UCLV
Universidad Central
"Marta Abreu" de Las Villas

MFC
Facultad de Matemática
Física y Computación

Licenciatura en Ciencia de la Computación

TRABAJO DE DIPLOMA

Integración entre el sistema de portadores energéticos EnerguX y sistema de control de flotas MovilWeb

Autor: Dario Deya Diaz

Tutores: Dra. Ana María García Pérez, Sub Directora de Investigación y Desarrollo Desoft VC
Lic. Mileidi Iglesias Cuéllar, líder del producto EnerguX Desoft

Santa Clara, junio, 2018

Hago constar que el presente trabajo fue realizado en la Universidad Central Marta Abreu de Las Villas como parte de la culminación de los estudios de la especialidad de Ciencia de la Computación, autorizando a que el mismo sea utilizado por la institución, para los fines que estime conveniente, tanto de forma parcial como total y que además no podrá ser presentado en eventos ni publicado sin la autorización de la Universidad.

_____

Firma del autor

Los abajo firmantes, certificamos que el presente trabajo ha sido realizado según acuerdos de la dirección de nuestro centro y el mismo cumple con los requisitos que debe tener un trabajo de esta envergadura referido a la temática señalada.

_____ _____

Firma del tutor Firma del jefe del Laboratorio

Resumen

El trabajo consiste en la programación de un sistema informático basado en web para extender las capacidades de una aplicación de control de portadores energéticos en uso mediante el empleo de datos GPS obtenidos desde un servicio web de tipo REST, diseñado con herramientas de código abierto, capaz de trabajar sobre una infraestructura ya existente con apenas cambios, y de manera autónoma.

En modernas tecnologías de programación en Java se centra el desarrollo de esta solución que permite dividir el desarrollo de una aplicación orientada a la Web en las tradicionales tres capas de arquitectura de desarrollo, que proveen a la vez múltiples ventajas para el desarrollador, calidad en la interfaz de usuario y facilidad de mantenimiento, utilizando las potencialidades de los marcos de trabajo existentes.

Abstract

This paper is about a computer application for the web to extend the capabilities of an existing application that control the usage of energetic sources using GPS data from a REST Web Service. It was designed and programmed using open source technologies, being able to work with an existing infrastructure with just a few small changes and it works autonomously.

This solution is based on Java. The techniques described here allows to divide the application into three traditional-and-well-defined layers bound together to ease the design of a Web application, which provides several advantages for the developer, it produces an easy-to-use interface and advantages for extensibility in future versions of this software by using the features of the existing frameworks.

Tabla de contenido

Tabla de ilustraciones

Introducción

En Cuba se le concede una importancia vital al control de los portadores energéticos con el objetivo de planificar de forma eficiente y prioritaria la distribución del combustible necesario para el trabajo empresarial diario. La informática en el mundo cada día gana en desarrollo y en particular en Cuba. Se hacen grandes esfuerzos por llevarla a todos los niveles de la sociedad para contribuir con la economía. Con este avance, se consigue que el desarrollo del trabajo y el procesamiento de la información sean superiores en cuanto a inmediatez y fiabilidad de los resultados. El desempeño por lograr aplicaciones más cercanas al usuario final, así como la facilidad a la hora de manipularlas, es el mayor propósito de los programadores y el soporte de las nuevas tecnologías, específicamente tecnologías basadas en la Web, más aún cuando la Web 2.0 es una realidad y toda la tecnología que trae aparejada este concepto.

EnerguX es un sistema de control de portadores energéticos creado por Desoft y usado por más de 500 entidades en el país. Su principal objetivo es el de controlar de una forma eficiente y segura los recursos de una empresa. Algunos de estos recursos son el gas, el agua, la electricidad, tarjetas magnéticas y el transporte (Rodríguez, 2010).

Las principales funcionalidades del sistema son:

- Control de consumo de Combustible por Tarjetas Magnéticas, que incluye:
  - Facturación
  - Control de consumo del combustible.
  - Cambio de portador
  - Ajustes
  - Traspaso de saldo entre tarjetas
- Control de hojas de ruta, que incluye la elaboración de Hojas de Ruta por vehículos, incluyendo la introducción de los detalles de viajes de cada una de ellas y los planes de consumo mensuales por vehículos.
- Emisión de comprobantes contables para uso del departamento económico de la entidad, a partir de las fuentes de información almacenadas en el sistema.

MovilWeb es una aplicación de localización de vehículos basada en Web para el seguimiento de móviles sobre cartografía vectorial y raster, diseñada para controlar flotas dentro de una arquitectura cliente-servidor. Esta herramienta permite el monitoreo

de móviles de manera remota sobre una red de comunicaciones, posibilitando reconstruir el comportamiento del vehículo en un determinado periodo de tiempo, reelaborando su trayectoria y analizando su velocidad, detenciones en destinos autorizados o no, etc. a través de la información almacenada en una base de datos histórica, dota al usuario de un grupo de herramientas para el manejo de mapas, similar a las herramientas de los software profesionales para el manejo de Sistemas de Información Geográfica. Su arquitectura está orientada a servicios, en ella se produce la integración y encadenamiento de varios servicios de procesamiento y datos geoespaciales distribuidos sobre la Web. Está implementada sobre plataformas de software libre, utiliza como gestor de base de datos PostgreSQL, Business Intelligence and Reporting Tools como herramienta para la generación de los reportes, GeoServer como servicio de mapas, todo se ejecuta sobre Apache Tomcat 6.0 como servidor para Internet, las nuevas utilidades han sido implementadas sobre Java y se nutre de los servicios geoespaciales disponibles en la Infraestructura de Datos Espaciales de la República de Cuba (IDERC), como los servicios de imágenes satelitales y cartografía vectorial. En la actualidad brinda servicio a 193 bases de transporte, con 900 usuarios conectados al sistema y un total de 7806 vehículos en todas las provincias del país (Fernández, 2011).

Problema de investigación

Las empresas actuales que usan el servicio de MovilWeb para sus flotas de vehículos y usan EnerguX como plataforma principal para el control del transporte tienen que trasladar la información brindada por MovilWeb a EnerguX de manera manual lo cual es lento y propenso a errores.

Objetivo general

- Desarrollar sobre el EnerguX la capacidad de consumir datos provenientes de la plataforma MovilWeb, para aquellas empresas que usen el EnerguX y tengan habilitado el servicio de Sistema de Posicionamiento Global (GPS) en sus vehículos.

Objetivos específicos

1. Determinar la granularidad adecuada para lograr un buen rendimiento del sistema.
2. Implementar un mecanismo de comunicación basado en Servicios Web usando REST como protocolo estándar de comunicación.

3. Hacer persistir la información procesada desde los servicios web que se consuman.
4. Realizar las pruebas necesarias para demostrar el correcto funcionamiento del servicio.

Preguntas de investigación

1. ¿Cuál es la granularidad adecuada de los servicios web requeridos que permita un manejo adecuado de los datos devueltos por los servicios, es decir, cómo lograr un balance entre cantidad de parámetros a usar y cantidad de registros devueltos que permita que el sistema se comporte con buen rendimiento en la ejecución?
2. ¿Cómo implementar un mecanismo de comunicación para un Servicio Web existente que usa un protocolo REST?
3. ¿Cómo lograr la persistencia de la información brindada por el servicio web, permitiendo datos correctos sobre la estructura de la Base de Datos ya existente?

Justificación

Debido a que la manera en que los datos son trasladados de una plataforma a la otra es de forma manual pueden ocurrir con mucha frecuencia inconsistencia entre los datos reportados por una plataforma y la otra. Dando lugar a largas revisiones que atrasa el trabajo de la empresa, además de que un error puede significar una pérdida de mucho dinero para la entidad y por los tanto para el país.

Este trabajo consta de tres capítulos.

El capítulo 1 se describe los principales conceptos, métodos y tecnologías, relacionados con la posible solución del problema de investigación planteado. Se llega a conclusiones sobre los métodos, tecnologías y software a desarrollar.

El capítulo 2 muestra la arquitectura de la solución, el proceso de planeamiento y la implementación del sistema acorde a la metodología escogida.

En el capítulo 3 se realizan las pruebas necesarias para comprobar el correcto funcionamiento de las principales funcionalidades de sistema.

Capítulo 1. Fundamentación teórica

En este capítulo se describen los principales conceptos, métodos, tecnologías y aplicaciones existentes en la actualidad, relacionados con la posible solución del problema de investigación planteado. Se llega a conclusiones sobre los métodos, tecnologías y software a desarrollar.

1.1 EnerguX

Los recursos energéticos se están agotando a nivel mundial y se encarecen cada día. Todo país que intente mantener su desarrollo y desarrollarse debe aplicar una política estable y sostenida de ahorro energético. Esto se traduce en llevar este ahorro a cada empresa, a cada área de la empresa, a cada equipo, etc. En las entidades cubanas se debe realizar diariamente un control del consumo de los portadores energéticos y valorar su comportamiento en el consejo de dirección mensual de la entidad, relacionándolo con los resultados productivos y de la prestación de los servicios, con el fin de tomar medidas dirigidas a la disminución de los índices de consumo por portador y la intensidad energética con respecto a lo alcanzado en cada periodo precedente, además, se realiza un diagnóstico energético con el objetivo del análisis de los equipos mayores consumidores y la puesta en práctica de las soluciones técnicas organizativas que permitan obtener una mayor eficiencia energética, conformando un plan de medidas de ahorro de energía y portadores energéticos.(Desoft, 2015)

La idea de controlar la energía consumida no es novedosa. Muchos son los sistemas informáticos existentes en el país. En el entorno cubano existe uno que constituye el antecesor natural de EnerguX. Se trata del Celador S2C, solución informática comercializada y desarrollada por Desoft, en explotación desde el año 2004 y que se explota, en la actualidad, por parte de unos 80 clientes, con más de 100 licencias en todo el territorio nacional. El sistema, sobre todo, pondera su uso en el control del consumo de combustibles por tarjeta magnética, funcionalidad principal que tiene la solución. Su versión más reciente es la 3.4 de marzo del 2010. (Rodríguez, 2010)

Sin embargo, como parte de un proceso que se comenzó en todo el país en el año 2008, se hace necesario migrar los sistemas informáticos actuales a tecnologías de código abierto. Además, a pedidos de los clientes se hace necesario incorporar o mejorar algunas funcionalidades no presentes en Celador S2C como: asociar los consumos de los vehículos a actividades propias de la empresa, la automatización del proceso de carga comenzando por la facturación de combustible, las operaciones descritas por la

Resolución 60 del 2009 y la configuración de los niveles y subniveles de cuentas para la generación del comprobante contable. Para satisfacer esta demanda se creó la aplicación informática EnerguX.

EnerguX es un software destinado al control estricto de los portadores energéticos y está diseñado para obtener una serie de datos estadísticos que pueden ser de gran utilidad para el control de estos. Se tendrá una referencia en tiempo real de cómo anda el comportamiento del uso de los portadores energéticos en las entidades cubanas, así como en cada una de las áreas que la forman, además de dar la contabilidad de ellos (Desoft, 2015).

El sistema maneja y controla los siguientes apartados de una empresa:

- Datos generales
- Tarjetas magnéticas
- Transporte
- Electricidad
- Portadores
- Operaciones
- Administración
- Agua
- Gas

En la Ilustración 1 se muestra la interfaz principal la cual resume las principales funcionalidades del sistema.

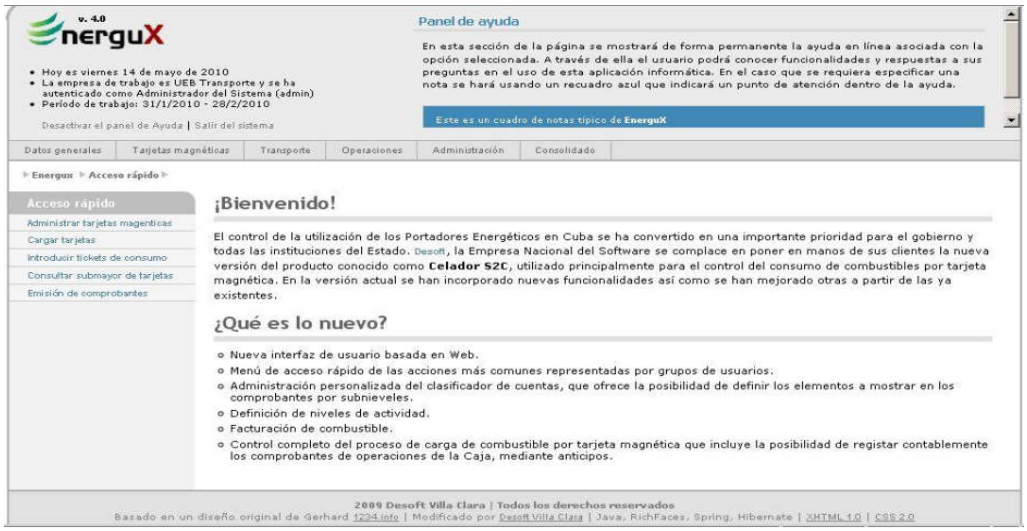

ILUSTRACIÓN 1. INTERFAZ PRINCIPAL ENERGUX V4.0

Transporte:

EnerguX dedica una parte de su sistema al control de Hoja de Ruta y Mantenimiento de vehículos. Es importante precisar que se trata solo de una implementación mínima de este instrumento y que en ningún caso la utilización de esta parte de la aplicación constituye una aproximación completa al tema, pues para ello existen aplicaciones especializadas en el manejo de estos indicadores. Lo que se puede controlar con la aplicación es el consumo de combustible por vehículo que tiene como objetivo final el cálculo de los indicadores de consumo. (Desoft, 2015)

En el sistema se implementan las siguientes características:

- Control básico de Hoja de Ruta para los vehículos de la entidad.
- Modelo de combustible habilitado y kilómetros recorridos para vehículos administrativos.
- Control básico de mantenimiento a vehículos.
- Cálculo de índices de consumo.
- Control de combustible utilizado de terceros y litros restantes en el tanque de vehículo al finalizar el mes.
- Control de las fechas de vencimiento de la validación técnica FICAV (Empresa de Administración Vial y Diagnóstico Automotor del Ministerio de Transporte) y de la LOT (Licencia Operativa de Transporte).

El equipo de desarrollo de EnerguX está trabajando para implementar un control más preciso sobre el transporte debido a la importancia en nuestro país. Una de las características de mayor necesidad es la de utilizar datos GPS para aumentar la fiabilidad de la información suministrada al sistema. Esta característica además implicaría una mayor automatización al proceso de control de hojas de rutas y vehículos reduciendo los márgenes de error aún más.

1.2 GPS

GPS son las siglas del inglés Global Position System, es un sistema de posicionamiento basado en terminal que permite conocer la situación de un objeto o persona en cualquier lugar del mundo. Se trata de una red de 27 satélites que emiten una señal con el tiempo de emisión y su posición. Esta señal llega al GPS con un cierto retraso, lo cual nos permite calcular de una manera aproximada la distancia del satélite, ya que se conoce que esta señal viaja a la velocidad de la luz. (Wikipedia, 2018b)

Fiabilidad de los datos:

La precisión intrínseca del sistema GPS depende del número de satélites visibles en un momento y posición determinados. Si se capta la señal de entre siete y nueve satélites, y si éstos están en una geometría adecuada (están dispersos), pueden obtenerse precisiones inferiores a 2,5 metros en el 95 % del tiempo. La funcionabilidad de los satélites es por medio de triangulación de posiciones para proporcionar la posición exacta de los receptores (celulares, vehículos, etc.) (Rizos, 1999).

1.3 Servicios Webs

Para definir que es un servicio web se toma como referencia el concepto que propone World Wide Web Consortium (W3C, 2018) que define al servicio web como una aplicación software identificada por un URI (Uniform Resource Identifier), cuyas interfaces se pueden definir, representar y descubrir mediante documentos XML, esto hace posible la interacción de aplicaciones, utilizando mensajes XML, invocados mediante protocolos estándares en internet. Los servicios web exponen funcionalidades que son enviados y recibidos por un agente, en el caso de la persona u organización que envía y recibe mensajes pueden ser proveedor (proporciona un agente para implementar un servicio) o solicitante (utiliza el servicio que proporciona el agente del proveedor). Además de web presentar aplicaciones informáticas mediante tecnologías y protocolos web estándares, proporcionan mecanismos de comunicación para presentar información dinámica al usuario, aquí la interoperabilidad va más allá de la capacidad de intercambiar información entre dos máquinas diferentes sino que también proporciona mecanismos para que los servicios sigan presentando la misma funcionalidad aunque los agentes hayan cambiado.

La utilización de estos sistemas trae consigo ventajas y desventajas ligadas a su naturaleza: (Barbero, 2014)

Ventajas de los Servicios Webs

- Aportan interoperabilidad entre aplicaciones de software independientemente de sus propiedades o de las plataformas sobre las que se instalen.
- Los servicios Web fomentan los estándares y protocolos basados en texto, que hacen más fácil acceder a su contenido y entender su funcionamiento.
- Permiten que servicios y software de diferentes compañías ubicadas en diferentes lugares geográficos puedan ser combinados fácilmente para proveer servicios integrados.

Desventajas de los Servicios Webs

- Para realizar transacciones no pueden compararse en su grado de desarrollo con los estándares abiertos de computación distribuida como CORBA (Common Object Request Broker Architecture).
- Su rendimiento es bajo si se compara con otros modelos de computación distribuida, tales como RMI (Remote Method Invocation), CORBA o DCOM (Distributed Component Object Model). Es uno de los inconvenientes derivados de adoptar un formato basado en texto. Y es que entre los objetivos de XML no se encuentra la eficacia de procesamiento.
- Al apoyarse en HTTP, pueden esquivar medidas de seguridad basadas en firewall cuyas reglas tratan de bloquear o auditar la comunicación entre programas a ambos lados de la barrera.

Los primeros Servicios Web fueron construidos para enviar llamadas a procedimientos remotos sobre la web. Esta idea avanzó rápidamente y resultó en una larga colección de estándares (comenzando con SOAP y WSDL). Estos estándares fueron definidos con poca consideración por la práctica contemporánea, a veces antes de que hubiera alguna implementación para estandarizar. El resultado final de esta prematura estandarización fue confusión, al contrario de orden que es lo que un estándar normalmente brinda.(Hafiz, 2011)

SOAP

SOAP es considerado como el formato para definir el intercambio de datos XML entre dos usuarios, independiente de la plataforma o lenguaje de programación de una forma simple y ligera mediante un modelo de empaquetado de datos modular y una serie de mecanismos de codificación de datos. Su estructura cuenta con variadas especificaciones y extensiones como son la seguridad, formato de entrega, procesamiento del mensaje, enrutamiento, etc. Es considerado junto con el lenguaje de definición de servicios WSDL, un estándar, completamente dependiente del formato XML para la codificación de datos suponiendo una sobrecarga de trabajo para su procesado, también para la transmisión de datos el protocolo HTTP (Hiper Text Transport Protocol), diseñado para trabajar bajo el esquema RPC, invocando funciones remotas.

Estructura de un mensaje

El formato de mensajes SOAP define una estructura de sobre o envoltura que se compone de un encabezado o cuerpo, los datos con los que se define el encabezado aumenta de la funcionalidad del mensaje tales como direccionamiento, seguridad y mensajería confiable. El cuerpo contiene datos a ser transmitidos y soporta bloques de elementos XML, texto u otro contenido como se muestra en la Ilustración 2

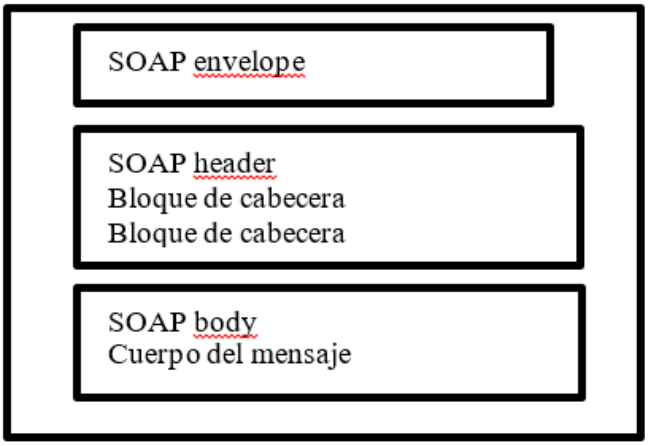

ILUSTRACIÓN 2. ESTRUCTURA DEL UN MENSAJE SOAP

El encabezado es opcional, aunque en un mensaje SOAP es de gran ayuda para su flexibilidad y el control que lleva de los bloques de encabezado. Para un recurso en REST no es de interés llevar un encabezado puesto que una de sus ventajas es utilizar lo mínimo en recursos. Esta información puede tener modificaciones en toda su vida útil, refiriéndose al modelo SOAP que permite el tránsito por puntos intermedios.

En el cuerpo del mensaje se transportan los datos en XML simples, texto o estructuras complejas que no sean binaria o multimedia, aprovechando al máximo las funcionalidades de la codificación XML.

Existe un código especial adjunto para transmitir contenido misceláneo, como documentos multimedia y binario es muy costoso en términos de procesamiento y tamaño de datos, para ello SOAP crea una estructura compuesta, que separa al contenido especial y al mensaje en dos secciones bien definidas. Para transportar este tipo de contenido SOAP utiliza el estándar Extensiones Multipropósito de Correo de Internet (MIME), para adjuntar documentos al mensaje.

Lenguaje de definición de servicios web WSDL

El WSDL nos permite tener una descripción de un servicio web. Especifica la interfaz abstracta a través de la cual un cliente puede acceder al servicio y los detalles de cómo se debe utilizar.(Barbero, 2014)

Elementos del WSDL:

- Types: es el contenedor de definiciones del tipo de datos que utiliza algún sistema de tipos (por ejemplo XSD).
- Message: definición abstracta y escrita de los datos que se están comunicando.
- Operation: descripción abstracta de una acción admitida por el servicio.
- Port Type: conjunto abstracto de operaciones admitidas por uno o más puntos finales.
- Binding: especificación del protocolo y del formato de datos para un tipo de puerto determinado.
- Port: punto final único que se define como la combinación de un enlace y una dirección de red.
- Service: colección de puntos finales relacionados

La estructura que tiene el lenguaje de definición de servicios se observa en la Ilustración 3.

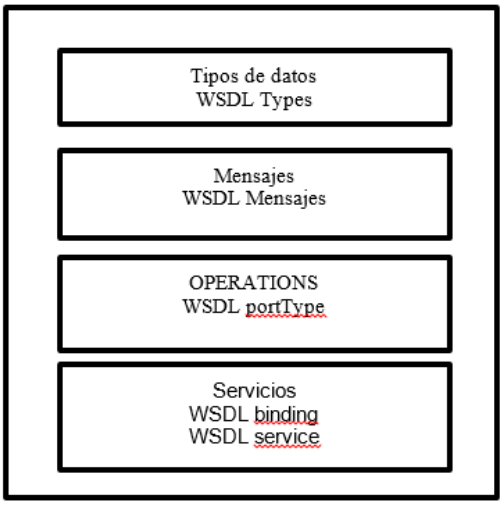

ILUSTRACIÓN 3. ESTRUCTURA DEL LENGUAJE DE DEFINICIÓN DE SERVICIOS

Tipos de datos en SOAP

Los datos que integran el mensaje SOAP siguen ciertas normas que deben ser cumplidas, estos datos son conocidos como datos simples que pueden ser datos complejos y estructurados pero empaquetados. Esto surge ante la necesidad de realizar un mapeo de los tipos de datos de java y el documento SOAP (J. Snell, 2010), la cantidad de tipos de datos es considerable los más destacados se muestran en la Tabla 1.

TABLA 1 COMPARACIÓN TIPOS DE DATOS

Tipo SOAP	Tipo JAVA
int	java.lang.Integer
long	java.lang.Long
short	java.lang.Short
string	java.lang.String
boolean	java.lang.Boolean
float	java.lang.Float
double	java.lang.Double
byte	java.lang.Byte

En respuesta, un estilo alternativo de Servicios Web construido de acuerdo a las reglas de la Web comenzaron a aparecer. Estos servicios (llamados RESTful) son maduros y motiva a las personas a retomar los principios ya probados y ciertos de los estándares webs (Hafiz, 2011).

REST

La Transferencia de Estado Representacional (Representational State Transfer, REST) es un estilo de arquitectura software para sistemas hipermedia distribuidos como la World Wide Web. El termino REST fue implementado por primera vez por Roy Fielding en una conferencia en el Universidad de California, al tratar de principios arquitectónicos distribuidos. Este tipo de servicios web exponen datos y funcionalidades mediante recursos identificados por URI, los clientes interactúan con los recursos mediante métodos de ingresos. Para Fielding REST es un Estilo Arquitectónico que consiste en clientes y servidores. Los clientes generan solicitudes a los servidores, estos la procesan, generando una respuesta apropiada. Las solicitudes y respuestas se construyen alrededor de la transferencia de representaciones de los recursos. Un recurso puede ser esencialmente cualquier concepto que pueda ser tratado. Una

representación de un recurso es típicamente un documento que captura el estado actual o previsto de un recurso (Fielding, 2000). A diferencia del Protocolo SOAP, REST consume un poco menos el ancho de banda porque no se analiza el documento XML como lo hace SOAP, además de que no requiere de cabeceras en el mensaje. Se puede decir que un servicio web RESTFULL es un diseño basado en la arquitectura REST direccionada a construir aplicaciones distribuidas, orientada a publicar e identificar recursos

Roy Fielding documentó REST basado en los principios de la Web a medida que esta evolucionaba. Él identificó cuatro principios fundamentales los cuales llamó "restricciones" (Fielding, 2000):

1. Identificación de recursos
2. Manipulación de recursos a través de representaciones.
3. Mensajes auto descriptivos.
4. Hipermedia como motor del estado de la aplicación.

Estos principios describen la arquitectura de los sistemas y las interacciones de la Web. Los bloques fundamentales que componen la Web son llamados *recursos* (resources). Un recurso es cualquier cosa que pueda ser blanco de un hipervínculo ya sea un archivo, un script o una colección de recursos. En respuesta a una solicitud por un recurso el cliente recibe una representación de ese recurso, que puede tener un formato diferente que el recurso poseído por el servidor. Los recursos son manipulados a través de *mensajes* que tienen un significado especial, en la Web estos mensajes son los métodos HTTP (Fielding, 2000).

REST afirma que la web ha disfrutado de escalabilidad como resultado de una serie de diseños fundamentales clave:

- Un protocolo cliente/servidor sin estado: cada mensaje HTTP contiene toda la información necesaria para comprender la petición. Como resultado, ni el cliente ni el servidor necesitan recordar ningún estado de las comunicaciones entre mensajes. Sin embargo, en la práctica, muchas aplicaciones basadas en HTTP utilizan cookies y otros mecanismos para mantener el estado de la sesión (algunas de estas prácticas, como la reescritura de URLs, no son permitidas por REST)

- Un conjunto de operaciones bien definidas que se aplican a todos los recursos de información: HTTP en sí define un conjunto pequeño de operaciones, las más importantes son POST, GET, PUT y DELETE.
- Una sintaxis universal para identificar los recursos. En un sistema REST, cada recurso es direccionable únicamente a través de su URI.
- El uso de hipermedios, tanto para la información de la aplicación como para las transiciones de estado de la aplicación: la representación de este estado en un sistema REST son típicamente HTML o XML. Como resultado de esto, es posible navegar de un recurso REST a muchos otros, simplemente siguiendo enlaces sin requerir el uso de registros u otra infraestructura adicional.

Utilización de métodos estándar de HTTP

HTTP expone los métodos estándar: GET, PUT POST, DELETE, que son utilizados por desarrolladores estableciendo relaciones con las operaciones de Leer, Actualizar, Crear y Borrar de la siguiente manera.

- GET: Para obtener un recurso del servidor
- PUT: Para cambiar el estado de un recurso o actualizarlo
- POST: Para crear un recurso en el servidor
- DELETE: Para eliminar un recurso en el servidor

Es importante tomar en cuenta que en método GET no debe ser utilizado para ejecutar alguna operación sobre el servidor, como se define solo se utiliza para obtener un recurso del servidor no para realizar modificación alguna sobre él.

Los clientes pueden manejar el formato en que desean que sus datos sean devueltos, para ello es necesario utilizar el atributo HTTP Accept en el encabezado del mensaje y definir el tipo en el content-type. Se utilizan tipo MIME (Multipurpose Internet Mail Extensions) como posibles tipos, en los cuales se retomara la respuesta en dicho formato. Entre los más utilizados se mencionan en la Tabla 2.

TABLA 2 TIPOS MIME COMUNES

Tipos MIME	Formato
Application/xml	XML
Application/json	JSON
Application/xhtml+xml	XHTML

Los diferentes tipos de datos que se pueden utilizar permiten que los servicios puedan ser consumidos desde cualquier otra plataforma.

Comparación entre REST y SOAP

La comparación será realizada basado en una conferencia impartida en la WWW International Conference de 2008.(Leymann, 2008). En la misma Pautasso los compara en tres niveles: principios de arquitectura, decisiones conceptuales, y decisiones tecnológicas.

En cuanto a *principios de arquitectura*, se analizan tres principios (capas de protocolos, manejo de heterogeneidad, y acoplado ligero) ambos estilos cumplen con los tres principios. Sin embargo solo hay un aspecto común entre ellos el acoplado ligero de locaciones (dynamic late binding). Por lo tanto no es posible tomar una decisión en este nivel. Al analizar las *decisiones conceptuales* los autores compararon nueve diferentes decisiones y encontró que servicios RESTful requiere que el diseñador implemente al menos 8 de ellas contra solo 5 para los servicios SOAP. Sin embargo los servicios SOAP tienen más alternativas que su competidor. Finalmente al comparar *decisiones tecnológicas*, se encontró que SOAP nuevamente ofrece mucho más alternativas. Basado en los resultados los autores recomiendan usar REST para integraciones ad hoc y SOAP para soluciones empresariales en donde las transacciones y seguridad a nivel de mensajes sean necesarios.

Otro criterio de comparación es tomado de un artículo científico (F. AlShahwan, 2010). El mismo se resume en la Tabla 3.

TABLA 3 COMPARACIÓN DE SERVICIOS WEB BASADOS EN SOAP Y RESTFUL

Criterio	Servicios web basados en SOAP	Servicios web basados en RESTful
Cliente/ servidor	Perfectamente acoplados	Débilmente acoplados
URI	Representa un único servicio en una URI	Representa cada recurso con una URI
Capa de transporte	Todos	Únicamente HTTP
cache	No soporta	Soporte
Interface	Interfaz no uniforme (WSDL)	Interfaz uniforme
Contexto de información	Informa al cliente el funcionamiento del servicio web	Funcionamiento de servicios web implícito
Tipo de datos	Necesitan conversión de datos	Soporta todos los tipos de datos directamente
Método de información	Body entity of HTTP	URI – HTTP

Descripción de web services	WSDL	WADL
Expandability	No expandible	Expandible sin necesidad de crear nuevos WS
Estándares usados	Especifica estándares (WDSLD, UDDI, WS-security)	Estándares web (URL, métodos HTTP, tipos MIME XML)
Seguridad/confidencialidad	Especifica estándares WS-security	HTTP secutiry

RESTful y SOAP tienen enfoques diferentes. RESTful es un estilo de arquitectura para generar aplicaciones cliente-servidor. SOAP es una especificación de protocolo para intercambiar datos entre dos extremos. De la comparación anterior se deduce que RESTful es más ligero y se basa en HTTP, y no tiene estado permitiendo escalabilidad, es compatible con los marcadores y el almacenamiento en memoria cache y mejora el rendimiento además de soportar todo topo de datos. Debido a que MovilWeb solo da acceso a su servicio por uso del método HTTP GET (obtención de recursos del servidor) no es necesaria la capacidad extra de SOAP en cuanto a transacciones y seguridad a nivel de mensajes.

1.4 Aplicaciones en capas

La estrategia tradicional de utilizar aplicaciones compactas causa gran cantidad de problemas de integración en sistemas de aplicaciones complejos como pueden ser los sistemas de gestión de una empresa o los sistemas de información integrados consistentes en más de una aplicación. Estas aplicaciones suelen encontrarse con importantes problemas de escalabilidad, disponibilidad, seguridad e integración. Para solventar estos problemas se ha generalizado la división de las aplicaciones en capas que normalmente serán tres: una capa que servirá para guardar los datos (modelo), una capa para centralizar la lógica de negocio (control) y por último una interfaz gráfica que facilite al usuario el uso del sistema (presentación).(Albin, 2003)

Si se establece una separación entre la capa de interfaz gráfica (cliente), replicada encada uno de los entornos de usuario, y la capa del modelo, que quedaría centralizada en un servidor de base de datos, se obtiene una potente arquitectura que otorga algunas ventajas:

- Centralización de los aspectos de seguridad y transaccionalidad, que serían responsabilidad del modelo.

- No replicación de la lógica de negocio en los clientes, lo que permite que las modificaciones y mejoras sean automáticamente aprovechadas por el conjunto de los usuarios, reduciendo los costos de mantenimiento.
- Mayor sencillez de los clientes.

La mayoría de las aplicaciones web comunes utilizan una arquitectura basada en la de tres capas extendida a las particularidades de la web.

La arquitectura MVC

La arquitectura Model-View-Controller surgió como patrón arquitectónico para el desarrollo de interfaces gráficas de usuario en entornos Smalltalk. Su concepto se basaba en separar el modelo de datos de la aplicación, de su representación de cara al usuario y de la interacción de éste con la aplicación, mediante la división de la aplicación en tres partes fundamentales:

- El modelo, que contiene la lógica de negocio de la aplicación
- La vista, que muestra al usuario la información que éste necesita.
- El controlador, que recibe e interpreta la interacción del usuario, actuando sobre modelo y vista de manera adecuada para provocar cambios de estado en la representación interna de los datos, así como en su visualización.

1.5 Plataforma Java

Java es el nombre de un entorno o plataforma de computación originaria de Sun Microsystems, capaz de ejecutar aplicaciones desarrolladas usando el Lenguaje de programación Java u otros lenguajes que compilen a código intermedio y un conjunto de herramientas de desarrollo. En este caso, la plataforma no es un hardware específico o un sistema operativo, sino más bien una máquina virtual encargada de la ejecución de aplicaciones, y un conjunto de bibliotecas estándares que ofrecen funcionalidad común.

La plataforma así llamada incluye:

- Edición Estándar (Java Platform, Standard Edition), o Java SE (antes J2SE)
- Plataforma Java, Edición Empresarial (Java Platform, Enterprise Edition), o Java EE (antes J2EE)
- Plataforma Java, Edición Micro (Java Platform, Micro Edition), o Java ME (antes J2ME)

La Plataforma Java se compone de un amplio abanico de tecnologías, cada una de las cuales ofrece una parte del complejo de desarrollo o del entorno de ejecución en tiempo

real. Por ejemplo, los usuarios finales suelen interactuar con la máquina virtual de Java y el conjunto estándar de bibliotecas. Además, las aplicaciones Java pueden usarse de forma variada, como por ejemplo ser incrustadas en una página Web. Para el desarrollo11 de aplicaciones, se utiliza un conjunto de herramientas conocidas como JDK (Java Development Kit, o herramientas de desarrollo para Java).(Froufe, 1997)

Java Runtime Environment

Un programa destinado a la Plataforma Java necesita dos componentes en el sistema donde se va a ejecutar: una máquina virtual de Java (Java Virtual Machine, JVM), y un conjunto de bibliotecas para proporcionar los servicios que pueda necesitar la aplicación. La JVM que proporciona Sun Microsystems, junto con su implementación de las bibliotecas estándares, se conocen como Java Runtime Environment (JRE) o Entorno en tiempo de ejecución para Java. El JRE es lo mínimo que debe contener un sistema para poder ejecutar una aplicación Java sobre el mismo.

En el concepto de máquina virtual se encierra el concepto común de un procesador "virtual" que ejecuta programas escritos en el lenguaje de programación Java. En concreto, ejecuta el código resultante de la compilación del código fuente, conocido como bytecode. Este "procesador" es la máquina virtual de Java o JVM, que se encarga de traducir (interpretar o compilar al vuelo) el bytecode en instrucciones nativas de la plataforma destino. Esto permite que una misma aplicación Java pueda ser ejecutada en una gran variedad de sistemas con arquitecturas distintas, siempre que con una implementación adecuada de la JVM. Este hecho es lo que ha dado lugar a la famosa frase: "write once, run anywhere" (escriba una vez, ejecute en cualquier parte). La condición es que no se utilicen llamadas nativas o funciones específicas de una plataforma y aun así no se asegura completamente que se cumpla una verdadera independencia de plataforma.(Pérez, 2009)

Desde la versión 1.2 de JRE, la implementación de la máquina virtual de Sun incluye un compilador JIT (Just In Time). De esta forma, en vez de la tradicional interpretación del código bytecode, que da lugar a una ejecución lenta de las aplicaciones, el JIT convierte el bytecode a código nativo de la plataforma destino. Esta segunda compilación del código penaliza en cuanto a tiempo, pero el código nativo resultante se ejecuta de forma más eficaz y rápida que si fuera interpretado. Otras técnicas de compilación dinámica del código durante el tiempo de ejecución permiten optimizar más aún el código, dejando atrás el estigma que caía sobre Java en cuanto a su lentitud y en sus últimas versiones

la JVM se ha optimizado a tal punto que ya no se considera una plataforma lenta en cuanto a ejecución de aplicaciones.

Java no fue la primera plataforma basada en el concepto de una máquina virtual, aunque es la que más amplia difusión ha gozado. El empleo de máquinas virtuales se había centrado principalmente en el uso de emuladores para ayudar al desarrollo de hardware en construcción o sistemas operativos, pero la JVM fue diseñada para ser implementada completamente en software, y al mismo tiempo hacer que fuera portable a todo tipo de hardware.(Campos, 1999)

1.6 Capa de datos

Es una de las capas de la arquitectura *Model-View-Controller* (MVC). El modelo o capa de datos contiene la lógica de negocio de la aplicación. A continuación se analizan las bases teóricas que se tuvieron en cuenta para crear esta capa así como las herramientas elegidas para su implementación.

PostgreSQL

PostgreSQL es un Sistema de gestión de bases de datos relacional orientado a objetos y libre, publicado bajo la licencia PostgreSQL, similar a la BSD o la MIT. Como muchos otros proyectos de código abierto, el desarrollo de PostgreSQL no es manejado por una empresa y/o persona, sino que es dirigido por una comunidad de desarrolladores que trabajan de forma desinteresada, altruista, libre y/o apoyada por organizaciones comerciales. Dicha comunidad es denominada el PGDG (PostgreSQL Global Development Group).

Algunas de sus principales características son, entre otras:

Alta concurrencia, mediante un sistema denominado MVCC (Acceso concurrente multiversión, por sus siglas en inglés) PostgreSQL permite que mientras un proceso escribe en una tabla, otros accedan a la misma tabla sin necesidad de bloqueos. Cada usuario obtiene una visión consistente de lo último a lo que se le hizo commit. Esta estrategia es superior al uso de bloqueos por tabla o por filas común en otras bases, eliminando la necesidad del uso de bloqueos explícitos.

Variedad de tipos, PostgreSQL provee nativamente soporte para:

- Números de precisión arbitraria
- Texto largo ilimitado

- Figuras geométricas (con una variedad de funciones asociadas)
- Direcciones IP (IPv4 e IPv6)
- Arreglos
- Direcciones MAC

Adicionalmente los usuarios pueden crear sus propios tipos de datos, los que pueden ser por completo indexables gracias a la infraestructura GiST de PostgreSQL. Algunos ejemplos son los tipos de datos GIS creados por el proyecto PostGIS.

ORM

Object-Relational mapping (ORM) es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y la utilización de una base de datos relacional como motor de persistencia. En la práctica esto crea una base de datos orientada a objetos virtual, sobre la base de datos relacional. Esto posibilita el uso de las características propias de la orientación a objetos (básicamente herencia y polimorfismo).

Hoy en día la programación orientada a objetos (POO) es un paradigma ampliamente aceptado para el desarrollo de aplicaciones empresariales. Este paradigma provee muchas facilidades para modelar objetos del mundo real en contraste a la programación estructurada por poner un ejemplo. Algunas de estas facilidades son los conceptos de clases, encapsulación de datos y comportamientos, herencia, y polimorfismo.(ORSÁG, 2006)

La persistencia de datos es proporcionada generalmente por un sistema gestor de base de datos. Como se trabaja con objetos la solución más trivial seria guardarlos en una base de datos orientada a objetos (OODBMS: Object-oriented database management system). Muchos creían en esta idea en los ‘90s, además de que los principales proveedores de este tipo de gestores prometieron que se convertirían en una tendencia en el futuro. En el presente no lo han hecho y existen dos razones por las cuales se cree que no lo lograron (ORSÁG, 2006):

- Ambientes heterogéneos. Los ambientes orientados a objetos y los ambientes de despliegue eran totalmente diferente tanto tecnológicamente como conceptualmente.
- Dependencia con sistemas anteriores (legacy systems). En los años ‘90s existía una gran expectación por un cambio hacia las bases de datos orientadas a objetos. Pero la mayoría de empresas en ese momento (bancos, aseguradoras,

grandes negocios) tenían sistemas anteriores basados en modelos de redes con COBOL como lenguaje principal y algunos en bases de datos relaciones las cuales para esta fecha ya estaban bastante esparcidas. Para estas empresas el cambio a esta nueva tecnología era muy difícil y costosa.

Hibernate

Hibernate es una herramienta de Mapeo objeto-relacional (ORM) para la plataforma Java (y disponible también para .Net con el nombre de NHibernate) que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML) o anotaciones en los beans de las entidades que permiten establecer estas relaciones. Hibernate genera las sentencias SQL y libera al desarrollador del manejo manual de los datos que resultan de la ejecución de dichas sentencias, manteniendo la portabilidad entre todos los motores de bases de datos con un ligero incremento en el tiempo de ejecución.

Está diseñado para ser flexible en cuanto al esquema de tablas utilizado, para poder adaptarse a su uso sobre una base de datos ya existente. También tiene la funcionalidad de crear la base de datos a partir de la información disponible. Hibernate ofrece también un lenguaje de consulta de datos llamado HQL (Hibernate Query Language), al mismo tiempo que una API para construir las consultas programáticamente (conocida como "criteria"). Soporta todos los sistemas gestores de bases de datos SQL y se integra de manera elegante y sin restricciones con los más populares servidores de aplicaciones J2EE y contenedores web, y por supuesto también puede utilizarse en aplicaciones standalone. Posee un alto rendimiento, tiene una caché de dos niveles y puede ser usado en un clúster. Permite inicialización perezosa (lazy) de objetos y colecciones y soporta la generación automática de llaves primarias.(Wikipedia, 2018a)

1.7 Inversión de control

Es un método de programación en el que el flujo de ejecución de un programa se invierte respecto a los métodos de programación tradicionales, en los que la interacción se expresa de forma imperativa haciendo llamadas a procedimientos o funciones. Tradicionalmente el programador especifica la secuencia de decisiones y procedimientos que pueden darse durante el ciclo de vida de un programa mediante llamadas a funciones. En su lugar, en la inversión de control se especifican respuestas deseadas a sucesos o solicitudes de datos concretas, dejando que algún tipo de entidad o arquitectura externa lleve a cabo las acciones de control que se requieran en el orden

necesario y para el conjunto de sucesos que tengan que ocurrir. La inversión de control sucede cuando es la biblioteca la que invoca el código del usuario.

Spring Framework

Es un marco de trabajo de aplicaciones Java/J2EE desarrollado por Rod Jonson, Juergen Hoeller, Justin Gehtland y Bruce A. Tate. Proporciona una potente gestión de configuración basada en JavaBeans, aplicando los principios de Inversión de Control (IoC). Esto hace que la configuración de aplicaciones sea rápida y sencilla. Ya no es necesario tener singletons ni ficheros de configuración, una aproximación consistente y elegante. Esta factoría de beans puede ser usada en cualquier entorno, desde applets hasta contenedores J2EE. Estas definiciones de beans se realizan en lo que se llama el contexto de aplicación. Presenta una capa genérica de abstracción para la gestión de transacciones, permitiendo gestores de transacción enchufables (pluggables), y haciendo sencilla la demarcación de transacciones sin tratarlas a bajo nivel.

Se incluyen estrategias genéricas para JTA y un único JDBC DataSource. En contraste con el JTA simple o EJB CMT, el soporte de transacciones de Spring no está atado a entornos J2EE. Spring además posee una gran integración con Hibernate, JDO e iBatis SQL Maps en términos de soporte a implementaciones DAO y estrategias con transacciones. Especial soporte a Hibernate añadiendo convenientes características de IoC, y solucionando muchos de los comunes problemas de integración de Hibernate. Todo ello cumpliendo con las transacciones genéricas de Spring y la jerarquía de excepciones DAO (Agüero, 2007).

La Programación Orientada a Aspectos (Aspect Oriented Programming, AOP, por sus siglas en inglés) está presente en el marco de trabajo y se encuentra totalmente integrada en la gestión de configuración de Spring. Se puede aplicar AOP a cualquier objeto gestionado, añadiendo aspectos como gestión de transacciones declarativa. Está basado sobre un marco de trabajo MVC (Model-View-Controller), construido sobre el núcleo de Spring. Este marco de trabajo es altamente configurable a través de interfaces y permite el uso de múltiples tecnologías para la capa vista como pueden ser JSP, Velocity, Tiles, iText o POI. De cualquier manera una capa modelo realizada con Spring puede ser fácilmente utilizada con una capa web basada en cualquier otro framework MVC, como Struts, WebWork o Tapestry. Toda esta funcionalidad puede usarse en cualquier servidor J2EE, y la mayoría de ella ni siquiera requiere su uso. El objetivo central de Spring es permitir que objetos de negocio y de acceso a datos sean reusables, no atados a servicios J2EE específicos.

La arquitectura en capas de Spring (Ilustración 4) ofrece gran flexibilidad. Toda la funcionalidad está construida sobre los niveles inferiores. Por ejemplo se puede utilizar la gestión de configuración basada en JavaBeans sin utilizar el marco de trabajo MVC o el soporte AOP.(Bagüés, 2009)

Contenedor de Inversión de Control de Spring

El Contenedor se encarga de gestionar los ciclos de vida de los objetos específicos: la creación de estos objetos, llamando a sus métodos de inicialización, y configurando estos objetos cableándolos juntos. Los objetos creados por el Contenedor también se denominan objetos gestionados o beans. El Contenedor se puede configurar mediante la carga de archivos XML o la detección de anotaciones Java específicas sobre la configuración de las clases. Estas fuentes de datos contienen las definiciones que proporcionan la información necesaria para la creación de las beans. Los objetos pueden ser obtenidos por cualquiera de los medios de dependencia de búsqueda o dependencia de inyección. Dependencia de búsqueda es un modelo donde se pide al objeto contenedor un objeto con un nombre específico o de un tipo específico. Dependencia de inyección es un modelo en el que el contenedor pasa objetos por nombre a otros objetos, ya sea a través de métodos constructores, propiedades, o métodos de la fábrica. No creas un objeto, sino describes la forma en que deben crearse, definiéndolo en el archivo de configuración de Spring. No llamas a los servicios y componentes, sino dices que servicios y componentes deben ser llamados, definiéndolos en los archivos de configuración de Spring. Esto hace el código fácil de mantener y más fácil de probar mediante Inversión de Control (IoC).

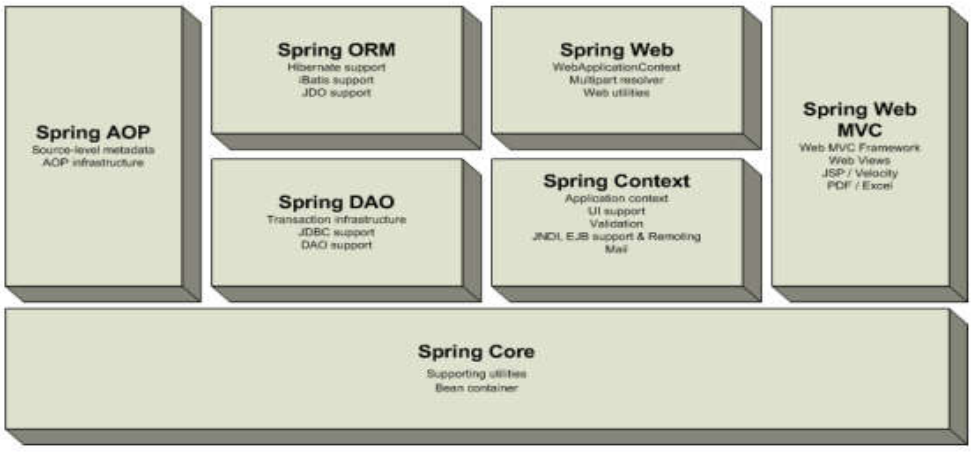

ILUSTRACIÓN 4. ARQUITECTURA EN CAPAS DE SPRING

1.8 Metodologías para el desarrollo de software

En la década de los noventa surgieron metodologías de desarrollo de software ligeras, más adelante nombradas como metodologías ágiles, que buscaban reducir la probabilidad de fracaso por subestimación de costos, tiempos y funcionalidades en los proyectos de desarrollo de software. Estas metodologías nacieron como reacción a las metodologías existentes con el propósito de disminuir la burocracia que implica la aplicación de las metodologías tradicionales en los proyectos de pequeña y mediana escala. Las metodologías tradicionales buscan imponer disciplina al proceso de desarrollo de software y de esa forma volverlo predecible y eficiente. Para conseguirlo se soportan en un proceso detallado con énfasis en planeación (Fowler, 2005) propio de otras ingenierías. El principal problema de este enfoque es que hay muchas actividades que hacer para seguir la metodología y esto retrasa la etapa de desarrollo.

Desarrollo ágil

Hablar de metodologías agiles implica hacer referencia a las metodologías de desarrollo de software tradicionales ya que las primeras surgieron como una reacción a las segundas; sus características principales son antagónicas y su uso ideal aplica en contextos diferentes.

Las metodologías tradicionales de desarrollo de software son orientadas por planeación. Inician el desarrollo de un proyecto con un riguroso proceso de selección de requerimientos, previo a etapas de análisis y diseño. Con esto tratan de asegurar resultados con alta calidad circunscritos a un calendario. En las metodologías tradicionales se concibe un solo proyecto, de grandes dimensiones y estructura definida; se sigue un proceso secuencial en una sola dirección y sin marcha atrás; el proceso es rígido y no cambia; los requerimientos son acordados de una vez y para todo el proyecto, demandando grandes plazos de planeación previa y poca comunicación con el cliente una vez ha terminado ésta.(H. Khurana, 2011)

Las metodologías ágiles son flexibles, pueden ser modificadas para que se ajusten a la realidad de cada equipo y proyecto. Los proyectos ágiles se subdividen en proyectos más pequeños mediante una lista ordenada de características. Cada proyecto es tratado de manera independiente y desarrolla un subconjunto de características durante un periodo de tiempo corto, de entre dos y seis semanas. La comunicación con el cliente es constante al punto de requerir un representante de él durante el desarrollo. Los proyectos son altamente colaborativos y se adaptan mejor a los cambios; de hecho, el cambio en los requerimientos es una característica esperada y deseada, al igual que las

entregas constantes al cliente y la retroalimentación por parte de él. Tanto el producto como el proceso son mejorados frecuentemente.(Ghosh, 2012)

Comparación entre metodologías

LA

Tabla 4 muestra aspectos relevantes de las metodologías de desarrollo tradicional contrastándolas con los aspectos relevantes de las metodologías de desarrollo ágil.

TABLA 4 COMPARACIÓN ENTRE METODOLOGÍAS

Métodos Tradicionales	Métodos Ágiles
Predictivos	Adaptativos
Orientados a procesos	Orientados a personas
Proceso rígido	Proceso flexible
Se concibe como un proyecto	Un proyecto es dividido en proyectos más pequeños
Poca comunicación con el cliente	Comunicación constante con el cliente
Entrega del software al finalizar el desarrollo	Entregas constantes del software
Documentación extensa	Poca documentación

Scrum

Su nombre no corresponde a una sigla, sino a un concepto deportivo, propio del rugby, relacionado con la formación requerida para la recuperación rápida del juego ante una infracción menor. Su primera referencia en el contexto de desarrollo data de 1986, cuando Takeuchi y Nonaka utilizan el Rugby Approach para definir un nuevo enfoque en el desarrollo de productos, dirigido a incrementar su flexibilidad y rapidez, a partir de la integración de un equipo interdisciplinario y múltiples fases que se traslapan entre sí(Nonaka, 1986). La metodología Scrum para el desarrollo ágil de software es un marco de trabajo diseñado para lograr la colaboración eficaz de equipos en proyectos, que emplea un conjunto de reglas y artefactos y define roles que generan la estructura necesaria para su correcto funcionamiento.

Scrum utiliza un enfoque incremental que tiene como fundamento la teoría de control empírico de procesos. Esta teoría se fundamenta en transparencia, inspección y adaptación; la transparencia, que garantiza la visibilidad en el proceso de las cosas que pueden afectar el resultado; la inspección, que ayuda a detectar variaciones indeseables en el proceso; y la adaptación, que realiza los ajustes pertinentes para minimizar el impacto de las mismas.(Alliance, 2012)

Scrum defne tres roles: el Scrum master, el dueño del producto y el equipo de desarrollo. El Scrum master tiene como función asegurar que el equipo está adoptando la metodología, sus prácticas, valores y normas; es el líder del equipo pero no gestiona el desarrollo. El dueño del producto es una sola persona y representa a los interesados, es el responsable de maximizar el valor del producto y el trabajo del equipo de desarrollo; tiene entre sus funciones gestionar la lista ordenada de funcionalidades requeridas o Product Backlog. El equipo de desarrollo, por su parte, tiene como responsabilidad convertir lo que el cliente quiere, el Product Backlog, en iteraciones funcionales del producto; el equipo de desarrollo no tiene jerarquías, todos sus miembros tienen el mismo nivel y cargo: desarrollador.

Conceptos básicos

Scrum defne un evento principal o Sprint (Ilustración 5) que corresponde a una ventana de tiempo donde se crea una versión utilizable del producto (incremento). Cada Sprint, como en el rugby, es considerado como un proyecto independiente. Su duración máxima es de un mes. Un Sprint se compone de los siguientes elementos: reunión de planeación del Sprint, Daily Scrum, trabajo de desarrollo, revisión del Sprint y retrospectiva del Sprint. En la reunión de Planeación del Sprint se defne su plan de trabajo: qué se va a entregar y cómo se logrará. Es decir, el diseño del sistema y la estimación de cantidad de trabajo. Esta actividad dura ocho horas para un Sprint de un mes. Si el Sprint tiene una duración menor, se asigna el tiempo de manera proporcional.

El Daily Scrum es un evento del equipo de desarrollo de quince minutos, que se realiza cada día con el fin de explicar lo que se ha alcanzado desde la última reunión; lo que se hará antes de la siguiente; y los obstáculos que se han presentado. Este evento se desarrolla mediante una reunión que normalmente es sostenida de pie con los participantes reunidos formando un círculo, esto, para evitar que la discusión se extienda (Sutherland, 2011). La Revisión del Sprint ocurre al fnal del Sprint y su duración es de cuatro horas para un proyecto de un mes (o una proporción de ese tiempo si la duración es menor). En esta etapa: el dueño del proyecto revisa lo que se hizo, identifca lo que no se hizo y discute acerca del Product Backlog; el equipo de desarrollo cuenta los problemas que encontró y la manera en que fueron resueltos, y muestra el producto y su funcionamiento. Esta reunión es de gran importancia para los siguientes Sprints.

El Product Backlog es una lista ordenada por valor, riesgo, prioridad y necesidad de los requerimientos que el dueño del producto define, actualiza y ordena. La lista tiene como

característica particular que nunca está terminada, pues evoluciona durante el desarrollo del proyecto.

XP (Extreme Programming)

Fue desarrollada por Kent Beck buscando guiar equipos de desarrollo de software pequeños o medianos, entre dos y diez desarrolladores, en ambientes de requerimientos imprecisos o cambiantes (Beck, 1999). XP tiene como base cinco valores: Simplicidad, Comunicación, Retroalimentación, Respeto y Coraje.

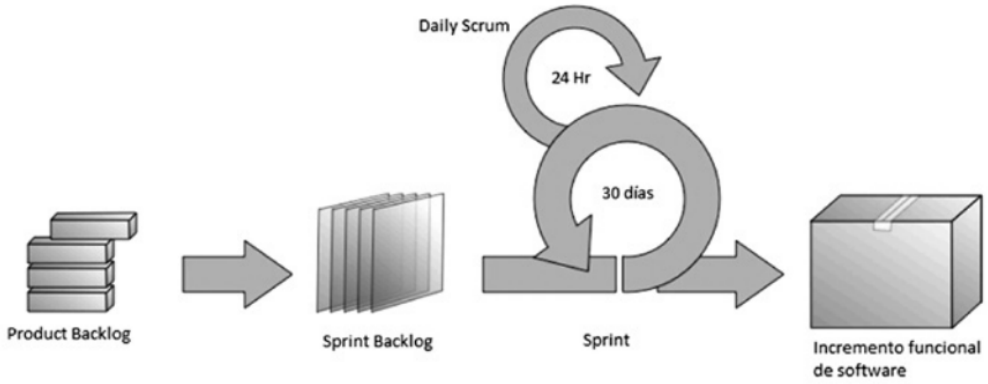

ILUSTRACIÓN 5. FASES SCRUMB

Estos valores, a su vez, son la base para la definición de sus principios. De ellos, los fundamentales son: la retroalimentación rápida, asumir simplicidad, el cambio incremental, la aceptación del cambio y el trabajo de calidad. Las prácticas de esta metodología se derivan de sus valores y principios y están enfocadas en darle solución a las actividades básicas de un proceso de desarrollo, esto es: escribir código, realizar pruebas, escuchar (planear) y diseñar. Las prácticas de XP incluyen: planning game, pequeñas entregas, diseño simple, programación en pareja, pruebas, refactoring, integración continua, propiedad común del código, paso sostenible, cliente en sitio, metáfora y estándares de código.

Conceptos básicos

Planning Game define el alcance y la fecha de cumplimiento de una entrega funcional completa es decir, la fecha de entrega de un release que pueda ser puesto en funcionamiento (Beck, 1999) y divide las responsabilidades entre el cliente y los desarrolladores. El cliente define, utilizando Historias de Usuario, una versión simplificada de los tradicionales Casos de Uso, los requerimientos de manera general, y precisa su importancia; Con base en ellas, los desarrolladores estiman el costo de implementarlas y se definen las características de una entrega y el número de

iteraciones que se necesitarán para terminarla. Para cada iteración el cliente define cuáles de las historias de usuario que componen la entrega funcional desea que se desarrollen. Se pueden crear o modificar historias de usuario en cualquier momento excepto cuando forman parte de una iteración en curso (Martin, 2006).

Entregas Pequeñas se refiere al uso de ciclos cortos de desarrollo (iteraciones) que le muestran software terminado al cliente y obtienen retroalimentación de él. La definición determinada está relacionada con la pruebas de aceptación.

Diseño Simple indica que el sistema debe ser tan simple como sea posible, en un momento determinado, lo cual implica que los desarrolladores deben preocuparse únicamente por las historias de usuario planeadas para la iteración actual, sin importar cuanto pueden cambiar por funciones futuras (Martin, 2006).

Las Pruebas son la guía del desarrollo del producto ya que los detalles de las historias de usuario (i.e., requerimientos específicos) se obtienen en el desarrollo de la prueba de aceptación. Las pruebas son lo primero que se desarrolla; con base en ellas, se desarrolla el código que las satisfaga. A este concepto se le denomina Desarrollo orientado a las pruebas(Martin, 2006).

Refactoring consiste en realizar cambios que mejoren la estructura del sistema sin afectar su funcionamiento. Para garantizar la no afectación, después de cada cambio se corre la prueba unitaria, con el fin de corroborar sus beneficios. Esta práctica ayuda a mantener el código simple.

Conclusiones del capítulo

1. Se implementará un servicio web de tipo RESTFul debido a su rendimiento superior y escalabilidad.
2. Se utilizará la tecnología ORM para la capa de datos debido a su flexibilidad y facilidad de uso para entornos de desarrollos orientados a objetos. Hibernate será el marco de trabajo escogido para implementar esta tecnología debido a su lenguaje de consulta propio y amplio soporte de la comunidad.
3. Para la capa de negocio se escogió la Inversión de Control como una decisión importante del punto de vista arquitectural debido a su flexibilidad al cambio y el acoplado ligero. Spring será el marco de trabajo que se utilizará para implementar esta arquitectura debido a su potente contenedor de inversión de control, además de integrarse muy bien con servicios RESTFul e Hibernate.

4. En cuanto a la metodología a usar se escogió XP debido a la velocidad con que es capaz de terminar el proyecto y a la simplificación de las tareas de planeamiento y estimación.

Capítulo 2. Análisis, diseño e implementación

En este capítulo se documentan los artefactos del proyecto, de acuerdo con las fases de la metodología ágil XP, elegida por su versatilidad y adaptabilidad al cambio.

2.1 Fase de análisis y diseño

En esta fase se determina los requerimientos funcionales y no funcionales del proyecto. Se planifica mediante un plan de iteraciones como lo demanda la metodología el orden y la cantidad de horas para el desarrollo de cada historia de usuario y sus pruebas correspondientes.

Requerimientos iniciales

Requerimientos funcionales

Tiene como objetivo establecer las funcionalidades, propiedades y propósito del software a desarrollarse, aquí se define el comportamiento del sistema de acuerdo a lo que el cliente desea o espera.

A continuación se muestran los requerimientos funcionales determinados a partir del problema de la investigación:

- Consumir los datos de GPS y Hojas de Ruta, de los vehículos de una empresa, del servicio de MovilWeb.
- Guardar los datos consumidos a una base de datos en los servidores de Desoft, de manera que sea accesible desde EnerguX sin necesidad de ningún cambio funcional en el mismo.
- Crear un servicio web el cuál a través de una tarea programada actualice los datos de todos los vehículos de los clientes de EnerguX de manera autónoma.
- El servicio además debe ser capaz de manejar solicitudes de actualización específicas mediante una URL especificando la fecha.
- El servicio debe presentar un listado de los datos de los vehículos y sus hojas de ruta a través de una plataforma web.

Requerimientos no funcionales

Tiene como objetivo determinar los requerimientos específicos para la implementación del proyecto, los mismos están determinados por la arquitectura del sistema que se desea implementar y por las características de los servicios EnerguX y MovilWeb.

A continuación se muestran los requerimientos no funcionales:

- El servicio es desarrollado en Java y desplegado sobre un servidor web Apache Tomcat.
- El servicio es de tipo RESTful.
- El servicio utiliza una base de datos ya existente basada en PostgreSQL.

Limitaciones del proyecto

Tiene como objetivo determinar las limitaciones del proyecto debido a los requerimientos y naturaleza del problema a resolver.

- La aplicación solo puede ser ejecutada en un servidor web basado en Java (JRE).
- La empresa cliente debe tener el servicio de EnerguX desplegado y funcional.
- La empresa debe tener contratado el servicio de Control de Flotas de MovilWeb y tener añadida sus bases de transporte en el servicio.
- El servicio debe tener salida ya sea a través del proxy de la empresa cliente o directamente a la red externa que aloja a MovilWeb.

Plan de iteraciones

El plan de iteraciones tiene como objeto delimitar las tareas, recursos y tiempo con la finalidad de cumplir con los requerimientos del proyecto. El mismo se muestra en la Tabla 5.

TABLA 5 PLAN DE ITERACIONES

Nombre	Duración	Comienzo	Fin
Fase de análisis y diseño	6 días		
Iteración 1	6 días		
Requerimientos iniciales	3 días		
Planear el desarrollo	3 días		
Fase de inicialización	15 días		
Iteración 1	11 días		
Seleccionar e instalar herramientas	1 día		
Historias de usuario	2 días		
Diseñar la arquitectura del sistema	4 días		

Diseñar cambios a la base de datos de EnerguX Diseñar el servicio web	1 día 3 días		
Iteración 2 Determinar granularidad de las solicitudes Monitorear el estado del proyecto Actualización de las historias de usuario	4 días 1 día 2 días 1 día		
Fase de producción	27 días		
Iteración 1 Implementar la base de datos Crear el servicio web Desarrollar interfaz administrativa del servicio	7 días 1 día 3 días 3 días		
Iteración 2 Implementar conexión a MovilWeb y consumo de datos XML. Implementar el parseo XML y persistencia de los datos. Fase de pruebas y evaluación	9 días 2 días 4 días 3 días		
Iteración 3 Implementar tarea programada. Implementar consulta de datos. Actualización de requerimientos adicionales Pruebas y evaluación.	11 días 3 días 4 días 1 día 3 días		
Fase de estabilización	8 días		
Iteración 1 Realizar correcciones al sistema. Pruebas y evaluación	8 días 3 días 3 días		

Analizar resultados	2 días		
Fase de pruebas	7 días		
Iteración 1 Pruebas del sistema Analizar resultados	7 días 5 días 2 días		

2.2 Fase de inicialización

En esta fase se crean las historias de usuario y se diseña la arquitectura general del sistema.

Configuración del Ambiente de Desarrollo:

En esta actividad se realiza la configuración de los elementos técnicos, en este caso para el entorno de desarrollo para aplicaciones web en Java.

- Tipo de proyecto: Aplicación Web en Java
- IDE: IntelliJ IDEA Community Edition
- Servidor web: Apache Tomcat 8.0

Marcos de trabajo y bibliotecas a usar:

- Spring Framework 5.0 – Proporciona la inversión de control del proyecto haciéndolo fácilmente extensible.
- Spring MVC 5.0 – Proporciona la infraestructura Modelo-Vista-Controlador para el proyecto.
- Hibernate ORM 5.2 – Persistencia de los objetos de negocio a la base de datos relacional.
- JAXB 2.3 – Serializador y deserializador de XML a objetos Java.
- Quartz 2.2 – Infraestructura para la programación de tareas autónomas para Java.
- Thymeleaf 3.0 – Generador de HTML dinámicos a partir de plantillas, presenta ventajas de prototipado sobre JavaServer Pages (JSP).
- Bootstrap 3.3.6 - Framework de diseño de aplicaciones web.

Historias de usuario

Las historias de usuario se han realizado en base al análisis de los requerimientos iniciales planteados en la fase de diseño y análisis.

TABLA 6. HISTORIA DE USUARIO 1

Historia de usuario #1	
Nombre: Interfaz administrativa del servicio	Usuario: Cliente
Prioridad en el negocio: Media	Riesgo en desarrollo: Alta
Estimación: 8 horas	Iteración: Producción-1
Programador responsable: Dario Deya Diaz	
Descripción: Interfaz para la configuración especifica del servicio.	
Fecha:	Firma:

TABLA 7. HISTORIA DE USUARIO 2

Historia de usuario #2	
Nombre: Actualizar datos GPS y Hojas de ruta	Usuario: Cliente
Prioridad en el negocio: Alta	Riesgo en desarrollo: Alta
Estimación: 32 horas	Iteración: Producción-2
Programador responsable: Dario Deya Diaz	
Descripción: Sistema responsable de actualizar los datos los vehículos dado una empresa y una fecha. Funcionalidad solo accesible a través de EnerguX.	
Fecha:	Firma:

TABLA 8. HISTORIA DE USUARIO 3

Historia de usuario #3	
Nombre: Actualizar datos GPS y Hojas de ruta de manera automática	Usuario: EnerguXGPS (Propio Sistema)
Prioridad en el negocio: Alta	Riesgo en desarrollo: Baja
Estimación: 24 horas	Iteración: Producción-3
Programador responsable: Dario Deya Diaz	
Descripción: Sistema que actualiza los datos diariamente cuando los servidores estén bajo poca carga. Sistema autónomo no necesita interacción humana.	
Fecha:	Firma:

TABLA 9. HISTORIA DE USUARIO 4

Historia de usuario #4	
Nombre: Sistema de consulta de datos	Usuario: Cliente
Prioridad en el negocio: Baja	Riesgo en desarrollo: Baja
Estimación: 16 horas	Iteración: Producción-3
Programador responsable: Dario Deya Diaz	
Descripción: Mecanismo para consultar los datos actualizados de manera independiente.	
Fecha:	Firma:

TABLA 10 RESUMEN HISTORIAS DE USUARIO

Número	Nombre	Prioridad	Riesgo	Iteración
1	Interfaz administrativa del servicio	Media	Alta	1
2	Actualizar datos GPS y Hojas de ruta	Alta	Alta	2
3	Actualizar datos GPS y Hojas de ruta de manera automática	Alta	Baja	3
4	Sistema de reporte	Baja	Baja	3

Arquitectura del sistema

En la Ilustración 6 se muestra como queda la estructura general del sistema. EnerguX coexiste junto al sistema implementado, los mismos tienen acceso una base de datos común basado en el gestor PostreSQL. El sistema se comunica con el servicio de MovilWeb a través del protocolo TCP/IP sobre una red previamente establecida intercambiando documentos XML. A pesar del que el sistema actualiza los datos directamente en la base de datos de manera autónoma, EnerguX puede solicitar por petición de un cliente la actualización de los datos de una empresa mediante una URL.

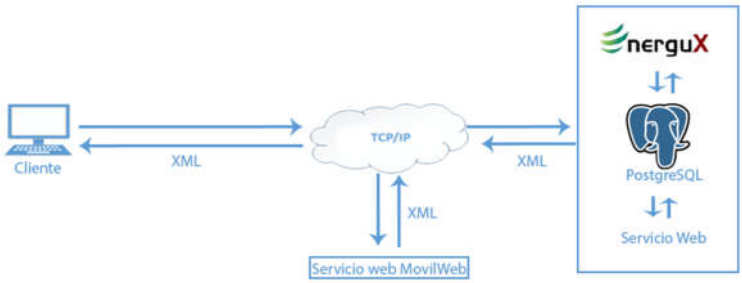

ILUSTRACIÓN 6. ARQUITECTURA DEL SISTEMA

Modelos

Modelo de dominio

A continuación en el Diagrama 1 se muestran las clases fundamentales que contienen la lógica de la aplicación y su relación entre ellas.

MovilWebDSource se encarga de realizar la petición REST a los servidores de MovilWeb. El mismo delega el parseo y deserialización del xml a las clases correspondientes, en este caso a las clases cuyo nombre tiene la estructura *nombre*XML. Las clases deserializadoras crean según la información parseada los objetos de negocio Vehiculo y HojaRuta. A su vez cada objeto HojaRuta tiene una dependecia a el Vehiculo y la Empresa asociada al misma.

Modelo lógico de la base de datos:

La base de datos de EnerguX tiene soporte parcial de los datos que provienen de MovilWeb específicamente los relacionados con GPS. A continuación se presenta las modificaciones que se necesita hacer al mismo para soportar los nuevos datos. El Diagrama 2 muestra solo una parte de la base de datos de EnerguX, debido a su extensión solo se mostrará las tablas que son usadas por el servicio y sus principales columnas. Las columnas resaltadas en verde representan las nuevas columnas a adicionar.

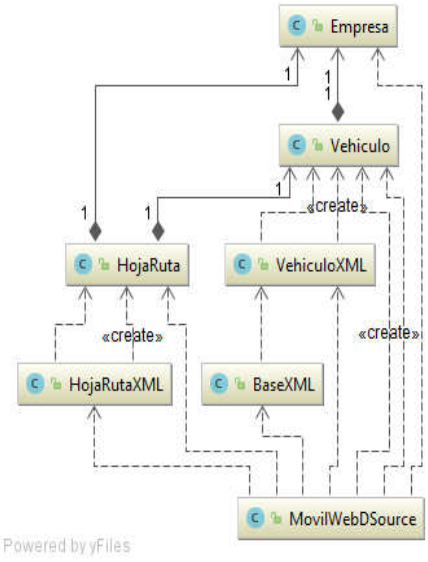

DIAGRAMA 1. MODELO DE DOMINIO

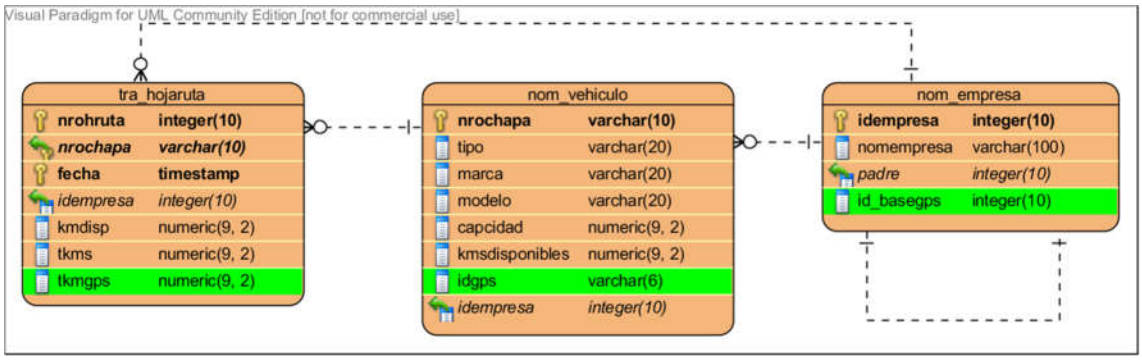

DIAGRAMA 2. CAMBIOS SOBRE EL MODELO LÓGICO DE LA BASE DE DATOS.

Diagramas

Diagrama de componentes:

En el Diagrama 3 se puede observar las dependencias de las bibliotecas y marcos de trabajos utilizados. Tal y como se plantea en el paradigma de la inversión de control Spring es el marco de trabajo que controla como todas estas dependencias coexisten e interactúan entre sí.

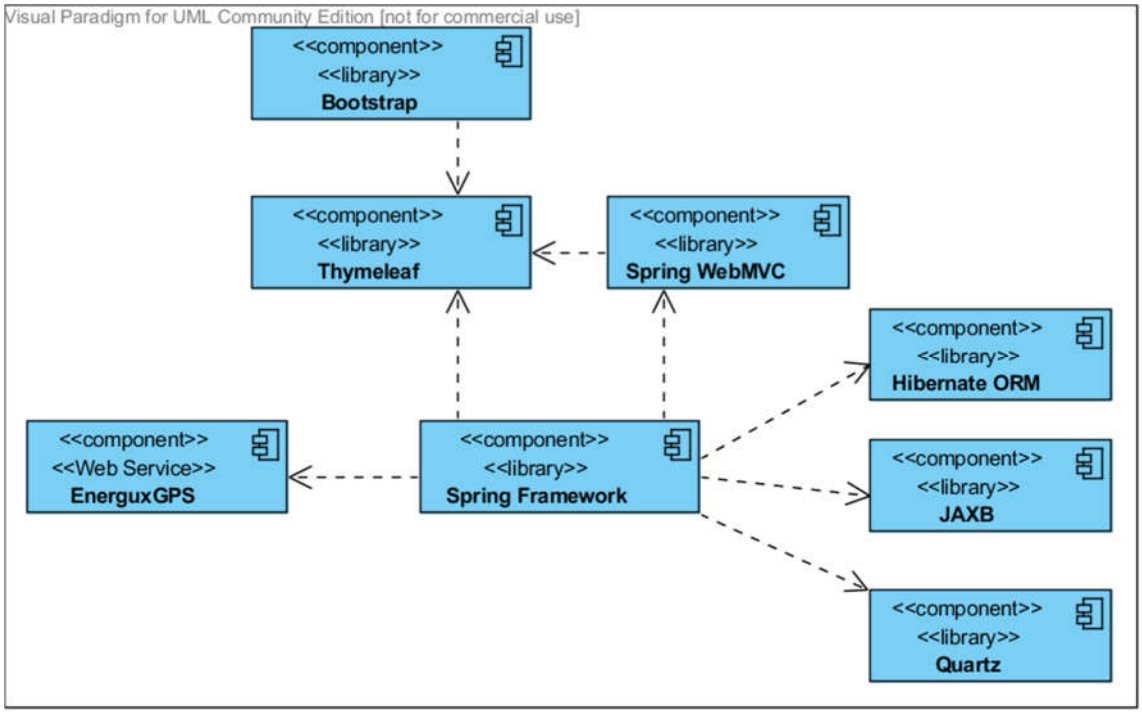

DIAGRAMA 3. DIAGRAMA DE COMPONENTES

Diagramas de secuencia:

Los diagramas de secuencia tienen como objetivo proporcionar una idea general de las principales operaciones y acciones que debe realizar la solución informática a implementar. En la historia de usuario #2 (Tabla 7) se muestra la función más importante que debe realizar el sistema, actualizar las hojas de ruta de los vehículos incluyendo sus datos GPS, los siguientes diagramas de secuencia ilustrara la secuencia de pasos a seguir para lograr este objetivo. Debido a su complejidad el mismo se divide en tres diagramas más sencillos.

La secuencia comienza cuando un usuario o cliente solicita al servidor web una actualización. La clase controladora asociada a esta acción procesa la petición y llama un método update en el cual se implementa la lógica de actualización correspondiente (Ver Diagrama 4). Finalmente una vez terminado el proceso de actualización se le notifica al usuario si se realizó satisfactoriamente o no.

La clase Updater es la responsable de solicitar y actualizar los datos según se requiera. Para esto se necesita conocer que empresa solicita la actualización y así obtener los datos de los vehículos que pertenecen a la misma y sus respectivas hojas de rutas. Los datos de estos vehículos y sus hojas de rutas son guardados localmente por los Objetos de Acceso a los Datos (Data Acces Objects, DAOs). Los mismos son responsables por recuperar y guardar cualquier información en la base de datos (Ver Diagrama 5).

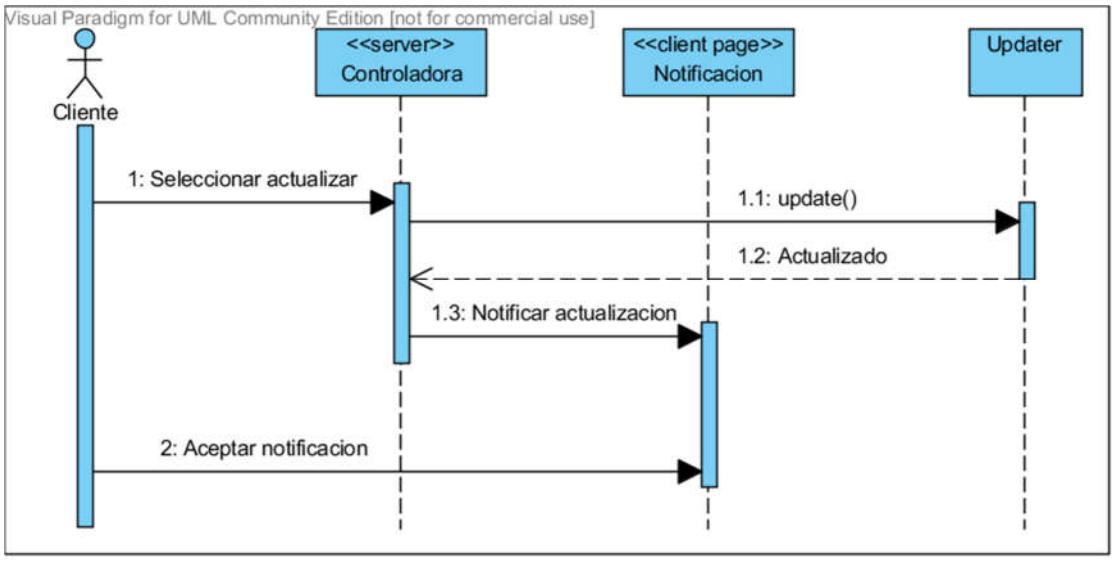

DIAGRAMA 4. DIAGRAMA DE SECUENCIA PARTE 1

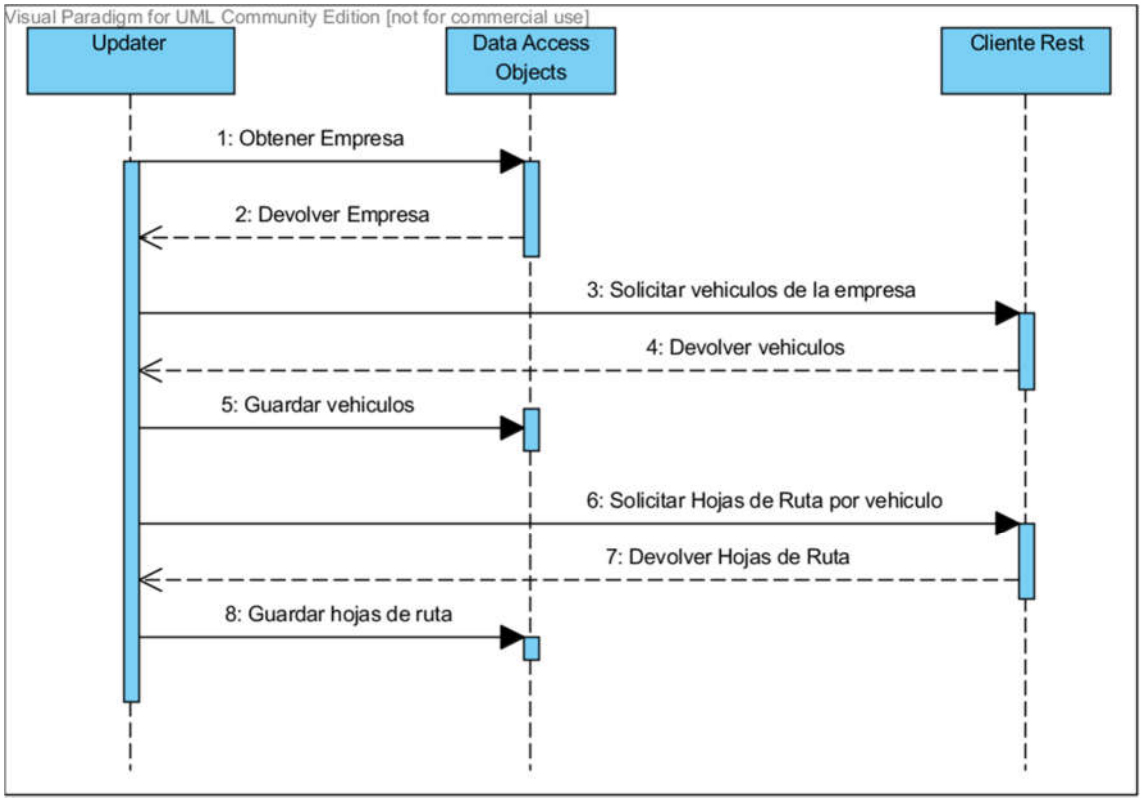

DIAGRAMA 5. DIAGRAMA DE SECUENCIA PARTE 2

Los datos de los vehículos y sus hojas de rutas son manejados por un servicio web basado un REST (MovilWeb), por lo que se necesita un cliente que genere las direcciones URL para cada solicitud con los parámetros adecuados y a su vez transforme la respuesta XML a simple objetos Java para su uso (Ver Diagrama 6).

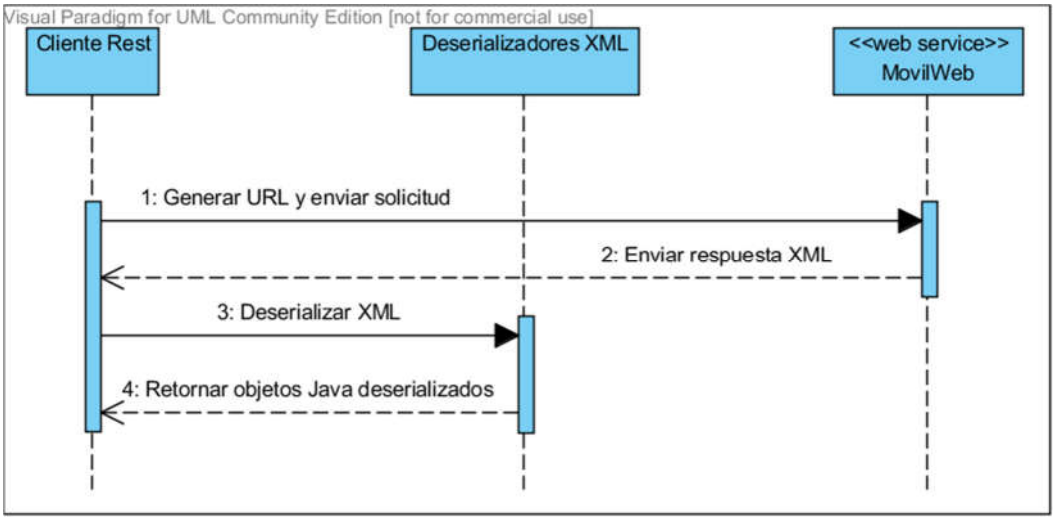

DIAGRAMA 6. DIAGRAMA DE SECUENCIA PARTE 3

Diseño del servicio web

Siguiendo el patrón de arquitectura de software Modelo-Vista-Controlador el servicio acepta un cierto número de peticiones las cuales son respondidas o manejadas por tres clases controladoras. Estas peticiones, sus repuestas y el diseño web correspondiente serán mostrados a continuación.

Consideraciones en cuanto al diseño de la web:

- Interfaz sencilla y minimalista.
- Colores brillantes y tonos pasteles.
- Responsive Design. El diseño debe adaptarse a varios tamaños de pantalla.
- Navegación entre páginas sencilla.

Controlador principal:

El controlador principal maneja la solicitud de actualización del servicio por parte del usuario mediante un botón que es accesible desde cualquier página web de manera fácil e intuitiva. Además, responde a la URL raíz de servicio mostrando la página de bienvenida.

Requisitos de la página de bienvenida:

- Barra de navegación.
- Botón de actualización de servicio.
- Información acerca de la última actualización ya sea manual o automática.
- Mensaje de bienvenida e información sobre la empresa Desoft y el software EnerguX.

Luego de analizar estos requisitos y teniendo en cuenta las consideraciones planteadas se construye el prototipo que se muestra en la Ilustración 7.

Controlador de datos:

El controlador está a cargo de mostrar los datos de los vehículos y las hojas de rutas en la cual su fecha final sea igual a la fecha actual. Además, permite al usuario mediante un formulario seleccionar la fecha manualmente.

Requisitos de la página de datos:

- Barra de navegación.

- Botón de actualización de servicio.
- Información acerca de la última actualización ya sea manual o automática.
- Permitir al usuario elegir la fecha de las hojas de ruta a mostrar.
- Navegación por páginas en caso de 25 o más entradas de datos.

El prototipo diseñado para cumplir los requerimientos planteados se puede ver en la Ilustración 8.

Controlador de configuración:

Su objetivo es la de configurar el servicio web para las condiciones que mejor se ajusten a la empresa. Permite mayor control en cuanto a cuándo y con qué frecuencia el servicio se actualiza automáticamente y sobre qué base de datos de EnerguX debe operar.

Requisitos de la página de configuración:

- Barra de navegación.
- Botón de actualización de servicio.
- Información acerca de la última actualización ya sea manual o automática.
- Formulario que permita configurar la conexión a la base de datos.
- Probar la conexión a la base de datos.
- Formulario que permita configurar a qué hora se actualiza el servicio y qué días.
- Formulario que permita configurar la salida a internet en caso de existir un proxy en la empresa.

El prototipo diseñado para cumplir los requerimientos planteados se puede ver en la Ilustración 9.

2.3 Fase de producción

En la producción se realiza la codificación de los procesos diseñados y la posterior integración entre cada una de ellas para obtener el total funcionamiento de la aplicación.

Configuración de Spring y Spring WebMVC

El marco de trabajo a utilizar para la creación del servicio web es Spring WebMVC el cual posibilita usar las potencialidades del paradigma de la inversión de control. El mismo consiste en permitir a una biblioteca o marco de trabajo, en este caso Spring, manejar la creación, resolución de dependencias y destrucción de los objetos. Para

lograr esto se debe instruir a Spring que objetos crear y cómo hacerlo, a las clases responsables de esto se le llama clases de configuración de Spring. (Ver Ilustración 10)

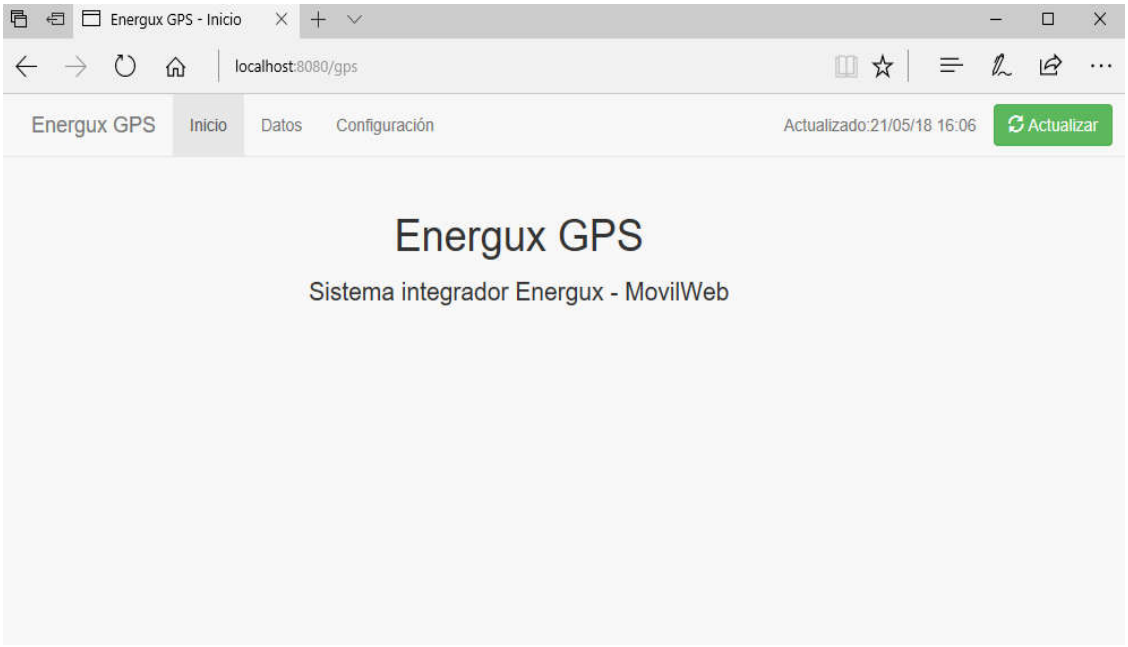

ILUSTRACIÓN 7. PROTOTIPO DE DISEÑO DE LA PÁGINA DE BIENVENIDA

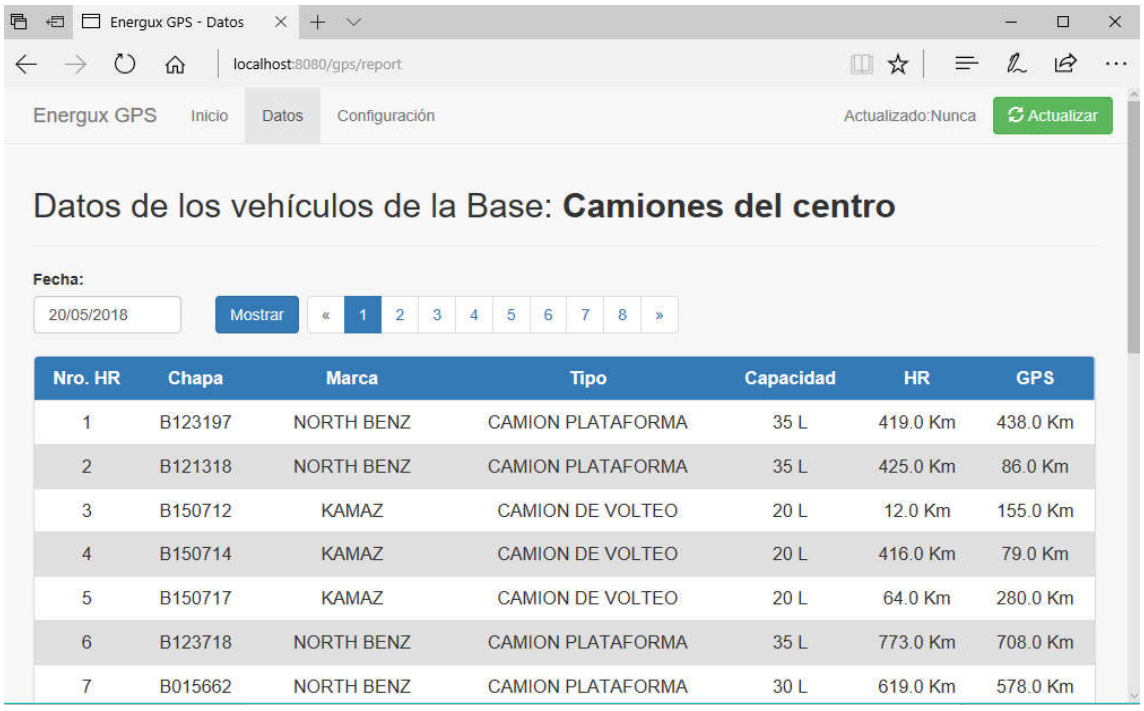

ILUSTRACIÓN 8. PROTOTIPO DE DISEÑO DE LA PÁGINA DE DATOS.

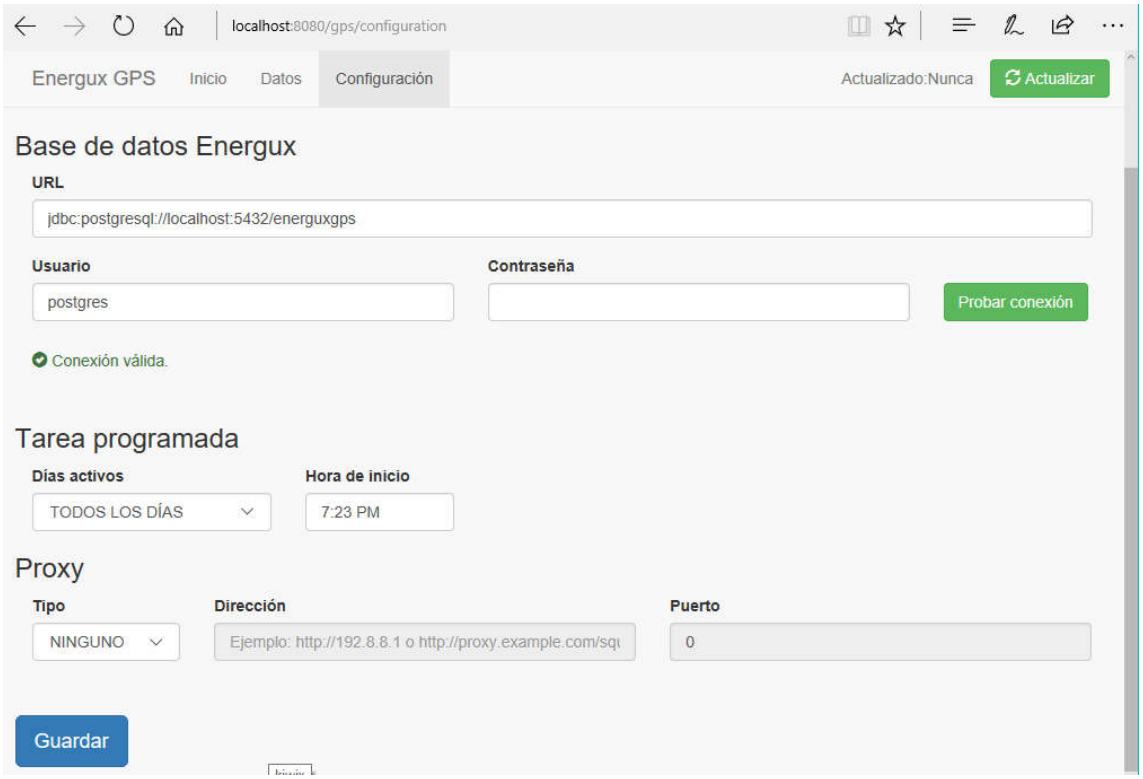

ILUSTRACIÓN 9. PROTOTIPO DE DISEÑO DE LA PÁGINA DE CONFIGURACIÓN.

Clases de configuración de Spring para el proyecto:

- WebInitializer – Esta clase es punto de entrada en la cual se inicializa el server y se envían las solicitudes URL asociadas a el servicio. Además se especifican cuáles son las clases de configuración de Spring.
- WebConfig – Esta clase es la principal configuración de Spring en el proyecto e instruye a Spring que objetos debe crear y cómo.
- RootConfig – Esta clase es una configuración secundaria para objetos que no dependan de ningún objeto del contexto web de la aplicación.

```
@Configuration
@EnableWebMvc
@ComponentScan({"cu.razor.ws.controllers", "cu.razor.ws.dao", "cu.razor.ws.task"})
@PropertySource(value = "classpath:/datasource.properties")
public class WebConfig implements WebMvcConfigurer {
```

ILUSTRACIÓN 10. CONFIGURACIÓN SPRING

La clase WebConfig implementa la interfaz de Spring WebMvcConfigurer, esta interfaz permite la implementación de algunos métodos para configurar parámetros del servicio, como la resolución de recursos estáticos como javascripts, css e imágenes.

Anotaciones importantes:

- @Configuration – Marca esta clase como una clase de configuración de Spring.
- @EnableWebMvc – Posibilita esta clase ser procesada por el módulo WebMVC de Spring y asocia el mismo con el contexto de una aplicación web.
- @ComponentScan – Spring es capaz de inicializar objetos mediante reflexión sin necesidad de decirle explícitamente cómo, a esta función se le conoce como autowiring. Esta anotación le dice a Spring que paquetes debe escanear en busca de objetos candidatos para autowiring. En el proyecto se escanearon las clases controladoras, los objetos de la capa de acceso a los datos y el paquete encargado de la tarea de actualización.

Capa de datos

En esta capa se implementa la capacidad para recuperar o guardar información desde una fuente. Las posibles fuentes de datos son:

- Base de datos
- Servicio web

Las clases asociadas a esta tarea se agruparan en el paquete “dao”. A continuación se explicaran la implementación de dos clases las cuales extraen datos de fuentes diferentes.

Acceso desde un servicio web:

La interfaz GPSDataSource define el comportamiento a seguir por las clases que quieran obtener los datos GPS desde una URL y es fácilmente extensible. (Ver Ilustración 11)

```
public interface GPSDataSource {
    List<Vehiculo> getVehiculosFromURL(String url, Empresa empresa);

    HojaRuta getHRFromURL(String url, Vehiculo vehiculo, String fecha);
}
```

ILUSTRACIÓN 11. INTERFAZ GPSDATASOURCE

La clase MovilWebDSource implementa esta interfaz y solicita los datos usando el protocolo REST. (Ver Ilustración 12)

RestTemplate es un clase parte del marco de trabajo Spring y posibilita de una manera fácil la obtención de recursos usando REST como protocolo. Para su construcción necesita un objeto que implemente la interfaz ClientHttpRequestFactory, este objeto

determina como se crea la conexión HTTP. En este caso el objeto estará configurado con un proxy en caso de existir uno en la empresa. El método getForObject utiliza como parámetros la URL a solicitar, la clase del deserializador a usar para parsear la respuesta y los parámetros de la URL, en este caso el ID de la empresa.

BaseXML parsea la información recursivamente con ayuda de VehiculoXML y retorna un arreglo con todos los vehículos de la base. Finalmente cada objeto VehiculoXML se transforma a un objeto modelo y se le asigna la empresa que le corresponde.

```
@Override
public List<Vehiculo> getVehiculosFromURL(String url, Empresa empresa) {
    BaseXML base = new RestTemplate(requestFactory)
            .getForObject(url, BaseXML.class, empresa.id);
    VehiculoXML[] vehiculosXML = base.getVehiculos();
    Vehiculo[] vehiculos = new Vehiculo[vehiculosXML.length];
    for (int i = 0; i < vehiculosXML.length; i++) {
        vehiculos[i] = vehiculosXML[i].toModel();
        vehiculos[i].empresa = empresa;
    }
    return Arrays.asList(vehiculos);
}
```

ILUSTRACIÓN 12. IMPLEMENTACIÓN ACCESO A VEHÍCULOS EN UN SERVICIO REST

Objetos Modelos:

Los objetos modelos se agrupan en el paquete “model” y su objetivo es mapear datos de una base de datos a un objeto simple de Java, Hibernate es quien realiza el mapeo. Vehiculo es un ejemplo de estos objetos y está asociado a la tabla nom_vehiculo de la base de datos de EnerguX. (Ver Ilustración 13)

Anotaciones importantes:

- @Entity – Describe a la clase como un objeto modelo y lo asocia con el nombre de la tabla que representa
- @Id – Anota el campo identificador del objeto, se usa para hacer consultas mediante identificadores.
- @Column – Mapea el atributo del objeto con una columna de la base de datos.
- @ManyToOne – Implica que el atributo representa una llave foránea de muchos a uno.
- @JoinColumn – Se utiliza como sustituto de @Column para las llaves foráneas y especifica el nombre de la columna en la tabla y el nombre de la columna a la que hace referencia.

```java
@Entity(name = "nom_vehiculo")
public class Vehiculo {
    @Id
    @Column(name = "nrochapa")
    public String chapa;

    @Column(name = "marca")
    public String marca;

    @Column(name = "tipo")
    public String tipo;

    @Column(name = "capacidad")
    public int capacidad;

    @Column(name = "idgps")
    public String idGPS;

    @Column(name = "hruta")
    public boolean hojaRuta;

    @Column(name = "actiivo")
    public boolean activo;

    @Column(name = "descripchapa")
    public String descChapa;

    @ManyToOne
    @JoinColumn(name = "idempresa", referencedColumnName = "idempresa")
    public Empresa empresa;
}
```

ILUSTRACIÓN 13. EJEMPLO DE UN OBJETO MODELO

Acceso desde una base de datos:

Para guardar o recuperar datos de una base datos relacional usando simple objetos Java se utiliza el marco de trabajo Hibernate. Para esto primeramente se necesita configurar a Spring para que maneje las sesiones automáticamente. En la clase de configuración WebConfig se crea un método anotado con @Bean, esta anotación le dice a Spring que es su responsabilidad llamar este método y guardar el objeto que devuelve en caso que ser necesitado por alguna otra dependencia, en este caso los objetos del paquete “dao”. (Ver Ilustración 14)

El objeto DataSource contiene la URL de la base de datos, nombre de usuario y contraseña. LocalSessionFactoryBean no es más que el objeto encargado de construir una sesión, esta sesión se utiliza como contexto para manejar las conexiones a la base de datos y las transacciones. Para poder utilizar correctamente Hibernate el

SessionFactory necesita conocer la fuente de los datos (DataSource), los paquetes que debe escanear para encontrar los objetos modelos y algunas propiedades extras de Hibernate.

```
@Bean
public LocalSessionFactoryBean sessionFactoryBean(DataSource dataSource) {
    LocalSessionFactoryBean localSessionFactoryBean = new LocalSessionFactoryBean();
    Properties props = new Properties();
    try (InputStream inputStream = getClass().getResourceAsStream( name: "/hibernate.properties")) {
        props.load(inputStream);
        localSessionFactoryBean.setHibernateProperties(props);
    } catch (IOException e) {
        e.printStackTrace();
    }
    localSessionFactoryBean.setPackagesToScan("cu.razor.ws.model");
    localSessionFactoryBean.setDataSource(dataSource);
    return localSessionFactoryBean;
}
```

ILUSTRACIÓN 14. CONFIGURACIÓN DE HIBERNATE EN SPRING

Ahora ya se puede solicitar un objeto SessionFactory a Spring en cualquier momento. En Hibernate toda consulta, actualización o inserción en la base de datos tiene que hacerse bajo el contexto de una sesión por lo que este objeto es muy importante. En la Ilustración 15 se muestra un fragmento de la implementación de VehiculoDAO encargado de guardar y recuperar el objeto modelo Vehiculo.

En el constructor se tiene como parámetro un objeto SessionFactory el cual nos proporcionara la sesión adecuada para interactuar con la base de datos. El método saveVehiculo toma un objeto modelo de tipo Vehiculo y lo guarda en la base de datos o lo actualiza en caso de existir previamente un vehículo con la misma chapa. Obsérvese que cualquier inserción o consulta en la base de datos debe hacerse bajo la sesión actual y durante el transcurso una transacción. Hasta que esta transacción nos es finalizada ningún cambio queda de manera permanente en la base de datos.

```
public VehiculoDAO(SessionFactory sessionFactory) { this.sessionFactory = sessionFactory; }

private Session session() { return sessionFactory.getCurrentSession(); }

@Override
public void saveVehiculo(Vehiculo vehiculo) {
    Transaction transaction = session().beginTransaction();
    session().saveOrUpdate(vehiculo);
    transaction.commit();
}
```

ILUSTRACIÓN 15. IMPLEMENTACIÓN DE UN OBJETO DE LA CAPA DE ACCESO

Capa lógica

En esta capa se implementa todo los requerimientos funcionales de la lógica del negocio. Durante este epígrafe se explica la implementación asociada a las historias de usuario más importantes la historia 2 y la 3 (Ver Tabla 7 y Tabla 8 respectivamente).

Actualización del servicio:

La actualización del servicio consiste en obtener desde MovilWeb la lista de vehículos de la empresa, actualizar los datos de los ya existentes y agregar los nuevos. Para cada vehículo solicitar la hoja de ruta correspondiente cuya fecha final es igual a la fecha en la que ocurre la actualización y guardarlo en la base de datos de EnerguX. Por último en caso de que todos los pasos anteriores fueran realizados satisfactoriamente guardar un registro de cuándo fue la última actualización.

La clase Updater es la responsable de la lógica de actualización, la misma necesita para su funcionamiento los objetos de acceso a los datos correspondientes a IEmpresaDAO, IVehiculoDAO, IHojaRutaDAO y GPSDataSource. Este último para extraer los datos desde un servidor REST como MovilWeb, los demás para acceder y guardar los datos en la base de datos de EnerguX. Además necesita un objeto Environment el cuál es instanciado por Spring y contiene las propiedades importadas desde un archivo “properties” por la anotación @PropertySource (Ver Ilustración 10). Este archivo de propiedades contiene las urls del servidor REST al cuál solicitar los datos GPS. Por último necesita un objeto de tipo GeneralSettings cuya responsabilidad es la de guardar la configuración y otros datos que deben ser accesible solo por el servicio por lo que no se guardaran en la base de datos de EnerguX, en este caso de usa para guardar y recuperar el registro de la última actualización. (Ver Ilustración 16)

Esta clase consta con un método update el cual lleva a cabo la actualización. El mismo retorna verdadero cuando la actualización se ha realizado correctamente. (Ver Ilustración 17)

La actualización puede fallar por dos razones fundamentales.

- La empresa no está registrada aún en la base de datos. (getEmpresa() devuelve nulo)
- Existe algún problema con los objetos daos. Ya sea porque hay un problema con la base de datos o con la conexión a las urls del servicio REST. En este caso particular el método lanza una excepción que es manejada fuera del método.

```
public Updater(Environment environment, GPSDataSource gpsDataSource,
               IVehiculoDAO vehiculoDAO, IHojaRutaDAO hojaRutaDAO,
               IEmpresaDAO empresaDAO, GeneralSettings settings) {
    this.environment = environment;
    this.gpsDataSource = gpsDataSource;
    this.vehiculoDAO = vehiculoDAO;
    this.hojaRutaDAO = hojaRutaDAO;
    this.empresaDAO = empresaDAO;
    this.preference = settings.getTaskSettings().getPreference();
    if (preference.getData().hasKey(LASTUPDATE_KEY)) {
        try {
            lastUpdate = DateFormat.getDateTimeInstance(
                    DateFormat.SHORT, DateFormat.SHORT).parse(
                    preference.getData().getString(LASTUPDATE_KEY)
            );
        } catch (ParseException e) {
            e.printStackTrace();
        }
    }
}
```

ILUSTRACIÓN 16. IMPLEMENTACIÓN DE LA CLASE UPDATER

```
public boolean update() {
    Empresa empresa = empresaDAO.getEmpresa();
    if (empresa != null) {
        List<Vehiculo> vehiculos = gpsDataSource.getVehiculosFromURL(
                environment.getProperty("urls.listadoMoviles"), empresa);
        vehiculoDAO.saveVehiculos(vehiculos);
        ArrayList<HojaRuta> hojaRutas = new ArrayList<>(vehiculos.size());
        Calendar calendar = Calendar.getInstance();
        String fecha = getDateFormatted(calendar);
        for (Vehiculo vehiculo : vehiculos) {
            HojaRuta hr = gpsDataSource.getHRFromURL(
                    environment.getProperty("urls.listadoHR"), vehiculo,
                    fecha);
            if (hr != null) {
                hojaRutas.add(hr);
            }
        }
        hojaRutaDAO.saveHRs(hojaRutas);
        logUpdate();
        return true;
    }
    return false;
}
```

ILUSTRACIÓN 17. IMPLEMENTACIÓN DE LA ACTUALIZACIÓN

Finalmente, logUpdate guarda la fecha formateada y de manera legible para el usuario en un archivo configuración. El archivo está conformado una relación clave-valor en el cuál se guarda de manera asincrónica. (Ver Ilustración 18)

```
private void logUpdate() {
    lastUpdate = new Date();
    preference.edit()
            .putString(LASTUPDATE_KEY,
                    DateFormat.getDateTimeInstance(DateFormat.SHORT, DateFormat.SHORT)
                            .format(lastUpdate))
            .apply();
}
```

ILUSTRACIÓN 18. GUARDADO DEL REGISTRO DE LA ÚLTIMA ACTUALIZACIÓN

Tarea programada:

Para la implementación de la tarea programada se usa la biblioteca Quartz, es de código abierto y está disponible para su uso y modificación en GitHub. Cómo la mayoría de las bibliotecas del proyecto primeramente se debe configurar a Spring para que pueda manejar automáticamente la tarea programada.

Configuración Spring para biblioteca Quartz:

Una tarea programada consta de tres partes u objetos fundamentales en la infraestructura de Quartz. A continuación se explican brevemente la función de cada uno y para luego mostrar la implementación de su configuración en Spring:

- Scheduler – Es quien administra la lista de tareas programadas y se encargado de ejecutarlas cuando se cumpla cierta condición. A esta condición se le llama trigger.
- Trigger – Representa el cuándo de la tarea. Es el encargado de determinar cuándo una tarea debería ejecutarse, es usado por el Scheduler para determinar si una tarea debió haberse ejecutado en un tiempo pasado, llamado missfire, y cuanto tiempo falta para la próxima posible ejecución.
- JobDetail – Representa el qué y cómo de la tarea programada. Contiene los detalles de la tarea, la clase que tiene el código a ejecutar los datos que necesita el mismo, entre otros.

Primeramente se debe determinar cuáles de estos objetos se necesita que Spring maneje y cuáles son responsabilidad del programador.

Objetos manejados por Spring:

El objeto Scheduler es manejado por Spring por dos razones fundamentales:

- El objeto es creado automáticamente cuando el servicio se inicie y será destruido llamando el método “shutdown” cuando el mismo finalice. Esta última característica es muy importante ya que Scheduler inicia un hilo que controla cuando se inicia las tareas y si no es apagado antes de cerrar el servicio puede provocar que el servidor Apache no finalice debido a hilos ejecutándose indefinidamente.
- Para crear el JobDetail y el Trigger se necesita objetos que ya son manejados por Spring por lo que son inyectados automáticamente.

Para su configuración solo se necesita crear el objeto Scheduler, programar una tarea proveyendo el JobDetail y el Trigger del mismo y por último iniciar el Scheduler con el método start().En la anotación @Bean se especifica que antes de destruir el objeto Spring debe llamar el método shutdown para garantizar que el Scheduler termine las tareas pendientes, libere recursos y finalice correctamente los hilos creados por él (Ver Ilustración 19).

```
@Bean(destroyMethod = "shutdown")
public Scheduler scheduler(Updater updater, GeneralSettings settings) throws SchedulerException {
    Scheduler s = StdSchedulerFactory.getDefaultScheduler();
    s.scheduleJob(jobDetail(updater), trigger(settings.getTaskSettings().getPreference().getData()));
    s.start();
    return s;
}
```

ILUSTRACIÓN 19. CONFIGURACIÓN DE QUARTZ EN SPRING

Objetos no manejados por Spring:

Para crear un JobDetail se debe asignar una identidad o identificador a la tarea, para esto se le asigna un nombre y a qué grupo pertenece. Se le indica que objetos o datos debe usar la tarea, se realiza en un mapa de tipo llave-valor. En este caso se suministra el objeto Updater, objeto responsable de actualizar el servicio. Por último, se le especifica la clase responsable de ejecutar la tarea, en este caso UpdaterTask. (Ver Ilustración 20)

Para crear el Trigger se debe saber qué días y a qué hora se quiere que se ejecute. Estos datos son proporcionados como un mapa llave-valor por GeneralSetting. Los datos son parseados y se crea un objeto CronSchedule dado una expresión de tiempo basado en la herramienta muy potente para crear tareas programadas llamada “cron” creada para UNIX. Al Trigger se le asigna una identidad y un grupo, la cual es útil para

luego poder remplazar el Trigger por otro en caso de que la configuración de hora y días cambien. (Ver Ilustración 21)

```
private static JobDetail jobDetail(Updater updater) {
    JobDataMap map = new JobDataMap();
    map.put("updater", updater);
    return JobBuilder
            .newJob(UpdateTask.class)
            .usingJobData(map)
            .withIdentity( name: "UpdateTask",  group: "Group1")
            .build();
}
```

ILUSTRACIÓN 20. CREACIÓN DE UN JOBDETAIL

```
public static Trigger trigger(Bundle b) {
    String days = b.getString( key: "task.trigger.days", TaskSettings.EVERY_DAY);
    String time = b.getString( key: "task.trigger.time",  defValue: "8:00");
    int hour = Integer.parseInt(time.substring(0, time.indexOf(':')));
    int minute = Integer.parseInt(time.substring(time.indexOf(':') + 1));
    CronScheduleBuilder cronScheduleBuilder;
    if (days.equals(TaskSettings.EVERY_DAY)) {
        cronScheduleBuilder = CronScheduleBuilder.dailyAtHourAndMinute(hour, minute);
    } else {
        cronScheduleBuilder = CronScheduleBuilder.cronSchedule("0 " + minute + " " + hour + " ? " MON-FRI");
    }
    cronScheduleBuilder.withMisfireHandlingInstructionDoNothing();
    return TriggerBuilder
            .newTrigger()
            .withIdentity( name: "MovilWebTrigger",  group: "group1")
            .withSchedule(cronScheduleBuilder)
            .startNow()
            .build();
}
```

ILUSTRACIÓN 21. CREACIÓN DE UN TRIGGER

La clase UpdateTask:

La clase responsable de llevar a cabo la lógica de la tarea tiene que implementar la interfaz Job de la biblioteca Quartz. En este caso lo único que se hace es llamar el método update del objeto Updater. (Ver Ilustración 22)

```
public class UpdateTask implements Job {

    @Override
    public void execute(JobExecutionContext jobExecutionContext) {
        JobDataMap map = jobExecutionContext.getJobDetail().getJobDataMap();
        Updater updater = ((Updater) map.get("updater"));
        updater.update();
    }
}
```

ILUSTRACIÓN 22. IMPLEMENTACIÓN DE LA CLASE UPDATETASK

Capa visual

En este epígrafe se explica construcción de las páginas html dinámicas usando Thymeleaf y las clases controladoras que responden ante las solicitudes que provienen de estas páginas.

Clases controladoras en Spring WebMVC:

Las clases controladoras se indican con la anotación @Controller al momento de declarar la clase en cuestión. Estas clases son candidatas para autowiring, por lo que al ser escaneado el paquete donde se encuentran se convierten en objetos manejados automáticamente por Spring (Ver @ComponentScan en Ilustración 10).

Los métodos de esta clase indican que páginas html se debe generar y tienen que ser anotados con @RequestMapping. Esta anotación tiene parámetros como “path”, este define la URL relativa a la cual se da respuesta con este método y el parámetro “method” determina el tipo de solicitud (GET, POST, UPDATE o DELETE) a la que se responderá. Por último estos métodos deben retornar una cadena de caracteres que indica inequívocamente la página a genera.

```
@RequestMapping(path = {"/", "/home", "/index"}, method = RequestMethod.GET)
public String homepage(Model model) {
    model.addAttribute( s: "active",  o: 1);
    model.addAttribute( s: "updateTime", Updater.getLastUpdateTime());
    return "home";
}
```

ILUSTRACIÓN 23. MAPEO DE URL EN CONTROLADOR SPRING

En la Ilustración 23 se muestra un ejemplo de un método que devuelve la página principal del servicio y forma parte de la clase MainController. El parámetro “path” puede mapear este método a más de una solicitud, siempre de manera relativa a la URL base del servicio. El objeto Model pasado como parámetro no es obligatorio pero permite pasar objetos al generador html de Thymeleaf para que genere la página en función de datos dinámicos.

Resolución de vistas:

Para que el ejemplo de la Ilustración 23 funcione correctamente Spring debe saber a partir de la cadena de caracteres retornada, “home” en este caso, que plantilla html se corresponde y quien es el encargado de generar a partir de ella la página web final que

será enviada al navegador. Para esto se configura un ViewResolver, objeto de Spring encargado de localizar la vista y de generar a además la página definitiva.

Para configurar Thymeleaf se debe usar un ViewResolver especial llamado ThymeleafViewResolver de la propia bliblioteca. El mismo necesita un motor para procesar las plantillas y este a su vez un TemplateResolver para poder encontrar las plantillas. La configuración se dividió en tres métodos para hacer su explicación más sencilla. Nótese que en el método htmlTemplateResolver se indica el prefijo del camino donde se encuentra las plantillas, el sufijo de los archivos y que tipo de archivo se quiere generar. En este caso la cadena “home” corresponde a la plantilla que está en la dirección relativa al servicio /WEB-INF/home.html. (Ver Ilustración 24)

```
@Bean
public ViewResolver htmlViewResolver() {
    ThymeleafViewResolver viewResolver = new ThymeleafViewResolver();
    viewResolver.setTemplateEngine(templateEngine(htmlTemplateResolver()));
    viewResolver.setCharacterEncoding("UTF-8");
    return viewResolver;
}

private SpringTemplateEngine templateEngine(SpringResourceTemplateResolver templateResolver) {
    SpringTemplateEngine templateEngine = new SpringTemplateEngine();
    templateEngine.setTemplateResolver(templateResolver);
    templateEngine.setEnableSpringELCompiler(true);
    return templateEngine;
}

private SpringResourceTemplateResolver htmlTemplateResolver() {
    SpringResourceTemplateResolver resolver = new SpringResourceTemplateResolver();
    resolver.setApplicationContext(applicationContext);
    resolver.setPrefix("/WEB-INF/");
    resolver.setSuffix(".html");
    resolver.setTemplateMode("HTML5");
    resolver.setCharacterEncoding("UTF-8");
    return resolver;
}
```

ILUSTRACIÓN 24. CONFIGURACIÓN THYMELEAF EN SPRING Y RESOLUCIÓN DE VISTAS

Resolución de recursos estáticos:

Se le llama recursos estáticos a archivos de los cuales depende una página html y no necesitan preprocesamiento alguno. Ejemplo imágenes, javascript, css y fuentes. Para esto solo se necesita mapear las url de estos recursos a la ubicación física de los archivos. Como se configura en Spring WebMVC se muestra en la Ilustración 25.

```
@Override
public void addResourceHandlers(ResourceHandlerRegistry registry) {
    registry.addResourceHandler( ...pathPatterns: "/css/**")
            .addResourceLocations("/WEB-INF/css/");
    registry.addResourceHandler( ...pathPatterns: "/js/**")
            .addResourceLocations("/WEB-INF/js/");
    registry.addResourceHandler( ...pathPatterns: "/images/**")
            .addResourceLocations("/WEB-INF/images/");
    registry.addResourceHandler( ...pathPatterns: "/fonts/**")
            .addResourceLocations("/WEB-INF/fonts/");
}
```

ILUSTRACIÓN 25. CONFIGURACIÓN PARA LA RESOLUCIÓN ESTÁTICA DE RECURSOS

Plantillas Thymeleaf:

Thymeleaf presenta tres ventajas que lo hacen ideal para este tipo de proyectos:

- Su integración con Spring. Permite usar expresiones de Spring para generar datos dinámicos e incluye un ViewResolver propio.
- Puede generar archivos de varios tipos a partir de plantillas. Algunos de ellos son archivos html, css, texto plano entre otros.
- Las plantillas HTML se basan en agregar atributos que ya existen en HTML pero con un nombre de espacio diferente. Es fácil de aprender por quien ya conoce HTML y no agrega etiquetas nuevas. Esto es especialmente bueno para la realización de prototipos, una plantilla por si sola puede ser visualizada sin problema en un navegador, el mismo solo ignorará los nombres de espacio que no conozca, y mantiene la mayoría de sus características.

En la Ilustración 26 se puede ver un fragmento de la plantilla que genera la tabla de datos de hojas de rutas. El nombre de espacio de Thymeleaf es “th” y además de replicar los atributos estándares de HTML agrega nuevas como th:if y th:each.

- th:if – Genera la etiqueta solo si la expresión es verdadera.
- th:each – Repite la etiqueta en la que se encuentra y su descendencia por cada objeto en una lista dada.

Formularios Thymeleaf:

La capacidad de implementar formularios usando Thymeleaf que se integran con la infraestructura de Spring hace muy sencillo el trabajo con formularios. Las principales características que permite Thymeleaf son:

- Especificar la url a la cual se envía los datos del formulario.

- Especificar un objeto el cual contiene los valores por defecto y además permite a Spring mapear las entradas y los atributos del mismo.
- Utilización de métodos útiles de Spring WebMVC de manera nativa y simple.

```
<table class="table table-responsive">
    <tr class="head">
        <th>Nro. HR</th>
        <th>Chapa</th>
        <th>Marca</th>
        <th>Tipo</th>
        <th>Capacidad</th>
        <th>HR</th>
        <th>GPS</th>
    </tr>
    <!--/*@thymesVar id="HRList" type="java.util.List<cu.razor.ws.model.HojaRuta>"*/-->
    <tr th:if="${HRList==null ||HRList.isEmpty()}">
        <td colspan="6">No se encontraron datos</td>
    </tr>
    <tr th:each="HR,idx:${HRList}" th:class="${idx.even}?'even'">
        <td th:text="${HR.nro}">45</td>
        <td th:text="${HR.vehiculo.chapa}">-</td>
        <td th:text="${HR.vehiculo.marca}">-</td>
        <td th:text="${HR.vehiculo.tipo}">-</td>
        <td th:text="${HR.vehiculo.capacidad}+' L'">-</td>
        <td th:text="${HR.totalKm}+' Km'">256</td>
        <td th:text="${HR.totalKmGPS}+' Km'">256</td>
    </tr>
</table>
```

ILUSTRACIÓN 26. FRAGMENTO DE PLANTILLA HTML DE THYMELEAF

```
<form method="post" class="form-group" th:action="${#mvc.url('CC#dbTest').build()}"
      th:object="${settings}">
    <div class="row"><h3>Base de datos Energux</h3></div>
    <div class="row form-group">
        <div class="col-md-12">
            <label for="dbUrl">URL</label>
            <input type="url" id="dbUrl" class="form-control"
                   placeholder="Ejemplo: jdbc:postgresql://localhost:5432/dbname"
                   th:field="*{dbSettings.url}">
        </div>
    </div>
    <div class="row">
        <div class="col-md-5">
            <label for="dbUser">Usuario</label>
            <input type="text" id="dbUser" class="form-control"
                   th:field="*{dbSettings.username}">
        </div>
        <div class="col-md-5">
            <label for="dbPass">Contraseña</label>
            <input type="password" id="dbPass" class="form-control"
                   th:field="*{dbSettings.password}">
        </div>
        <div class="col-md-2">
            <input class="btn btn btn-success" type="submit" value="Probar conexión" style="...">
        </div>
    </div>
```

ILUSTRACIÓN 27. FRAGMENTO FORMULARIO DE CONEXIÓN A LA BASE DE DATOS

En la Ilustración 27 se muestra un fragmento de la plantilla utilizada para el formulario de configuración de la aplicación. En el mismo se puede observar las principales características que hacen de Thymeleaf una opción ideal para su integración en proyectos Spring.

- th:action – Indica el url al cuál se enviará el formulario una vez el usuario presione el botón de tipo “submit”.
- #mvc.url(‘CC#dbTest’) – La expresión se utiliza para indicar que se quiere construir una url según indica el método dbTest en la clase ConfigController cuyas iniciales son ‘CC’. Esto es un método de Spring WebMVC que Thymeleaf integra con una sintaxis especial para facilitar su uso.
- th:object – Indica el objeto que servirá como referencia por el formulario. En sus atributos se guardarán y obtendrán los valores de cada componente. Este objeto será añadido a través de la clase Model por el controlador encargado de solicitar esta vista.
- th:select – La sintaxis de este atributo es un poco diferente ya que utiliza el carácter * y no $. La razón de esto es que este operador no evalúa la expresionn en su interior si no que selecciona, de ahí su nombre en inglés “selector operator”, un atributo del objeto declarado en el th:object. Su objetivo es el de especificar que atributo se usará para guardar y recuperar el valor del componente en el que se encuentra.

Conclusiones del capítulo

1. Se implementaron todos los requerimientos funcionales a partir de las historias de usuario.
2. Se mostró y explicó la implementación de las principales funcionalidades del sistema pasando por cada una de las capas de la arquitectura modelo-vista-controlador.

Capítulo 3. Pruebas y evaluación

En este capítulo se realizan las pruebas necesarias para comprobar los requisitos funcionales y la calidad de la solución propuesta.

Durante el capítulo se mostrarán los resultados y conclusiones de tres tipos de pruebas.

- Prueba de granularidad – Tiene como objetivo determinar el tiempo de reacción de los servidores de MovilWeb dada una cierta solicitud con cantidad de parámetros variables.
- Pruebas unitarias – Tiene como objetivo comprobar el correcto funcionamiento de las unidades lógicas.
- Prueba de interfaz – Tiene como objetivo comprobar que la interfaz sea amigable en varios dispositivos siguiendo con el paradigma Responsive Design.

3.1 Prueba de granularidad

La prueba fue realizada durante la fase de inicialización del proyecto (Ver Tabla 5) para determinar antes de la implementación del proyecto el tiempo de respuesta del servidor MovilWeb para determinados parámetros.

El tiempo de respuesta está determinado por la siguiente relación:

- Menor especificidad de los parámetros (Poca granularidad) – Provoca mayor cantidad de datos obtenidos como respuesta a la solicitud, esto implica generar un XML más grande y un mayor uso de la red.
- Mayor especificidad de los parámetros (Mucha granularidad) – Provoca una consulta en la base de datos de MovilWeb más compleja y puede que se necesite realizar más de una solicitud para completar la tarea que se desee realizar.

Se necesita encontrar un balance entre las relaciones anteriores ya que ambas presentan ventajas y desventajas propias. Se realizaron tres casos de prueba en la empresa Camiones de Centro, la base cuenta con 195 vehículos, y los resultados obtenidos se encuentran en la Tabla 11.

- Caso 1 – Parámetros: Fecha final de la hoja de ruta.
- Caso 2 – Parámetros: Id de la empresa, fecha final de la hoja de ruta.
- Caso 3 – Parámetros: Fecha final de la hoja de ruta, chapa de vehículo.

TABLA 11. RESULTADO PRUEBA DE GRANULARIDAD

Casos de pruebas	Tamaño de la repuesta (Kilobytes)	Tiempo promedio de repuesta (milisegundos)	Cantidad de solicitudes	Tiempo total (milisegundos)
Caso 1	2652 Kb	51235 ms	1	51235 ms
Caso 2	138 Kb	2200 ms	1	2200 ms
Caso 3	0,8 Kb	11,2 ms	195	2194 ms

Conclusiones de la prueba:

Los resultados del caso de prueba 1 muestra que no es práctico porque al solicitar todas las hojas de rutas sin especificar la empresa que pide la solicitud la respuesta resultante es inmensa en tamaño. Los resultados para los casos de prueba 2 y 3 son muy similares en cuanto a tiempo total. El caso 3 tiene mejor tiempo total y a pesar de la pequeña la diferencia con el caso 2 la red se satura menos ya que cada solicitud solo ocupa 0,8 Kilobytes cada 11,2 milisegundos permitiendo otros datos utilizar la red durante ese espacio de tiempo. Esto se puede ver como una analogía a como se emula varias tareas en una sola unidad de procesamiento en una CPU, dando a cada tarea un pequeño tiempo de ejecución y luego pausándola para que otra tome su lugar. Esto ayuda a un uso eficiente de los recursos ya sea el procesador o la red en este caso en cuestión.

3.2 Pruebas unitarias

Las pruebas unitarias fueron realizadas durante las fechas previstas en la Tabla 5 correspondientes a la fase de pruebas.

Herramientas para la realización de pruebas unitarias:

- TestNG – Framework para pruebas en Java basado en JUnit (para Java) y NUnit (para .NET).
- Spring Test – Módulo parte de Spring Framework para realización de pruebas. Tiene soporte para pruebas TestNG y JUnit.

Integración de TestNG y Spring Framework:

Antes de explicar cómo integrar estas herramientas se debe explicar que son los perfiles (profiles) en Spring y su importancia durante las pruebas.

Perfiles de Spring:

Spring es capaz de crear automáticamente objetos Java a partir de clases de configuración y crea los mismos usando métodos anotados con @Bean. Pero que sucede si se desea que Spring cree algunos objetos de manera diferente según el contexto en el que Spring se ejecuta para esto existen los perfiles. Para asociar un objeto a un perfil solo hay que anotarlo con @Profile pasando cómo parámetro una cadena de caracteres que indica el nombre del perfil en el cual se creará este objeto. Los @Bean no anotados con @Profile se crearán siempre sin importar que perfil esté activo.

La principal importancia de los perfiles para las pruebas es que permite crear objetos que dependen de situaciones imposibles o difíciles de recrear en una prueba unitaria. En este caso el método anotado con @Bean de tipo DataSource necesita ser creado a partir de un archivo de configuración que indica la URL de la base de datos, su usuario y contraseña. Este archivo no existe durante la prueba ya que se necesita una previa configuración del servicio web y crear de esta manera el DataSource no es una opción. Para nuestras pruebas se conocen estos datos de antemano por lo que se puede crear el DataSource usando datos estáticos. Para esto se crea dos métodos que retornan un objeto DataSource pero asociados a dos perfiles diferentes, uno asociado a la producción “prod” y uno asociado a las pruebas “test” (Ver Ilustración 28).

```
@Profile("test")
@Bean
public PGSimpleDataSource dataSourceTest() {
    PGSimpleDataSource dataSource = new PGSimpleDataSource();
    dataSource.setUrl("jdbc:postgresql://localhost:5432/energuxgps");
    dataSource.setUser("postgres");
    dataSource.setPassword("admin");
    return dataSource;
}

@Profile("prod")
@Bean
public PGSimpleDataSource dataSource(GeneralSettings settings) {
    PGSimpleDataSource dataSource = new PGSimpleDataSource();
    settings.getDbSettings().updateDataSource(dataSource);
    return dataSource;
}
```

ILUSTRACIÓN 28. PERFILES DE SPRING

Creación de una prueba:

Para crear una prueba TestNG compatible con Spring se debe crear una clase que extienda de la clase AbstractTestNGSpringContextTests del módulo Spring Test. A su vez se debe anotar la clase con @ContextConfiguration la cual indica la clase de

configuración que se debe usar en la prueba, la anotación @WebAppConfiguration para indicar que se debe crear un contexto web falso para simular la tarea, esto permite que la prueba no necesite de un servidor web para ejecutarse. Finalmente se puede o no especificar qué perfil estará activo durante la ejecución de la prueba mediante @ActiveProfiles (Ver Ilustración 29).

```
@ContextConfiguration(classes = WebConfig.class)
@WebAppConfiguration
@ActiveProfiles("test")
public class UpdateTest extends AbstractTestNGSpringContextTests {
```

ILUSTRACIÓN 29. CONFIGURACIÓN DE TESTNG EN SPRING

Luego de un análisis de los principales componentes de sistema se determinó realizar cuatro pruebas (Ver Tabla 12).

TABLA 12. LISTADO DE PRUEBAS UNITARIAS

Prueba	Objetivo	Resultado esperado
Controladores MVC	Probar las clases controladoras	Dada una URL el controlador devuelva la vista adecuada
Inserción y recuperación de la base de datos	Probar objetos DAO que trabajan sobre el marco de trabajo Hibernate	Poder insertar un objeto Java a una base de datos relacional y recuperar el mismo
Recuperación de datos del servicio MovilWeb	Probar objetos DAO que trabajan sobre un servicio web de tipo REST	Poder recuperar a partir de una URL un listado de los vehículos de una empresa
Correcto guardado de la configuración	Probar el paquete manejador de preferencias	Crear una preferencia guardar datos en ella y luego recuperarlos correctamente

Controladores MVC:

Esta prueba se realiza comúnmente para comprobar que el mecanismo MVC de Spring funcione correctamente. Para esta prueba se necesita probar tres controladores y a su vez cada uno puede tener más de una solicitud URL por lo que utilizó una función avanzada de TestNG los proveedores de datos (data providers). Los proveedores de datos sirven para realizar pruebas que piden parámetros de entradas permitiendo realizar pruebas más generales y simplificar el código (Ver Ilustración 30).

```
@DataProvider(name = "controllerData")
public Object[][] controllersData() {
    MainController mainController =
            new MainController(updater, dataSource);
    DataController dataController =
            new DataController(hojaRutaDAO, empresaDAO, dataSource);
    ConfigController configController =
            new ConfigController(simpleClientHttp, scheduler, dataSource, generalSettings);
    return new Object[][]{
            {mainController, "/", "home"},
            {mainController, "/index/", "home"},
            {mainController, "/home/", "home"},
            {dataController, "/report/", "report"},
            {configController, "/configuration/", "configuration"}
    };
}
```

ILUSTRACIÓN 30. PROVEEDOR DE DATOS PARA LA PRUEBA DE CONTROLADORES MVC

La implementación de la prueba de controladores se puede ver en la Ilustración 31. El módulo Spring Test proporciona múltiples objetos falsos (mock objects) que sirven para emular el funcionamiento de un servicio web sin necesidad de uno real. En este caso se crea un objeto falso MockMvc en el cual se realiza una petición de tipo GET a la url utilizando MockMvcRequestBuilders.

```
@Test(dataProvider = "controllerData")
public void testControllers(Object controller, String url, String expectedViewName) throws Exception {
    MockMvc mockMvc = standaloneSetup(controller).build();
    mockMvc.perform(MockMvcRequestBuilders.get(url))
            .andExpect(new ViewNameMatcher(expectedViewName));
}
```

ILUSTRACIÓN 31. PRUEBA DE CONTROLADORES MVC

La Ilustración 32 muestra el resultado de esta prueba.

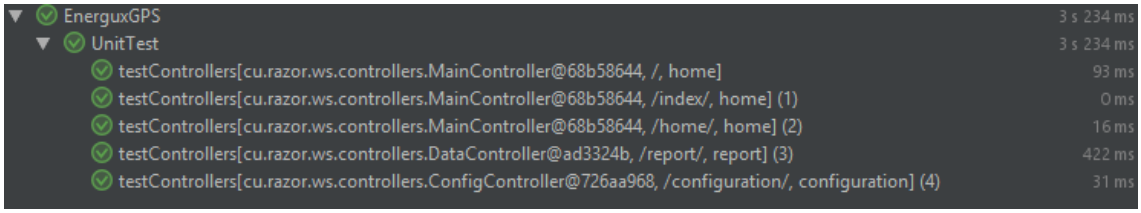

ILUSTRACIÓN 32. RESULTADOS DE PRUEBA DE CONTROLADORES MVC

Inserción y recuperación de la base de datos:

Para esta prueba se realiza la inserción y recuperación de varios objetos del mismo tipo y se comprueba el identificador insertado es igual al recuperado. Para generar los identificadores se recurre nuevamente a un proveedor de datos (Ver Ilustración 33). La prueba comienza por borrar todos los registros de vehículos de la base de datos. Crea

un objeto Vehiculo a partir de la chapa pasada por parámetro e inserta el mismo. Luego recupera de la base de datos un vehículo con la misma chapa y comprueba que no sea null y que el vehículo devuelto no coincida con el insertado (Ver Ilustración 34).

```
@DataProvider(name = "chapaGen")
public Object[][] chapaGenerator() {
    return new Object[][]{
            {"G52B89"},
            {"893B89"},
            {"GYDN89"},
            {"23B896"},
            {"78WPL3"},
            {"ZX52B8"},
            ["Q02X89"],
            {"8O4L3V"}
    };
}
```

ILUSTRACIÓN 33. PROVEEDOR DE DATOS PARA LA PRUEBA DE INSERCIÓN Y RECUPERACIÓN

```
@Test(dataProvider = "chapaGen")
public void testInsertAndQuery(String chapa) {
    Transaction transaction = sessionFactory.getCurrentSession().beginTransaction();
    sessionFactory.getCurrentSession().createQuery( s: "DELETE FROM nom_vehiculo ").executeUpdate();
    transaction.commit();
    Vehiculo vehiculo = new Vehiculo();
    vehiculo.chapa = vehiculo.descChapa = chapa;
    vehiculo.empresa = empresaDAO.getEmpresa();
    vehiculoDAO.saveVehiculo(vehiculo);
    Vehiculo byChapa = vehiculoDAO.getByChapa(chapa);
    if (byChapa == null || !byChapa.chapa.equals(chapa)) {
        Assert.fail("Can't find inserted vehicle with plate " + chapa);
    }
    transaction = sessionFactory.getCurrentSession().beginTransaction();
    sessionFactory.getCurrentSession().createQuery( s: "DELETE FROM nom_vehiculo ").executeUpdate();
    transaction.commit();
}
```

ILUSTRACIÓN 34. PRUEBA DE INSERCIÓN Y RECUPERACIÓN

La Ilustración 35 muestra el resultado de esta prueba.

ILUSTRACIÓN 35. RESULTADOS DE LA PRUEBA DE INSERCIÓN Y RECUPERACIÓN

Recuperación de datos del servicio MovilWeb:

Esta prueba utiliza un objeto de acceso a los datos para recuperar una lista de vehículos desde un servidor web basado en REST. El mismo solo comprueba que la lista resultante contenga al menos un vehículo (Ver Ilustración 36).

```
@Test
public void testGPSDataSource() {
    if (gpsDataSource.getVehiculosFromURL(
            env.getProperty("urls.listadoMoviles"), empresaDAO.getEmpresa()).isEmpty()) {
        Assert.fail("GPS Data source failed to retrieve vehicles");
    }
}
```

ILUSTRACIÓN 36. PRUEBA DE ACCESO A SERVICIO REST

La Ilustración 37 muestra el resultado de esta prueba.

ILUSTRACIÓN 37. RESULTADO DE PRUEBA DE ACCESO A SERVICIO REST

Correcto guardado de la configuración:

Para el guardado de la configuración se utiliza un objeto Preference el cual se encarga de guardar en un archivo binario datos en forma clave-valor. Para comprobar el mismo se crea una configuración nueva se le adicionan datos y se guarda en un hilo independiente. Debido a esto luego de llama a PreferenceManager.shutdownSafe() el cuál bloquea el hilo hasta que se complete el guardado siempre y cuando el proceso de guardado no demore más de un minuto (timeout), en cuyo caso se detiene el hilo y la prueba falla. Finalmente, se carga la configuración nuevamente y se comprueba la presencia de los datos esperados (Ver Ilustración 38).

```
@Test
public void testPreferenceManager() {
    Preference testPref = PreferenceManager.getPreference( pref_name: "test.config");
    testPref.edit()
            .putString("test.string", "test")
            .putFloat("test.float", .5f)
            .apply();
    PreferenceManager.shutdownSafe();
    testPref = PreferenceManager.getPreference( pref_name: "test.config");
    Bundle data = testPref.getData();
    try {
        if (!data.getString( key: "test.string").equals("test") || data.getFloat( key: "test.float") != .5f) {
            Assert.fail("Preference not saved");
        }
    } finally {
        if (!testPref.clearPref( keepFile: false)) {
            Assert.fail("Preference saved correctly but no deleted.");
        }
        PreferenceManager.shutdownSafe();
    }
}
```

ILUSTRACIÓN 38. PRUEBA DE CONFIGURACIÓN

La Ilustración 39 muestra el resultado de esta prueba.

ILUSTRACIÓN 39. RESULTADOS DE PRUEBA DE CONFIGURACIÓN

Conclusiones de las pruebas unitarias:

Las pruebas unitarias son una muestra del correcto funcionamiento de las principales funcionalidades de un sistema. Las pruebas realizadas fueron completadas exitosamente para cada uno de sus parámetros.

3.3 Prueba de interfaz

Esta prueba consiste en visualizar la interfaz web en varios tipos de dispositivos en los cuáles el navegador, las dimensiones y resolución de la pantalla sean diferentes. Para determinar un resultado satisfactorio de esta prueba se accederá al servicio web desde diferentes dispositivos los cuáles tienen relaciones de aspecto muy diferentes. Se comprobará que la navegación sea accesible, la información sea legible y que los campos de los formularios se ajusten adecuadamente al tamaño disponible.

Dispositivos a probar:

TABLA 13. DISPOSITIVO DE PRUEBA 1

Modelo	Xiaomi Redmi Note 4X
Tipo de dispositivo	Teléfono
Sistema operativo	Android 7.0(Nougat)
Navegador	Google Chrome
Resolución de pantalla	1080x1920
Pantalla en diagonal	5.5 pulgadas

TABLA 14. DISPOSITIVO DE PRUEBA 2

Modelo	LO-T1073
Tipo de dispositivo	Tableta
Sistema operativo	Android 4.2.2(Jelly Bean)
Navegador	Google Chrome
Resolución de pantalla	800x480

Pantalla en diagonal	6.7 pulgadas

TABLA 15. DISPOSITIVO DE PRUEBA 3

Modelo	Acer Generic-100 (Monitor)
Tipo de dispositivo	PC
Sistema operativo	Microsoft Windows 10 x64
Navegador	Microsoft Edge
Resolución de pantalla	1366x768
Pantalla en diagonal	21 pulgadas

Debido a que en el dispositivo de pruebas 3 tiene las mismas características que los dispositivos en los cuáles se realizaron los prototipos no se mostrarán imágenes del mismo pues no sufrieron cambio alguno al probarlos.

Página de bienvenida:

En la página de bienvenida se puede comprobar como la barra de navegación pasa a un modo compacto permitiendo visualizar la información en un modo de lista desplegable (Ver Ilustración 40).

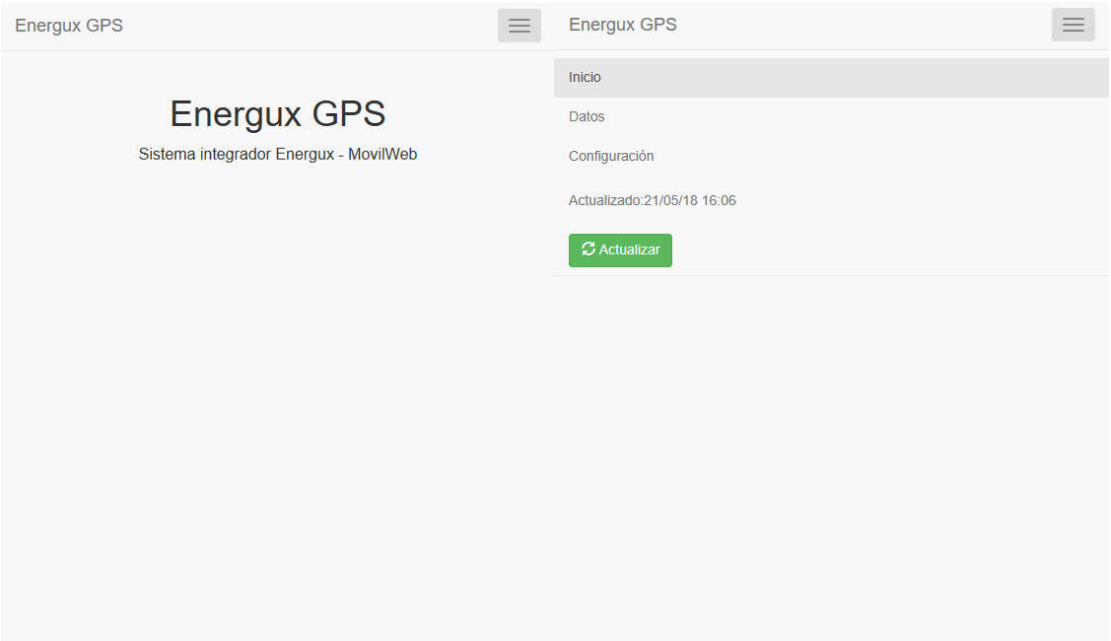

ILUSTRACIÓN 40. PÁGINA DE BIENVENIDA EN TELÉFONO XIAOMI Y BARRA DE NAVEGACIÓN EN MODO COMPACTO

En la tableta (dispositivo 2) la barra de navegación no pasa a modo compacto sólo disminuye su tamaño y ajusta su contenido según el espacio disponible. (Ilustración 41)

ILUSTRACIÓN 41. PÁGINA DE BIENVENIDA EN TABLETA LO-T1073

Página de datos:

En la página de datos el formulario, el botón mostrar y los botones de paginación pasan de modo horizontal a modo vertical debido al poco ancho. La tabla se ajusta al espacio disponible creando celdas multi-líneas para textos largos. (Ver Ilustración 42)

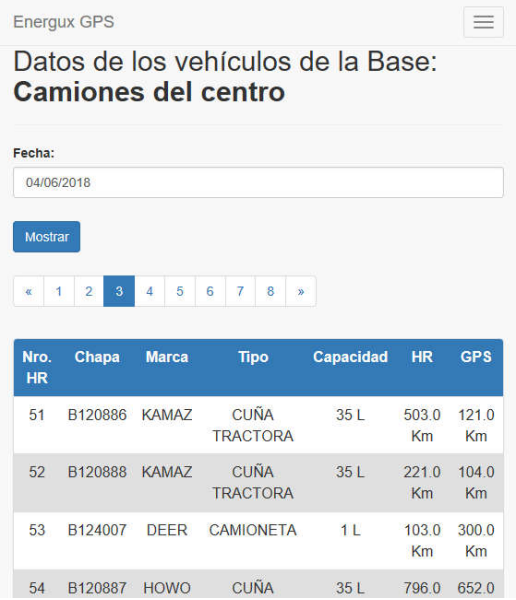

ILUSTRACIÓN 42. PÁGINA DE DATOS EN TELÉFONO XIAOMI

Los componentes en el dispositivo 2 se muestran de la misma manera sólo la barra de navegación pasa su modo estándar. (Ver Ilustración 43)

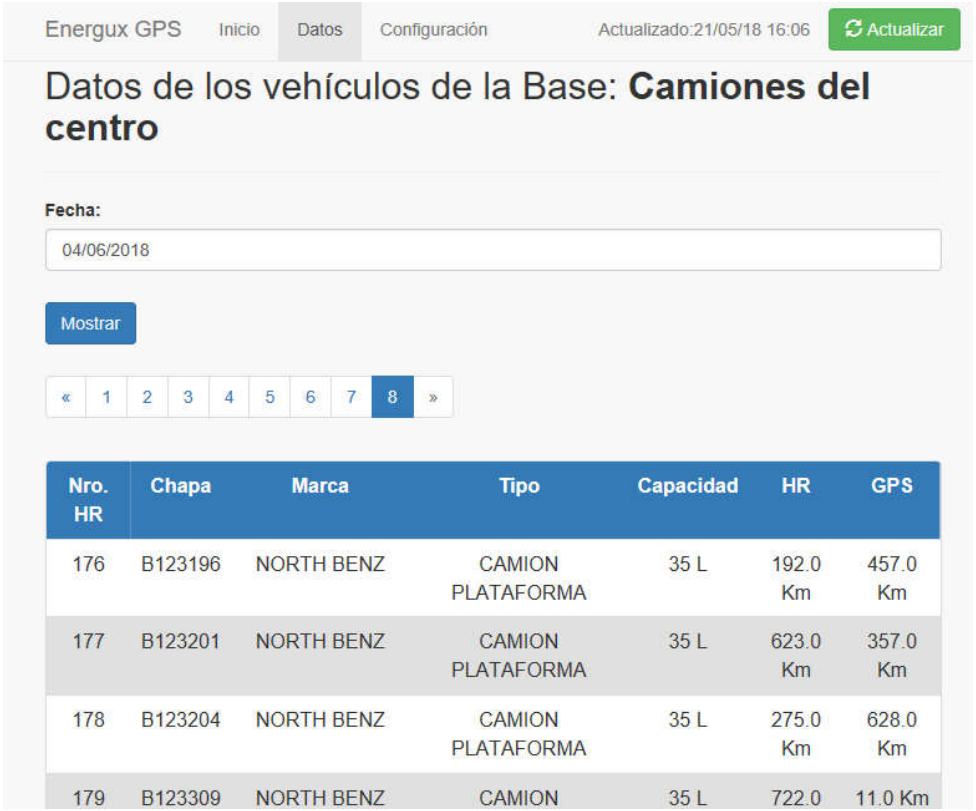

ILUSTRACIÓN 43. PÁGINA DE DATOS EN TABLETA LO-T1073

Página de configuración:

En el dispositivo 1 la configuración se muestra de manera vertical permitiendo que todas las entradas de datos sean perfectamente visibles (Ver Ilustración 44).

ILUSTRACIÓN 44. PÁGINA DE CONFIGURACIÓN EN TELÉFONO XIAOMI

Los componentes en el dispositivo 2 se muestran de la misma manera sólo la barra de navegación pasa su modo estándar (Ver Ilustración 45).

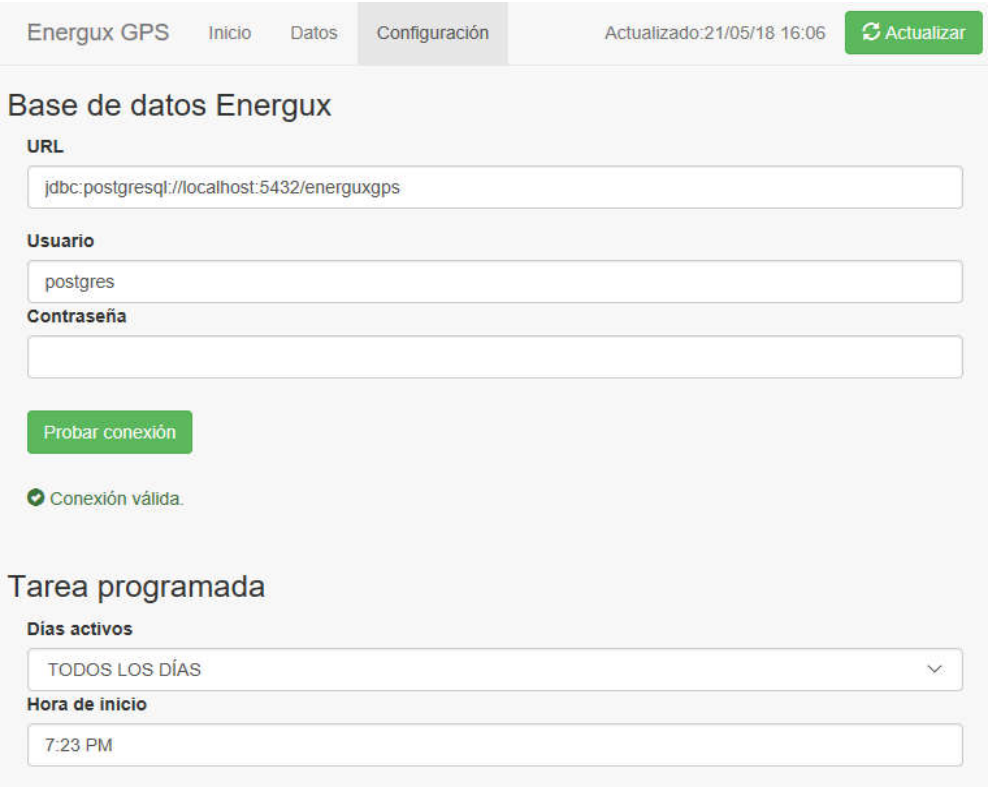

ILUSTRACIÓN 45. PÁGINA DE CONFIGURACIÓN EN TABLETA LO-T1073

Conclusiones del capítulo

1. Se eligió el grado de granularidad óptimo para las necesidades y requerimientos basado en el tiempo de respuesta y la buena utilización de los recursos de la red.
2. Se crearon pruebas unitarias para probar el funcionamiento de las principales funcionalidades del sistema y todas ellas se realizaron satisfactoriamente.
3. Se comprobó el diseño de las páginas en diferentes dispositivos para así tener una buena experiencia sin importar tamaño de pantalla o navegador.

Conclusiones

Como resultado de este trabajo se concluye:

1. Se determinó la granularidad adecuada para las necesidades de la aplicación logrando un buen tiempo de respuesta y un uso eficiente de la red.
2. Se implementó un mecanismo de comunicación basado en REST para la obtención de datos GPS desde el servicio MovilWeb.
3. Se logró persistir la información extraída en la base de datos existente de EnerguX.
4. Se desarrolló un servicio web para dotar a EnerguX con la capacidad de usar datos GPS provenientes de MovilWeb de manera autónoma e independiente para aquellas empresas con ambos servicios.
5. Mediante la realización de las pruebas correspondientes al tipo de aplicación desarrollada y metodología usada se comprobó el correcto funcionamiento de las principales funcionalidades del servicio.

Recomendaciones

1. Crear un sistema de autentificación (basado en los usuarios registrados de EnerguX) para garantizar la protección de la configuración del servicio ante usuarios no autorizados.
2. Modificar EnerguX para que pueda solicitar una actualización manual al servicio desde su propia interfaz.
3. Integrar el sistema implementado como una funcionalidad más en la próxima versión de EnerguX, esto permitirá facilidad de despliegue e integración.

Referencias Bibliográficas

AGÜERO, M. 2007. Introducción a Spring Framework. Universidad de Palermo.

ALBIN, S. 2003. The Art of Software Architecture: Design methods and techniques. *Wiley*.

ALLIANCE, S. 2012. *Scrum: the basics* [Online]. Available: http://www.scrumalliance.org/pages/what_is_scrum [Accessed 2 de febrero 2018].

BAGÜÉS, R. 2009. Proactiva Calidad - Framework Spring.

BARBERO, C. 2014. *Plataformas De Integración. Servicios Web REST y SOAP*.

BECK, K. 1999. Extreme Programming Explained: Embrace Change [1ª ed.]. *Addison Wesley*.

CAMPOS, M. A. 1999. *Guía de Iniciación al Lenguaje Java.* Universidad de Burgos.

DESOFT 2015. *Manual de Usuario: EnerguX*.

F. ALSHAHWAN, M. M., & CARREZ 2010. *Evaluation of Distributed SOAP and RESTful MobileWeb Services. International Journal on Advances in Networks and Services.*

FERNÁNDEZ, G. G. S. R. C. I. J. L. C. 2011. MovilWeb: Aplicación para el control de flotas basada en PostgreSQL. *Revista Cubana de Ciencias Informáticas,* 5**,** 1-12.

FIELDING, R. 2000. *Architectural Styles and the Design of Network-based Software Architectures.* Doctoral dissertation, University of California, Irvine.

FOWLER, M. 2005. *The new methodology* [Online]. Available: http://martinfowler.com/articles/newMethodology.html [Accessed 7 de enero 2018].

FROUFE, A. 1997. *Tutorial de Java* [Online]. Universidad de las Palmas de gran Canarias Available: http://www.ulpgc.es/otros/tutoriales/java/Intro/tabla.html [Accessed 5 de diciembre 2017].

GHOSH, S. 2012. *Systemic comparison of the application of EVM in traditional and agile software project.*

H. KHURANA, J. S. S. 2011. Agile: The necessitate of contemporary software developers. *International Journal of Engineering Science & Technology*.

HAFIZ, P. A. P. H. S. R. E. J. M. 2011. REST and Web Services: In Theory and Practice. *REST: From Research to Practice.*

J. SNELL, D. T., & KULCHENKO 2010. *Programming Web Services with SOAP*, O'Reilly Media, Inc.

LEYMANN, C. P. O. Z. F. 2008. RESTful Web Services vs. "Big" Web Services: Making the Right Architectural Decision. *WWW '08: Proceeding of the 17th international conference on World Wide Web.* New York, NY, USA: ACM.

MARTIN, R. M. M. 2006. Agile principles, paterns, and practices in C#. *Prentice Hall*.

NONAKA, H. T. I. 1986. The new product development game. *Harvard Business Review***,** 137-146.

ORSÁG, J. 2006. *OBJECT-RELATIONAL MAPPING.* COMENIUS UNIVERSITY.

PÉREZ, A. 2009. *Aplicaciones Web con Java* [Online]. Available: http://www.monografias.com//trabajos71/aplicaciones-web-java/aplicacionesweb-java.shtml [Accessed 24 de enero 2018].

RIZOS, C. 1999. GPS Stellite Signals. *University of New South Wales*.

RODRÍGUEZ, L. A. P. 2010. *EnerguX Control de Portadores Energéticos.* Máster en Computación Aplicada, Universidad Central "Marta Abreu" de Las Villas.

SUTHERLAND, K. S. J. 2011. *The Scrum guide* [Online]. Available: http://www.scrumguides.org/ [Accessed 21 de febrero 2018].

W3C, W. 2018. *Web Services* [Online]. Available: http://www.w3.org/2002/ws/ [Accessed 5 de enero 2018].

WIKIPEDIA. 2018. *Hibernate* [Online]. Available: http://es.wikipedia.org/wiki/Hibernate [Accessed 2 de abril 2018].

WIKIPEDIA. 2018. *Sistema de posicionamiento global* [Online]. Available: http://es.wikipedia.org/wiki/Sistema_de_posicionamiento_global.html [Accessed 4 de abril 2018].