLUMNA_E_COLUMNA foreignkey (ID_COLUMNA)
references COLUMNA (ID_COLUMNA)
ondeleterestrictonupdaterestrict;

altertable FOREIGNKEY
addconstraint FK_FOREIGNK_TABLA_TIE_TABLA foreignkey (ID_TABLA)
references TABLA (ID_TABLA)
ondeleterestrictonupdaterestrict;
```

2.2.3.2 Segundo Sprint

Como resultado del segundo sprint, se tiene la creación de la clase “ControladorConexión”, la cual contiene los atributos y métodos que permitirán establecer la conexión con la base de datos del usuario.

A continuación se presentaun fragmento representativo del código de la clase java, que cumple con el proceso de establecer la conexión.

```
import java.io.IOException;
import java.sql.Connection;
import java.sql.DatabaseMetaData;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
```

```
publicclass ConexionControlador {

	// Variables deConexión
	private Connection conexion;
	private Statement instanciaConexion;
	private DatabaseMetaData metaData;
	@NotNull(message = "Debe ingresar el nombre del Driver de la BDD")
	private String driverJdbc;
	@NotNull(message = "Debe ingresar el url de la BDD")
	private String urlBdd;
	@NotNull(message = "Debe ingresar el nombre de usuario de la BDD")
	private String usuarioBdd;
	@NotNull(message = "Debe ingresar el password de la BDD")
	private String passwordBdd;
	private String esquemaSeleccionado;
	// Listas
	private List<String>esquemas;


	publicvoid establecerConexion() {
		try {
			// Cargadel JDBC driver
			Class.forName(driverJdbc);
			// EstableciendolaConexión a la BDD
			conexion = DriverManager.getConnection(urlBdd, usuarioBdd,
					passwordBdd);
			instanciaConexion = conexion.createStatement();
			metaData = conexion.getMetaData();
			// Guardandoen DataManager
			codgenDataManager.setMetaData(metaData);

	Mensajes.mostrarMensajeInformacion(MensajesInformacion.CONEXION_EXITOSA);
			// Llamada a Listaresquemas
			listarEsquemas();


	RequestContext.getCurrentInstance().execute("wvEsquemas.show()");

		} catch (Exception e) {
			e.printStackTrace();

	Mensajes.mostrarMensajeError(MensajesError.CONEXION_FALLIDA);

		}
	}
```

2.2.3.3 Tercer Sprint

En este sprint se han desarrollado métodos que permiten extraer los metadatos de la base de datos del usuario, y poderlos almacenar en la base de CODGEN.

A continuación se muestra un fragmento representativo del código que cumple la tarea de listar esquemas y tablas.

```
publicvoid listarEsquemas() {

		try {
			ResultSet esquemasRs = metaData.getSchemas();
			while (esquemasRs.next()) {
				esquemas.add(esquemasRs.getString(1));
			}
		} catch (SQLException e) {
			e.printStackTrace();
		}

}

/***
* Métodoqueextraelosnombresdelastablasdeuna BDD, tomadadel
* esquemapúblico. Tomacomoparámetrolametadatadeuna BDD
* @param metaData
* @returnListade String, conlosnombresdelasTablas
*/

publicvoid obtenerNombreTablas() {

		if (!esquemaSeleccionado.equals("")) {
			// GuardandoEsquemaen DataManager
			codgenDataManager.setEsquema(esquemaSeleccionado);

	RequestContext.getCurrentInstance().execute("wvEsquemas.hide()");
			String tipoTablas[] = { "TABLE" };
			ResultSet tablasBase;

			try {
				tablasBase = metaData.getTables(null,
esquemaSeleccionado, "%",
						tipoTablas);

				while (tablasBase.next()) {

					nombreTablas.add(tablasBase.getString(3));
				}

		RequestContext.getCurrentInstance().execute("wvTablas.show()");

			} catch (SQLException e) {
				e.printStackTrace();
			}
		} else {

	Mensajes.mostrarMensajeError(MensajesError.SELECCION_ESQUEMAS);
		}
	}
```

De la misma manera, a continuación se muestra una parte del código desarrollado para extraer: nombres de columnas, tipos de datos, claves primarias, claves

foráneas y restricciones NOT NULL. De igual forma se presenta código del método que permite guardar los datos recopilados, en la base de datos de CODGEN.

```
publicvoid guardarColumnas(Tabla tabla) {
		String nombreColumna;
		String nombreAtributo;
		String tipoDato;
		Boolean esNull;
		Boolean esSerial;
		try {
			ResultSet columnasBase = metaData.getColumns(esquema, null,
					tabla.getNombreBase(), "%");
			while (columnasBase.next()) {

				columna = new Columna();
				nombreColumna = columnasBase.getString(4);
				nombreAtributo = nomColumnaAtributo(nombreColumna);
				tipoDato =
obtenerNombreTipo(Integer.parseInt(columnasBase
						.getString(5)));
				esNull = columnasBase.getString(18).equals("NO") ?
true
						: false;
				esSerial = columnasBase.getString(23).equals("YES") ?
true
						: false;
				columna.setNombreBase(nombreColumna);
				columna.setNombreJava(nombreAtributo);
				columna.setTipoDato(tipoDato);
				columna.setNotNull(esNull);
				columna.setTabla(tabla);
				columna.setEsSerial(esSerial);
				columna.setEsPk(false);
				columna.setEsFk(false);
				servicioColumna.insertar(columna);
			}

		} catch (Exception e) {
			e.printStackTrace();
		}
	}

	publicvoid verificarPKs(Tabla tabla) {
		// Verificando PKs
		ResultSet columnasPk;
		String nombreColumnaPk = "";
		try {
			columnasPk = metaData.getPrimaryKeys(esquema, null,
					tabla.getNombreBase());
			while (columnasPk.next()) {
				nombreColumnaPk = columnasPk.getString(4);

				if
(servicioColumna.buscarPorNombreBase(nombreColumnaPk,
```

```
                                        tabla.getIdTabla()) != null) {
                                columna = servicioColumna.buscarPorNombreBase(
                                            nombreColumnaPk,
tabla.getIdTabla());
                                columna.setEsPk(true);
                                servicioColumna.actualizar(columna);
                          }

                    }

              } catch (Exception e) {
                    e.printStackTrace();
              }
       }

       publicvoid verificarFKs(Tabla tabla) {
              // Verificando FKs
              ResultSet columnasFk;
              try {
                    columnasFk = metaData.getImportedKeys(esquema, null,
                                tabla.getNombreBase());

                    String nombreColumnaFk = "";
                    Foreignkey foreignkey;
                    while (columnasFk.next()) {
                          nombreColumnaFk = columnasFk.getString(8);

                          if
(servicioColumna.buscarPorNombreBase(nombreColumnaFk,
                                        tabla.getIdTabla()) != null) {
                                columna = servicioColumna.buscarPorNombreBase(
                                            nombreColumnaFk,
tabla.getIdTabla());
                                columna.setEsFk(true);
                                servicioColumna.actualizar(columna);

                                foreignkey = new Foreignkey();
                                foreignkey.setTabla(servicioTabla

       .buscarPorNombreBase(columnasFk.getString(3)));
                                foreignkey.setColumna(columna);
                                foreignkey.setNombre(columnasFk.getString(12));
                                servicioForeignkeys.insertar(foreignkey);
                          }

                    }
              } catch (Exception e) {
                    e.printStackTrace();
              }
       }
```

2.2.3.4 Cuarto Sprint

Para la primera parte del código de la aplicación, es necesaria la consulta de la base de datos.

A continuación se muestran partes del código usadas para realizar esta tarea.

```
@ViewScoped
@ManagedBean
publicclass GeneracionEntidadesControlador {
	public String ubicacion;
	public String nombrePaquete;
	privateintauxiliar;
	private Columna columna;

	@ManagedProperty("#{entidadesDataManager}")
	private EntidadesDataManager entidadesDataManager;
	private List<Columna>columnasPk;
	private List<Columna>columnasFk;
	private List<Columna>columnasClaveCompuesta;
	private List<Columna>columnasHijas;
	private Tabla tabla;
	private List<Columna>columnas;
	private List<Tabla>tablas;
	@EJB
	private ServicioTabla servicioTabla;
	@EJB
	private ServicioColumna servicioColumna;
	@EJB
	private ServicioForeignkey servicioForeignKey;
	@EJB
privateServicioGeneracionMetodosservicioGeneracionMetodos;
	publicvoid generarEntidades() throws Exception {

		nombrePaquete = entidadesDataManager.getNombrePaquete();
		ubicacion = entidadesDataManager.getUbicacion();
		tablas = new ArrayList<Tabla>();
		tablas = servicioTabla.recuperarTodos();
		entidadesDataManager.setTablas(tablas);
```

```
            for (int i = 0; i <tablas.size(); i++) {

                  String nombreTabla = tablas.get(i).getNombreJava();
                  columna = null;
                  tabla = tablas.get(i);
                  columnas =
servicioColumna.buscarPorTabla(tabla.getIdTabla());
                  // pkcompuesta
                  for (Columna columna : columnas) {
                        if (columna.getEsFk() && columna.getEsPk()) {
                              columnasClaveCompuesta.add(columna);
                        }
                  }

                  for (Columna c : columnasClaveCompuesta) {
                        columnas.remove(c);
                  }
                  // pk
                  for (Columna columna : columnas) {
                        if (columna.getEsPk() == true&&
columna.getEsFk() == false) {
                              columnasPk.add(columna);
                        }
                  }
                  for (Columna c : columnasPk) {
                        columnas.remove(c);
                  }
                  // fk
                  for (Columna columna : columnas) {
                        if (columna.getEsFk() == true&&
columna.getEsPk() == false) {
                              columnasFk.add(columna);

                        }
                  }
                  for (Columna c : columnasFk) {
                        columnas.remove(c);
                  }
```

```
                    // Columnas

                    definirUbicacion(ubicacion, nombrePaquete,
tabla.getNombreJava());
                    columnas = new ArrayList<Columna>();
                    columnasClaveCompuesta = new ArrayList<Columna>();
                    columnasFk = new ArrayList<Columna>();
                    columnasPk = new ArrayList<Columna>();
              }

            }
```

Para la siguiente fase que se refiere a la creación del archivo se creó el siguiente método, del cual se muestra a continuación el código:

```
publicvoid definirUbicacion(String ubicacion, String nombrePaquete,
                    String nombreTabla) {
              // Ubicaciondelpaquete
              String ubicacionDelPaquete = nombrePaquete.replace('.',
'/');
              // Ubicaciondelarchivo
              String ubicacionDelArchivo;
              if (ubicacion == null) {
                    ubicacionDelArchivo = ubicacionDelPaquete +
"/entidades/";
              } else {
                    ubicacionDelArchivo = ubicacion + '/' +
              ubicacionDelPaquete
                                + "/entidades/";
              }
              File tFile = new File(ubicacionDelArchivo +
tabla.getNombreJava()
                          + ".java");
              BufferedWriter bw = null;
              try {
                    new File(ubicacionDelArchivo).mkdirs();
                    tFile.createNewFile();
```

```
                    bw = new BufferedWriter(new
FileWriter(ubicacionDelArchivo
                                + tabla.getNombreJava() + ".java"));
                    bw.write(escribirEnEntidades(ubicacionDelArchivo,
              nombrePaquete,
                                tabla));
                    bw.close();
              } catch (Exception e) {
                    e.printStackTrace();

       Mensajes.mostrarMensajeError(MensajesError.PATH_NO_VALIDO);
              } finally {
                    try {
                          bw.close();
                    } catch (Exception e) {
                          e.printStackTrace();
                    }
              }
            }
```

A continuación se muestran partes del código del método utilizado para escribir el código necesario en las entidades:

```
public String escribirEnEntidades(String ubicacionDelArchivo,
                    String nombrePaquete, Tabla tabla) {

              StringBuffer sb = new StringBuffer("");

              // Paquete
              sb.append("package " + nombrePaquete + ".entidades;\n");
              sb.append("\n");

              // imports
              sb.append("import java.util.*;\n");
              sb.append("import javax.persistence.*;\n");
              sb.append("import java.math.BigDecimal;\n");
              sb.append("import java.io.Serializable;\n");
              sb.append("import
```

```
javax.validation.constraints.NotNull;\n");
            sb.append("\n");

            // declaraciondelaclase
            sb.append("@Entity\n");
            sb.append("@Table(name=\"" + tabla.getNombreBase() +
"\")\n");

            if (columnasClaveCompuesta.size() > 1) {
                  sb.append("@IdClass(value=" + tabla.getNombreJava()
+ "PK.class)\n");
            }
            sb.append("public class " + tabla.getNombreJava()
                        + " implements Serializable {\n");

            sb.append("\n  private static final long serialVersionUID
= 1L;\n");

            // Obtenciondeclaveprimaria.

            if (columnasPk.size() > 0) {
                  for (Columna c : columnasPk) {
                        sb.append("@NotNull (message=\"Campo
            Requerido\")\n");
                        sb.append("@Id\n");
                        if (c.getEsSerial()) {
      sb.append("@GeneratedValue
(strategy=GenerationType.IDENTITY)\n");
                        }
                        if (c.getTipoDato().equals("Date")) {
                              sb.append("@Temporal
(TemporalType.DATE)\n");
                              c.setTipoDato("Date");
                              servicioColumna.actualizar(c);
                        } elseif (c.getTipoDato().equals("Time")) {
            sb.append("@Temporal(TemporalType.TIME)\n");
                              c.setTipoDato("Date");
                              servicioColumna.actualizar(c);
                        } elseif
```

```
 (c.getTipoDato().equals("Timestamp")) {

                                sb.append("@Temporal
(TemporalType.TIMESTAMP)\n");
                                c.setTipoDato("Date");
                                servicioColumna.actualizar(c);
                          }

// GeneraciondelasFks.

            if (columnasFk.size() > 0) {

                  for (Columna c : columnasFk) {
                        if (c.getNotNull())
                                sb.append("@NotNull (message=\"Campo
Requerido\")\n");
                        Tabla t =
servicioForeignKey.buscarDeDondeVieneLaClave(c
                                          .getIdColumna());
                        sb.append("//Relacion bidireccional con la
                        tabla "
                                          + t.getNombreJava() + "\n");
                        sb.append("@ManyToOne\n");
                        sb.append("@JoinColumn(name=\"" +
                        c.getNombreBase() + "\")\n");
                        sb.append("private " + t.getNombreJava() + " "
                                          + (t.getNombreJava().charAt(0) +
"").toLowerCase()
                                          + t.getNombreJava().substring(1) +
";\n");

                  }

            }


//Métodos más relevantes

public String escribirClaveCompuestaSinForaneas(String nombrePaquete,
```

```
                  Tabla tabla, List<Columna> columnas) {

            StringBuffer sb = new StringBuffer("");
            // Paquete
            sb.append("package " + nombrePaquete + ".entidades;\n");
            sb.append("\n");

            // imports
            sb.append("import java.util.*;\n");
            sb.append("import javax.persistence.*;\n");
            sb.append("import java.io.Serializable;\n");
            sb.append("\n");

            // declaracion de la clase

            String nombreClase = tabla.getNombreJava() + "PKey";

            sb.append("public class " + nombreClase
                        + " implements Serializable {\n");

            sb.append("\n  private static final long serialVersionUID
= 1L;\n");

            for (Columna c : columnas) {

                  sb.append("private " + c.getTipoDato() + " " +
 c.getNombreJava()
                              + ";\n");
            }

            // Aniadir setters y getters

            if (columnas != null) {
                  for (Columna c : columnas) {


servicioGeneracionMetodos.generarSettersYGetters(sb,
                                    c.getNombreJava(),
c.getTipoDato(), false, false);
```

```
                }
            }
            sb.append("}\n");
            return sb.toString();
      }


public String escribirClaveCompuestaConForaneas(String nombreClase,
                String nombrePaquete, Tabla tabla, List<Columna>
                columnas) {

            StringBuffer sb = new StringBuffer("");
            // Paquete
            sb.append("package " + nombrePaquete + ".entidades;\n");
            sb.append("\n");

            // imports
            sb.append("import java.util.*;\n");
            sb.append("import javax.persistence.*;\n");
            sb.append("import java.io.Serializable;\n");
            sb.append("\n");

            // declaracion de la clase

            sb.append("public class " + nombreClase
                        + "PK implements Serializable {\n");

            sb.append("\n  private static final long serialVersionUID
= 1L;\n");

            for (Columna c : columnas) {

                Tabla td =
servicioForeignKey.buscarDeDondeVieneLaClave(c
                            .getIdColumna());
                sb.append("private " + td.getNombreJava() + " "
                            + td.getNombreJava().toLowerCase() +
";\n");
```

```
            }

            // Aniadir setters y getters

            if (columnas != null) {
                  for (Columna c : columnas) {

                        Tabla td =
servicioForeignKey.buscarDeDondeVieneLaClave(c
                                    .getIdColumna());

servicioGeneracionMetodos.generarSettersYGetters(sb, td
                                    .getNombreJava().toLowerCase(),
                                    td.getNombreJava(),
                                    false, false);
                  }
            }
            sb.append("}\n");
            return sb.toString();
      }
```

2.2.3.4.1 *Control de Calidad*

Al realizar la ejecución del presente módulo, y contrastar los resultados con las plantillas detalladas en los**Anexos 1 y 5**, se pudo constatar que el código generado se ajusta al código propuesto.

Por lo tanto se puede concluir que el código fue generado correctamente.

2.2.3.5 Quinto Sprint

El siguiente código muestra el método para crear el servicioBase.java.

```
publicvoid generarServicioBase() {
            new File(ubicacionServicios).mkdirs();
            File destino = new File(ubicacionServicios +
"ServicioBase.java");
            InputStream in = null;
```

```
            OutputStream output = null;
            try {
                  in =
FacesContext.getCurrentInstance().getExternalContext()

      .getResourceAsStream("/recursos/ServicioBase.txt");
                  output = new FileOutputStream(destino);

                  String nombrePaqueteServicios1 = "package "
                              + nombrePaqueteServicios + ";\n";
                  output.write(nombrePaqueteServicios1.getBytes(), 0,

      nombrePaqueteServicios1.getBytes().length);
                  byte[] buf = newbyte[1024];
                  int len;

                  while ((len = in.read(buf)) > 0) {

                        output.write(buf, 0, len);
                  }
            } catch (FileNotFoundException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
            } catch (IOException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();

      Mensajes.mostrarMensajeError(MensajesError.PATH_NO_VALIDO);
            } finally {

                  try {
                        in.close();
                        output.close();
                  } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                  }

            }
```

```
            }
```

En el siguiente fragmento de código, se puede ver el método para la generación de cada uno de los archivos con extensión .java, que se utilizarán en este caso como servicios.

```
publicvoid definirUbicacion(Tabla tabla) {

            File tile = new File(ubicacionServicios + "Servicio"
                        + tabla.getNombreJava() + ".java");
            BufferedWriter bw = null;
            try {
                  tFile.createNewFile();
                  bw = new BufferedWriter(new
FileWriter(ubicacionServicios
                              + "Servicio" + tabla.getNombreJava() +
".java"));

                  List<Columna> compuestas = new ArrayList<Columna>();
                  for (Columna c :
servicioColumna.buscarPorIdTabla(tabla
                              .getIdTabla())) {

                        if (c.getEsPk() && c.getEsFk()) {

                              compuestas.add(c);
                        }
                  }

                  if (compuestas.size() > 1) {


      bw.write(escribirEnServiciosClaveCompuesta(tabla));
                  } else {

                        bw.write(escribirEnServicios(tabla));

                  }
```

```
                bw.close();
            } catch (Exception e) {
                e.printStackTrace();

      Mensajes.mostrarMensajeError(MensajesError.PATH_NO_VALIDO);
            } finally {
                try {
                    bw.close();
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
          }
```

En el siguiente código se muestra un fragmento del método para escribir en los archivos generados.

```
public String escribirEnServicios(Tabla tabla) {

            StringBuffer sb = new StringBuffer();

            // Paquete
            sb.append("package " + nombrePaqueteServicios + ";\n");
            sb.append("\n");

            // imports
            sb.append("import " +
entidadesDataManager.getNombrePaquete()
                        + ".entidades." + tabla.getNombreJava() +
";\n");
            sb.append("import javax.ejb.Stateless;\n");
            sb.append("\n");
            // declaraciondelaclase

            sb.append("@Stateless\n");
            sb.append("public class Servicio" + tabla.getNombreJava()
                        + " extends ServicioBase<" +
```

```
            tabla.getNombreJava() + "> {\n");
            sb.append("\n");

            sb.append("public Servicio" + tabla.getNombreJava() + "()
            {\n");
            sb.append("super(" + tabla.getNombreJava() + ".class,
Servicio"
                        + tabla.getNombreJava() + ".class);\n");
            sb.append("}\n");
            sb.append("\n");

            sb.append("}\n");
            return sb.toString();
          }
```

2.2.3.5.1 *Control de Calidad*

Al realizar la ejecución del módulo de generación de servicios, y contrastar los resultados con la plantilla detallada en los **Anexos 2 y 3**, se pudo constatar que el código generado se ajusta al código propuesto.

Por lo tanto se puede concluir que el código fue generado correctamente.

2.2.3.6 Sexto Sprint

El siguiente código muestra el método para crear los archivos con extensión .java.

```
publicvoid definirUbicacion(Tabla tabla, Tabla tablaAInsertar,
                  Tabla tablaAElegir, boolean compuesta) {

            if (compuesta == true) {

                  File tFile = new File(ubicacionControladores +
"Controlador"
                              + tablaAInsertar.getNombreJava() +
".java");
                  new File(ubicacionControladores).mkdirs();
                  BufferedWriter bw = null;
```

```
                try {
                        tFile.createNewFile();
                        bw = new BufferedWriter(new
 FileWriter(ubicacionControladores
                                        + "Controlador" +
                        tablaAInsertar.getNombreJava()
                                        + ".java"));


      bw.write(escribirEnControladoresRompimiento(tablaAInsertar,
                                        tablaAElegir, tabla));
                        // //
                        Tabla t5 = null;
                        for (Tabla t : tablas) {
                                if (t.getNombreJava()

      .equals(tablaAInsertar.getNombreJava())) {
                                        t5 = t;
                                }
                        }
                        tablas.remove(t5);
                        bw.close();
                } catch (Exception e) {
                        e.printStackTrace();

      Mensajes.mostrarMensajeError(MensajesError.PATH_NO_VALIDO);
                } finally {
                        try {
                                bw.close();
                        } catch (Exception e) {
                                e.printStackTrace();
                        }
                }

            } else {

                File tFile = new File(ubicacionControladores +
"Controlador"
                                + tabla.getNombreJava() + ".java");
```

```
                    new File(ubicacionControladores).mkdirs();
                    BufferedWriter bw = null;
                    try {
                          tFile.createNewFile();
                          bw = new BufferedWriter(new
FileWriter(ubicacionControladores
                                      + "Controlador" +
                    tabla.getNombreJava() + ".java"));

                          bw.write(escribirEnControladores(tabla));
                          // /
                          Tabla t5 = null;
                          for (Tabla t : tablas) {
                                if
                    (t.getNombreJava().equals(tabla.getNombreJava())) {
                                      t5 = t;
                                }
                          }
                          tablas.remove(t5);
                          bw.close();
                    } catch (Exception e) {
                          e.printStackTrace();

      Mensajes.mostrarMensajeError(MensajesError.PATH_NO_VALIDO);
                    } finally {
                          try {
                                bw.close();
                          } catch (Exception e) {
                                e.printStackTrace();
                          }
                    }
              }
            }
```

A continuación se muestra un fragmento de código, que forma parte del método para escribir el código en los controladores generados.

```
public String escribirEnControladores(Tabla tabla) {
```

```
            StringBuffer sb = new StringBuffer();
            new File(ubicacionControladores).mkdirs();
            // Paquete
            sb.append("package " + nombrePaqueteControladores + ";\n");
            sb.append("\n");

            // imports

            sb.append("import javax.ejb.EJB;\n");
            sb.append("import javax.faces.bean.ManagedBean;\n");
            sb.append("import javax.faces.bean.ViewScoped;\n");
            sb.append("import
org.primefaces.context.RequestContext;\n");
            sb.append("import javax.annotation.PostConstruct;\n");
            sb.append("import java.util.ArrayList;\n");
            sb.append("import java.util.List;\n");
            sb.append("import " +
entidadesDataManager.getNombrePaquete()
                        + ".entidades.*;\n");
            sb.append("import " +
entidadesDataManager.getNombrePaquete()
                        + ".servicios.*;\n");
            sb.append("\n");
            // declaraciondelaclase
            sb.append("@ViewScoped\n");
            sb.append("@ManagedBean\n");
            sb.append("public class Controlador" +
tabla.getNombreJava() + "{\n");
            sb.append("\n");
            sb.append("\n");

            sb.append("@EJB\n");
            sb.append("private Servicio" + tabla.getNombreJava() + "
            servicio"
                        + tabla.getNombreJava() + ";\n");

            columnas =
servicioColumna.buscarPorTabla(tabla.getIdTabla());
```

```
            recursiva = false;
            for (Columna columna : columnas) {

                  if (columna.getEsFk() && columna.getEsPk() == false)
            {

                        int i =
servicioForeignkey.buscarDeDondeVieneLaClave(

            columna.getIdColumna()).getIdTabla();

                        Tabla t = servicioTabla.buscarPorId(i);

                        if
            (t.getNombreJava().equals(tabla.getNombreJava())) {

                              recursiva = true;
                        }

                        if (recursiva == false) {
                              sb.append("@EJB\n");
                              sb.append("private Servicio" +
                              t.getNombreJava()
                                          + " servicio" +
                              t.getNombreJava() + ";\n");

                        }

                  }

            }
```

2.2.3.6.1 *Control de Calidad*

Después de ejecutar el módulo de generación de Controladores, y contrastar los resultados con las plantillas detalladas en los**Anexos 6 y 7**, se pudo constatar que el código generado se ajusta al código propuesto. Sin embargo, al existir lazos (for) y bloques de decisión (if) en el código generado, se vio la necesidad de

realizar pruebas del camino básico, con la finalidad de comprobar que el código generado funcione en su totalidad.

Se elaboraron 2 pruebas del camino básico[24], cada una de ellas con 3 caminos alternativos. En base a estos caminos se han diseñado 5 pruebas unitarias, las mismas que comprueban todas las alternativas posibles. Estas pruebas unitarias, detalladas en el **Anexo 7**, se cumplieron con éxito, por lo tanto se puede concluir que el código fue generado correctamente.

2.2.3.7 Séptimo Sprint

El siguiente código, es utilizado para la creación de los archivos con extensión .xhtml.

```
publicvoid definirUbicacion(Tabla tabla, Tabla tablaAInsertar,
                  Tabla tablaAElegir, boolean compuesta) {

            if (compuesta == true) {

                  File tFile = new File(ubicacionPaginas + "Gestion"
                              + tablaAInsertar.getNombreJava() +
".xhtml");
                  new File(ubicacionPaginas).mkdirs();
                  BufferedWriter bw = null;
                  try {
                        tFile.createNewFile();
                        bw = new BufferedWriter(
                                    new FileWriter(ubicacionPaginas +
"Gestion"
                                                +
                     tablaAInsertar.getNombreJava() + ".xhtml"));

               bw.write(escribirEnPaginasDeRompimiento(tablaAInsertar,
                           tablaAElegir, tabla));
                        // //
                        Tabla t5 = null;
                        for (Tabla t : tablas) {
```

```
                                if (t.getNombreJava()

                        .equals(tablaAInsertar.getNombreJava())) {
                                        t5 = t;
                                }
                        }
                        tablas.remove(t5);
                        bw.close();
                } catch (Exception e) {
                        e.printStackTrace();
                } finally {
                        try {
                                bw.close();
                        } catch (Exception e) {
                                e.printStackTrace();
                        }
                }

        } else {

                File tFile = new File(ubicacionPaginas + "Gestion"
                                + tabla.getNombreJava() + ".xhtml");
                new File(ubicacionPaginas).mkdirs();
                BufferedWriter bw = null;
                try {
                        tFile.createNewFile();
                        bw = new BufferedWriter(new
FileWriter(ubicacionPaginas
                                        + "Gestion" +
                        tabla.getNombreJava() + ".xhtml"));
                        bw.write(escribirEnPaginas(tabla));

                        Tabla t5 = null;
                        for (Tabla t : tablas) {
                                if
(t.getNombreJava().equals(tabla.getNombreJava())) {
                                        t5 = t;
                                }
                        }
```

```
                        tablas.remove(t5);
                        bw.close();
                  } catch (Exception e) {
                        e.printStackTrace();
                  } finally {
                        try {
                              bw.close();
                        } catch (Exception e) {
                              e.printStackTrace();
                        }
                  }
            }
                                                                  }
```

El siguiente fragmento de código forma parte del método que permite escribir el código en los archivos generados.

```
public String escribirEnPaginas(Tabla tabla) {

            StringBuffer sb = new StringBuffer();
            new File(ubicacionPaginas).mkdirs();
            // Paquete

            columnas =
servicioColumna.buscarPorIdTabla(tabla.getIdTabla());

            String cuerpo = "";
            cuerpo += "<p:panelGrid columns=\"2\">\n";
            for (Columna c : columnas) {
                  if (!c.getEsSerial() && c.getEsFk() == false) {
                        cuerpo += "<h:outputText value=\""
                                    + (c.getNombreJava().charAt(0) +
"").toUpperCase()
                                    + c.getNombreJava().substring(1) +
"\" />\n";
                        if (c.getTipoDato().equals("Date")) {

                              cuerpo += " <p:calendar value=\"#"
```

```
                                        + "{controlador"
                                        + tabla.getNombreJava()
                                        + "."
                                        +
            (tabla.getNombreJava().charAt(0) + "")
                                                    .toLowerCase()
                                        +
            tabla.getNombreJava().substring(1) + "."
                                        + c.getNombreJava() + "}\"
/>\n";

                        } elseif (c.getTipoDato().equals("Boolean"))
{

                              cuerpo += " <p:selectBooleanCheckbox
                               value=\"#"
                                        + "{controlador"
                                        + tabla.getNombreJava()
                                        + "."
                                        +
            (tabla.getNombreJava().charAt(0) + "")
                                                    .toLowerCase()
                                        +
            tabla.getNombreJava().substring(1) + "."
                                        + c.getNombreJava() + "}\"
/>\n";

                        } else {

                              cuerpo += " <p:inputText value=\"#"
                                        + "{controlador"
                                        + tabla.getNombreJava()
                                        + "."
                                        +
             (tabla.getNombreJava().charAt(0) + "")
                                                    .toLowerCase()
                                        +
             tabla.getNombreJava().substring(1) + "."
                                        + c.getNombreJava() + "}\"
```

```
/>\n";
                        }
                    }

                    // Combos
                    if (c.getEsPk() == false&& c.getEsFk() == true) {

                        cuerpo += "<h:outputText value=\""
                                + (c.getNombreJava().charAt(0) +
"").toUpperCase()
                                + c.getNombreJava().substring(1) +
"\" />\n";

                        int i =
servicioForeignkey.buscarDeDondeVieneLaClave(
                                c.getIdColumna()).getIdTabla();

                        Tabla t = servicioTabla.buscarPorId(i);
                        Columna pk = null;
                        List<Columna> columnasCombos =
servicioColumna.buscarPorTabla(t
                                .getIdTabla());
                        for (Columna c1 : columnasCombos) {
                            if (c1.getEsPk() && c1.getEsFk() ==
false) {
                                pk = c1;
                            }
                        }
                        cuerpo += "<p:selectOneMenu value=\"#" +
"{controlador"
                                + tabla.getNombreJava() + "."
                                + (t.getNombreJava().charAt(0) +
"").toLowerCase()
                                + t.getNombreJava().substring(1) +
"Seleccion}\">\n";

                        if
(tabla.getNombreJava().equals(t.getNombreJava())) {
                            cuerpo += "<f:selectItem
```

```
                    itemLabel=\"Seleccione\" itemValue=\"\"/>";
                            cuerpo += "<f:selectItems value=\"#" +
"{controlador"
                                    + tabla.getNombreJava() +
"."
                                    +
                    (t.getNombreJava().charAt(0) + "").toLowerCase()
                                    +
                    t.getNombreJava().substring(1)
                                    + "Padres}\" var=\"cmb" +
                    t.getNombreJava()
                                    + "\"\n" + "itemValue = \"#"
 + "{cmb"
                                    + t.getNombreJava() + "." +
                    pk.getNombreJava()
                                    + "}\" itemLabel=" + "\"#" +
"{cmb"
                                    + t.getNombreJava() + "." +
                    pk.getNombreJava()
                                    + "}\"/>\n";

                        } else {
                            cuerpo += "<f:selectItems value=\"#" +
"{controlador"
                                    + tabla.getNombreJava() +
"."
                                    +
                    (t.getNombreJava().charAt(0) + "").toLowerCase()
                                    +
                    t.getNombreJava().substring(1) + "s}\" var=\"cmb"
                                    + t.getNombreJava() + "\"\n"
                    + "itemValue = \"#"
                                    + "{cmb" + t.getNombreJava()
                    + "."
                                    + pk.getNombreJava() + "}\"
                     itemLabel=" + "\"#"
                                    + "{cmb" + t.getNombreJava()
                     + "."
                                    + pk.getNombreJava() +
```

```
"}\"/>\n";

                        }
                        cuerpo += "</p:selectOneMenu>\n";
                    }
                }

                cuerpo += " </p:panelGrid>\n ";

                    cuerpo += " <p:commandButton value=\"Insertar \"
                    action=\"#"
                        + "{controlador" + tabla.getNombreJava() + "."
                    + "insertar"
                        + tabla.getNombreJava() + "()" + "}\"
                    update=\"@form\" />\n";

                    cuerpo += " <p:commandButton value=\"Actualizar \"
                    action=\"#"
                        + "{controlador" + tabla.getNombreJava() + "."
 + "actualizar"
                        + tabla.getNombreJava() + "()" + "}\"
                    update=\"@form\" />\n";
}
```

2.2.3.7.1 *Control de Calidad*

Al realizar la ejecución del presente módulo, y contrastar los resultados con la plantilla detallada en el **Anexo 4**, se pudo constatar que el código generado se ajusta al código propuesto.

Por lo tanto se puede concluir que el código fue generado correctamente.

2.2.3.8 Octavo Sprint

Como resultado de este sprint se ha desarrollado un método que permitirá comprimir todos los archivos generados. El comprimido tendrá extensión .zip.

A continuación se presenta un fragmento representativo de código que permite cumplir con el proceso de compresión y descarga.

```
/**
	 * Métodoparaañadirarchivosalcomprimidozip
	* @param path
	 * @param origenArchivo
	 * @throws Exception
	 */

	publicvoid aniadirArchivos(String path, String origenArchivo)
			throws Exception {

		File carpeta = new File(origenArchivo);
		if (carpeta.isDirectory()) {
			aniadirDirectorioa(path, origenArchivo);
		}
	}

	/**
	 * Métodoparaañadirdirectorioalcomprimidozip
	* @param path
	 * @param origenDirectorio
	 * @throws Exception
	 */
	publicvoid aniadirDirectorioa(String path, String origenDirectorio)
			throws Exception {
		File carpeta = new File(origenDirectorio);

		for (String nombreArchivo : carpeta.list()) {
			if (path.equals("")) {
				aniadirArchivos(carpeta.getName(), origenDirectorio +
"/"
						+ nombreArchivo);
			} else {
				aniadirArchivos(path + "/" + carpeta.getName(),
						origenDirectorio + "/" + nombreArchivo);

				if (carpeta.getName().equals("entidades")) {
					TreeNode entidad = new
DefaultTreeNode("document",
							new Document(nombreArchivo, "",
									"Archivo Generado con
CODGEN"), entidades);
				} elseif (carpeta.getName().equals("controladores"))
{
					TreeNode controlador = new
DefaultTreeNode("document",
							new Document(nombreArchivo, "",
									"Archivo Generado con
CODGEN"),
							controladores);

				} elseif (carpeta.getName().equals("paginas")) {
					TreeNode pagina = new
DefaultTreeNode("document",
```

```
                                                new Document(nombreArchivo, "",
                                                            "Archivo Generado con
CODGEN"), paginas);

                              } elseif (carpeta.getName().equals("servicios")) {
                                    TreeNode servicio = new
DefaultTreeNode("document",
                                                new Document(nombreArchivo, "",
                                                            "Archivo Generado con
CODGEN"), servicios);

                              } elseif (carpeta.getName().equals("recursos")) {
                                    TreeNode controlador = new
DefaultTreeNode("document",
                                                new Document(nombreArchivo, "",
                                                            "Archivo Generado con
CODGEN"), recursos);

                              }
                        }
                  }
            }

/**
       * Métodoparadescargar el archivozipquecontienelosarchivosgenerados.
      *
       * @return
       * @throws FileNotFoundException
       */

      public StreamedContent getFile() throws FileNotFoundException {
            InputStream stream = new FileInputStream(new File(
                        entidadesDataManager.getUbicacion() + ".zip"));
            archivoZip = new DefaultStreamedContent(stream, "application/zip",
                        entidadesDataManager.getUbicacion() + ".zip");
            returnarchivoZip;
      }
```

2.2.3.9 Noveno Sprint

En este sprint se han desarrollado métodos que permitirán realizar cambios a los atributos de la tabla y almacenar dichos cambios.

A continuación se muestra una parte representativa del código de dichos métodos.

```
/**
       * Métodoquepermiteguardarloscambiosrealizadospor el usuario
       */
      publicvoid guardarCambios() {

            try {
```

```
                    for (Columna columna : columnas) {
                          servicioColumna.actualizar(columna);
                    }

      Mensajes.mostrarMensajeInformacion(MensajesInformacion.GUARDAR_CAMBIOS);
            } catch (Exception e) {
                    e.printStackTrace();
            }

      }



<p:panelid="pnlTabla"
                                 header="Atributos de la Tabla
#{personalizacionControlador.tablaSeleccionada.nombreJava}">
                                 <p:commandButton

      action="#{personalizacionControlador.eliminarSufijo()}"
                                       value="Quitar
Sufijo"update=":frmPersonalizacion:pnlTabla"/>
                                 <p:commandButton

      action="#{personalizacionControlador.eliminarPrefijo()}"
                                       value="Quitar
Prefijo"update=":frmPersonalizacion:pnlTabla"/>
                                 <p:commandButton

      action="#{personalizacionControlador.guardarCambios()}"

      value="Guardar"update=":frmPersonalizacion:pnlTabla"/>
                                 <p:commandButtonvalue="Cancelar"
                                       update=":frmPersonalizacion:pnlTabla"/>
                                 <p:dataTableid="tbAtributos"

      value="#{personalizacionControlador.columnas}"var="atributo"
                                       editable="true"editMode="cell"
                                       emptyMessage="Seleccione una Tabla para
mostrar sus atributos.">
                                       <p:columnheaderText="Nombre Base">

      <p:outputLabelvalue="#{atributo.nombreBase}"/>
                                       </p:column>
                                       <p:columnheaderText="Nombre Java">
                                             <p:cellEditor>
                                                   <f:facetname="output">

      <p:outputLabelvalue="#{atributo.nombreJava}"/>
                                                   </f:facet>
                                                   <f:facetname="input">

      <p:inputTextvalue="#{atributo.nombreJava}"/>
                                                   </f:facet>
                                             </p:cellEditor>
                                       </p:column>
                                       <p:columnheaderText="Tipo">
                                             <p:cellEditor>
```

```
                                                <f:facetname="output">

<p:outputLabelvalue="#{atributo.tipoDato}"/>
                                                </f:facet>
                                                <f:facetname="input">

<h:selectOneMenuvalue="#{atributo.tipoDato}"

style="width:100%">

<f:selectItemsvalue="#{personalizacionControlador.tipoDatos}"

var="tipoDato"itemLabel="#{tipoDato}"

itemValue="#{tipoDato}"/>
                                                      </h:selectOneMenu>
                                                </f:facet>
                                          </p:cellEditor>
                                    </p:column>
                                    <p:columnheaderText="Not Null">
                                          <p:cellEditor>
                                                <f:facetname="output">

<p:outputLabelvalue="#{atributo.notNull}"/>
                                                </f:facet>
                                                <f:facetname="input">

<h:selectOneMenuvalue="#{atributo.notNull}"style="width:100%">

<f:selectItemitemLabel="true"itemValue="true"/>

<f:selectItemitemLabel="false"itemValue="false"/>
                                                      </h:selectOneMenu>
                                                </f:facet>
                                          </p:cellEditor>
                                    </p:column>
                                    <p:columnheaderText="PK">

<p:outputLabelvalue="#{atributo.esPk}"/>
                                    </p:column>
                                    <p:columnheaderText="FK">

<p:outputLabelvalue="#{atributo.esFk}"/>
                                    </p:column>
                                    <p:columnheaderText="Serial">

<p:outputLabelvalue="#{atributo.esSerial}"/>
                                    </p:column>
                              </p:dataTable>
                        </p:panel>
```

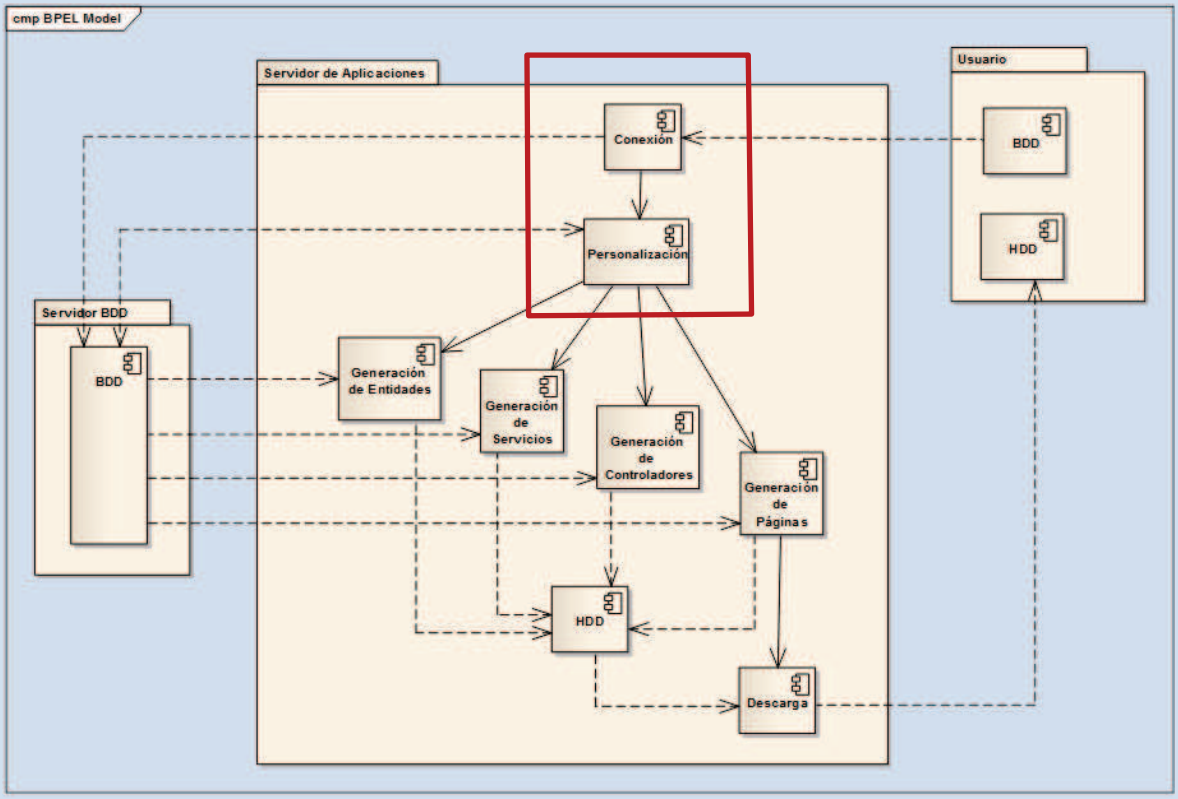
2.3 PRUEBAS DE INTEGRACIÓN

Estas pruebas se realizan con la finalidad de, comprobar que los distintos módulos desarrollados trabajen correctamente en conjunto, las pruebas se harán

de manera incremental, es decir que se van a probar módulos, añadidos a lo que ya está probado anteriormente[25]. Para realizar estas pruebas, se va a seguir un formato, el cual consta de los siguientes campos.

- Título: Se muestra la palabra prueba, seguida de un número distintivo.
- Módulos a integrar: Se muestran los módulos que se incluyen en la prueba, además de un gráfico que muestra los módulos que se integrarán dentro de un esquema del sistema.
- Descripción: es una explicación detallada de la prueba.
- Parámetros de entrada: Se refiere a los datos que deberán ser ingresados por el usuario.
- Flujo de Ejecución: Detalla las acciones y el orden en que estas se realizarán durante la prueba.
- Prerrequisitos: se refiere a los requisitos que deben ser cubiertos antes de efectuar la prueba.
- Módulo de Control de la Prueba:Aquí se muestra la interfaz que se utilizará para el propósito indicado, en el caso de ser necesaria una nueva interfaz.
- Resultados esperados: se refiere a los resultados que se desea obtener.
- Resultado final: se refiere al resultado que dio la prueba como tal.

Prueba 1
Módulos a integrar: Conexión y Edición



Descripción:

Esta prueba permitirá visualizar la integración del módulo que permite la conexión a la base de datos, con el modulo que permite realizar la edición de la información que ha sido obtenida.

Parámetros de Entrada:

- Driver
- JDBC
- Usuario
- Password

Flujo de Ejecución:

Se mostrarán las pantallas que forman parte del módulo de conexión, una vez que se haya realizado la conexión, se navegará a la pantalla de edición, en la cual se podrán editar los campos, y se almacenaráen la base de datos la información modificada.

Prerrequisitos :

- Base de datos en el usuario.
- Si la aplicación se está ejecutando de manera remota, el cliente debe tener habilitado el acceso remoto a la base de datos.

Resultados esperados:

En caso de que la conexión sea correcta se mostrará un mensaje indicando que la

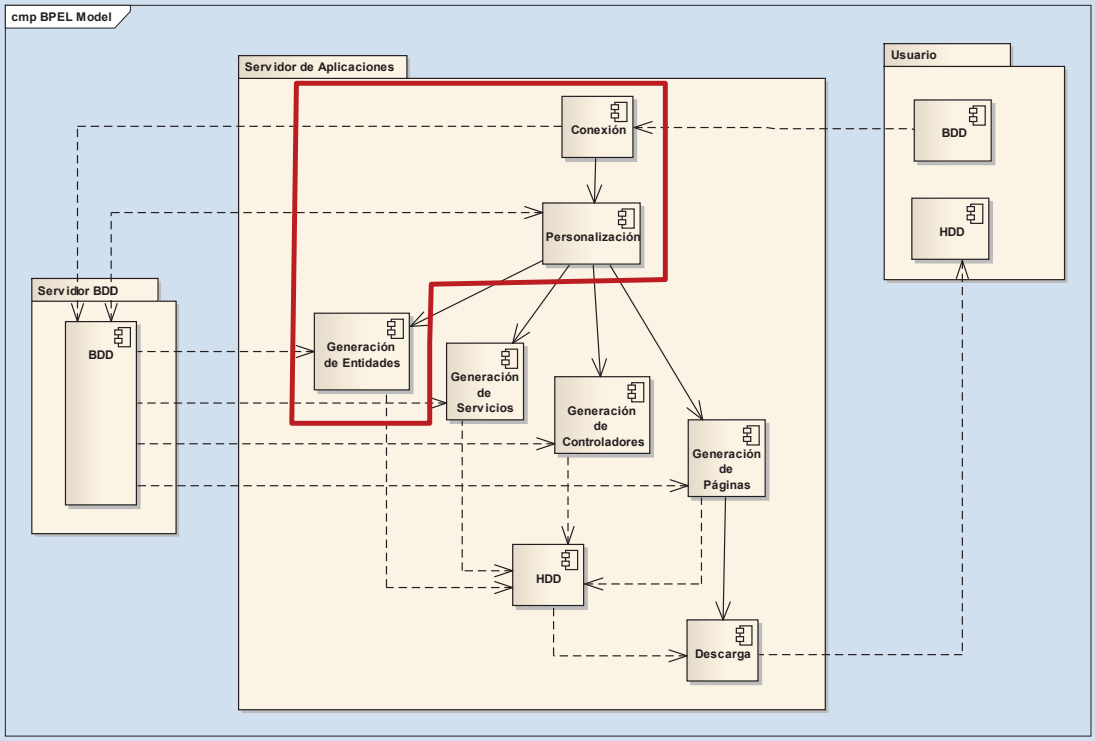
conexión es correcta, además de mostrarse los esquemas. Una vez que se seleccione un esquema, se mostrarán las tablas existentes en el mismo. En caso de que los parámetros sean incorrectos, se mostrará un mensaje indicando que se presentó un error. Luego se verá una pantalla en la que se puede seleccionar las tablas, y al seleccionar una, los campos existentes en dicha se mostrarán y los campos establecidos podrán ser editados. Estos cambios se verán reflejados en la base de datos.

Resultado final:

La prueba se realizó de manera correcta, obteniendo los resultados esperados.

Prueba 2

Módulos a integrar: Conexión, Edición y Generación de Entidades.



Descripción:

Se deberá realizar el procedimiento especificado en la Prueba 1, además se integrará con el módulo de generación de entidades.

Parámetros de Entrada:

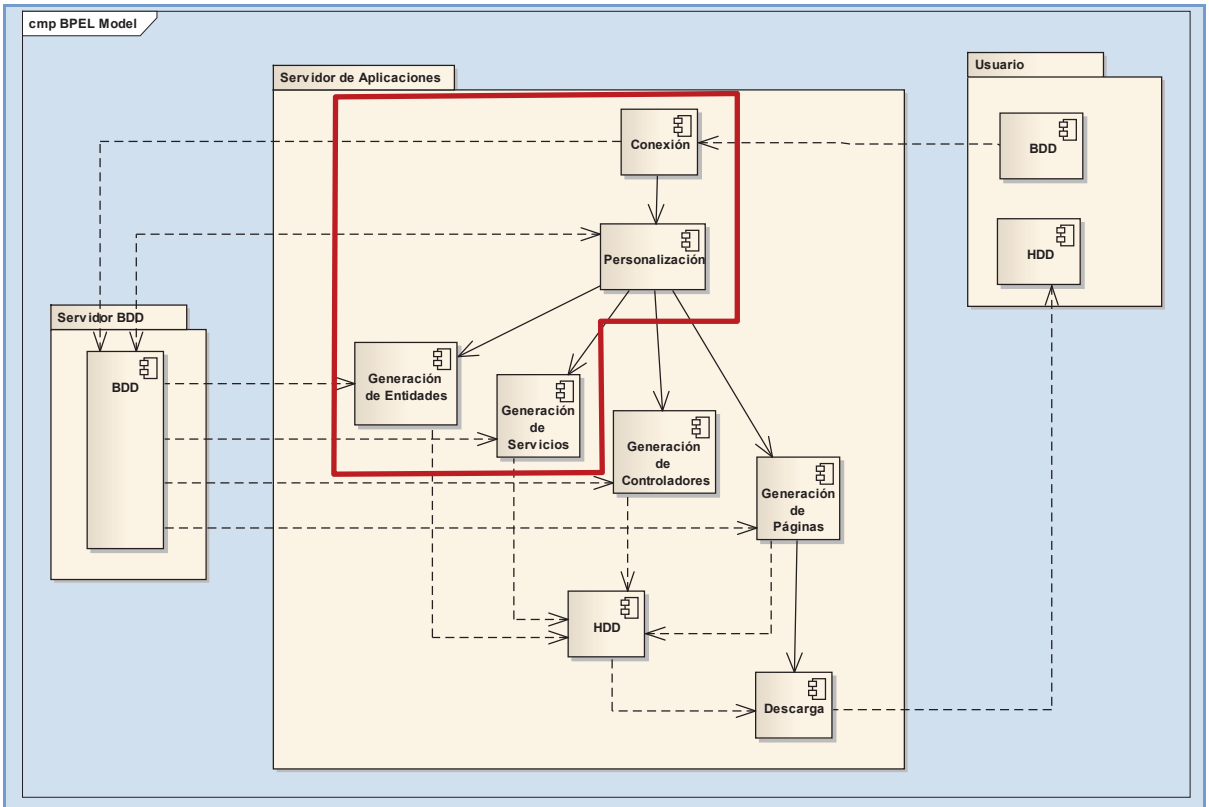
- Driver
- JDBC
- Usuario
- Password
- Nombre del Proyecto
- Nombre del Paquete

Flujo de Ejecución:

A partir del final de la Prueba 1, se debe navegar a la pantalla para generación de entidades.

• Datasource
Prerrequisitos : • Base de datos en el cliente. • Si la aplicación se está ejecutando de manera remota, el cliente debe tener habilitado el acceso remoto a la base de datos.
Resultados esperados: Además de los resultados esperados en la Prueba 1, se deberá generar la carpeta con el nombre del proyecto, dentro de ella se generaran tantas carpetas, como se haya especificado en el nombre del paquete, luego se generará una carpeta de nombre Entidades, y dentro de esta carpeta, se alojarán los archivos que se generen, de acuerdo al número de tablas existentes en la base de datos, además de los archivos que se generan en caso de que existan tablas de rompimiento.
Resultado final: La prueba se realizó de manera correcta, obteniendo los resultados esperados.

Prueba 3
Módulos a integrar: Conexión, Edición, Generación de Entidades y Generación de Servicios.



Descripción:

Se deberá realizar el procedimiento especificado en la Prueba 2, añadiendo el módulo de Generación de Servicios.

Parámetros de Entrada:

- Driver
- JDBC
- Usuario
- Password
- Nombre del Proyecto
- Nombre del Paquete
- Datasource

Flujo de Ejecución:

A partir de donde se llega en la Prueba 2, se navegará hacia la pantalla de generación de servicios.

Prerrequisitos :

- Base de datos en el cliente.
- Si la aplicación se está ejecutando de manera remota, el cliente debe tener habilitado el acceso remoto a la base de datos.

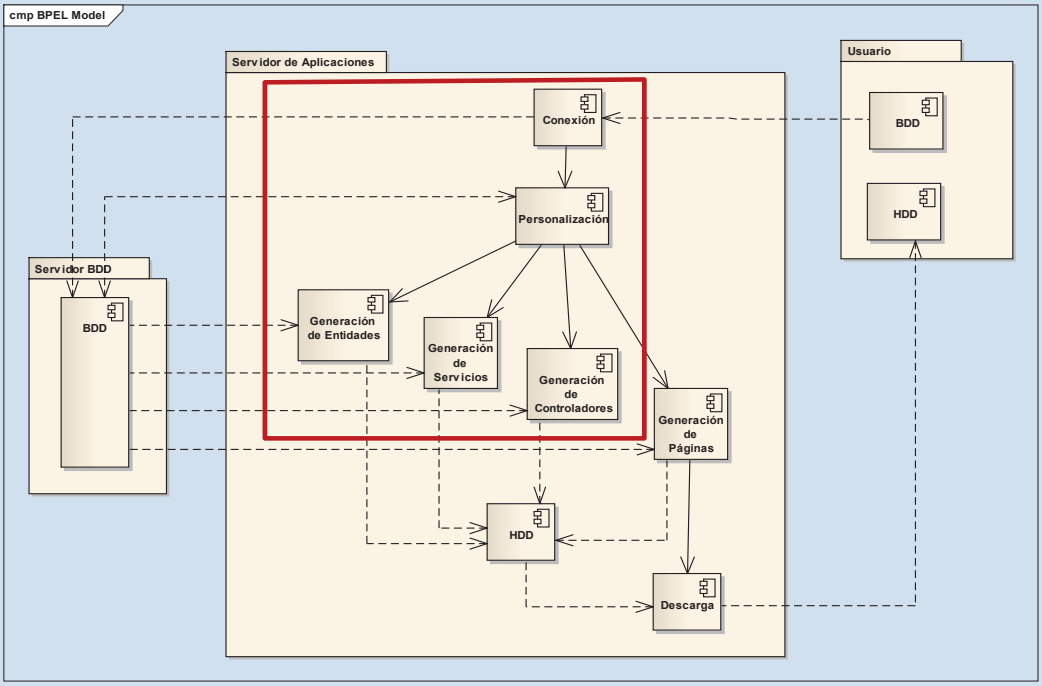
Resultados esperados:

Además de los resultados esperados en la Prueba 2, se deberá generar una carpeta de nombre servicios, de acuerdo con la jerarquía de carpetas, especificada en el nombre del paquete ingresado, la cual contendrá los archivos generados para los servicios, de acuerdo a las tablas con las que cuente la base de datos.

Resultado final:

La prueba se realizó de manera correcta, obteniendo los resultados esperados.

Prueba 4
Módulos a integrar: Conexión, Edición, Generación de Entidades, Generación de Servicios y Generación de Controladores.



Descripción:

Se deberá realizar el procedimiento especificado en la Prueba 3, se agregará el módulo de Generación de Controladores.

Parámetros de Entrada:

- Driver
- JDBC
- Usuario
- Password
- Nombre del Proyecto
- Nombre del Paquete
- Datasource

Flujo de Ejecución:

A partir de donde se llega en la Prueba 3, se debe navegar hacia la pantalla de generación de controladores.

Prerrequisitos :

- Base de datos en el cliente.
- Si la aplicación se está ejecutando de manera remota, el cliente debe tener habilitado el acceso remoto a la base de datos.

Resultados esperados:

Además de los resultados esperados en la Prueba 3, se deberá generar una carpeta de nombre controladores, de acuerdo con la jerarquía de carpetas, especificada en el nombre del paquete ingresado, la cual contendrá los archivos generados para los

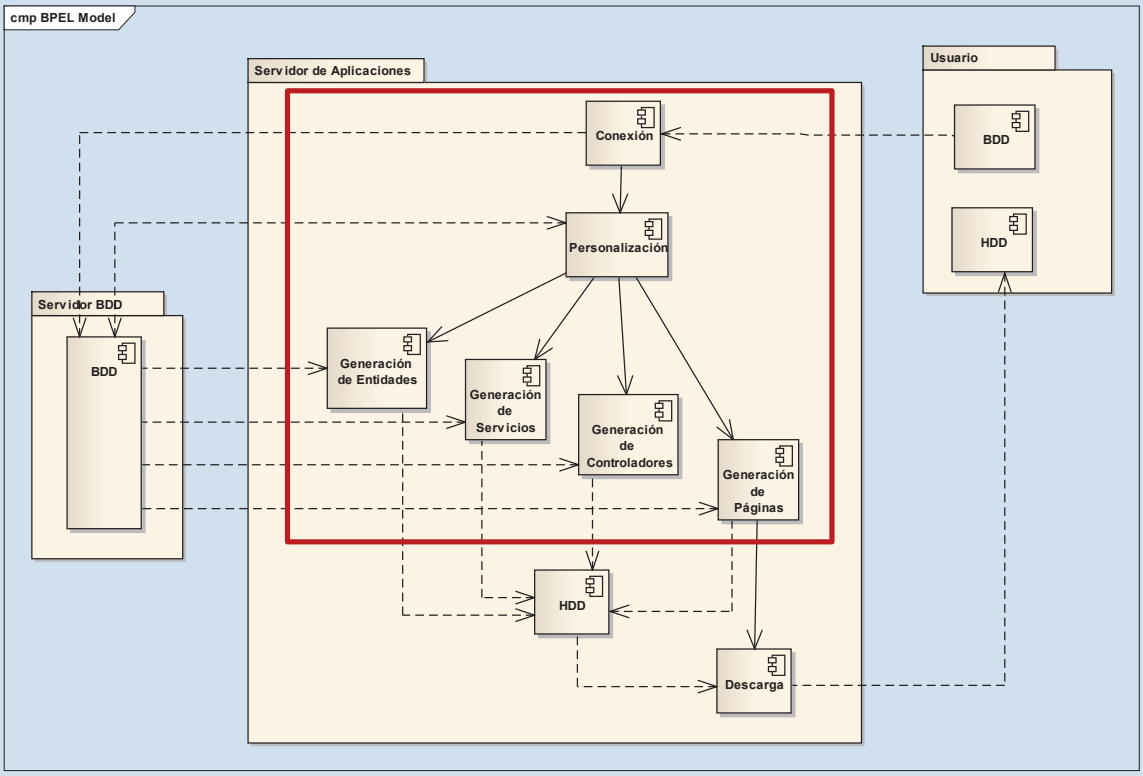
controladores, existirán tantos archivos, como tablas existan en la base de datos, exceptuando las tablas de rompimiento, ya que su tratamiento es diferente.

Resultado final:

La prueba se realizó de manera correcta, obteniendo los resultados esperados.

Prueba 5

Módulos a integrar: Conexión, Edición, Generación de Entidades, Generación de Servicios, Generación de Controladores y Generación de Páginas.



Descripción:

Se deberá realizar el procedimiento especificado en la Prueba 4, se agregará el módulo de Generación de Páginas.

Parámetros de Entrada:

- Driver
- JDBC
- Usuario
- Password
- Nombre del Proyecto
- Nombre del Paquete
- Datasource

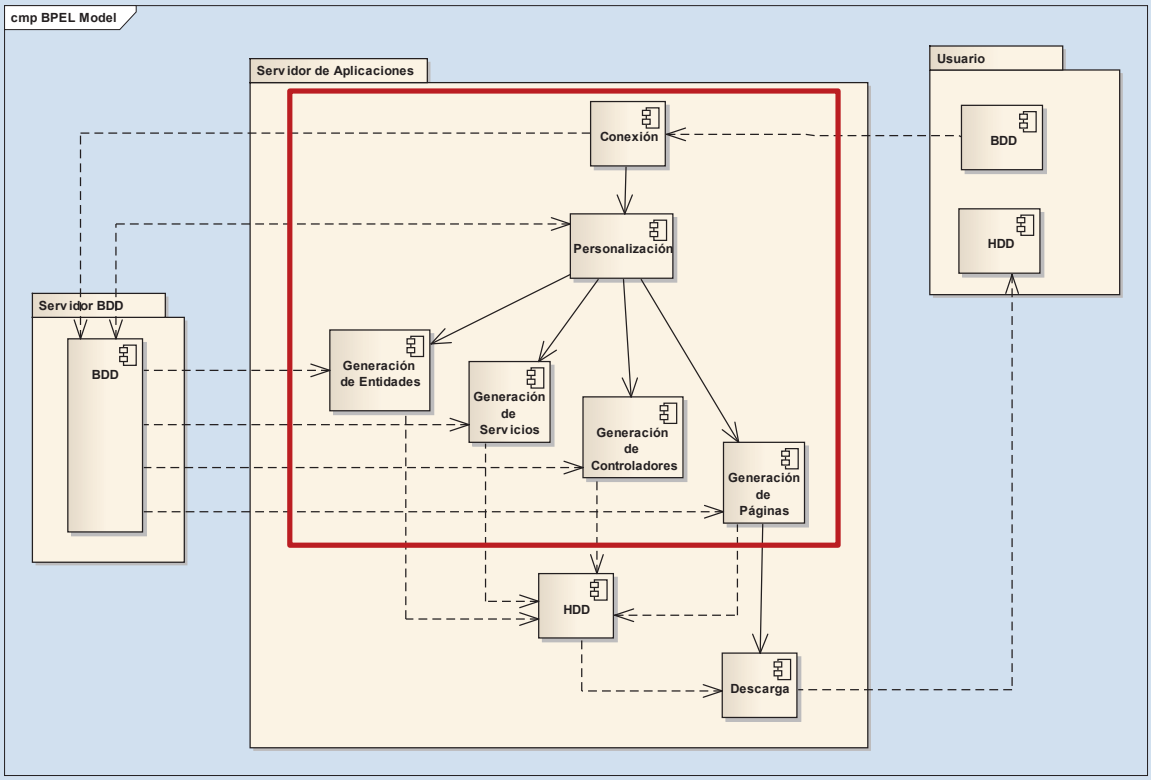
Flujo de Ejecución:

A partir de donde se llega en la Prueba 4, se va a navegar hacia la pantalla de generación de páginas.

Prerrequisitos :

• Base de datos en el cliente. • Si la aplicación se está ejecutando de manera remota, el cliente debe tener habilitado el acceso remoto a la base de datos.
Resultados esperados: Además de los resultados esperados en la Prueba 4, se deberá generar una carpeta de nombre páginas, de acuerdo con la jerarquía de carpetas, especificada en el nombre del paquete ingresado, la cual contendrá los archivos generados para los páginas, el número de páginas, será el mismo que el número de controladores.
Resultado final: La prueba se realizó de manera correcta, obteniendo los resultados esperados.

Prueba 6
Módulos a integrar: Conexión, Edición, Generación de Entidades, Generación de Servicios, Generación de Controladores y Generación de Páginas.



Descripción:

Se deberá realizar el procedimiento especificado en la Prueba 4, se agregará el módulo de Generación de Páginas. La diferencia de esta prueba con la anterior es que, esta prueba se hará con el objetivo de que los módulos de Generación de Entidades, Generación de Servicios, Generación de Controladores, Generación de Páginas, se ejecuten a la vez utilizando un solo botón, para aminorar el número de interfaces utilizadas, y darle más comodidad al usuario al momento de generar los archivos.

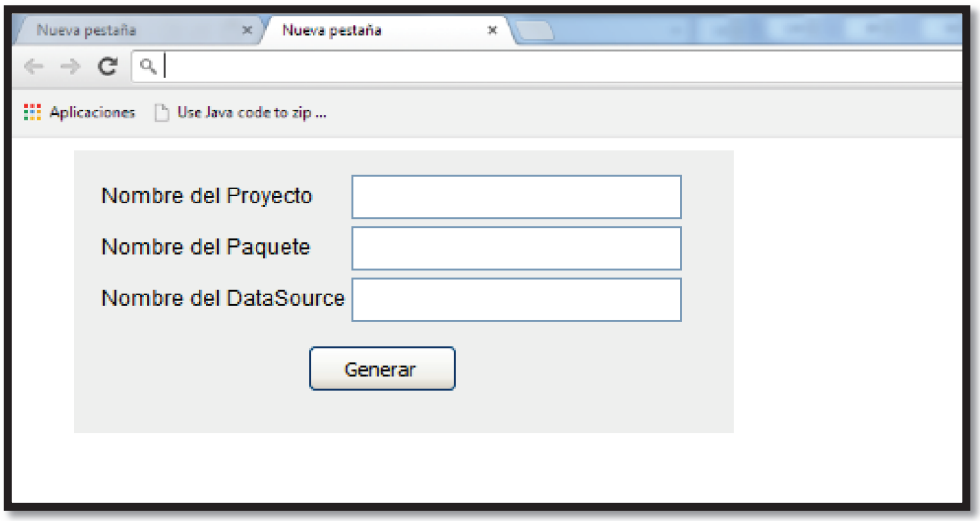
Parámetros de Entrada:

- Driver
- JDBC
- Usuario
- Password
- Nombre del Proyecto
- Nombre del Paquete
- Datasource

Flujo de Ejecución:

Se partirá desde el resultado de la Prueba 1, luego se navegará a una interfaz que permitirá la generación de todos los elementos.

Módulo de Control de la Prueba



Prerrequisitos :

- Base de datos en el cliente.
- Si la aplicación se está ejecutando de manera remota, el cliente debe tener habilitado el acceso remoto a la base de datos.

Resultados esperados:

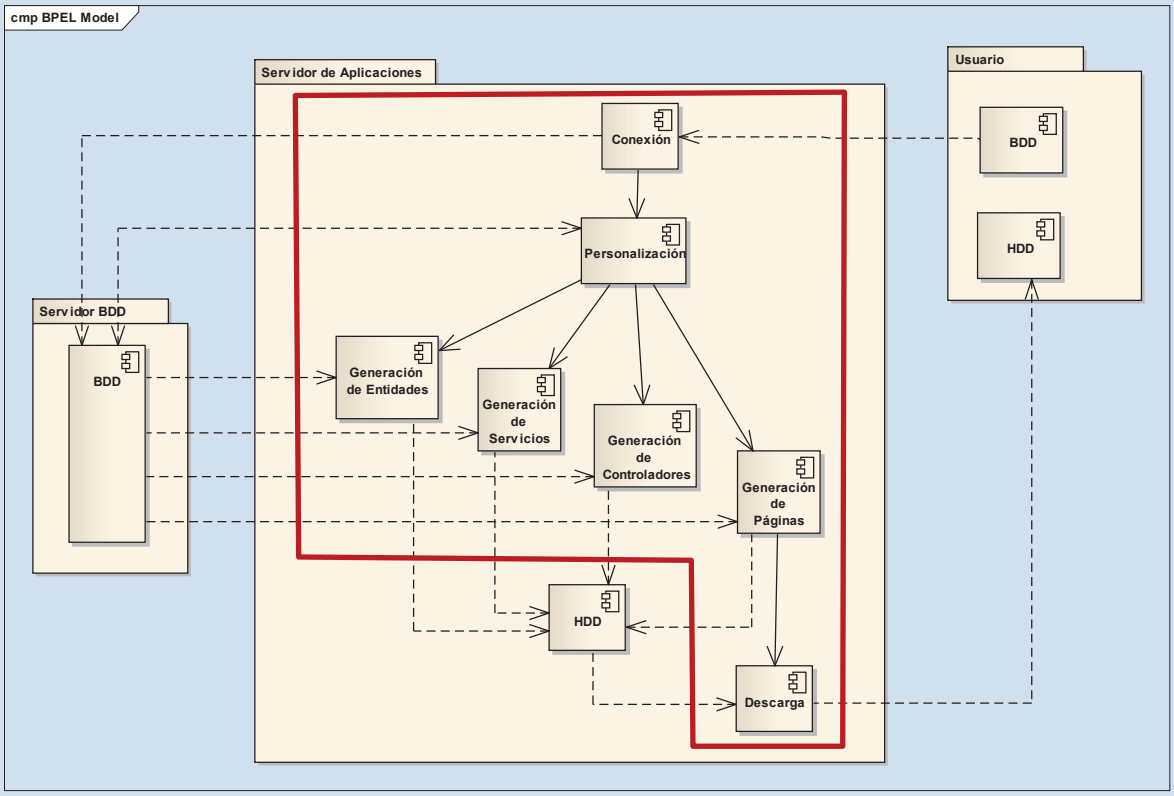
Se deberá tener los mismos resultados obtenidos en la prueba 5. Es decir todos los archivos generados en sus respectivas carpetas, de acuerdo a la jerarquía especificada en el paquete. Además se generará una carpeta recursos, la misma que contendrá el archivo Persistence.xml, un template que será usado como el template por defecto, y un archivo de instrucciones de uso.

Resultado final:

La prueba se realizó de manera correcta, obteniendo los resultados esperados.

Prueba 7
Módulos a integrar: Conexión, Edición, Generación de Entidades, Generación de

Servicios, Generación de Controladores, Generación de Páginas y Descarga.



Descripción:

Se deberá realizar el procedimiento especificado en la Prueba 5, se agregará el módulo de Descarga.

Parámetros de Entrada:

- Driver
- JDBC
- Usuario
- Password
- Nombre del Proyecto
- Nombre del Paquete
- Datasource

Flujo de Ejecución:

A partir de donde se llega en la Prueba 5, se va a navegar hacia la página de descarga. La interfaz a utilizar será la misma que se especificó en el diseño del sprint.

Prerrequisitos :

- Base de datos en el cliente.
- Si la aplicación se está ejecutando de manera remota, el cliente debe tener habilitado el acceso remoto a la base de datos.

Resultados esperados:

Además de los resultados esperados en la Prueba 5, se permitirá la descarga de todo el contenido generado. El contenido generado será comprimido en un archivo .zip, para luego ser descargado desde el navegador en que la aplicación se esté ejecutando.

Resultado final:

La prueba se realizó de manera correcta, obteniendo los resultados esperados.

CAPÍTULO 3

3 SIMULACIÓN DEL SISTEMA

Finalizada la construcción del sistema, se procede a realizar la simulación del mismo, definiendo casos específicos, con la finalidad de verificar que los requerimientos han sido cubiertos y que el sistema cumple con su cometido. La simulación parte con el levantamiento del ambiente, luego se realizará la ejecución de cada uno de los casos de simulación y se finalizará con el análisis de los resultados obtenidos.

3.1 EJECUCIÓN DEL PROTOTIPO

3.1.1 HARDWARE EMPLEADO EN LA SIMULACIÓN

Servidor

En este servidor se alojaran la base de datos, el servidor de aplicaciones y el prototipo CODGEN, sobre el cual los usuarios trabajarán.

Características

IP: 192.168.0.1.

RAM: 2 GB.

PROCESADOR: Intel Core I5 3470 CPU 3.20GHz.

DISCO: 500 GB.

INTERFAZ DE RED: Realtek RTL8168/8111 PCI-E Gigabit Ethernet NIC.

Ordenador Usuario

Este será un computador desde el cual los usuarios realizaran las peticiones al prototipo CODGEN.

Características

IP: 192.168.0.2.

RAM: 4 GB.

PROCESADOR: Intel Core I3 2310M 2.10GHz.

DISCO: 500 GB.

INTERFAZ DE RED: Realtek RTL8168/8111 PCI-E Gigabit Ethernet NIC.

3.1.2 SOFTWARE EMPLEADO EN LA SIMULACIÓN

Servidor

Características

SISTEMA OPERATIVO: Windows 7 Ultimate 64 bits.

DBMS: postgresql-9.3.2.

SERVIDOR DE APLICACIONES: jboss-as-7.1.1.Final.

Ordenador Usuario

Características

SISTEMA OPERATIVO: Windows 8.1.

DBMS: postgresql-9.3.2.

NAVEGADOR: Chrome 33.0.1750.154.

3.1.3 CASOS DE SIMULACIÓN

3.1.3.1 Caso de Simulación Número Uno - Base de Datos con relación de muchos a muchos.

3.1.3.1.1 *Estructura de la Base de Datos*

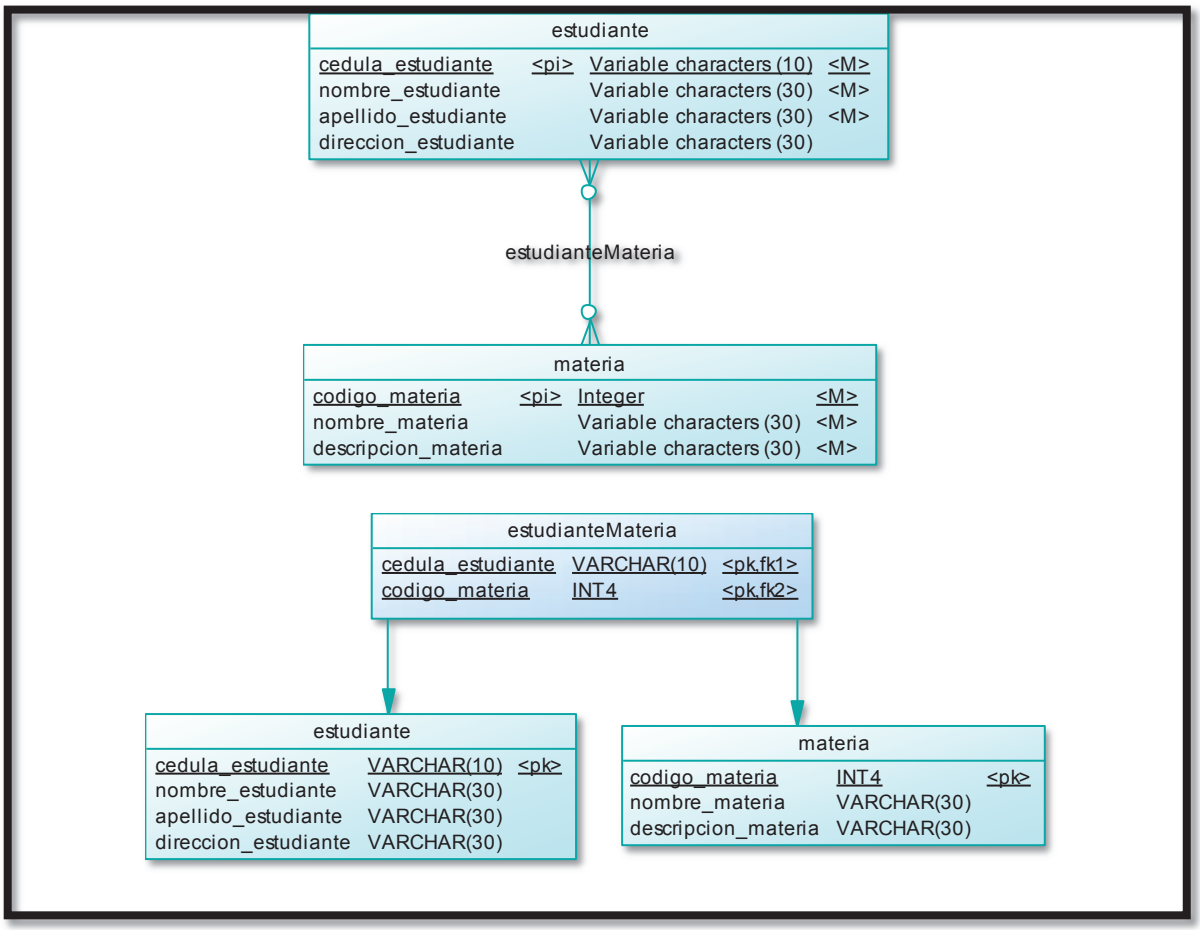


Figura 47 Diagrama Conceptual y Físico de la Base de Datos Con Relación de muchos a muchos.

3.1.3.1.2 *Pantallas de simulación*

Figura 48 Pantalla de Conexión.

Figura 49 Pantalla de Selección de Esquemas.

Figura 50 Diálogo de Tablas Encontradas.

Figura 51 Pantalla de Edición de Tablas.

Figura 52 Pantalla de Ingreso de Datos del Proyecto.

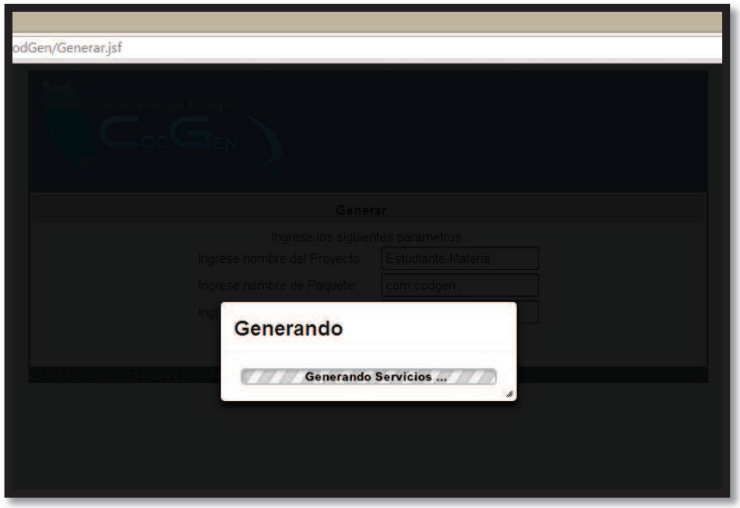


Figura 53 Pantalla de Espera.

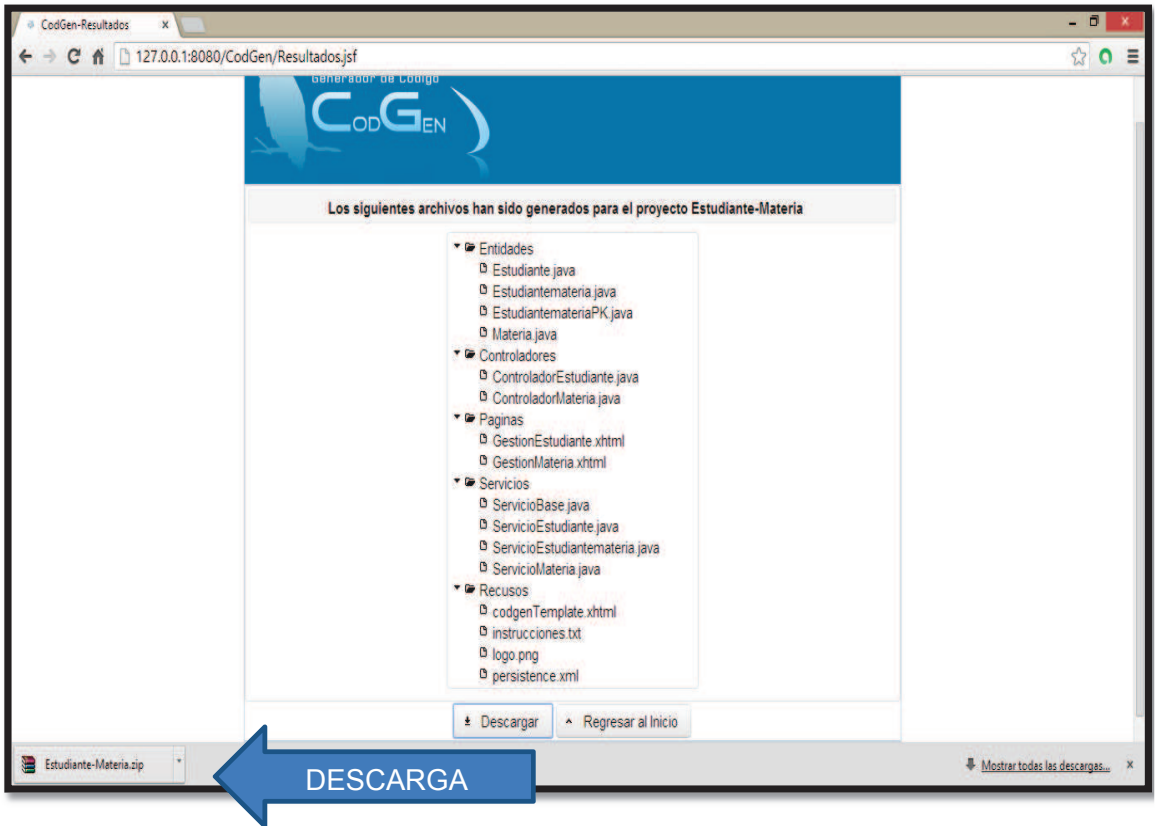


Figura 54 Pantalla de Información de Resultados y Descarga.

3.1.3.1.3 *Resultado*

Se pueden ver los archivos y las carpetas generadas en la jerarquía indicada. El resultado de la simulación fue satisfactorio.

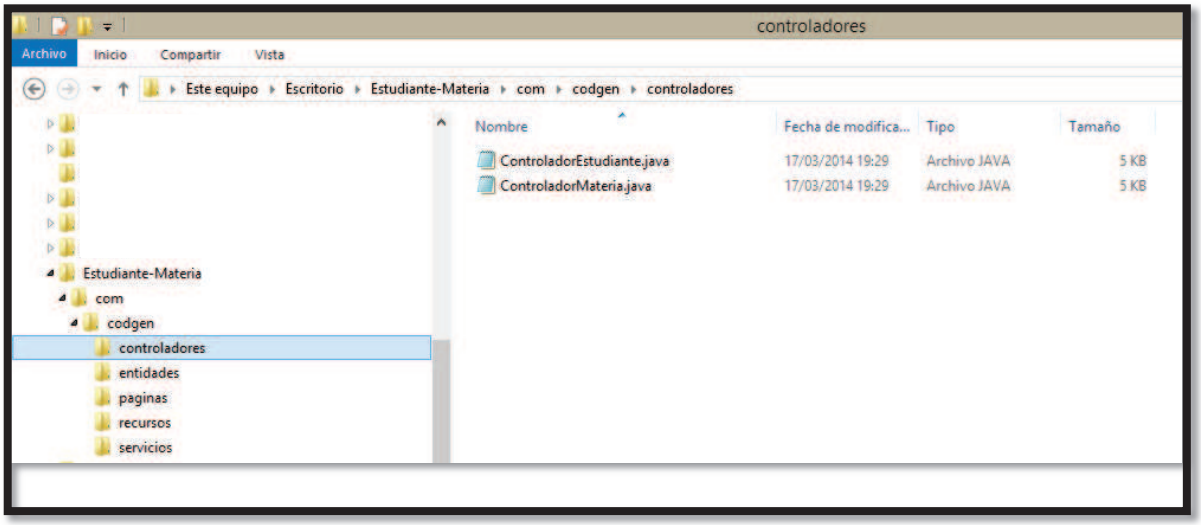


Figura 55 Captura de los Resultados Simulación Uno.

3.1.3.2 Caso de Simulación Número Dos - Base de Datos con relación de uno a muchos.

3.1.3.2.1 *Estructura de la Base de Datos*

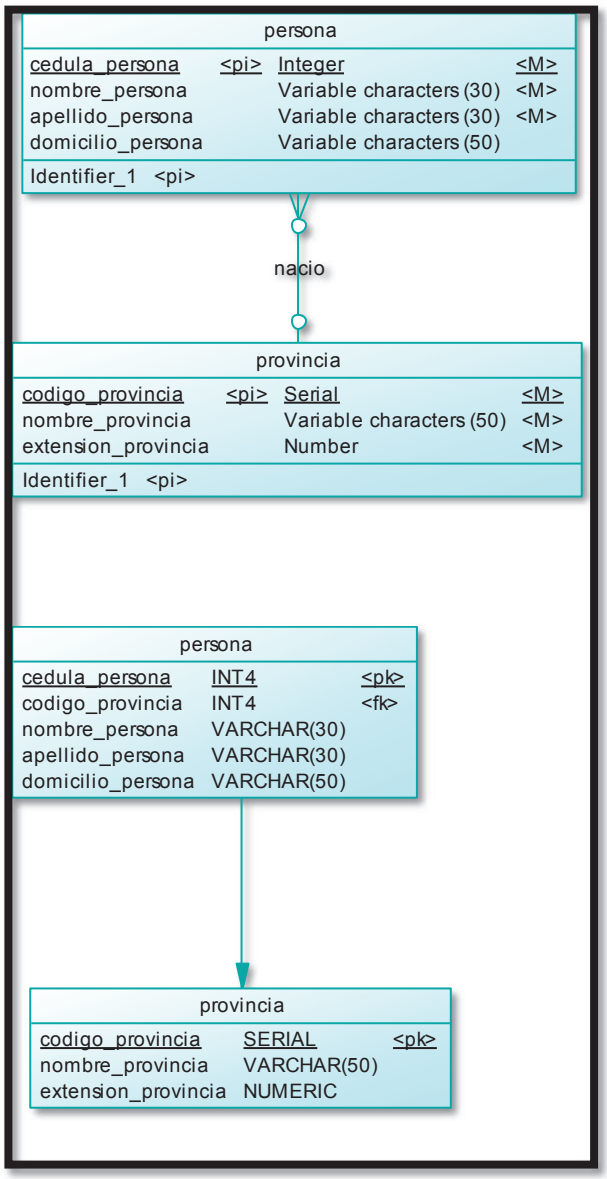


Figura 56 Diagrama Lógico y Físico de la Base de Datos Con Relación de Uno a Muchos.

3.1.3.2.2 *Pantallas de simulación*

Se mostrarán únicamente algunas pantallas, ya que el flujo es el mismo que en la simulación anterior.

Figura 57Pantalla de Edición de Tablas (Simulación Dos).

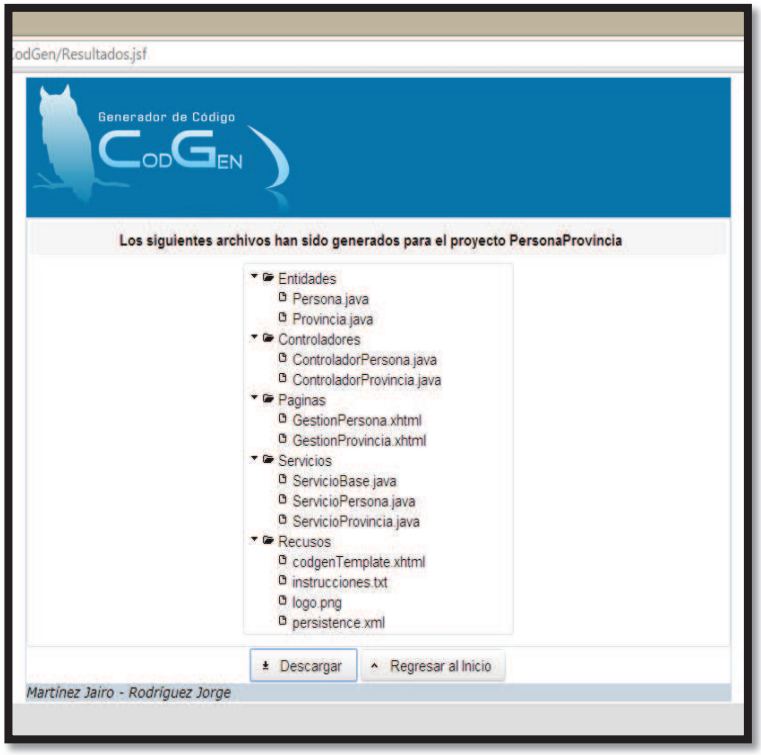


Figura 58Pantalla de Información de Resultados y Descarga (Simulación Dos).

3.1.3.2.3 *Resultados*

En la siguiente imagen se puede ver los archivos y las carpetas generadas de acuerdo a la jerarquía especificada. El resultado de esta simulación fue exitoso.

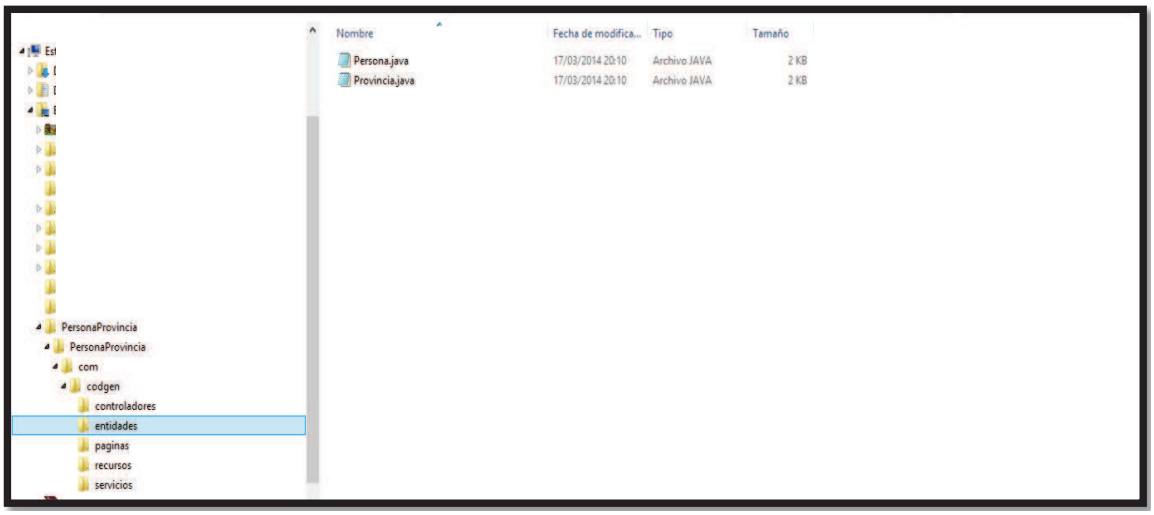


Figura 59Captura de los Resultados Simulación Dos.

3.1.3.3 Caso de Simulación Número Tres - Base de Datos con relación de uno a muchos a muchos, tabla recursiva y tabla suelta.

3.1.3.3.1 *Estructura de la Base de Datos*

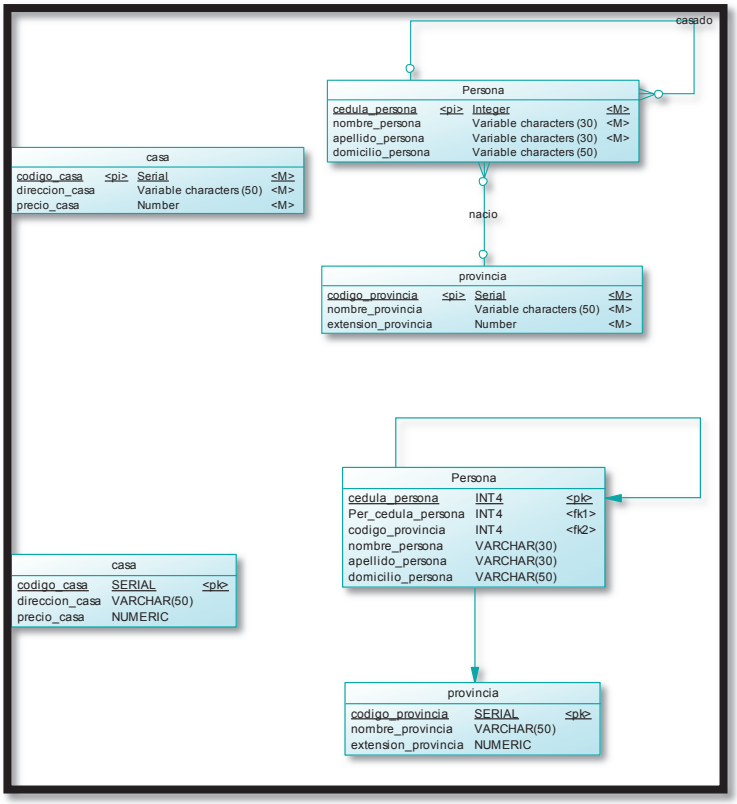


Figura 60 Diagrama Lógico y Físico de la Base de Datos con relación de uno a muchos a muchos, tabla recursiva y tabla suelta.

3.1.3.3.2 *Pantallas de simulación*

Se mostrarán únicamente algunas pantallas, ya que el flujo es el mismo que en la simulación anterior.

Pantalla que muestra un error en los parámetros de conexión.

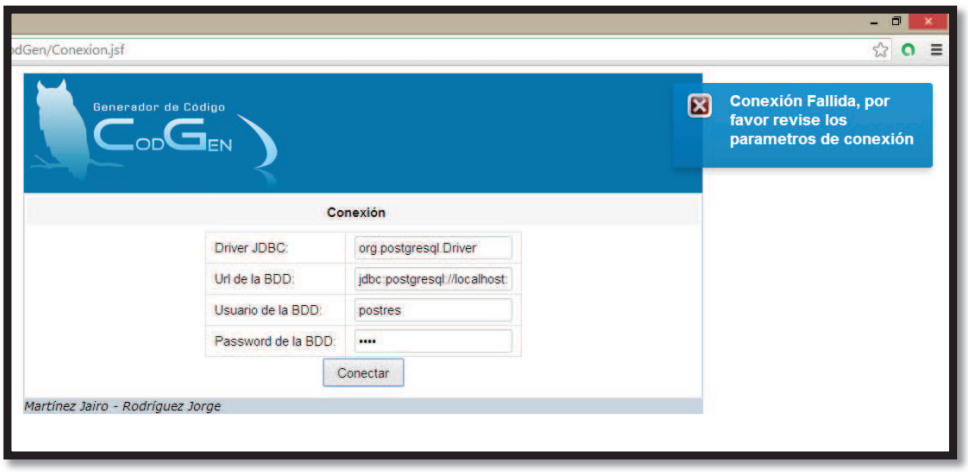


Figura 61 Pantalla de Conexión Simulación Tres (Muestra de Error en la Conexión).

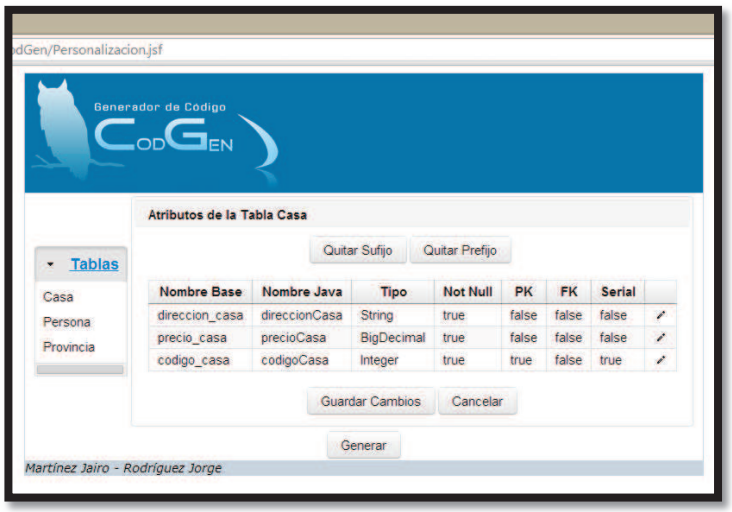


Figura 62 Pantalla de Edición de Tablas (Simulación Tres).

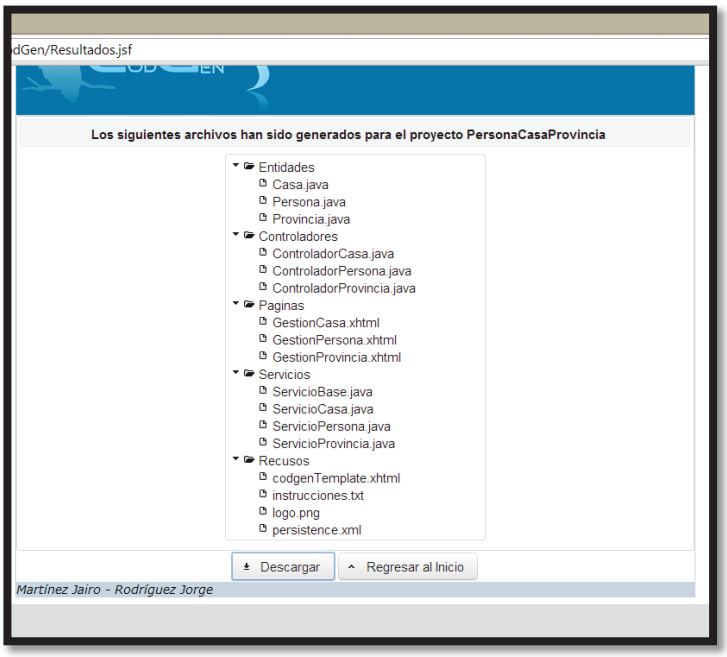


Figura 63 Pantalla de Resultados y Descarga (Simulación Tres).

3.1.3.3.3 *Resultados*

En la siguiente imagen se puede ver los archivos y las carpetas generadas de acuerdo a la jerarquía especificada. El resultado de esta simulación fue exitoso.

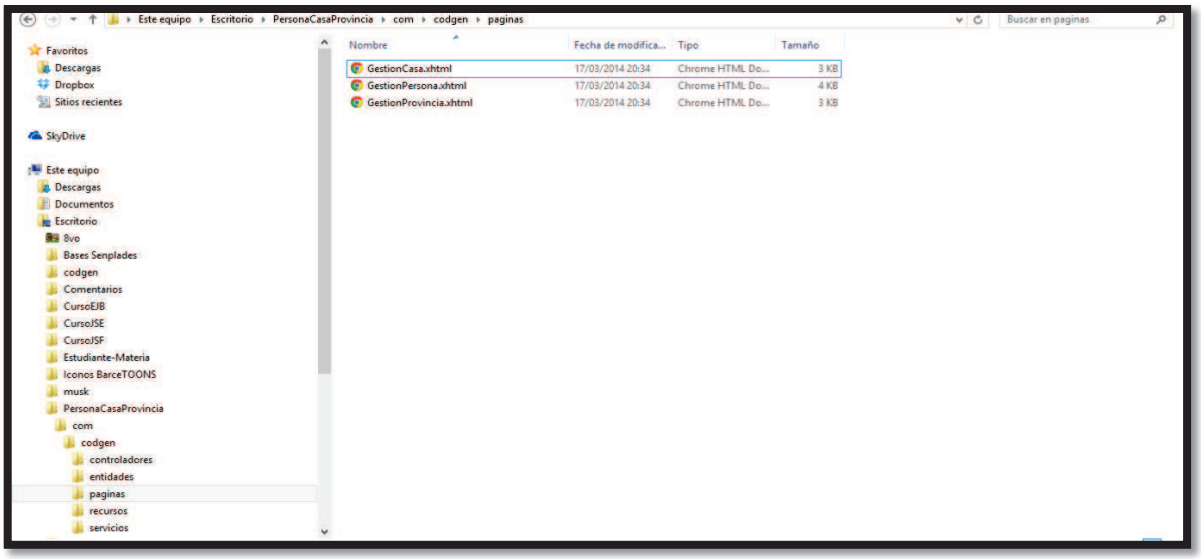


Figura 64 Captura de Resultados (Simulación Tres).

3.1.3.4 Caso de Simulación Número Cuatro - Base de Datos con tipos de datono considerados por la aplicación.

3.1.3.4.1 *Estructura de la Base de Datos*

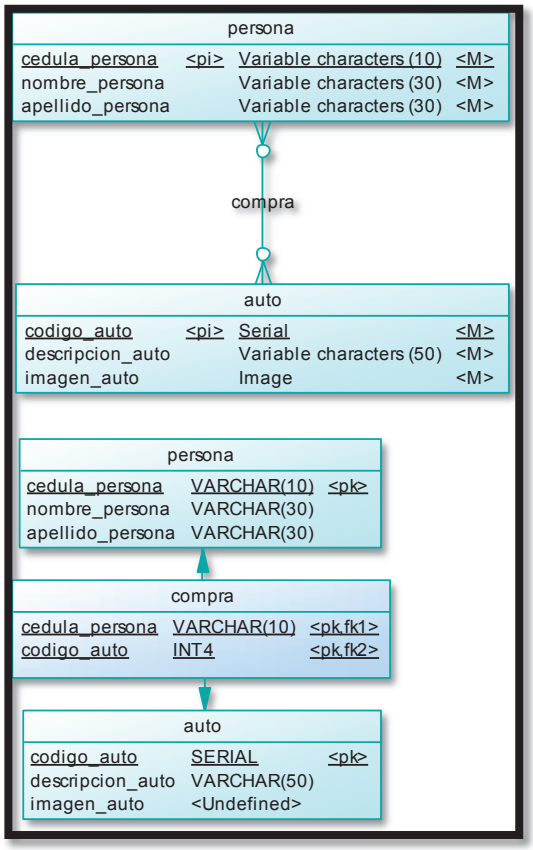


Figura 65 Diagrama Lógico y Físico de la Base de Datos con tipos de dato no considerados por la aplicación.

3.1.3.4.2 *Pantallas de simulación*

Figura 66 Pantalla de Error - Tipos de Datos no reconocidos por la Aplicación.

3.1.3.4.3 *Resultados*

Al existir un tipo de dato que no está considerado en la aplicación, el mismo es detectado y la aplicación se detiene. El resultado de la simulación fue exitoso.

3.1.3.5 Caso de Simulación Número Cinco - Base de Datos sin tablas.

3.1.3.5.1 *Pantallas de simulación*

Figura 67 Pantalla de Error - Base de Datos Sin Tablas.

3.1.3.5.2 *Resultados*

Cuando la base de datos no tiene tablas, la aplicación muestra una pantalla mostrando el error, y termina el proceso. El resultado de la simulación fue exitoso.

3.1.3.6 Caso de Simulación Número Seis - Base de Datos con relación de uno a uno.

3.1.3.6.1 *Estructura de la Base de Datos*

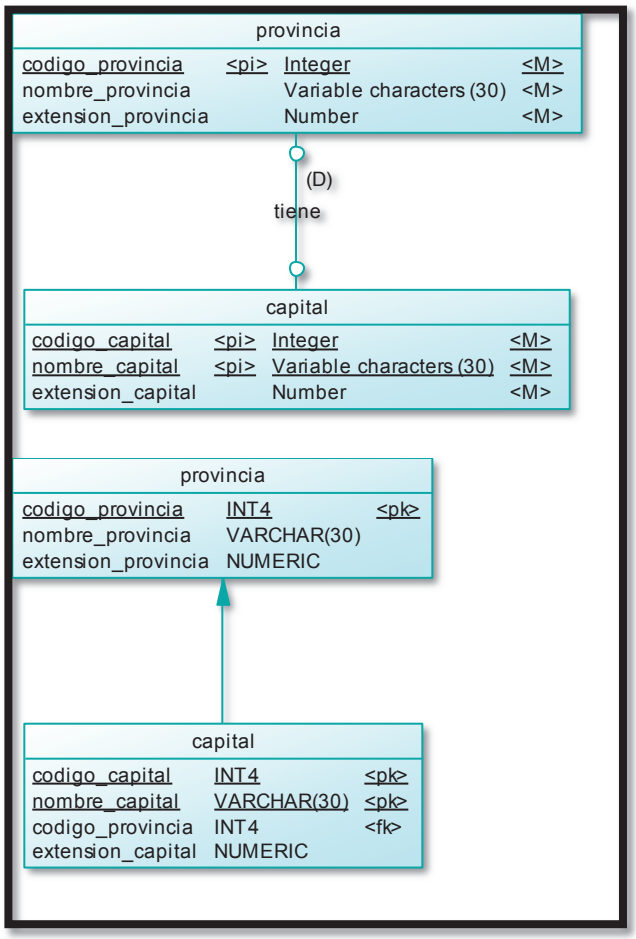


Figura 68 Diagrama Lógico y Físico de la Base de Datos con relación de uno a uno.

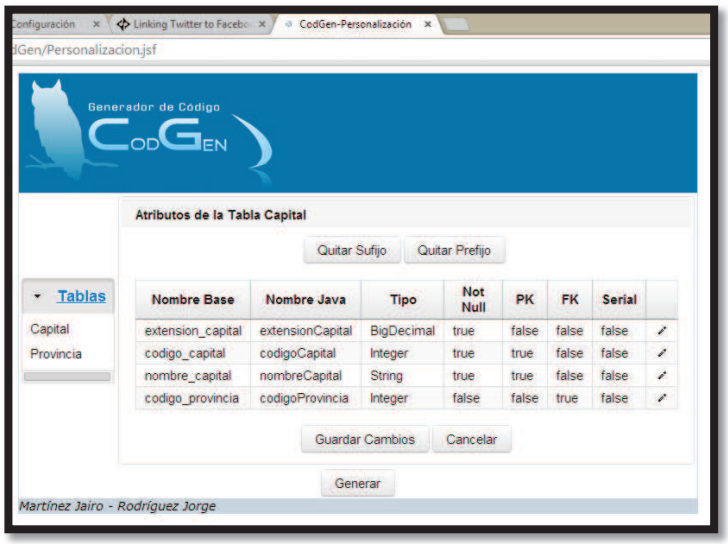


Figura 69 Pantalla de Edición (Simulación Seis).

Figura 70 Pantalla de Resultados y Descarga (Simulación Seis).

3.1.3.6.2 *Resultados*

Se puede ver en la imagen siguiente que se generan las carpetas y los archivos según la jerarquía indicada, por lo que se puede concluir que el resultado de la simulación fue exitoso.

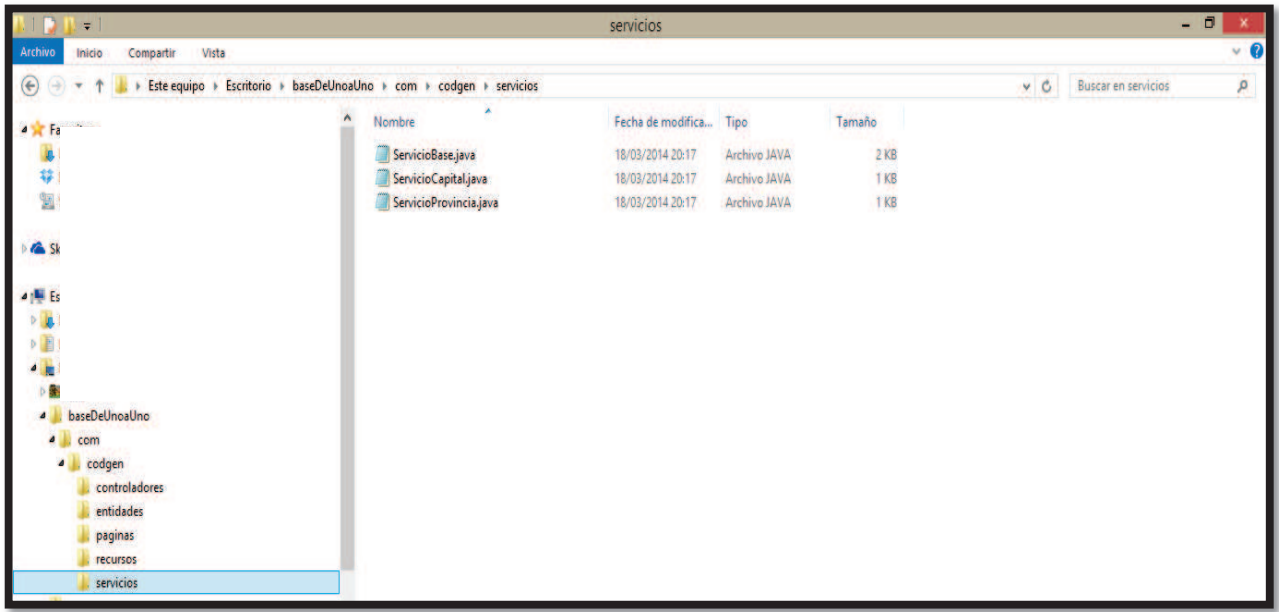


Figura 71 Captura de Resultados (Simulación Seis).

3.2 ANÁLISIS DE RESULTADOS

Después de haber realizado la simulación del prototipo, se procede a realizar el análisis de los resultados obtenidos.

Para lo cual se ha realizado la siguiente tabla, en la cual se catalogará los resultados obtenidos, entre, Excelente, Bueno y Malo.

Caso de Simulación	Excelente	Bueno	Malo
Base de Datos con relación de muchos a muchos.	x		
Base de Datos con relación de uno a muchos.	x		
Base de Datos con relación de uno a muchos a muchos, tabla recursiva y tabla suelta.	x		
Base de Datos con tipos de dato no considerados por la aplicación.	x		
Base de Datos sin tablas.	x		
Base de Datos con relación de uno a uno.	x		

Tabla 29 Análisis de Resultados.[58]

De un total de 7 simulaciones, las 7 fueron catalogadas como excelentes. De este análisis, se puede concluir que el prototipo cumple de manera eficiente y satisfactoria con los requerimientos expuestos por el usuario y está en capacidad de ser utilizado en los proyectos que la empresa lo requiera.

CAPÍTULO 4

4 CONCLUSIONES Y RECOMENDACIONES

4.1 CONCLUSIONES

[58]Tabla realizada por los Autores.

- La participación del cliente durante el desarrollo, que es uno de los principales lineamientos del marco de trabajo Scrum, facilitó la labor, y permitió que el producto final cumpla con las especificaciones de manera correcta. Desde la fase de levantamiento de requerimientos se notó la importancia de recoger la mayor cantidad de información de la manera precisa, para que luego los cambios no influyan drásticamente en el desarrollo realizado en cada sprint.

- Todos los casos realizados en la simulación del prototipo, arrojaron buenos resultados, por lo que se pudo concluir que todos los requerimientos especificados se cumplieron a cabalidad, tanto para satisfacción del cliente, como de los desarrolladores.

- El marco de trabajo utilizado para el desarrollo del prototipo permitió constatar que la confianza en el equipo es de vital importancia. No hubo necesidad de establecer controles rigurosos para que cada uno de los miembros cumpliera con sus obligaciones de la manera adecuada y en el tiempo establecido.

- Las reuniones realizadas durante el proceso de desarrollo del prototipo ayudaron a que los problemas que se suscitaban fueran solucionados en conjunto, haciendo del desarrollo un proceso más eficiente.

- El lenguaje Java, que fue utilizado para el desarrollo, tiene gran variedad de componentes que permiten que el desarrollo se haga de manera ágil. Cabe mencionar que existe gran cantidad de información acerca de sus capacidades, lo que permitió facilitar el desarrollo del prototipo.

- El servidor de aplicaciones Jboss, tiene varias funcionalidades que hacen que el prototipo se desempeñe de una manera correcta en un ambiente web. La configuración del servidor y su ejecución permiten fijarse más en la lógica del negocio y no en aspectos externos. Cabe mencionar que este

servidor ayuda a controlar la transaccionalidad, la concurrencia, entre otros aspectos, los cuales son complejos y a la vez muy importantes en el desarrollo de aplicaciones web.

- El uso de plantillas para realizar las pruebas unitarias y de integración fue adecuado para comprobar que el código que se genera es totalmente adecuado, pues al contrastarlo con las plantillas coincide satisfactoriamente. De la misma forma, se realizaron simulaciones que ayudaron a la validación general del prototipo.

  Los resultados anteriores permitieron concluir que CODGEN cumple con el objetivo trazado.

- El ahorro de tiempo que esta aplicación proporciona es significativa para las labores de desarrollo de aplicaciones web, ya que permite al desarrollador enfocarse más a los aspectos referentes a la lógica del negocio, es decir, aquellos que llaman la atención del usuario, ahorrándose el tiempo que toma el desarrollo de la parte de gestión del sistema.

4.2 RECOMENDACIONES

- Se recomienda el uso de Scrum para incluir al cliente dentro del proceso de desarrollo de un sistema tipo web, lo que permitirá que el sistema se ajuste de manera muy precisa a sus requerimientos y se reducirá en gran medida las inconformidades al momento de la entrega.

- Es recomendable realizar entregas frecuentes de partes funcionales del sistema, así el cliente verá que el sistema va avanzando, hecho que lo mantendrá satisfecho, asimismo, en caso de encontrar errores, se pueden

corregir en cada fase y de esta manera se lograría evitar el costo que representaría hacer los cambios al final del desarrollo.

- Para obtener un código correctamente generado, usando CODGEN, es recomendable previamente realizar un buen diseño de la base de datos, verificar el nombramiento de tablas y atributos, así también que las relaciones guarden una lógica correcta.

- Se recomienda la utilización de este prototipo para contribuir de manera significativa al desarrollo de cualquier sistema web, ayudando en la generación de los módulos referentes a la gestión, y permitiendo a los desarrolladores enfocarse más en la lógica del negocio del sistema.

- Si se desea aumentar la funcionalidad de este prototipo para versiones futuras, se recomienda permitir que el usuario pueda conectarse a cualquier base de datos, ya que por el momento y como requerimiento del cliente el prototipo se conecta solamente con el motor de base de datos Postgres. Además, es posible aumentar controles que identifiquen el uso de palabras reservadas del lenguaje Java, en el nombramiento de tablas y atributos, debido a que estas pueden causar errores de compilación cuando se inserte el código generado por CODGEN, en un IDE.

GLOSARIO

Sprint

Se refiere a un ciclo o periodo de tiempo dentro del cual se realizan tareas establecidas.

Stakeholder

Son todos los interesados en un proyecto de desarrollo de un sistema.

Retroalimentación

Proceso que sirve para controlar y tomar medidas correctivas de ser necesario.

Plugins

Son complementos que aumentan funcionalidades determinadas a un programa.

Trigger

Procedimiento que se ejecuta una vez que se cumpla una condición.

Xhtml

eXtensible HyperText Markup Language (Lenguaje de Marcas de Hipertexto extensible).

Metadatos

Datos acerca de los datos.

Esquema

Describe la organización y estructura de la base de datos.

IDE

Integrated Development Environment (Entorno de Desarrollo Integrado)

BDD

Base de Datos

DBMS

Database Management System (Sistema de Administración de Base de Datos)

JSF

Java Server Faces

JDK

Java Development Kit

HDD

Hard Disk Drive (Disco Duro)

API

Application Programming Interface (Interfaz de programación de aplicaciones)

URL

Uniform Resource Locator (Localizador Uniforme de Recursos)

Paquete

Contenedor que permite agrupar clases y recursos del sistema cuya funcionalidad es similar

BIBLIOGRAFÍA

[1] C. BAUER y G. KING, «Hibernate in Action,» [En línea]. Available: http://www.cpe.ku.ac.th/~plw/oop/e_book/hibernate_in_action.pdf. [Último acceso: 07 Enero 2014].

[2] O. ED, «Introducing the Java EE 6 Platform,» [En línea]. Available: http://www.oracle.com/technetwork/articles/javaee/javaee6overview-141808.html. [Último acceso: 07 Enero 2014].

[3] I. Scrum Alliance, «Why Scrum?,» [En línea]. Available: http://www.scrumalliance.org/why-scrum. [Último acceso: 07 Enero 2014].

[4] M. COHN, «SCRUM,» [En línea]. Available: http://www.mountaingoatsoftware.com/agile/scrum. [Último acceso: 07 Enero 2014].

[5] S. STAMENKOVIĆ, «Eclipse vs NetBeans for PHP Development,» [En línea]. Available: http://dev.umpirsky.com/eclipse-vs-netbeans-for-php-development/. [Último acceso: 14 Enero 2014].

[6] D. BOLTON, «Battle of the Java IDEs: Eclipse vs. NetBeans vs. Intellij IDEA,» [En línea]. Available: http://slashdot.org/topic/bi/battle-of-the-java-ides-eclipse-vs-netbeans-vs-intellij-idea/. [Último acceso: 14 Enero 2014].

[7] Wikipedia, «Eclipse (software).,» [En línea]. Available: http://es.wikipedia.org/wiki/Eclipse_(software). [Último acceso: 14 Enero 2014].

[8] Wikipedia, «NetBeans,» [En línea]. Available: http://es.wikipedia.org/wiki/NetBeans. [Último acceso: 14 Enero 2014].

[9] K. MIKOLUK, «MySQL vs PostgreSQL: Why MySQL Is Superior To PostgreSQL.,» [En línea]. Available: https://www.udemy.com/blog/mysql-vs-

postgresql/. [Último acceso: 14 Enero 2014].

[10] 2. Ltd, «PostgreSQL vs MySQL,» [En línea]. Available: http://www.2ndquadrant.com/es/postgresql/postgresql-vs-mysql/. [Último acceso: 14 Enero 2014].

[11] Wikipedia, «MySQL,» [En línea]. Available: http://es.wikipedia.org/wiki/MySQL. [Último acceso: 14 Enero 2014].

[12] Wikipedia, «PostgreSQL,» [En línea]. Available: http://es.wikipedia.org/wiki/PostgreSQL. [Último acceso: 14 Enero 2014].

[13] F. H. T. LLC., «JBoss vs. Tomcat: Choosing A Java Application Server,» [En línea]. Available: http://www.futurehosting.com/blog/jboss-vs-tomcat-choosing-a-java-application-server/. [Último acceso: 14 Enero 2014].

[14] Wikipedia, «Jboss,» [En línea]. Available: http://es.wikipedia.org/wiki/JBoss. [Último acceso: 14 Enero 2014].

[15] Wikipedia, «Apache Software Foundation.,» [En línea]. Available: http://es.wikipedia.org/wiki/Apache_Software_Foundation. [Último acceso: 14 Enero 2014].

[16] Wikipedia, «RichFaces,» [En línea]. Available: http://en.wikipedia.org/wiki/RichFaces. [Último acceso: 14 Enero 2014].

[17] M. EISELE, «Enterprise Software Development with Java.,» [En línea]. Available: http://blog.eisele.net/2009/12/software-quality-jsf-component_09.html. [Último acceso: 14 Enero 2014].

[18] Wikipedia, «PrimeFaces,» [En línea]. Available: http://es.wikipedia.org/wiki/PrimeFaces. [Último acceso: 14 Enero 2014].

[19] J. Palacio, «NAVEGAPOLIS,» [En línea]. Available: http://www.navegapolis.net/. [Último acceso: 10 Febrero 2014].

[20] ORACLE, «JDBC Overview,» [En línea]. Available: http://www.oracle.com/technetwork/java/overview-141217.html. [Último acceso: 10 Febrero 2014].

[21] UNAM, «JDBC,» [En línea]. Available: http://profesores.fi-b.unam.mx/carlos/java/JDBC.html. [Último acceso: 10 Febrero 2014].

[22] ORACLE, «Class DriverManager,» [En línea]. Available: http://docs.oracle.com/javase/7/docs/api/java/sql/DriverManager.html. [Último acceso: 10 Febrero 2014].

[23] A. Developer, «JDBC,» [En línea]. Available: http://www.artima.com/javaseminars/modules/JDBC/. [Último acceso: 10 Febrero 2014].

[24] R. S. Pressman, Ingeniería del Software un Enfoque Práctico, México: McGrawHill, 2005.

[25] J. Rojas, «Pruebas de Integración,» [En línea]. Available: http://200.69.103.48/comunidad/grupos/arquisoft/fileadmin/Estudiantes/Pruebas/HTML%20-%20Pruebas%20de%20software/node31.html. [Último acceso: 24 02 2014].

ANEXOS

Las siguientes plantillas se han desarrollado en base a la forma y estilo de programación que utiliza la empresa Clearminds Consultores Cía. Ltda, con el objetivo de contrastar las mismas con el código generado por CODGEN y así, poder determinar si el código se ha generado de manera correcta. Además estas plantillas se acoplan al estilo camellCase, por lo tanto se debe tener muy en cuenta las mayúsculas y minúsculas, en el nombrado de clases, atributos, métodos, etc.

Anexo 1

PLANTILLA PARA ENTIDADES

Código para poder verificar que la generación de entidades, se haya cumplido de manera correcta. A continuación se presenta una descripción del código:

- Paquete: nombre y jerarquía de paquetes.
- Tabla: nombre de la tabla que será mapeada.
- TablaClaveCompuesta: nombre de la tabla que contiene una clave compuesta.
- TablaRompimiento: nombre de la tabla de rompimiento generada por la relación de muchos a muchos.
- TipoDato: tipo de dato de la columna.
- Columna: nombre de la columna de la tabla.
- TablaRecibeCF: nombre de la tabla que recibe una clave foránea.
- TablaDaCF: nombre de la tabla a la que se da una clave foránea.

```
packagepaquete.entidades;

import java.util.*;
import javax.persistence.*;
import java.math.BigDecimal;
import java.io.Serializable;
import javax.validation.constraints.NotNull;

@Entity
@Table(name = "tabla")
```

```
@IdClass(value = tablaClaveCompuesta+PKey.classO
tablaRompimiento+PK.class)
//Solamente si es una Tabla de Rompimiento o con clave compuesta

publicclasstablaimplements Serializable {

        privatestaticfinallongserialVersionUID = 1L;

        //Para columnas que son claves primarias.

        @NotNull(message = "Campo Requerido")
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY) //Solamente
        si la columna es SERIAL
        @Column(name = "columna")
        privateTipoDatocolumna;



        //Para columnas que son claves foráneas.

        // Relacion bidireccional con la tabla tablaRecibeCF.
        @NotNull(message = "Campo Requerido")
        @ManyToOne
        @JoinColumn(name = "columna")
        privateTablaRecibeCF tablaRecibeCF;

        //Para tablas a las que se da claves foráneas.

        // Relacion muchos a uno con la tabla tablaDaCF
        @OneToMany(mappedBy = "tabla")
        private List< TablaDaCF>tablaDaCF+s;

        //Para columnas comunes.

        @NotNull(message = "Campo Requerido") //Solamente si la columna
        es NOT NULL
        @Column(name = "columna")
        @Temporal(TemporalType.DATE o TIME o TIMESTAMP) //Solamente si
        el Tipo de Dato es Date, Time o TimeStamp respectivamente
```

```
        privateTipoDatocolumna;

        //Los SETTERS y GETTERS de cada uno de los atributos se generan
        en base al formato establecido por java.
}
```

PLANTILLA PARA ENTIDADES CON CLAVE COMPUESTA

Para poder mapear una Tabla que tiene una clave compuesta, es necesario crear una entidad extra, que permitirá agrupar todas las columnas que forman parte de dicha clave.

El código es prácticamente sencillo y a continuación se presenta una descripción.

- Paquete: nombre y jerarquía de paquetes.
- TablaClaveCompuesta: nombre de la tabla que contiene una clave compuesta.
- TablaRompimiento: nombre de la tabla de rompimiento generada por la relación de muchos a muchos.
- TipoDato: tipo de dato de la columna.
- Columna: nombre de la columna de la tabla.
- TablaRecibeCF: nombre de la tabla que recibe una clave foránea.

```
package paquete.entidades;

import java.util.*;
import javax.persistence.*;
import java.io.Serializable;

public class tablaClaveCompuesta+PKeyO tablaRompimiento+PK implements
Serializable {

        private static final long serialVersionUID = 1L;

        //En el caso de que la clave compuesta no este conformada por
        claves foráneas
```

```
        private TipoDato columna;

        //En el caso de que la clave compuesta este conformada por
        claves foráneas
        private TablaRecibeCFtablaRecibeCF;

        //Los SETTERS y GETTERS de cada uno de los atributos se generan
        en base al formato establecido por java.


}
```

Anexo 2

PLANTILLA PARA EL SERVICIO BASE

El código del Servicio Base, será el mismo para todos los casos, la única variante será el nombre del paquete.

```
package paquete.servicios;



import java.util.List;
import javax.annotation.PostConstruct;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.criteria.CriteriaQuery;
import org.jboss.logging.Logger;

public abstract class ServicioBase <T>{
@PersistenceContext
protected EntityManager em;
private Class<T> tipoEntidad;
private Class<?> tipoServicio;

public ServicioBase(Class<T> tipoEntidad, Class<?> tipoServicio) {
        this.tipoEntidad = tipoEntidad;
        this.tipoServicio = tipoServicio;
        }
```

```
        public void insertar(T entidad) {
        em.persist(entidad);
        em.flush();
        }

        public void actualizar(T entidad) {
        em.merge(entidad);
        }

        public void eliminar(T entidad) {
        em.remove(em.merge(entidad));
        }

        public T buscarPorId(Integer id) {
        return em.find(tipoEntidad, id);
        }

        public T buscarPorIdString(String id) {
        return em.find(tipoEntidad, id);
        }

        @SuppressWarnings({ "unchecked", "rawtypes" })
        public List<T> buscarTodos() {
        CriteriaQuery cq = em.getCriteriaBuilder().createQuery();
        cq.select(cq.from(tipoEntidad));
        return em.createQuery(cq).getResultList();
        }

}
```

Anexo 3

PLANTILLA PARA SERVICIOS

Código de Servicio, el cual será utilizado para contrastar los resultados del código generado para cada uno de los servicios.

Un servicio, al heredar del servicio base que se diseñó para la aplicación, tiene un código bastante simple, el cual se muestra a continuación:

- Entidad: nombre de la entidad.
- Paquete: nombre y jerarquía de paquetes.

```
packagepaquete;

importpaquete.entidades.Entidad;
import javax.ejb.Stateless;

@Stateless
publicclass ServicioEntidadextendsServicioBase<Entidad> {

	public ServicioEntidad() {
		super(Entidad.class, ServicioEntidad.class);
	}

}
```

Anexo 4

PLANTILLA PARA PÁGINAS

La siguiente plantilla se ha establecido para poder verificar la validez del código generado para las páginas. En seguida se presenta un detalle del código.

- Entidad: nombre de la entidad.
- Atributo: nombre del atributo de la entidad.
- EntidadRompimiento: nombre de la entidad de rompimiento.
- EntidadAsociada: nombre de la entidad asociada a la entidad de rompimiento.
- Variable: nombre de la variable local empleada en algunos elementos de la página.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
	xmlns:h="http://java.sun.com/jsf/html"
```

```
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:p="http://primefaces.org/ui">

<ui:composition template="/templates/codgenTemplate.xhtml">
<ui:define name="tituloPagina">Gestion de entidad</ui:define>
<ui:define name="tituloFormulario">Gestion de entidad</ui:define>"
<ui:define name="autor"> Martinez Jairo, Rodriguez Jorge </ui:define>
<ui:define name="contenido">
    <h:form id="frmPrincipal">

        <p:panelGrid columns="2">
            //El siguiente código se generará por cada uno de los
            atributos de la entidad
            <h:outputText value="atributo" />
            <p:inputText value="#{controladorEntidad.entidad.atributo}" />
        </p:panelGrid>

        //Este código del panel será generado, solamente si la Entidad
        Forma parte de una Tabla de Rompimiento.
        <p:panel header="entidadRompimiento">
          <p:panelGrid columns="2">
          <p:outputLabel value="entidadAsociada" />
          <p:selectOneMenu value="#{controladorEntidad.entidad+sSeleccion}">
            <f:selectItem itemValue="" itemLabel="Seleccione..." />
            <f:selectItems              value="#{controladorEntidad.entidad+s}"
            var="entidadAsociada"              itemValue="#{variable.atributo}"
            itemLabel="#{variable.atributo}" />
          </p:selectOneMenu>

          //El siguiente código se generará por cada uno de los atributos
          de la entidad de rompimiento
          <h:outputText value="atributo" />
          <p:inputText
          value="#{controladorEntidad.entidadRompimiento.atributo}" />
          </p:panelGrid>

          <p:commandButton value="Insertar"
```

```
        action="#{controladorEntidad.agregar()}" update="@form" />
        <p:dataTable
        value="#{controladorEntidad.entidadRompimiento+s}"var="entidadRompimi
        ento">
            <p:column headerText="atributo">
            <p:outputLabel value="#{variable.entidadAsociada.atributo}" />
            </p:column>
            <p:column headerText="Eliminar">
            <p:commandButton icon="ui-icon-trash" update="@form"
            process="@this"
            action="#{controladorEntidad.eliminarRompimiento()}">
            <f:setPropertyActionListener
            target="#{controladorEntidad.entidadRompimiento}"
            value="#{variable}" />
            </p:commandButton>
            </p:column>
        </p:dataTable>
    </p:panel>

    <p:commandButton value="Insertar "
    action="#{controladorEntidad.insertarEntidad()}" update="@form" />
    <p:commandButton value="Actualizar "
    action="#{controladorEntidad.actualizarEntidad ()}" update="@form" />

    <p:dataTable value="#{controladoEntidad.entidad+s}" var="entidad">

        // El siguiente código se generará por cada uno de los
        atributos de la entidad
        <p:column headerText="atributo">
        <p:outputLabel value="#{variable.atributo}" />
        </p:column>

        <p:column headerText="Seleccionar">
        <p:commandButton icon="ui-icon-pencil" update="@form"
        process="@this"action="#{controladorEntidad.metodoParaSeleccionar()
        }">
        <f:setPropertyActionListener target="#{controladorEntidad.entidad}"
        value="#{variable}" />
```

```
            </p:commandButton>
            </p:column>
            <p:column headerText="Eliminar">
            <p:commandButton icon="ui-icon-trash" update="@form"
            process="@this" onclick="dlgEliminar.show()">
            <f:setPropertyActionListener target="#{controlador Entidad.entidad
            }"
            value="#{ variable }" />
            </p:commandButton>
            </p:column>
        </p:dataTable>
    </h:form>
    <h:form id="frmEliminar">
        <p:dialog header="Confirmacion" widgetVar="dlgEliminar"
        closable="false" resizable="false" modal="true" draggable="false">
        <h: outputText value="Esta seguro que desea eliminar?" />
        <p:commandButton value="SI"
        action="#{controladorEntidad.eliminarEntidad ()}" update=":frmPrincipal"
        />
        <p:commandButton value="NO" onclick="dlgEliminar.hide()" />
        </p:dialog>
    </h:form>
</ui:define>
</ui:composition>
</html>
```

Anexo 5

PLANTILLA PARA EL ARCHIVO PERSISTENCE

Mediante este código verificamos la generación del archivo Persistence, el cual no posee mayor variación que las siguientes:

- Datasource: nombre del DataSource.
- Paquete: nombre y jerarquía de paquetes.
- Entidad: nombre de la entidad.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0"
   xmlns="http://java.sun.com/xml/ns/persistence"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
   http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">
   <persistence-unit name="cmd">
      <jta-data-source> datasource</jta-data-source>
      //El siguiente código se generará para cada entidad.
      <class>paquete.entidades.entidad</class>
      <properties>
          <property                                    name="hibernate.dialect"
          value="org.hibernate.dialect.PostgreSQLDialect" />
          <property name="hibernate.show_sql" value="true" />
          <property name="hibernate.format_sql" value="true" />
      </properties>
   </persistence-unit>
</persistence>
```

Anexo 6

PLANTILLA PARA CONTROLADORES

Código de Controlador, que permitirá analizar los resultados obtenidos al realizar la generación de controladores.

Las palabras que están en color azul, serán las que se deben reemplazar cuando se genere el código, por los datos apropiados.

Las palabras que se indican a continuación se usan dentro del código:

Paquete: El nombre del paquete ingresado por el usuario al momento de realizar la generación.

Entidad: Cada controlador está atado a una entidad en particular, por tanto este será el nombre de la entidad. Para el nombre de la instancia, este nombre se

utilizará con la primera letra minúscula, y se aumentará una letra s al final en el caso de que sea una lista.

EntidadForanea: Se refiere a las claves foráneas que están presentes en la entidad para la que se genera el controlador. Para la instancia este nombre deberá iniciar con letra minúscula, y en el caso de que sea una lista, se deberá añadir una letra s al final.

TipoDeDato: Es el tipo de dato que tenga la clave primaria de la tabla a la que se hace referencia.

```
packagepaquete.controladores;

import javax.ejb.EJB;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.ViewScoped;
import org.primefaces.context.RequestContext;
import javax.annotation.PostConstruct;
import java.util.ArrayList;
import java.util.List;
importpaquete.entidades.*;
importpaquete.servicios.*;

@ViewScoped
@ManagedBean
publicclass ControladorEntidad {

      @EJB
      private ServicioEntidadservicioCliente;
// Las siguientes dos líneas se pueden repetir tantas veces como sea necesario, de acuerdo al número de claves foráneas que tenga la entidad.
      @EJB
      private ServicioEntidadForaneaservicioEntidadForanea;
      privateEntidad entidad;
      private List<Entidad>entidad+s;
     // Se pueden añadir tantas listas, como claves foráneas exista en
```

```
la entidad.
	private List<EntidadForanea>entidadForanea+s;
	privatetipoDeDatoentidadForaneaSeleccion;

	// Constructor
	public ControladorEntidad() {
		entidad = newEntidad();
	}

	// PostConstructor
	@PostConstruct
	publicvoid ejecutar() {
		// Se deben inicializar todas las listas existentes de
claves foráneas.
		entidadForanea+s= servicioEntidadForanea.buscarTodos();
		entidad+s = servicioEntidad.buscarTodos();

	}

	publicvoid insertarEntidad() {
		// Se debe agregar para todas las claves foráneas
existentes.
		entidad.setEntidadForanea(servicioEntidadForanea
				.buscarPorId(entidadForaneaSeleccion));
		//
		servicioEntidad.insertar(entidad);
		entidad+s = servicioEntidad.buscarTodos();
		entidad = new Entidad();
	}

	publicvoid actualizarEntidad() {
		// Se debe agregar para todas las claves foráneas existentes.
		entidad.setTipoEmpresa(servicioEntidadForanea
				.buscarPorId(entidadForaneaSeleccion));
		servicioEntidad.actualizar(entidad);
		entidad+s = servicioEntidad.buscarTodos();
		entidad = newEntidad();

	}
```

```
      publicvoid eliminarEntidad() {
            servicioEntidad.eliminar(entidad);
            entidad+s = servicioEntidad.buscarTodos();

      RequestContext.getCurrentInstance().execute("dlgEliminar.hide()");
            entidad = new Entidad();

      }

// Setters y getters, las instancias de los servicios no deben
tenerlos.
```

Anexo 7

Código del Controlador en el caso de que exista una entidad de muchos a muchos.

Las palabras que están en color azul, serán las que se deben reemplazar cuando se genere el código, por los datos apropiados.

Las palabras que se indican a continuación se usan dentro del código:

Paquete: El nombre del paquete ingresado por el usuario al momento de realizar la generación.

Entidad: Cada controlador está atado a una entidad en particular, por tanto este será el nombre de la entidad. Para el nombre de la instancia, este nombre se utilizará con la primera letra minúscula, y se aumentará una letra s al final en el caso de que sea una lista.

EntidadRelacionada: Con este nombre nos referimos a la segunda entidad que forma parte de la relación de muchos a muchos. Para la instancia este nombre deberá iniciar con letra minúscula, y en el caso de que sea una lista, se deberá añadir una letra s al final.

EntidadDeRompimiento: Cuando existe una tabla de rompimiento, se crea una entidad que hace referencia a esta tabla, esta no tiene un controlador asociado,

pero es necesaria en los controladores de las entidades que en cierta forma la “componen”. Para la instancia este nombre deberá iniciar con letra minúscula, y en el caso de que sea una lista, se deberá añadir una letra s al final.

TipoDeDato: Es el tipo de dato que tenga la clave primaria de la tabla a la que se hace referencia.

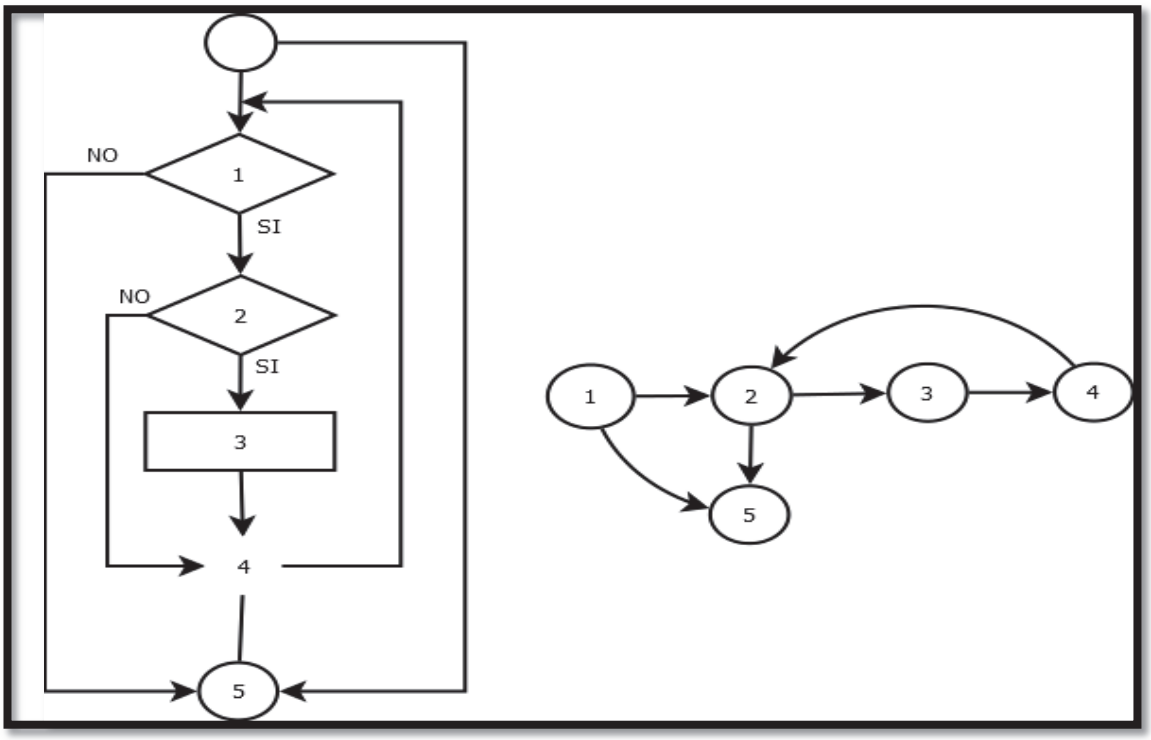
```
packagepaquete.controladores;

import javax.ejb.EJB;
import org.primefaces.context.RequestContext;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.ViewScoped;
import java.util.List;
import java.util.ArrayList;
import javax.annotation.PostConstruct;
importpaquete.entidades.*;
importpaquete.servicios.*;

@ViewScoped
@ManagedBean
publicclass ControladorEntidad{



@EJB
private ServicioEntidad servicioEntidad;
@EJB
private ServicioEntidadRelacionada servicioEntidadRelacionada;
@EJB
private ServicioEntidadDeRompimiento servicioEntidadDeRompimiento;
private Entidad entidad;
privateEntidadRelacionadaentidadRelacionada;
privateEntidadDeRompimientoentidadDeRompimiento;
private List<Entidad>entidad+s;
private List<EntidadDeRompimiento>entidadDeRompimiento+s;
private tipoDeDatoentidadRelacionada+sSeleccion;
private List<EntidadDeRompimiento>entidadDeRompimiento+s;
private List<EntidadDeRompimiento> comparaciones;
//Constructor
```

```
public ControladorEntidad(){
entidad = new Entidad();
entidadRelacionada = newEntidadRelacionada();
entidadDeRompimiento = newEntidadDeRompimiento();
entidadDeRompimiento+s= new ArrayList<EntidadDeRompimiento>();
comparaciones= new ArrayList<EntidadDeRompimiento>();
}
//PostConstructor
@PostConstruct
publicvoid ejecutar (){
entidadRelacionada+s = servicioEntidadRelacionada.buscarTodos();
entidad+s = servicioEntidad.buscarTodos();

}
publicvoid agregar (){
entidadDeRompimiento.setEntidadRelacionada(servicioEntidadRelacionada.met
odoParaBuscarPorId(entidadRelacionadaSeleccion));
entidadDeRompimiento.setEntidad(entidad);
entidadDeRompimiento+s.add(entidadDeRompimiento);
entidadDeRompimiento =  newEntidadDeRompimiento();

}
publicvoid insertarEntidad (){
servicioEntidad.insertar(entidad);
if(!(entidadDeRompimiento+s==null || entidadDeRompimiento+s.size()==0)){
for(EntidadDeRompimiento p : entidadDeRompimiento+s) {
p.setEntidad(entidad);
servicioEntidadDeRompimiento.insertar(p);
}
}
entidad= newEntidad();
entidadDeRompimiento= newEntidadDeRompimiento();
entidadDeRompimiento+s= new ArrayList<EntidadDeRompimiento>();
entidad+s = servicioEntidad.buscarTodos();

}
publicvoid actualizarEntidad (){
servicioEntidad.actualizar(entidad);
for  (EntidadDeRompimiento b : comparaciones) {
```

```
if(entidadDeRompimiento+s.contains(b) == false){
servicioEntidadDeRompimiento.eliminar(b);
}
}
for (EntidadDeRompimiento p : entidadDeRompimiento+s) {
servicioEntidadDeRompimiento.insertar(p);
}
Entidad+s = servicioEntidad.buscarTodos();
entidad = newEntidad();
entidadDeRompimiento = newEntidadDeRompimiento();
entidadDeRompimiento+s = new ArrayList<EntidadDeRompimiento>();

}
publicvoid eliminarEntidad (){
servicioEntidad.eliminar(entidad);
RequestContext.getCurrentInstance().execute("dlgEliminar.hide()");
entidadDeRompimiento = newEntidadDeRompimiento();
entidad+s = servicioEntidad.buscarTodos();
entidad= newEntidad();
entidadDeRompimiento+s = new ArrayList<EntidadDeRompimiento>();

}
publicvoid eliminarRompimiento (){
entidadDeRompimiento+s.remove(entidadDeRompimiento);

}
publicvoid metodoParaSeleccionar (){
entidadDeRompimiento+s = new ArrayList<EntidadDeRompimiento>();
comparaciones= new ArrayList<EntidadDeRompimiento>();
entidadDeRompimiento+s.addAll (entidad.getEntidadDeRompimiento+s());
comparaciones.addAll(entidad.getEntidadDeRompimiento+s());

}
}
```

PRUEBA DEL CAMINO BÁSICO PARA EL MÉTODO “INSERTARENTIDAD”

1.- Diagrama de Flujo y Diagrama de Estados.



Cálculo del número de caminos:

A partir del grafico de estados, se tiene que el número de caminos será igual al número de aristas – nodos +2.

6-5+2=3.

Entonces se tienen tres caminos, los cuales serían los siguientes.

1-5

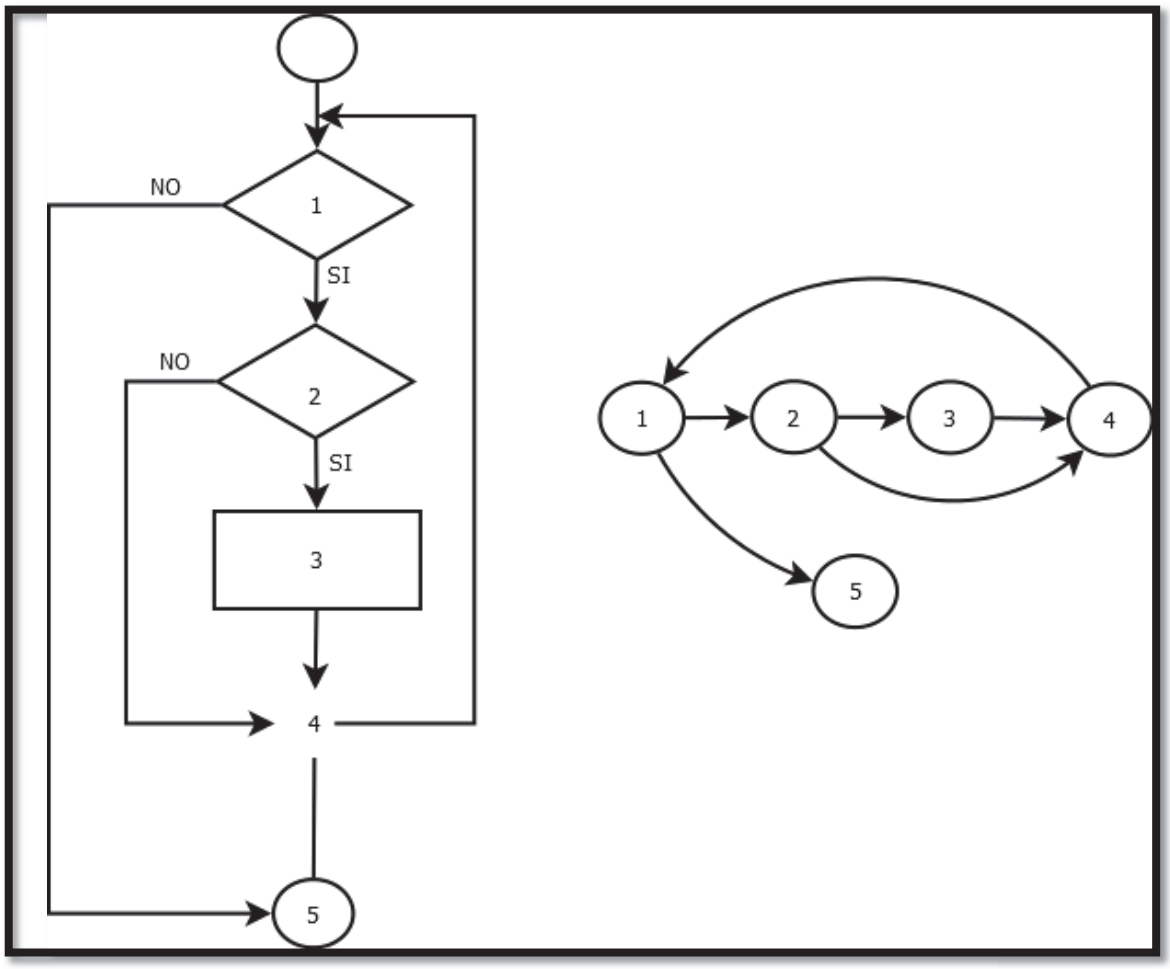
1-2-3-4-2-5

1-2-5

Con estos se podrá realizar más adelante las pruebas unitarias.

PRUEBA DEL CAMINO BÁSICO PARA EL MÉTODO “ACTUALIZARENTIDAD”

2.- Diagrama de Flujo y Diagrama de Estados.



Calculo del número de caminos:

A partir del grafico de estados, se tiene que el número de caminos será igual al número de aristas – nodos +2.

6-5+2=3.

Entonces se tienen tres caminos, los cuales serían los siguientes.

1-5

1-2-3-4-1-5

1-2-4-1-5

Con estos se podrá realizar más adelante las pruebas unitarias.

Pruebas Unitarias

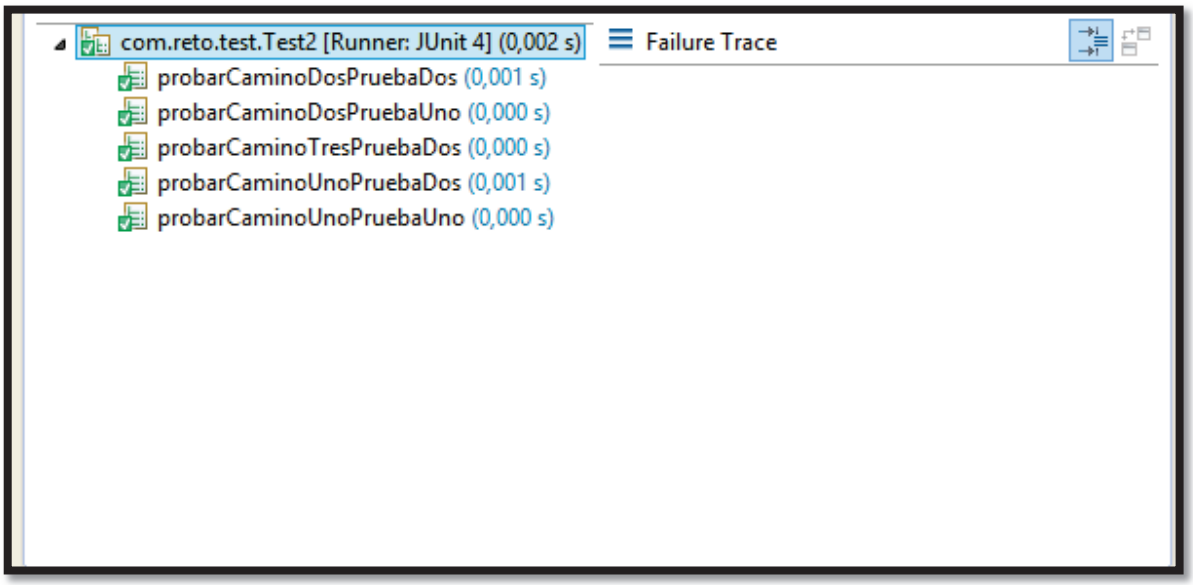
```
package com.reto.test;

import java.util.ArrayList;
import java.util.List;

import junit.framework.Assert;

@SuppressWarnings("deprecation")
publicclass Test2 {

	private List<String>entidadRompimientos;
	private List<String>comparaciones;

	/**
	 *
	 * @param lista
	 *            Para el metodouno solo se van a
especificardoscaminos, esto
	 *            debido a que el tercercamino, enestecaso no
sepuede
	 *            realizar, porquelaprimeracondicionloimpide.
	 *
	 *            Estemetodopermiteverquesucedeconlacondiciondellazo
if
	 *           analizalosresultadosconlasdoscondiciones.
	* @return
	 */
	publicboolean metodoUnoCaminoUno(List<String> lista) {

		if (!(lista == null || lista.size() == 0)) {

			returntrue;
		}
		returnfalse;
	}
	/**
	 * Estemetodopermiteverquepasacuandosecumpleconlacondiciondel
```

```
if, y seentraal for.
	*
	 * @return
	 */
	public String metodoUnoCaminoDos() {
		String resultado = "";
		entidadRompimientos = new ArrayList<String>();
		entidadRompimientos.add("a");
		if (!(entidadRompimientos == null ||
entidadRompimientos.size() == 0)) {

			for (String s : entidadRompimientos) {
				resultado += s;
			}
		}
		return resultado;
	}

	/**
	 * Estemetodosirveparapobrarloquesucedeen el lazo for
esdecirsila
	 * condicionsecumple o no.
	 *
	 * @return
	 */
	publicboolean metodoDosCaminoUno() {
		entidadRompimientos = new ArrayList<String>();
		boolean retorno = false;
		for (String s : entidadRompimientos) {
			retorno = true;
		}
		return retorno;
	}

	/**
	 * Estemetodopermiteversiunavezqueentraallazo for, entra a
comparar el lazo if
	 * y no cumplelacondicionnecesaria.
	 *
```

```
	 * @return
	 */
	publicboolean metodoDosCaminoDos() {
		entidadRompimientos = new ArrayList<String>();
		entidadRompimientos.add("a");
		comparaciones = new ArrayList<String>();
		comparaciones.add("a");
		boolean retorno = false;
		for (String s : comparaciones) {

			if (entidadRompimientos.contains(s) == false) {
				retorno = true;
			}
		}
		return retorno;
	}

	/**
	 * Estemetodopermiteversiunavez que entra al lazo for, entra a
comparar el lazo if
	 * y cumple con la condicion necesaria para entrar al if.
	 * @return
	 */
	publicboolean metodoDosCaminoTres() {
		entidadRompimientos = new ArrayList<String>();
		entidadRompimientos.add("a");
		comparaciones = new ArrayList<String>();
		comparaciones.add("s");
		boolean retorno = false;
		for (String s : comparaciones) {

			if (entidadRompimientos.contains(s) == false) {
				retorno = true;
			}
		}
		return retorno;
	}

	@org.junit.Test
```

```
	publicvoid probarCaminoUnoPruebaUno() {
		Assert.assertEquals(false,
metodoUnoCaminoUno(entidadRompimientos));
		entidadRompimientos = new ArrayList<String>();
		Assert.assertEquals(false,
metodoUnoCaminoUno(entidadRompimientos));
		entidadRompimientos.add("s");
		Assert.assertEquals(true,
metodoUnoCaminoUno(entidadRompimientos));
	}

	@org.junit.Test
	publicvoid probarCaminoDosPruebaUno() {
		Assert.assertEquals("a", metodoUnoCaminoDos());
	}

	@org.junit.Test
	publicvoid probarCaminoUnoPruebaDos() {
		Assert.assertEquals(false, metodoDosCaminoUno());
	}

	@org.junit.Test
	publicvoid probarCaminoDosPruebaDos() {
		Assert.assertEquals(false, metodoDosCaminoDos());
	}

	@org.junit.Test
	publicvoid probarCaminoTresPruebaDos() {
		Assert.assertEquals(true, metodoDosCaminoTres());
	}
}
```

A continuación se muestran los resultados de la ejecución de las mismas:


com.reto.test.Test2 [Runner: JUnit 4] (0,002 s)
probarCaminoDosPruebaDos (0,001 s)
probarCaminoDosPruebaUno (0,000 s)
probarCaminoTresPruebaDos (0,000 s)
probarCaminoUnoPruebaDos (0,001 s)
probarCaminoUnoPruebaUno (0,000 s)
Failure Trace