UNIVERSIDAD SIMÓN BOLÍVAR
DECANATO DE ESTUDIOS PROFESIONALES
COORDINACIÓN DE INGENIERÍA DE LA COMPUTACIÓN

DISEÑO Y DESARROLLO DE UN PROTOTIPO DE CORE BANCARIO UTILIZANDO TECNOLOGÍA WEB

Por:
JON ANDER RICCHIUTI URRESTI

Realizado con la asesoría de:
Tutor Académico: PROF. LEONID TINEO
Tutor Industrial: ING. ALEJANDRO HERNÁNDEZ

INFORME DE PASANTÍA LARGA
Presentado ante la Ilustre Universidad Simón Bolívar
como requisito parcial para optar al título de
Ingeniero en Computación

Sartenejas, FEBRERO de 2014

Resumen

Los sistemas bancarios se componen por complejos subsistemas que realizan labores complementarias y sustentan el negocio bancario. En su centro se encuentra un sistema conocido como Core Bancario. El Core realiza las tareas vitales para las entidades financieras, tiene información de cuentas y clientes, así como también las transacciones y el dinero que está representado de manera virtual. Tradicionalmente los sistemas de Core son desarrollados sobre sistemas de tipo mainframe, en lenguajes cercanos a la arquitectura del computador, esto para tratar de aprovechar al máximo los recursos del sistema.

Los sistemas bancarios fueron diseñados principalmente para resolver necesidades planteadas hace cincuenta años. Con el fuerte desarrollo de la tecnología en los últimos años, las entidades financieras se han visto en la necesidad de adaptar sus sistemas para satisfacer las necesidades del mercado. Sin embargo, esto lo han hecho agregando capas de software que sirven de interfaz de comunicación entre las viejas y las nuevas tecnologías. Por lo tanto esto le ha agregado una demora y un nivel de complejidad mucho mayor a un sistema bancario.

Con la llegada de la tecnología web, las sedes físicas de las instituciones financieras han pasado a un segundo plano y cada día es más común ver a los clientes realizar las operaciones principales que ofrece un banco, a través de sus celulares y de dispositivos conectados al internet.

Este trabajo consiste en desarrollar un prototipo que permita explorar la posibilidad de realizar un Core Bancario utilizando las herramientas más innovadoras en el mundo web. Estas nuevas tecnologías aportan características que pueden ser muy bien apreciadas por el negocio bancario, características tales como: replicas y distribución de datos, escalabilidad dinámica del sistema, interacción con casi cualquier dispositivo que tenga capacidad de conexión al internet, entre otras. De esta manera se busca absorber las características que aportan las nuevas tecnologías y de esta manera construir un modelo base para la banca del futuro.

A nuestros padres.

Porque nos dieron la vida y nos han guiado a ser quienes somos hoy.

Agradecimientos

...

Índice general

Índice de cuadros

Índice de figuras

Introducción

Un banco es una empresa financiera que se encarga de captar recursos y prestar dinero. Los sistemas bancarios son un conjunto de entidades financieras que se encargan de ofrecer el servicio de banco. La gran necesidad de los sistemas bancarios en la sociedad actual, ha hecho que las entidades financieras tengan que manejar una gran cantidad de datos, y por ello, han recurrido a sistemas computarizados que se encarguen del procesamiento de los mismos. El Core Bancario es el sistema que realiza las actividades principales para el negocio bancario y por lo tanto es el centro de toda entidad financiera.

Los sistemas bancarios fueron desarrollados para responder a las necesidades financieras que existían hace unas cuantas décadas atrás. Sin embargo, con la evolución de la tecnología, todo el paradigma utilizado se vuelve obsoleto. La empresa Synergy-GB, es una empresa muy cercana al negocio bancario y ha visto la posibilidad de cambiar la tecnología obsoleta que se utiliza actualmente en la banca venezolana por una tecnología mejor adaptada a nuestros tiempos.

Actualmente casi todo dispositivo tecnológico es capaz de conectarse al internet, entonces teniendo en cuenta este hecho, la empresa Synergy-GB ha visto la oportunidad de crear un banco diferente a los conocidos hoy en día. Esto es, un banco pensado para ofrecer sus servicios utilizando tecnologías web y que no se requiera sumas de dinero absurdas para lograrlo. Un banco hecho a la medida de la web, le permitiría a los clientes realizar operaciones con cuentas en sus bancos de una manera mas directa. Esto es, comunicarse con el Core Bancario

sin tener capas intermedias que traduzcan del lenguaje web al lenguaje Core. Al no tener que pasar por un sin número de capas intermediarias que que realicen la traducción entre los lenguajes mencionados anteriormente, se ahorraría tiempo y de esta forma se le brindaría una experiencia mucho más grata al usuario final.

Lo anterior sienta las bases que hacen posible la realización de este proyecto, el cual tiene como meta: "Diseñar y desarrollar un prototipo de Core Bancario utilizando tecnologías web".

Los objetivos específicos que se seguirán para cumplir con la meta planteada son los siguientes:

- Investigar los estándares y regulaciones de la industria bancaria.
- Diseñar el tipo de operaciones e interfaces del sistema, siguiendo estándares de la industria.
- Deseñar una arquitectura para que el sistema sea de altas prestaciones, escalable, fácil de conectar, y auditable.
- Desarrollar el sistema utilizando tecnología web.
- Estudiar el rendimiento del sistema comparándolo con los estándares del mercado.

Capítulo 1

Entorno Empresarial

El presente artículo describe el ambiente donde se desarrolló el proyecto de prototipo de Core Bancario. A continuación se presentan aspectos fundamentales de la empresa, con la intención de hacer entender el interés de la misma en desarrollar un proyecto dirigido hacia la banca.

1.1. Descripción de la empresa

"Somos una empresa proveedora de soluciones informáticas y desarrollo en Movilidad, Web 2.0 y consultoría en BPM, BI e Integración Corporativa, con experiencia en el desarrollo de proyectos de gran envergadura, que desde el 2007 ofrece una sólida base de conocimientos en estrategia tecnológica, soportada en el uso de herramientas innovadoras de última generación." [SGB]

También, "SYNERGY-GB es un conglomerado de empresas que nace en el año 2007, que forma parte del Grupo Corporativo SYNERGY-GB Corporation, mismo que cuenta con más de 6 años de experiencia en el desarrollo, comercialización e Integración de Aplicaciones

Corporativas enfocadas en manejar toda la información de negocio que requiere la empresa y su trabajador." [SGB]

En pocas palabras, Synergy-GB es una empresa innovadora que ofrece soluciones tecnológicas que se adaptan a modelos de negocios principalmente corporativos. La empresa se ha posicionado en el mercado como una de las principales proveedoras en las ramas de banca móvil y banca web.

1.2. Misión y Visión

En toda empresa existen pilares fundamentales y son estos los que la define. En entre los aspectos fundamentales de Synergy-GB están la misión y la visión.

Su misión es "Crear, desarrollar y apoyar modelos de negocio que mejoren la competitividad y productividad de nuestros clientes". [SGB]

Su visión es "Convertir a Synergy-GB en una organización de alcance hispanoamericano, que combine en forma creativa ideas, talento, tecnología, visión empresarial, innovación y excelencia en el servicio". [SGB]

La empresa se organiza en una arquitectura bien definida y se apoya en una serie de valores que le permitirán trabajar conforme a su visión y misión.

1.3. Arquitectura y Valores

La arquitectura de la empresa se divide según el tipo de soluciones que presta. Estas soluciones se van de la mano con las necesidades del cliente y su correspondiente alcance. Principalmente Synergy-GB ofrece soluciones en cuatro áreas fundamentales, estas son:

- Experiencia del usuario

Esta área se divide principalmente en Aplicaciones de Movilidad y Desarollo de Portales 2.0.

- Aplicaciones de Movilidad

  Este dominio se basa en el "desarrollo y comercialización de movilidad para el mercado masivo y el mercado corporativo". [SGB]

- Desarrollo de Portales 2.0

  Define soluciones que se adapten a las necesidades del cliente sobre Web 2.0

- Consultoría

  El área de consultoría de la empresa trabaja sobre tres dominios. Estos servicios son Integración SOA (Service Oriented Architecture), Consultoría BPM (Business Process Management) e Inteligencia de Negocios.

  - Integración SOA

    Se encarga de "Desarrollo en servicio de integración, desde arquitectura hasta desarrollo de servicios SOA (SOAP y REST)". [SGB]

  - Consultoría BPM

    Se encarga de "Desarrollo de servicios de consultoría BPM con una arquitectura consistente a la tecnología de integración postulada por la empresa". [SGB]

  - Inteligencia de Negocio

    Se encarga de "Desarrollo de servicios de Base de Datos a nivel de Data Mining, Machine Learning e IA y el Desarrollo de aplicativos de juegos de nivel corporativo". [SGB]

- Integración Móvil Corporativa

Este dominio se encarga de acoplar las soluciones móviles a los servicios que manejan la lógica del negocio. "Definición de arquitecturas de integración SOA (Service Oriented Architecture), desarrollo de servicios web REST y SOAP". [SGB]

- Inteligencia de Negocio

  Ofrece soluciones con respecto al manejo de grandes cantidades de datos. "Depuración, integración y análisis de datos. Minería de Datos. Machine Learning y esquemas de Inteligencia Artificial". [SGB]

Para ofrecer los mejores productos, los valores que definen a Synergy-GB como empresa son:

- Compromiso con la Calidad.
- Compromiso con la Satisfacción al Cliente.
- Compromiso con la Generación de un Legado.
- Sentido de Propiedad.
- Sentido Emprendedor.
- Integridad y Honestidad.
- Orientación a Resultados.
- Compromiso con la Innovación y el Desarrollo de nuevas ideas.
- Proactividad.
- Empowerment.
- Trabajo en Equipo.

- Socialmente Responsables.
- Comercialmente Astutos: Nosotros incentivamos el sentido empresarial y la innovación y premiamos al equipo por desarrollar competencias orientadas a desarrollar relaciones a largo plazo con clientes.
- Diversión como elemento catalizador: Tomamos nuestra vocación muy en serio, pero nos tomamos el tiempo para disfrutar del trabajo, de nuestro tiempo libre, de la compañía de los compañeros, amigos y de nuestras familias.

Los miembros de la empresa deben compartir los valores antes mencionados, de esta forma se desarrollarán productos de calidad y en cada etapa en el proceso de la creación de un producto será desarrollará de manera clara y ordenada.

Capítulo 2

Marco Teórico

En el siguiente capitulo se dan a conocer los conceptos básicos que fueron estudiados para la realización del proyecto. Estos conceptos proporcionan un conocimiento teórico sobre los cuales se basan las tecnologías utilizadas en el mismo. Por esta razón, el estudio de los siguientes conceptos conforma una parte muy relevante en cuanto al desarrollo de este proyecto.

2.1. Web 2.0

El concepto de la Web 2.0 no está concretamente definido. "No existe una arquitectura o especificación formal que defina explicitamente la Web 2.0, y no parece que nunca llegue a estarlo". [GHN09] La Web 2.0 se puede ver como una nueva forma de utilizar el Internet y más especificamente el Internet en conjunción con la Web.

Una de las principales diferencias entre la Web 2.0 y la Vieja Web es como involucran al usuario. La Web 2.0 involucra al usuario y permite que este aporte a la Web. Mientras tanto en la Vieja Web el usuario era un espectador que solo consumía información pero

no era capaz de aportar. James Governor en su libro "*Web 2.0 Architectures*" menciona un un articulo publicado por Tim O'Reilly en Septiembre del 2005. Tim en su artículo contrasta aplicaciones de la Vieja Web con su homologo en la Web 2.0. Entre los ejemplos que dio a conocer Tim, se encuentra *Britanica Online* en contraste con *Wikipedia*. *Britanica Online* ofrecía información pero la fuente de la misma provenía de expertos. Mientras tanto, *Wikipedia* ofrecía información que provenía de cualquier usuario, y esta información era verificada posteriormente por una serie de expertos. Como resultado final *Wikipedia* tenia información actualizada en todo momento mientras que su homologo no.

Según James Governor en la Web 2.0 se han identificado algunos de los siguientes patrones que se mencionarán a continuación y de entre los cuales se profundizara sobre algunos posteriormente en este capitulo.

- Service-Oriented Architecture (SOA)
- Software as a Service (SaaS)
- Participation-Collaboration
- Asynchronous Particle Update
- Mashup
- Rich User Experience (RUE)
- The Synchronized Web
- Collaborative Tagging
- Declarative Living and Tag Gardening
- Semantic Web Grounding

- Persistent Rights Management
- Structured Information

2.2. Arquitectura Orientada a Servicios

La Arquitectura Orientada a Servicios (del ingle, SOA), "es un paradigma arquitectónico, es una forma de construir un estructura que responda a las necesidades y capacidades". [GHN09] Una estructura SOA permite utilizar servicios que están totalmente desacoplados para ofrecer una solución. La característica de independencia de servicios le da flexibilidad a un sistema y esta flexibilidad es muy valorada en sistemas que están cambiando constantemente.

2.3. Transferencia de estado representacional

La Transferencia de estado representacional (del ingles, REST) es un estilo arquitectónico para sistemas distribuidos de hipermedia."REST es un estilo híbrido derivado de varios estilos arquitectónicos sobre redes y combinado con restricciones adicionales que definen una interfaz de conexión uniforme". [Fie00]

Las restricciones que debe cumplir un sistema para cumplir con el estilo arquitectónico REST son:

- **Client-Server**

  Esta restricción establece que debe existir una separación bien delimitada en cuanto a las ocupaciones que competen tanto servidores como a los clientes. En el entorno web la aplicación cliente se puede ver como una interfaz (el navegador) y el servidor el sitio de donde se obtiene la información que es mostrada por el cliente. Fielding dice

lo siguiente, “Al separar lo que compete a la interfaz de usuario de lo que compete a el almacenamiento de datos, entonces se mejora la probabilidad de tener interfaces de usuario en diversas plataformas y se mejora la escalabilidad al simplificar los componentes de los servidores”. [Fie00] Esto es así ya que si un servidor no tiene que estar al tanto de la interfaz de usuario o del estado del usuario, entonces los servidores podrían ser mas simples y más escalables. Esta independencia permite que los clientes sean desarrollados en diferentes plataformas.

- **Stateless**

  La información de contexto de un cliente no puede ser almacenada en el servidor entre peticiones. “Cada petición de un cliente a un servidor debe contener toda la información necesaria para que la petición sea entendida, y no se puede utilizar ningún contexto almacenado en el servidor”. [Fie00]

- **Cacheable**

  Como lo explica Fielding en su disertación de doctorado,“Architectural Styles and the Design of Network-based Soft-ware Architectures”, la información de una respuesta a una petición debe ser marcada como cacheable o no. De esta forma si la respuesta es cacheable, entonces para ciertas peticiones del mismo estilo no es necesario realizar una nueva petición, el cliente tiene el derecho de usar la respuesta previa.

- **Uniform Interface**

  “La característica principal que distingue a REST de cualquier otro estilo arquitectónico basado en en redes es su énfasis en una interfaz uniforme entre componentes”. [Fie00] Fielding especifica que para lograr una interfaz uniforme es necesario cumplir estas condiciones:

  - Todo recurso debe estar identificado.

- La manipulación de los recursos debe hacerse bajo representaciones.
- Los mensajes deben ser auto descriptivos.
- La hipermedia debe ser el motor del estado de la aplicación.

- **Layered System**

  El sistema debe estar divido en capas y los diferentes componentes que conforman el sistema deben estar distribuidos en esas capas. De esta forma ningún componente será capaz de ver información que se encuentre en una capa diferente a la cual el pertenece.

2.4. Arquitectura Orientada a Recursos

La Arquitectura Orientada a Recursos (del ingles, ROA) es una arquitectura orientada a recueros. Un recurso es cualquier cosa lo suficientemente importante como para ser referenciada como algo per se. [LS07] "Usualmente un recurso es algo que puede ser almacenado en una computadora y representado como una secuencia de bits". [LS07] Una arquitectura orientada a objetos es importante porque puede servir como base para construir un sistema que cumpla con las características REST.

2.5. Programación Orientada a Eventos

"La programación orientada a eventos es un paradigma de programación en el cual el flujo de trabajo de un programa es determinado por eventos como acciones del usuario". [EDP] Una aplicación orientada a eventos, es una aplicación en la cual las operaciones que se van a realizar no las define el programa. Las operaciones a realizar las define cierta entrada (input) y la aplicación proveer una forma de reaccionar según la entrada recibida.

Las herramientas que proveen soporte para programación orientada a eventos, suelen ofrecer mecanismos como funciones que retornan callbacks. Generalmente en una aplicación orientada a eventos existe un hilo de ejecución principal que se encuentra dedicado a atender eventos, y cuando uno ocurre, es despachado en una función que se ejecutada en un entorno diferente al hilo principal y retorna un callback. El callback es el mecanismo que tiene la función para avisar al hilo principal que el evento ya fue atendido.

2.6. Base de datos relacionales y NOSQL

"Una base de datos es una colección de datos". [RG03] Los datos pertenecientes a esta colección suelen estar relacionados. Para facilitar el manejo de los datos, se crearon los *database management system* (DBMS). "Un DBMS es software diseñado para ayudar en el mantenimiento y utilización de grandes colecciones de datos". [RG03]

Hoy en día existen dos grandes tipos de manejadores de bases de datos, los relacionales y los NOSQL. Los manejadores de base de datos relacionales fueron los pioneros en el área y modelan los datos a almacenar en tablas de relaciones. Los NOSQL son manejadores más recientes y se basan en documentos almacenados por clave. Los manejadores relacionales ofrecen una gran robustez en cuanto al manejo de los datos pero escalan verticalmente. Mientras tanto, los manejadores NOSQL fueron diseñados para escalar horizontalmente, esto hace que sea muy simple manejar datos que están distribuidos (en varias computadoras interconectadas).

Capítulo 3

Marco Tecnológico

Para la elaboración de este proyecto se utilizaron diversas herramientas tecnológicas que lo hicieron posible. En este capitulo se mencionan las herramientas utilizadas y sus principales características.

3.1. Twisted

Twisted es una herramienta de programación orientada a eventos que está desarrollada sobre el lenguaje de programación Python. Esta herramienta puede ser utilizada tanto como un marco de trabajo (framework) o como una simple librería de Python. "Twisted es una plataforma para desarrollar aplicaciones de internet". [Tea13b]

Algunas de las características principales que aporta Twisted serán mencionadas a continuación.

3.1.1. Multiplataforma

Twisted está escrito en Python, esto no solo hace que Twisted adquiera las características de fácil de escribir, de leer y de correr. Como Python es multiplataforma esto hace que Twisted también lo sea, y por lo tanto, cualquier aplicación desarrollada en Twisted es capaz de correr en Linux, Windows, Unix y Mac. [MF13]

3.1.2. Asíncrono

Como se menciono anteriormente, Twisted es orientada a eventos y esto lo logra mediante a su carácter asíncrono. Esto hace que el programador pueda escribir un programa reactivo mientras sigue procesando eventos que provienen de múltiples conexiones de red. Twisted abstrae al desarrollador el tedioso proceso de usar hilos. [MF13]

3.1.3. Flexible y Extensible

Twisted está pensado para la integración, por esto, toda toda las funcionalidades deberían ser accesibles a través de cualquier protocolo. La herramienta cuenta con la capacidad de manejar protocolos de Web, SSl/TLS, RPC, entre otros. Adicional a esto, modificar los protocolos existentes o incluso crear nuevos protocolos, es muy sencillo. [Tea13b]

3.2. Couchbase

Couchbase es un manejador de base de datos orientado a documentos. Esta optimizado para manejar usuarios concurrentes que interactuarán con la información almacenada en la base de datos. Este tipo de base de datos provee un modelo de datos que se adapta fácilmente a cambios y fueron concebidos con la idea de almacenar información de manera distribuida.

Couchbase presenta dos características principales como lo son su flexibilidad y escalabilidad.

3.2.1. Flexibilidad

Para almacenar la información no es requerido un esquema previo que le de estructura a la información. Esta información es almacenada en documentos de tipo JSON y pueden ser usados directamente por las tecnologías web y movil hoy en día. Como la información almacenada en Couchbase no es estructurada, a los documentos se les pueden añadir campos adicionales sin necesidad realizar ningún tipo de cambio. En otras palabras, el formato de la información que será insertada puede cambiar en cualquier momento sin afectar la aplicación.

3.2.2. Escalabilidad

La información relacionada a los usuarios ha adquirido un mayor valor con el paso del tiempo. Hoy en día es muy importante almacenar la mayor cantidad de información posible por la cantidad de análisis y deducciones que se pueden realizar a partir de esta información. Para lograr almacenar tanta información existen dos tendencias. Uno de ellos es crecer verticalmente, esto implica centralización de la información y por lo tanto servidores más y más grandes. Por otro lado está crecer verticalmente, que implica un crecimiento distribuido (descentralizado) es decir, los servidores no tienen que aumentar sus capacidades, para agregar más capacidad es necesario añadir un nuevo servidor.

Entre los dos modelos mencionados anteriormente, Couchbase implementa el modelo que crece horizontalmente.

Este modelo aporta las siguientes características:

- Distribución automática de la información entre los servidores.

- Soporte de consultas distribuidas.
- Mantiene memoria en caché para dar respuestas más rápidas.
- Tolerante a fallas.
- Económico.

3.3. PostgreSQL

"PostgresSQL es un sistema manejador de base de datos objeto-relacional (ORDMS)". [Gro] Un ORDMS es un sistema manejador de base de datos que soporta estructuras relacionales o de objetos.

PostgreSQl necesita un modelo de datos estructurado para poder almacenar la información. Debido a esto es necesario suministrarle al manejador la estructura que tendrá la información antes de poder comenzar a trabajar con ella. Como el manejador conoce la estructura que tendrá la información y tiene la certeza que esta no cambiara, entonces se pueden realizar optimizaciones tanto a nivel físico (forma de almacenamiento en disco) como a nivel lógico (software). Estas optimizaciones repercuten directamente en los tiempos de respuestas del manejador.

Unas de las principales características que provee PostgreSQL es que garantiza el cumplimiento de las propiedades ACID, siglas que se derivan de los nombres en ingles para *atomicidad, consistencia, aislamiento y durabilidad*. En el libro *"DATABASE MANAGEMENT SYS-TEMS"* describen estas propiedades como lo siguiente:

- **Atomicidad:** Cada transacción debe ejecutarse como atómica. Esto es, todas las acciones dentro de una transacción se llevan a cabo o ninguna lo hace.

- **Consistencia:** Asegura que al ejecutarse una transacción esta lleva a la base de datos de un estado consistente a otro.
- **Aislamiento:** El resultado de ejecutar una transacción es independiente de las transacciones que se estén ejecutando de manera concurrente. Si bien es cierto que las acciones de diversas transacciones se pueden ejecutar de manera intercalada, el resultado sera igual a que las transacciones se realicen una después de la otra.
- **Durabilidad:** Una vez que el manejador de base de datos informa que una transacción se realizó satisfactoriamente, este efecto debe persistir aun cuando el sistema colapse antes de que estos cambios sean reflejados en disco.

3.4. Elasticsearch

Elasticsearch es una flexible y poderosa herramienta distribuida que permite realizar búsquedas en tiempo real y realizar análisis sobre ellas. Fue diseñada con el fin de ser usada en ambientes distribuidos donde la escalabilidad y confiabilidad son primordiales. Esta herramienta permite realizar busquedas que van mucho más allá de búsquedas sobre texto sencillas. [ELK]

3.5. Logstash

Logsatsh ayuda en el proceso de la obtención de *logs* y otro tipo de eventos basados en fechas que ocurren en cualquier sistema, y permite guardarlos en un lugar para realizar transformaciones adicionales y realizar procesamiento. Logstash se encarga de analizar los *logs* y transformarlos en un formato mas sencillo de leer como lo es el JSON. [ELK]

3.6. Kibana

Kibana es una herramienta Web que permite visualizar la información almacenada en Elasticsearch. La herramienta permite interactuar con la información a través de tableros configurables. Estos tableros se comportan de manera dinámica, se pueden grabar, compartir, exportar y son capaces de mostrar cambios a las consultas sobre Elasticsearch en tiempo real. En otras palabras, se pueden realizar análisis de la información almacenada a través de una interfaz elegante. [ELK]

3.7. NGINX

NGINX es un acelerador de aplicaciones web de alto rendimiento que ayuda a un 37 % de los sitios web más ocupados. NGINX es instalado principalmente entre la capa de red y la capa de aplicación reduciendo la carga de concurrencia procesando cambios de URL, balanceando tráfico HTTP y gestionando políticas de seguridad. [NGI]

3.8. Xen Project

La comunidad del Xen Project elabora un orquestador de máquinas virtuales que hace posible correr varias instancias de un sistema operativo o de diferentes sistemas operativos simultáneamente en una única máquina. El proyecto es de código abierto. Este orquestador de máquinas virtuales utiliza diferentes herramientas (algunas de carácter comercial y otras de código abierto) como base para el producto final. La idea es permitirle al usuario final la utilización de servidores, granjas de servidores, reducir la complejidad y reducir el costo total. [XEN]

Capítulo 4

Marco Metodológico

La realización de este proyecto está dividido esencialmente en tres partes fundamentales. La primera de ellas estuvo dedicada únicamente a la investigación de las nociones fundamentales que rigen al mundo Web de hoy, y al entendimiento de las restricciones inherentes al desarrollo de un software para la industria bancaria. Las dos partes restantes corresponden al diseño y desarrollo del producto. Una vez conociendo las restricciones inherentes al producto a desarrollar y con unas bases teóricas bien fundadas sobre los principios usados en la Web, se procedió a crear un diseño inicial que sirviera de base para poder llevar a cabo la implementación de la solución, en la cual se utilizó una metodología scrum.

Para llevar a cabo la metodología Scrum Alejandro Hernandez participa como el Dueño del Producto, Alexander Ramírez como el Experto y Jon Ricchiutti como el equipo de desarrollo. A continuación se presenta una breve introducción a la metodología ya que según los pasos que ella dictamina se llevaron a cabo los procesos de diseño, implementación, rediseño y adaptación necesarios para la realización del prototipo.

4.1. Scrum

Scrum es un marco de trabajo para elaborar productos complejos. “Es un marco de trabajo en el cual las personas pueden manejar problemas complejos de manera flexible, mientras se entregan productos con el mayor valor posible de manera creativa y productiva”. [Tea13a]

Scrum se caracteriza por ser:

- Ligero
- Simple de entender
- Difícil de dominar

4.1.1. Fundamentos

Scrum se basa en que el conocimiento viene de la experiencia y de las decisiones que se toman basadas en lo que se conoce. Scrum emplea una metodología de trabajo iterativa para de esta forma generar experiencia. Del resultado de cada iteración se obtiene un incremento del producto final, pero también se logran obtener estimaciones más precisas en cuanto a predicción y control de riesgos.

Para lograr generar la experiencia para desarrollar un producto de calidad, ésta metodología concibe los siguientes procesos.

4.1.1.1. Transparencia

Los aspectos del proceso deben ser visibles para todos los participantes en el mismo. Para lograr esto, es necesario definir un lenguaje común y tanto para los que desarrollan un producto como para los que lo aceptan debe existir un concepto claro de lo que es un trabajo terminado.

4.1.1.2. Inspección

Los participes de Scrum deben revisar constantemente los resultados obtenidos luego de terminada una iteración, de esta forma se evitan resultados indeseados. Es importante tener en cuenta que la inspección no debe ser demasiado frecuente porque esto puede entorpecer el desarrollo del producto.

4.1.1.3. Adaptación

Si después de realizada alguna inspección se determina que el aspecto revisado no se encuentra dentro de los estándares aceptados, se debe replantear y ajustar. Los ajustes deben realizarse temprano posible para evitar desvíos en los desarrollos futuros.

4.1.2. Roles

En Scrum existen tres roles principales, el dueño del producto, el equipo de desarrollo y el experto.

4.1.2.1. Dueño del producto

Representa al cliente. Él es el responsable de hacer que el equipo entregue un producto con un máximo valor. El dueño del producto es el encargado de decidir cuales son los elementos más importantes de un producto y los agrega a la lista de tareas a realizar (*backlog*) según su prioridad.

4.1.2.2. Equipo de desarrollo

El equipo de desarrollo es el encargado de realizar el producto. Es el responsable de que al termino de cada sesión de desarrollo (*sprint*) exista un incremento del producto.

Se recomienda que los equipos de desarrollo contengan entre tres y nueve personas. Una característica importante sobre el equipo de desarrollo es que se organiza por cuenta propia.

4.1.2.3. Experto

El experto sobre Scrum es el encargado de eliminar todos los obstáculos e impedimentos que puedan tener los miembros del equipo, de esta forma puedan enfocarse en realizar su trabajo. El experto se encarga que el proceso se lleve a cabo de manera correcta y de hacer que cada miembro del equipo rinda al máximo.

4.1.3. El Sprint

La metodología prescribe una serie de eventos. "Los eventos son utilizados por Scrum para crear regularidad y para minimizar la necesidad de reuniones que se encuentran fuera de las definidas por Scrum" [Tea13a] Todos los eventos definidos en Scrum se deben realizar en un tiempo preestablecido.

El Sprint es un evento que suele durar un mes o menos y en el donde se produce un incremento del producto. La meta de todos Sprint es que a su termino se pueda entregar una parte (preestablecida) del producto, funcional y completa. Para lograr lo anterior un Sprint está compuesto por una serie de eventos como los son:

- Planificación del Sprint

  Es donde se planifican las tareas a realizar durante un Sprint.

- Scrum Diario

  Son reuniones que se llevan a cabo diariamente mientras se está en un Sprint. Los miembros del proyecto se reúnen y comparten información acerca de lo que se realizó el día

anterior y lo que se planifica realizar el presente día. También es una buena oportunidad para compartir los problemas que se han presentado y las soluciones tomadas.

- Revisión del Sprint

  Es la etapa del Sprint donde se manifiesta el proceso de la inspección. Durante este evento se revisa el incremento del producto y se realizan las adaptaciones requeridas.

- Retrospectiva del Sprint

  Ocurre después de la revisión de un Sprint. Es una oportunidad para el equipo de Scrum se examine y se tomen las medidas para generar una mejor dinámica de equipo.

4.1.4. Artefactos

"Los artefactos representan trabajo o valor para agregarle transparencia y oportunidades a la inspección y a la adaptación. Los artefactos definidos por Scrum son especialmente diseñados para maximizar la transparencia de la información clave y de esta forma todos tienen el mismo entendimiento del artefacto" [Tea13a]

4.1.4.1. Product Backlog

Es una lista de tareas ordenadas según su importancia y el valor que le agregan al producto final. El dueño del producto es el encargado de modificar esta lista, bien sea para añadirle un mayor valor al producto final o para realizar cambios requeridos por adaptaciones necesarias.

4.1.4.2. Sprint Backlog

Es un subconjunto de el Product Backlog. Es la lista de tareas en las cuales el equipo de desarrollo se debe enfocar durante un Sprint. "El Srint Backlog hace visible todo el trabajo que el equipo de desarrollo identifica como necesario para alcanzar la meta del Sprint". [Tea13a]

Dentro de un Sprint Backlog se contemplan los siguientes elementos:

- Incremento

  Es la suma de todos los elementos completados del Product Backlog durante un Sprint y de los anteriores.

- Monitoreo del Proceso del Sprint

  Es la suma de los elementos que falta por realizarse del Sprint Backlogy se realiza durante el Scrum Diario. De esta forma se puede llevar un seguimiento sobre lo que falta para cumplir la meta y así el equipo de desarrollo puede manejar su progreso.

Capítulo 5

Diseño

En este capitulo se presentará el proceso de concepción y diseño de un prototipo funcional de Core Bancario. Este proceso se realizó en dos etapas fundamentales, la primera etapa consistió en elaborar una solución general al problema sin tomar en cuenta el detalle. De esta forma se obtiene un modelo del cual partir y poder con el paso del tiempo ir agregándole valor. La segunda etapa de diseño se realizó utilizando la metodología Scrum. Durante el proceso iterativo que la metodología prescribe, se fue desarrollando la aplicación y refinando el modelo inicial de diseño.

En esta sección no se pretende mostrar decisiones en cuanto a tecnologías utilizadas, por el contrario, el proceso de toma de decisiones en cuanto a las tecnologías fue concebido en la etapa de desarrollo, que será presentada en la siguiente sección.

5.1. Concepción

En esta fase toman parte dos etapas fundamentales. La primera de ellas es la etapa de captura de requerimientos sobre los cuales se diseñará la solución y la segunda, es la elaboración de una primera aproximación de la solución. De la solución inicial se obtuvo una

solución final producto del refinamiento.

Al inicio de esta etapa se realizaron reuniones que permitieran recaudar material existente sobre algún producto similar. Una vez con este material a la mano, se procede al análisis y entendimiento del mismo. Posteriormente se realizaron reuniones periódicas con el *dueño del producto* (Alejandro Hernandez) que fue el encargado de dar a conocer los requerimientos principales necesarios para realizar el prototipo. Es importante destacar que durante la realización de las reuniones participaron miembros de la empresa que están muy familiarizados con el ambiente bancario. La información suministrada por los participantes fue clave en lo que respecta al entendimiento del proyecto.

A partir de las reuniones mencionadas anteriormente, se determinaron los requerimientos funcionales y los actores del sistema. Quedó claro que al arquitectura del prototipo no subyace únicamente en la elaboración de un sistema de software. También se debía elaborar una arquitectura de un sistema que pudiera trabajar de manera distribuida y fuera escalable.

5.1.1. La Arquitectura del Sistema

De las diferentes reuniones se capturaron una serie de requerimientos que sugerían el diseño de una arquitectura de sistema en lugar a una arquitectura de software. El sistema debe cumplir con las siguientes características:

- **Escalable**

  Es muy importante que el sistema pueda crecer según van creciendo las necesidades del banco. Incrementar el poder de procesamiento del banco debe ser un proceso ajustable y sencillo.

- **Económico**

  En la actualidad los bancos utilizan supercomputadoras con un gran poder de procesa-

miento pero estas computadoras son muy costosas y los sistemas que corren sobre ellas utilizan tecnologías muy viejas. Con tecnologías Web se desea crear un sistema que brinde las mismas capacidades que tienen los cores bancarios pero que sea económico.

- **Seguro**

  Los canales sobre los cuales se transmite la información deben ser seguros.

- **Alcanzable**

  Se desea realizar un banco moderno que se pueda comunicar con cualquier dispositivo capaz de conectarse a la Web.

- **Auditable**

  Los bancos manejan mucho dinero, por lo tanto, es necesario que toda operación que se realice quede registrada para poder realizar auditorias posteriores. Las auditorias deben ser fáciles de realizar.

Dadas las restricciones mencionadas anteriormente, se diseñó una arquitectura de sistema que cumpliera con los requerimientos, tal como muestra la figura 5.1.A continuación se muestra como se llegó Luego a tal solución, que realmente fue la conjunción de diversas soluciones específicas.

5.1.1.1. Solución Distribuida

Se decidió optar por un sistema distribuido para cumplir con los requerimientos de costos y escalabilidad. Un sistema distribuido es un sistema de software en el cual componentes (nodos) dentro de una red se comunican a través de mensajes para lograr un objetivo en común. Los nodos pertenecientes a la red tienen tareas especificas que cumplir y el objetivo en común es alcanzado cuando se unen los resultados de todas estas tareas.

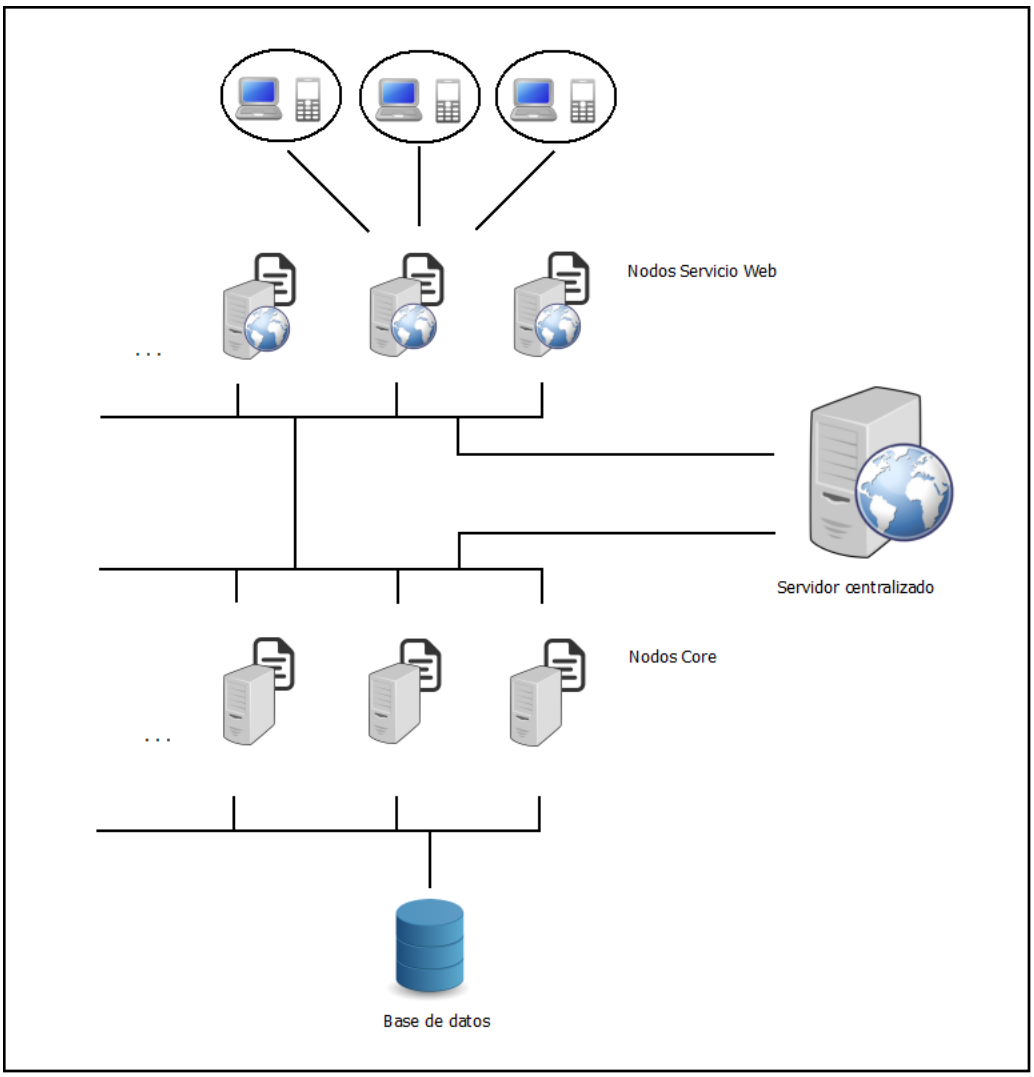

Figura 5.1: Solución general.

Un sistema basado en una arquitectura distribuida ofrece mayor flexibilidad. Un banco pequeño podría comenzar con un sistema de unos pocos nodos e ir añadiéndole según sus necesidades. Entre las principales características que aporta este modelo tenemos:

- Por su flexibilidad y escalabilidad, permite que la empresa crezca según le sea necesario.
- Agregar poder de computo es tan sencillo como encender un nuevo nodo y agregarlo a la red.

- Tener el poder de computo distribuido es más económico por las siguientes razones:
  - Este modelo tiene planteado utilizar computadoras comunes en contraposición a supercomputadoras que tienen unos altos precios en el mercado.
  - Si un supercomputador se ha quedado corto a nivel de computo, es necesario utilizar uno más poderoso para alcanzar los objetivos. Migrar un sistema de un supercomputador a otro requiere una fuerte inversión de dinero. Mientras tanto con un sistema distribuido, no es necesario realizar ninguna migración, basta con agregar la cantidad de nodos necesarios para seguir trabajando normalmente.

En la figura 5.2 se muestra una primera aproximación de diseño para darle solución a los requerimientos de escalabilidad y costos económicos. El esquema muestra básicamente una serie de *nodos core* que se encargaran de realizar todas las operaciones básicas de un banco, como lo son depósitos, transferencias, creación de cuentas, entre otras. Estos nodos también se encargaran de toda la comunicación hacia la base de datos. Por otro lado se encuentran los *nodos de servicios web*, estos se encargan de realizar las verificaciones correspondientes de la información suministrada. Los nodos core y los nodos de servicio web se comunican directamente. La comunicación de agentes externos (aplicaciones web, moviles, entre otras) con el core bancario, se realizará a través de los nodos de servicios web que expondrán una interfaz clara de comunicación al exterior.

5.1.1.2. Base de Datos

En cuanto a la administración de la información se planteaban dos opciones. Una de ellas era utilizar un manejador de base de datos relacional y por otro lado se planteaba utilizar una base de datos nosql. La base de datos nosql aporta escalabilidad ya que estas suelen ser distribuidas y flexibilidad porque no utilizan un modelo de datos rígido. Por otro lado, los

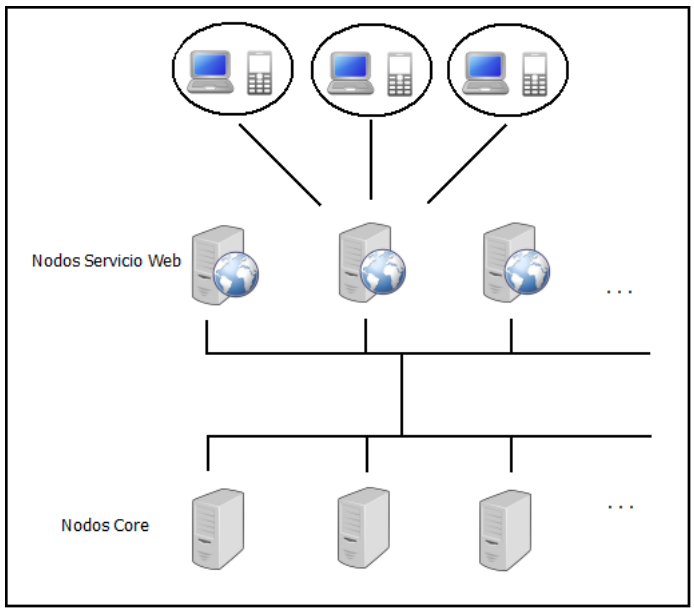

Figura 5.2: Diseño inicial de una solución distribuida.

manejadores relacionales implementan soporte transaccional (cumplen con las propiedades ACID) como fue mencionado en el marco tecnológico (capitulo 3).

Durante el proceso de concepción se decidió que la información almacenada por el banco perduraría y no convenía eliminarla. La información tiene un valor muy preciado y actualmente utilizado BI (Business Intelligence) se pueden realizar una cantidad de funcionalidades que le agreguen valor al producto. Esta decisión dificultaba el uso de de una base de datos relacional ya que ellas al administrar una gran cantidad de información sus tiempos de respuesta empeoran. No era conveniente utilizar una base de datos nosql porque ellas no garantizan las propiedades ACID y en el negocio bancario el manejo del dinero debe ser muy cuidadoso y no se pueden permitir errores como lecturas sucias.

Por estas razones se tomo la decisión de utilizar los dos manejadores de base de datos. Se utilizaría el manejador de base de datos relacional para garantizar la transaccionalidad y solo administraría la información que hace referencia al dinero. Se utilizaría la base de datos nosql para administrar el resto de la información. De esta manera se obtendría la escalabilidad que

proporcionan las bases de datos nosql y el manejo transaccional que aportan los manejadores relacionales. Al incluir la base de datos nosql en el esquema de administración de los datos se libera de una gran carga al manejador relacional y de esta forma los tiempos de respuesta serian mejores.

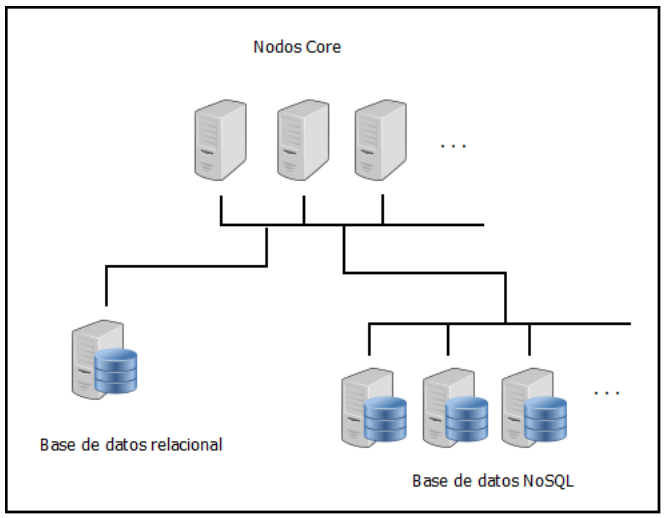

Figura 5.3: Esquema de distribución de datos.

La figura 5.3 muestra una representación de la arquitectura de administración de datos que interactua con el sistema bancario.

5.1.1.3. Seguridad

Unos de los requerimientos principales de cualquier sistema bancario es que sea seguro. Se debe garantizar que la información no será alterada ni vista por usuarios sin los permisos correspondientes. Por esta razón, todos los canales de comunicación existentes entre los nodos del sistema bancario estarán cifrados. Se decidió utilizar el protocolo TLS. Este protocolo utiliza criptografía asimétrica para negociar el intercambio de claves necesarias para utilizar posteriormente criptografía convencional (simétrica). Es importante mencionar que para realizar el intercambio mencionado anteriormente se utilizan certificados X.509 y existe todo un proceso para obtener un certificado avalado por alguna autoridad. Cuando se habla de

certificados X.509 no se puede dejar de mencionar a las autoridades certificadoras. Estas autoridades son entidades que emiten certificados digitales. Esto permite a otras dos entidades que quieren comunicarse confiar en las credenciales mostradas, si estas han sido firmadas por alguna autoridad certificadora de confianza en común. De esta manera se puede realizar la comunicación segura entre dos entidades desconocidas si las dos confían en alguna autoridad certificadora reconocida.

Para el sistema se toma como autoridad certificadora a la entidad bancaria en si. Los actores que quieren intercambiar información son los *nodos core* y los *nodos de servicio web*. Por esta razón un nodo core se comunica con un nodo de servicio web si y solo si cada estos nodos poseen un certificado digital firmado por la entidad bancaria. Por lo tanto, en este esquema existen dos tipos de certificados, los que pertenecen a los nodos de servicio web y los que pertenecen a los core.

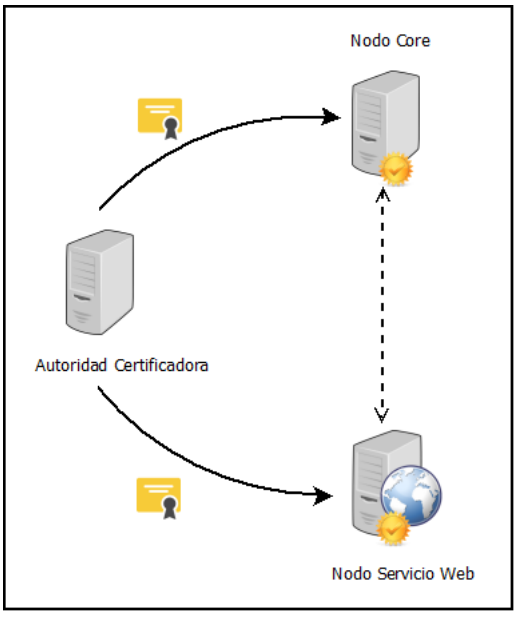

Figura 5.4: Conexión entre dos nodos que tienen certificados firmados por la autoridad.

La figura 5.4 Muestra a un nodo cliente (nodo de servicio web) y uno servidor (nodo core) comunicándose. Cada uno utiliza un certificado firmado por la autoridad certificadora (la

entidad bancaria). Es importante aclarar que la comunicación no se puede realizar utilizando certificados auto firmados (firmados por el mismo nodo) y tampoco si alguno de los nodos carece de certificado.

5.1.1.4. Alcance

Para asegurar que el sistema sea alcanzable es necesario tener en cuenta que dispositivos utilizarán el mismo. Se desea que el sistema esté orientado a brindar comodidad a los clientes de la entidad financiera mientras realizan operaciones bancarias. Hoy en día esta comodidad se ve reflejada mediante el uso de dispositivos telefonicos, tabletas, computadoras personales, entre otros. Todos estos dispositivos son capaces de manejar diferentes tecnologías pero todos coinciden en el soporte para la tecnología web. Por esta razón se decidió que el protocolo http será usado en las comunicaciones entre la interfaz del banco y las aplicaciones que la utilizan. De esta forma se garantiza que cualquier dispositivo moderno sea capaz de interactuar con el banco.

5.1.1.5. Auditabilidad

Uno de los requerimientos más importantes a cumplir es el que compete a toda operación realizada por el sistema. Estas operaciones deben poder ser verificadas en cualquier momento para detectar anomalías. Si el sistema actúa de manera errada por alguna razón esto garantiza que el error puede ser rastreado y solucionado rápidamente.

Debido que el sistema bancario está compuesto por una red de nodos, es necesario que cada uno de ellos mantenga registrada la información particular que maneja. Sin embargo resulta inconveniente tener que verificar cada nodo para encontrar la información que se desea. No resulta práctico tener la información registrada pero que el proceso para obtenerla sea complicado. Es de considerar que las auditorias que se realicen sobre el sistema las hará en su mayoría un ser humano. Debido a que ese proceso suele ser tedioso, es de intención

que el sistema bancario brinde al usuario la mayor comodidad al momento de presentar la información. Por esto se decidió que que la información aún cuando puede ser registrada locamente en cada nodo perteneciente al sistema, será almacenada también en un servidor dedicado a manejar esta información y presentarla cómodamente.

La figura 5.5 muestra como cada nodo perteneciente al sistema bancario está en la obligación de registrar las operaciones que realizan. Luego esta información es enviada a un servidor centralizado que se encargará de facilitarla vía web.

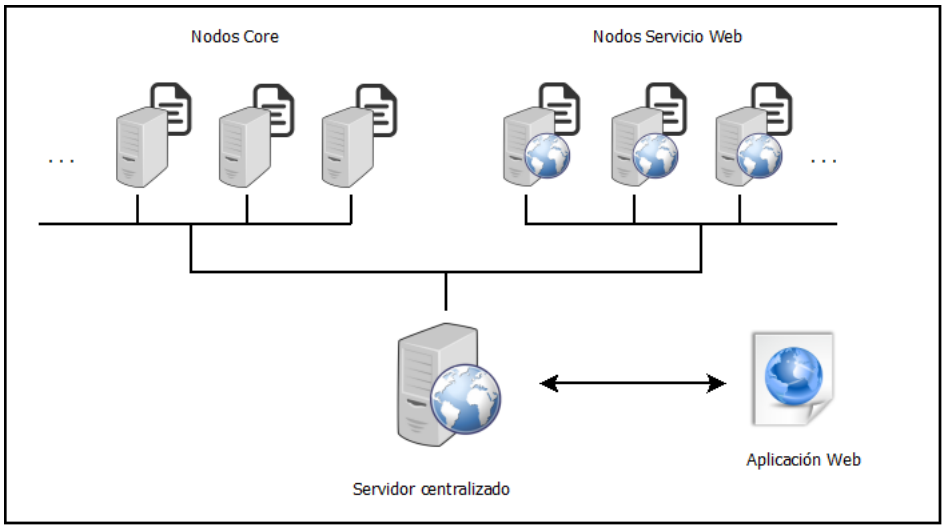

Figura 5.5: Arquitectura del sistema de logs.

5.1.2. La Arquitectura de Software

Esta etapa contempla el diseño de la estructura de datos sobre la cual se basará todo el sistema y las funcionalidades que serán expuestas. Esta estructura funcionará en la arquitectura del sistema que fue mencionada anteriormente. Se desea destacar que las funcionalidades resultantes de este proceso de diseño se pretende sean los mas simples y atómicas posibles. El sistema no pretende ser algo complejo, la simplicidad del mismo radica en que no está concebido para ser utilizado por seres humanos (como usuarios finales). El sistema pretende

ser utilizado por aplicaciones desarrolladas para y por los seres humanos. La simplicidad y atomicidad de las funcionalidades tiene un propósito claro, brindar flexibilidad de uso. Dicha flexibilidad permitirá unir los resultados de las diferentes funcionalidades expuestas por el sistema, de esta forma elaborar funcionaliades mucho más complejas, que exploten al máximo las bondades que ofrece la estructura de datos y que se realicen fácilmente.

5.1.2.1. Estructura de Datos

La estructura de datos es la base de la mayoría de los sistemas utilizados hoy en día. La información es altamente valorada en la actualidad y un banco debe ser garante de proteger, tanto la información de sus clientes como su dinero. Hoy en día grandes empresas de software, como lo son google y facebook prestan servicios gratuitos a sus usuarios pero se lucran a través de la venta de información o prestando servicios de publicidad utilizando la misma, entre otras cosas. Bien es sabido que la información que almacenan los bancos es privada y no puede ser vendida, pero no deja de ser muy valiosa. Con ella se pude realizar procesos de inteligencia de negocios que permitan, por ejemplo, otorgar tarjetas de crédito (o aumentos de crédito) más fácilmente a clientes que hayan cambiado de residencia de una zona pobre a una zona de clase media. Por lo tanto, con el diseño de la estructura de datos se pretende lo siguiente:

- Ofrecer las operaciones que tradicionalmente ofrece un sistema bancario actual.
- Poder crear funcionalidades más amigables que le permitan al usuario sentirse cómodo con los sistemas que utilicen esta plataforma bancaria.
- La información debe persistir, de esta forma se pueden desarrollar procesos de inteligencia de negocios que le brinden mayores utilidades a la entidad financiera y una mayor comodidad a el usuario final.

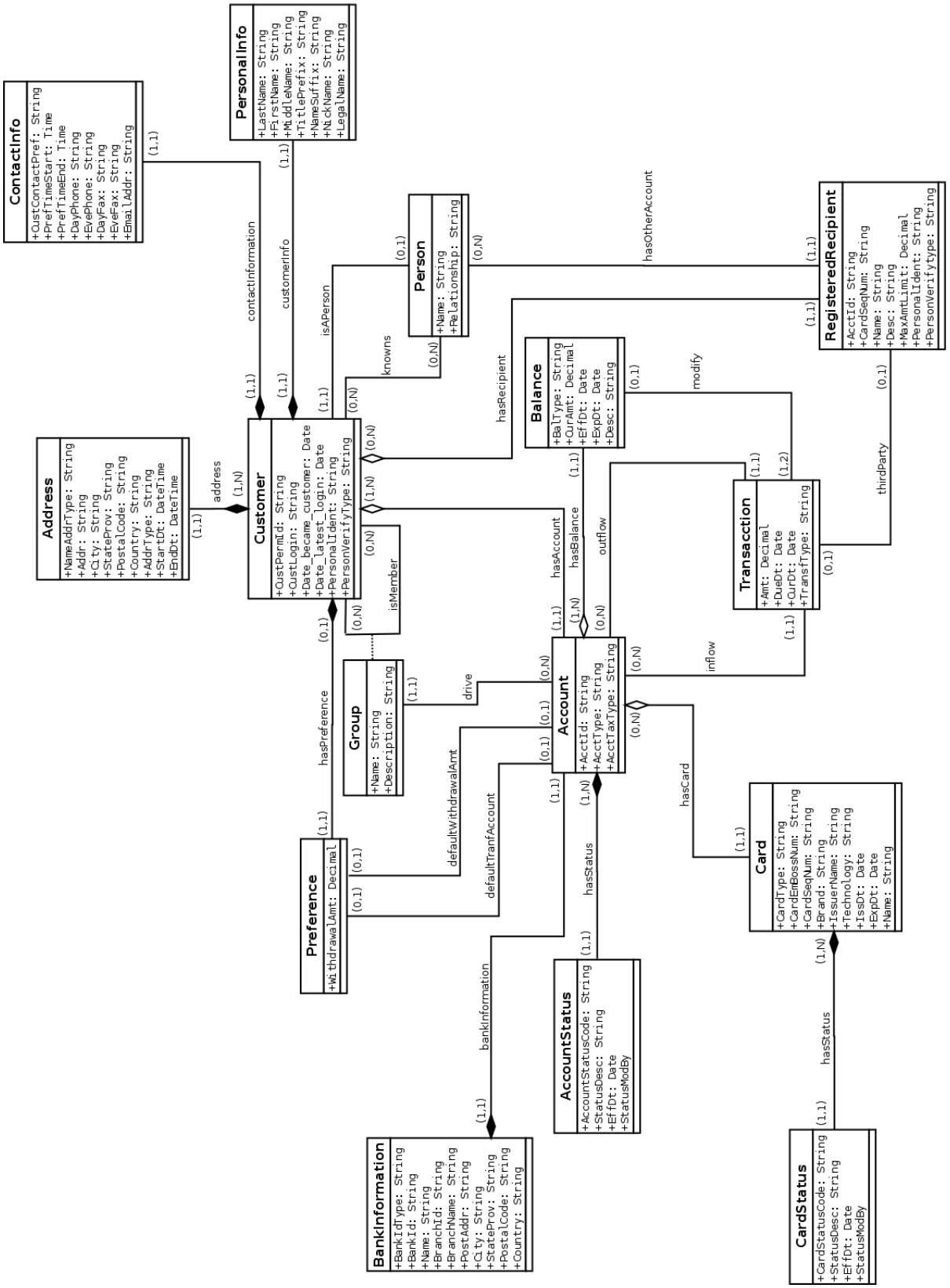

Figura 5.6: Estructura de datos del sistema bancario.

La figura 5.6 muestra la estructura de datos diseñada con el fin de garantizar lo mencionado anteriormente. Con esta arquitectura de datos se pueden realizar funcionalidades novedosas ademas de las que convencionalmente utiliza un banco. Alguna de estas funcionalidades podrían ser:

- Realizar transferencias a persona de las cuales se desconoce el numero de cuenta.
- Permite definir un monto por defecto para realizar retiros de efectivo rápidamente por los cajeros automáticos.
- Definir a que cuenta será transferido dinero proveniente de cierto grupo de personas.

5.1.2.2. Funcionalidades

Las funcionalidades que expondrá el prototipo bancario son funcionalidades muy simples y estas serán expuestas utilizando el protocolo IFF (ver anexos). Al ser este un sistema que utiliza tecnología web, se decidió ver todas las clases mostradas en la figura 5.6 como recursos y se diseñó un protocolo de comunicación basado en el protocolo IFX y utilizando los principios de una arquitectura REST. El protocolo IFF prescribe que todo (en el modelo de datos) es un recurso y para realizar alguna operación sobre los recursos, se debe realizar una petición y se retornará una respuesta. Cada petición está asociada a un recurso y sobre estos recursos se pueden realizar las siguientes operaciones:

- Add (añadir)
- Mod (modificar)
- Del (eliminar)
- Can (cancelar)
- Inq (consultar)

Es de vital importancia utilizar un protocolo de comunicación para interactuar con el sistema bancario. Esto brindará unas reglas de comunicación claras lo cual facilitará el uso del sistema y la aceptabilidad del mismo. Si el protocolo se llega a estandarizar en diversos sistemas bancarios (como lo hizo el IFX en Estados Unidos) el resultado será que con una sola aplicación un usuario podrá acceder a sus cuentas en múltiples entidades financieras y realizar las operaciones de su agrado.

Capítulo 6

Desarrollo

En este capitulo se mencionaran las tecnologías utilizadas para realizar lo especificado en el capitulo anterior. Básicamente el desarrollo del prototipo bancario se dividió en dos etapas. La primera etapa fue cumplir con la creación del sistema bancario, dejando a un lado parte del *logging*. La segunda consistió en implementar una infraestructura que brindara el mecanismo de *logging* a la aplicación.

6.1. Sistema Bancario

Para desarrollar un sistema distribuido que permitiera la flexibilidad descrita en el capitulo de diseño, la primera interrogante a resolver fue ¿Qué lenguaje de programación se debía utilizar? Esta no es una pregunta fácil de responder, sobre todo cuando existen tantos lenguajes de programación y cada uno brinda ventajas particulares. Por ejemplo, el lengua C es un lenguaje de alto nivel que corre muy rápido cuando es compilado. Mientras que lenguajes más flexibles aportan una mayor facilidad de programación pero suelen correr más lentamente. Se optó por utilizar Python como lenguaje. Es cierto que Python es un lenguaje interpretado y por esta característica es lento. Es bien sabido que los sistemas bancarios ne-

cesitan atender peticiones lo más rápido posible y esto podría ser contraproducente. Lo que se debe tener en mente es que se desarrolló un prototipo de sistema bancario y la intención era probar la arquitectura del sistema más que su velocidad. Como Python es un lenguaje muy flexible, lo hace indicado para probar prototipos.

Inicialmente se comenzó dividiendo la estructura de datos en dos partes. La parte que se iba a almacenar en una base de datos relacional y la que se almacenaría en una base de datos nosql. La figura 6.1 muestra esta división.

Se utilizó Postgresql como manejador para la base de datos relacional y Couchbase como manejador para la base de datos nosql.

En Postgresql se crearon las tablas correspondientes con sus debidas restricciones. También se creó un *trigger* encargado de llenar la tabla "balance". La información contenida en esta tabla es el resultado de cualquier transacción realizada sobre una cuenta. Las bases de datos nosql carecen de estructura, por esta razón no era necesario crear tabla alguna en Couchbase. Sin embargo no se pueden realizar consultas sobre documentos utilizando campos que no sean los identificadores sin antes crear indices sobre los documentos. Por esto se procedió a crear los indices necesarios para realizar consultas en Couchbase por otros campos diferentes a los identificadores.

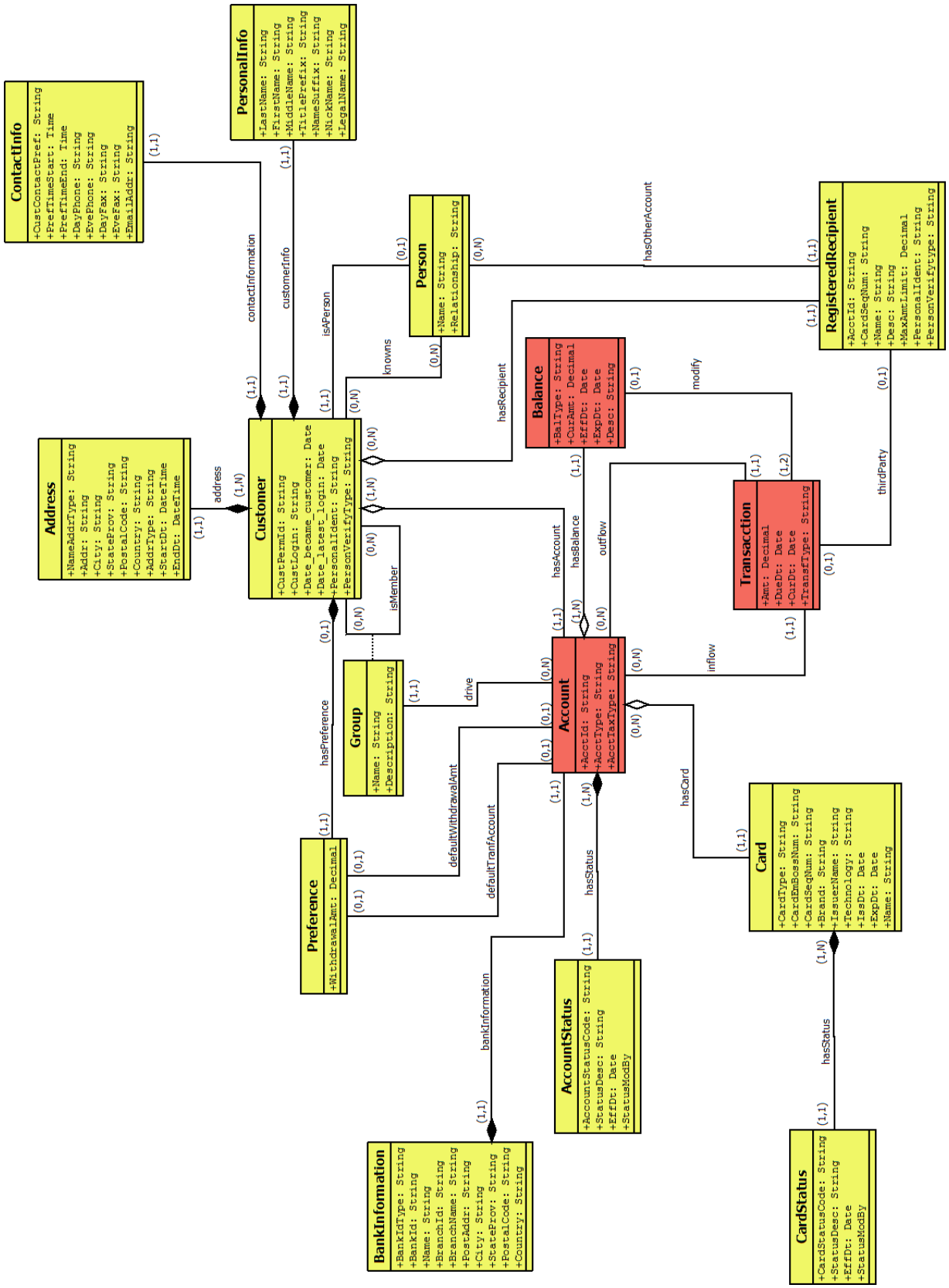

Figura 6.1: División de la estructura de datos. El rojo indica lo perteneciente a la base de datos relacional mientras lo amarillo muestra lo perteneciente a la base de datos nosql.

Para el desarrollo del sistema en Python se utilizó Twisted. Twisted es un framework orientado a eventos que facilita la implementación de comunicaciones sobre redes. Esta herramienta es muy útil pero para funcionar realmente es necesario que todos los procedimientos que corran en el hilo principal de ejecución sean de carácter asíncrono. Las librerías para la comunicación con Couchbase no eran asíncronas. Debido a esto y utilizando como base la la librería "adbapi"(para comunicación con bases de datos relacionales), se implementó una librería similar para el manejo de las conexiones con Couchbase.

Se implementaron los nodos core (ver figura, 5.1), estos nodos son los encargados de realizar toda la lógica de las operaciones bancarias y comunicarse con las respectivas bases de datos. Las operaciones implementadas fueron las correspondientes a añadir, eliminar y consultar (del IFF, ver anexo) sobre cada recurso mostrado (ver figura 5.6)

Los nodos de servicio web fueron implementados para servir de interfaz hacia las aplicaciones finales. Estos nodos se encargan de recibir las peticiones y enviarlas a los nodos core correspondientes. Se tenía pensado que toda la lógica de verificaciones se realizará en estos nodos. Dado a que este no sería un producto final ya que se está hablando de un prototipo, se decidió dejar estas verificaciones a un lado para avanzar en funcionalidades de mayor interés. Entre estas funcionalidades tenemos por ejemplo, el *loggin* del cual se hablará más adelante. Los nodos de servicios web exponen una interfaz web que permite utilizar las operaciones de agregar sobre cada recurso de la base de datos.

Las comunicaciones entre los nodos se realizaron utilizando *websockets*. Estas viajan por un canal seguro de comunicación que utiliza el protocolo TLS. Es necesario que los distintos nodos tengan su certificado digital firmado por la entidad bancaria para poder establecer la comunicación (como se especifica en el capitulo 5). Para finalmente llevar a acabo la comunicación se utilizó RPC sobre lo descrito anteriormente. El nodo de servicio web manda una petición al nodo core para utilizar un procedimiento. Este es un procedimiento despachador,

se encarga de entender la petición y encontrar el manejador adecuado para atenderla.

6.2. Infraestructura de Logging

Una vez desarrollado el prototipo de sistema bancario era necesario necesario implementar un sistema que registrara todas las operaciones que se realizaran en el banco. Una vez registrada esta información era necesario hacerla accesible de manera fácil y rápida. Por lo tanto se desarrollo un sistema de *loggin* como se muestra en la figura 5.5.

Inicialmente cada nodo del prototipo de sistema bancario registraba las operaciones de manera local. Esto cumplía con la restricción de registrar la información pero no permitía que esta fuera fácilmente consultada. Utilizando Logstash se inspeccionó los archivos donde cada nodo registraba las operaciones realizadas. La herramienta detecta los cambios realizados en los archivos y estos inmediatamente son enviados a un servidor que almacena cada operación.

El servidor donde se almacena cada registro utilizó Elasticsearch. Esta herramienta se encarga de almacenar la información y posee un potente motor de búsqueda. Sin embargo las búsquedas no se pueden realizar de manera sencilla. Elasticsearch implementa las búsquedas en un lenguaje con formato JSON y realizar consultas de esta forma no es sencillo para un usuario final (un auditor). Por esta razón se utilizó Kibana.

Kibana se utilizó como una interfaz web que se encarga de traducir lo que quiere el usuario final al lenguaje de Elasticsearch. De esta manera el usuario puede realizar búsquedas de manera sencilla. Además Kibana provee una alta configurabilidad y permite al usuario predefinir gráficos, tablas y consultas. Esto resulta ser muy ventajoso porque una de las maneras más sencillas para detectar anomalías es mediante a la observación de gráficos.

Capítulo 7

Pruebas y Resultados

Una vez habiendo implementado el prototipo, el siguiente paso es simular la arquitectura del sistema y así contar con un ambiente de pruebas que permita analizar su comportamiento. Las pruebas que se realizaron en esta etapa permiten observar el comportamiento normal del sistema. También se desea evaluar su comportamiento cuando se presenta una fuerte carga de peticiones en muy corto tiempo. Synergy-GB suministró datos de algunos bancos venezolanos referentes a concurrencia en horas pico. La identidad de estas instituciones financieras no será revelada por razones de confidencialidad. De cualquier forma, estos datos servirán para realizar comparaciones entre aquellos sistemas y el prototipo implementado en este trabajo de grado.

7.1. Construcción del Ambiente de Pruebas

Para crear el ambiente de pruebas la compañía proporcionó tres computadoras con las características que se muestran en la tabla 7.1.

Una de las computadoras se utilizó como base de datos. Tanto la base de datos del prototipo como la del sistema de *loggin*. Estas bases de datos no fueron optimizadas y recursos

Procesador	Memoria	Sistema Operativo	Disco Duro
Intel Core i5-2310 CPU @ 2.90GHz x 4	4 GB RAM	Ubuntu 12.04	750 GB 7200 RPM

Cuadro 7.1: Computadoras utilizadas.

para cada una de ellas fueron los que el sistema operativo les asignó por defecto.

En las otras dos computadoras se crearon máquinas virtuales utilizando la herramienta Xen. Una de las computadoras fue destinada para alojar a los nodos core. Por esta razón se crearon dos máquinas virtuales y en cada una de ellas se alojó un nodo core. Cada nodo core contaba con 1200 MB de memoria RAM y cincuenta GB de disco duro. A cada nodo le fue asignado un procesador (de los cuatro que poseía la maquina anfitriona) dedicado y uno adicional que era compartido entre los dos nodos. Por lo tanto un nodo core poseía lo mostrado en la tabla 7.2

Procesador	Memoria	Sistema Operativo	Disco Duro
Intel Core i5-2310 CPU @ 2.90GHz x 1.5	1200 MB RAM	Ubuntu 12.04	50 GB 7200 RPM

Cuadro 7.2: Características de un nodo core.

En la otra computadora se crearon tres maquinas virtuales. Dos de ellas alojaron nodos de servicio web y la tercera un balanceador. Se consideró necesario el balanceador ya que no es la idea que un servidor esté muy cargado de peticiones mientras el otro se encuentra esperando. El balanceador se configuró utilizando la herramienta Nginx y con él se garantiza que la redirección de peticiones de manera proporcional hacia los nodos de servicio web. A cada una de las máquinas virtuales se le asignó un procesador. A los nodos de servicio web les fue asignado 900 MB de memoria RAM y 30 GB de disco duro. Al balanceador le fue asignado 600 MB de memoria RAM y 20 GB de disco duro. Estos datos se pueden observar

de una mejor manera en la tabla 7.3

	Procesador	**Memoria**	**Sistema Operativo**	**Disco Duro**
Nodo web	Intel Core i5-2310 CPU @ 2.90GHz x 1	900 MB RAM	Ubuntu 12.04	30 GB 7200 RPM
Nodo web	Intel Core i5-2310 CPU @ 2.90GHz x 1	900 MB RAM	Ubuntu 12.04	30 GB 7200 RPM
Balanceador	Intel Core i5-2310 CPU @ 2.90GHz x 1	600 MB RAM	Ubuntu 12.04	20 GB 7200 RPM

Cuadro 7.3: Distribución de los recursos de una computadora hacia los nodos de servicio web y al balanceador.

Los recursos que no fueron asignados a ninguna máquina virtual los utilizaron los virtualizadores para gestionar las máquinas virtuales.

El ambiente de pruebas utilizado se puede observar en la figura 7.1. Se utilizaron las computadoras y máquinas virtuales descritas anteriormente. En la figura se puede observar que existe una conexión directa entre cada nodo de servicio web con uno core. Los nodos core almacenan la información en la máquina que aloja las bases de datos. Cada nodo, por medio de Logstash, se encarga de registrar en Elasticsearch la información de las operaciones que realizan. Elasticsearch expone la información registrada a través de la web. El balanceador se encarga de que las peticiones realizadas por la computadora cliente sean distribuidas de manera equitativa entre los dos nodos de servicio web.

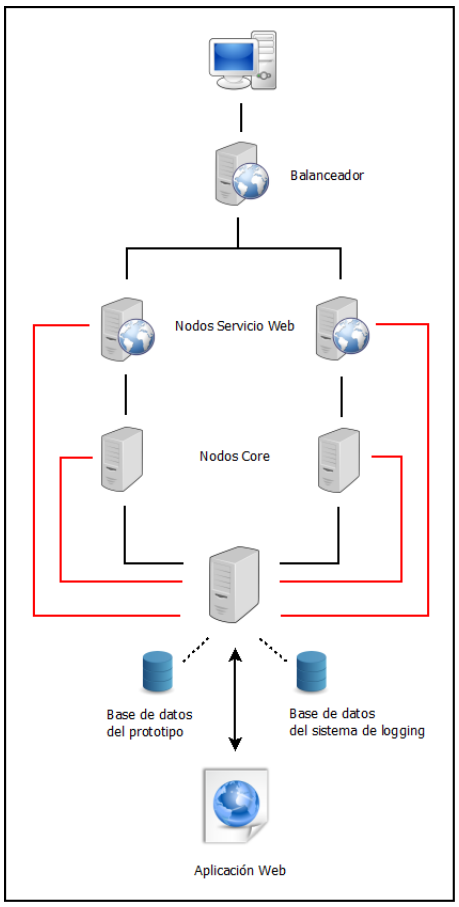

Figura 7.1: Ambiente de pruebas utilizado.

7.2. Pruebas y Resultados

En esta fase se utilizó el ambiente descrito anteriormente y mostrado en la figura 7.1 para realizar las pruebas. La intención de estas pruebas era obtener resultados que permitieran comparar el rendimiento del prototipo con sistemas de diferentes bancos venezolanos. La tabla 7.4 muestra información sobre cantidad de peticiones atendidas en horas pico sobre entidades bancarias que fueron agrupadas en tres niveles, grandes, medianas y pequeñas. Esta información no se suministra de manera detallada por políticas de privacidad.

	Cantidad de Peticiones Atendidas
Grandes	350000
Medianas	150000
Pequeñas	15000

Cuadro 7.4: Peticiones en hora pico.

El objetivo principal de las pruebas es evaluar el comportamiento del sistema en función del número de peticiones que suelen responder los sistemas bancarios venezolanos en hora pico.

Capítulo 8

Retos Enfrentados y Logros Adicionales

En este capitulo se mencionan los principales retos enfrentados durante el proceso de elaboración del proyecto. También se mencionaran los logros alcanzados que no fueron contemplados durante la etapa de concepción del proyecto.

Un reto importante durante la elaboración de este proyecto fue encontrado en la etapa de diseño de la arquitectura del sistema. Esta etapa fue particularmente retadora ya que se debían utilizar todos los conocimientos sobre el área y realizar un análisis exhaustivo para formular el mejor resultado posible. La amplitud del proyecto abre paso a un universo muy amplio de posibilidades y esto puede hacer que el objetivo principal se pierda. Lograr plantear una solución que permitiera probar los aspectos principales utilizando todos los conocimientos previos, fue sin duda uno de los principales retos.

Otro reto que se presentó tiene que ver con el proceso de pruebas del sistema. La empresa proporcionó tres computadoras (adicionales a donde se implementó el prototipo) pero por un periodo muy corto de tiempo. Se contaba con cuatro días desde el momento en el cual se tuvo acceso a las computadoras, hasta el momento en el cual tenían que ser devueltas. En

este periodo se debía realizar lo siguiente:

- Crear el ambiente de pruebas descrito en la sección 7.1
- Realizar los *scripts* que permitieran probar el rendimiento del sistema.
- Realizar y monitorear el proceso de pruebas.
- Obtener la información resultante.

Esta tarea fue retadora debido al poco tiempo con el que se contaba para realizar las tareas descritas anteriormente. Esto implicaba que todo debía realizarse de manera meticulosa para no cometer errores. De presentarse algún error, bien sea por una mala configuración del ambiente de pruebas, o por problemas no detectados en la etapa de implementación, se debía resolver inmediatamente ya que no se contaba con otra oportunidad para realizar este proceso.

Un logro adicional que se alcanzó, fue el desarrollo de un pequeño sistema web que permitiera probar alguna de las funcionalidades para las cuales se diseñó la estructura de datos. Se utilizó el *framework* Django que utiliza como lenguaje Python. Este sistema web se representa las aplicaciones que se pueden construir teniendo como base este core bancario. No formaba parte del alcance de la pasantía pero se consideró muy importante su desarrollo porque permite entender como se ha pensado sea utilizado el prototipo.

Bibliografía

[EDP] Event-driven programming. `http://en.wikipedia.org/wiki/Event-driven_programming`, consultado el 16 de marzo de 2014.

[ELK] The elasticsearch elk stack. `http://www.elasticsearch.org/overview/`, consultado el 16 de marzo de 2014.

[Fie00] Roy Thomas. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000.

[GHN09] James. Governor, Dion. Hinchcliffe, and Duane. Nickull. *Web 2.0 Architectures*. Adobe Developer Library, 1 edition, 2009.

[Gro] The PostgreSQL Global Development Group. Postgresql 9.1.9 documentation. `http://www.postgresql.org/docs/manuals/`, consultado el 16 de marzo de 2014.

[LS07] Richardson Leonard and Ruby Sam. *RESTful Web Services*. O'Reilly Media, Inc, 1 edition, 2007.

[MF13] Jessica McKellar and Abe Fettig. *Twisted Network Programming Essentials*. O'Reilly Media, Inc., 2 edition, 2013.

[NGI] Why nginx? `http://nginx.com/`, consultado el 16 de marzo de 2014.

[RG03] Raghu Ramakrishnan and Johannes Gehrke. *DATABASE MANAGEMENT SYSTEMS*. McGraw-Hili, 3 edition, 2003.

[SGB] Synergy-gb. `http://synergy-gb.com/`, consultado el 16 de marzo de 2014.

[Tea13a] The Scrum Team. The definitive guide to scrum: The rules of the game. `https://www.scrum.org/Portals/0/Documents/Scrum%20Guides/2013/Scrum-Guide.pdf`, consultado el 16 de marzo de 2014, Julio 2013.

[Tea13b] The Twisted Development Team. The twisted documentation. `https://twistedmatrix.com/trac/wiki/Documentation`, consultado el 16 de marzo de 2014, Junio 2013.

[XEN] Just what is the xen project hypervisor? `http://www.xenproject.org/users/virtualization.html`, consultado el 16 de marzo de 2014.