UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Desarrollo, implementación y optimización de un sistema de mensajería SMS

INFORME DE ACTIVIDADES PROFESIONALES

Que para obtener el título de

Ingeniero en Computación

P R E S E N T A

Jaime Esteban Beltrán Rosales

ASESORA DEL INFORME

Ing. Laura Sandoval Montaño

Ciudad Universitaria, Cd. Mx., 2019

Agradecimientos

Antes que nada, quisiera agradecer a mi padre que sin su ayuda y sacrificio yo no hubiera tenido la oportunidad de estudiar en esta maravillosa universidad, a mi fallecida madre ya que sin sus enseñanzas e inspiraciones tan nobles y duraderas jamás habría podido llegar tan lejos y a mi hermano por siempre estar ahí para mí, aunque las cosas se pusieran difíciles.

También me gustaría agradecer a todos los profesores que me apoyaron durante la realización y revisión de este trabajo. A Laura Sandoval Montaño por apoyarme con las múltiples revisiones y versiones por las que pasó este trabajo y por su alta disposición. A Elba Karen Saenz García por apoyarme durante el resto del proceso en representación de la profesora Laura. A Eduardo Espinosa Ávila por apoyarme con la revisión de este trabajo y por todas esas clases y pláticas tan interesantes y enriquecedoras que me permitieron tener una formación mas integral como profesionista. A Jaime Érik Castañeda De isla Puga por todo su legado educación, pero principalmente por ser una persona tan humilde, dispuesta a apoyar en cualquier momento y a dar ánimos cuando parece que no se puede continuar. A Tanya Itzel Arteaga Ricci por ayudarme a confirmar que me encontraba en la licenciatura correcta.

Agradecer también a Daniel Trejo Medina, uno de los profesores que nos ayudó a ver el mundo empresarial dentro de la ingeniería en computación, que nos enseñó que no todo en la carrera profesional ni en la vida son únicamente líneas de código, pero principalmente porque siempre nos impulsó a ser mejores sin dejar de creer y tener confianza en nosotros.

Finalmente, a todos mis amigos y conocidos con los cuales tuve la fortuna de compartir mi vida universitaria.

Contenido

1. Introducción

La formación académica obtenida en la Facultad de Ingeniería, específicamente en la carrera de Ingeniería en Computación, me ha permitido desempeñarme como un profesionista a lo largo de 3 años en una empresa de consultoría. Es por ello que en el presente trabajo describo, de forma detallada, mis actividades profesionales en dicha empresa, y particularmente mi participación como desarrollador junior en el proyecto de desarrollo e implementación de una plataforma de envíos masivos de mensajería SMS.

El proyecto surge en la empresa como resultado de realizar un estudio de mercado donde se identificó que varias empresas e instituciones, públicas o privadas, requerían realizar comunicación con sus clientes, tanto de una vía como de dos, a través de envío masivo de mensajes por medio de telefonía celular. Por lo que la empresa decidió desarrollar su propia plataforma SMS que cuente con diversas capas para permitir a clientes técnicos que quieran conectarse por medio de una API REST el utilizarlo sin problemas, así como a clientes promedio que quieran realizar el envío de mensajes masivos por medio de una interfaz más amigable, en este caso, una página web.

Los apartados del presente trabajo muestran, de forma estructurada, tanto mi participación profesional en la empresa como la elaboración y puesta en marcha de la plataforma SMS en todas sus etapas, ajustándose a la metodología SCRUM, así como la descripción y aplicación de las tecnologías de software y hardware empleadas en el desarrollo, optimización e implementación del sistema.

2. Objetivo

Presentar mis actividades profesionales realizadas como participante en el proyecto: “Desarrollo e implementación de una plataforma de envíos masivos de mensajería SMS”, utilizando tecnología y recursos con que cuenta la empresa de consultoría y que dé como resultado un sistema eficiente, óptimo, confiable y sencillo de usar; esto aplicando conocimientos y habilidades obtenidas en mi formación académica.

3. Descripción de la empresa

La empresa es un grupo de consultores especializados en diseñar e implantar soluciones puntuales a problemas específicos, a través de establecer una estrecha relación socio-cliente compenetrándonos para el logro de los objetivos de negocio.

El éxito de la empresa se sustenta en su capital humano: Sus socios y asociados cuentan con vasta experiencia derivada de proyectos exitosos puestos en marcha tanto en México como en el extranjero.

Un equipo de profesionales experimentados con visión de negocio en diversas industrias y disciplinas a la disposición de nuestros clientes, con el objeto de incorporar las mejores prácticas y lograr los mejores resultados.

- Misión: Generar alianzas con sus clientes para identificar sus áreas de oportunidad de mayor valor, atender sus retos más importantes y ayudarlos hacia una transformación de sus empresas.

- Visión: Ser la empresa de consultoría de nicho de referencia en México gracias a los resultados entregados y el alto valor percibido por nuestros clientes.

- Experiencia: Sus principales socios y asociados han ocupado puestos directivos en áreas de negocio, riesgos, operaciones, TI, crédito, finanzas, mercados, entre otras.

4. Descripción del puesto de trabajo

Desempeño el cargo de consultor de tecnologías de la información y desarrollador de software desde hace casi 3 años y mis responsabilidades principales del puesto son el desarrollo y soporte de aplicativos web y móviles, principalmente en el framework de Microsoft de ASP.NET y Entity Framework por medio de C#, JavaScript, HTML5 y Xamarin, apoyándonos en el Framework de Telerik. Por otro lado, para motor de base de datos utilizamos principalmente SQL Server.

Sin embargo, debido a que se trata de una empresa pequeña, la capacidad de autogestión se vuelve un factor muy importante para la administración de nuestros proyectos. En mi caso, he tomado el rol de Scrum Master para poder implementar Scrum como metodología de gestión y seguimiento de proyectos.

Derivado también del tamaño de la empresa y asesorado por un desarrollador sr., he tomado el rol de administrador auxiliar de servidores (en nuestro caso, Windows Server), incluyendo actividades como mantenimiento, seguridad, liberación de nuevos aplicativos a un servidor productivo, entre otras.

También he tomado la función de arquitecto de software para el caso de desarrollos, esto con la intención de poder diseñar un nuevo sistema, incluyendo el diseño del aplicativo, sus interacciones internas, externas y la base de datos.

Finalmente, como consultor de tecnologías de la información realizo las actividades de explicar y orientar a nuestros clientes en cualquier desarrollo tecnológico actual o futuro, las implicaciones y limitantes de sus decisiones (posibilidad de escalabilidad) así como de ofrecer recomendaciones tecnológicas y en ocasiones incluso seguimiento de proyectos, según corresponda.

5. Marco Teórico

5.1 .NET C#

Es un lenguaje de programación compilado que utiliza el paradigma orientado a objetos (POO) con tipo seguro (type-safe). Este nos permite construir aplicativos robustos y seguros que corren en el framework de .net, por ejemplo, aplicaciones para clientes de Windows, servicios web en XML y REST, componentes distribuidos, aplicaciones de cliente-servidor, aplicaciones de base de datos y más. Es un lenguaje que hereda características de C, C++ y Java, pero también provee poderosas características como delegados, expresiones lambda y accesos directo a memoria, las cuales no se encuentran en Java. C# soporta métodos y tipos genéricos que proveen un incremento en la seguridad de tipo y rendimiento.

Como un lenguaje orientado a objetos, C# soporta los conceptos de encapsulamiento, herencia y polimorfismo. Todas las variables y métodos, incluyendo el método main, el punto de entrada de la aplicación, son encapsulados dentro de las definiciones de la clase. Una clase puede heredar directamente de una clase padre, pero puede implementar cualquier número de interfaces. Los métodos que reemplazan o sobrescriben a los métodos virtuales en una clase padre requieren la palabra clave override como una forma de evitar la redefinición accidental. En C#, una estructura es como una clase ligera, es un tipo que está asignado como stack (stack-allocated) que implementa interfaces pero no permite herencia.

El proceso de compilación de C# es simple comparado con C y C++ y más flexible que en java. No hay archivos encabezados independientes, y no requieren que los métodos y tipos sean declarados en un orden en particular. Un archivo código fuente de C# puede definir cualquier número de clases, estructuras, interfaces y eventos.

5.2 .NET Framework

Los programas escritos en C# corren en el Framework .NET, un componente integral de Windows que incluye un sistema de ejecución virtual llamado common language runtime (CLR) y un conjunto unificado de bibliotecas de clases. El CLR es la implementación comercial por Microsoft del common language infrastructure (CLI), un estándar internacional que es la base para crear ambientes de ejecución y desarrollo en el cual lenguajes y bibliotecas trabajan juntos sin problemas.

El código fuente escrito en C# es compilado en un lenguaje intermedio (IL por su nombre en inglés) que se ajusta a la especificación del CLI. El código IL y sus recursos, tal como bitmaps y cadenas son guardados en disco en un archivo ejecutable llamado “assembly”, normalmente con una extensión .exe o .dll. Un “assembly” contiene un manifiesto que provee información acerca de los tipos del assembly, su versión, cultura y requerimientos de seguridad.

Cuando el código C# es ejecutado, el assembly es cargado en el CLR, el cual puede tomar diversas acciones basado en la información del manifiesto. Después, si los requerimientos de seguridad se cumplen, el CLR ejecuta una compilación just in time (JIT) para convertir el código IL a instrucciones nativas de la máquina. El CLR también provee otros servicios relacionados con recolección de basura

automática, manejo de excepciones y manejo de recursos. El código que es ejecutado por el CLR es algunas veces referido como “código administrado”, en contraste con “código no administrado” el cual es compilado en lenguaje nativo de la máquina que se dirige a un sistema en específico. El siguiente diagrama ilustra el tiempo de compilación y las relaciones de ejecución de un archivo de código fuente de C#, las clases de librerías del .NET Framework, los ensamblados y el CLR.

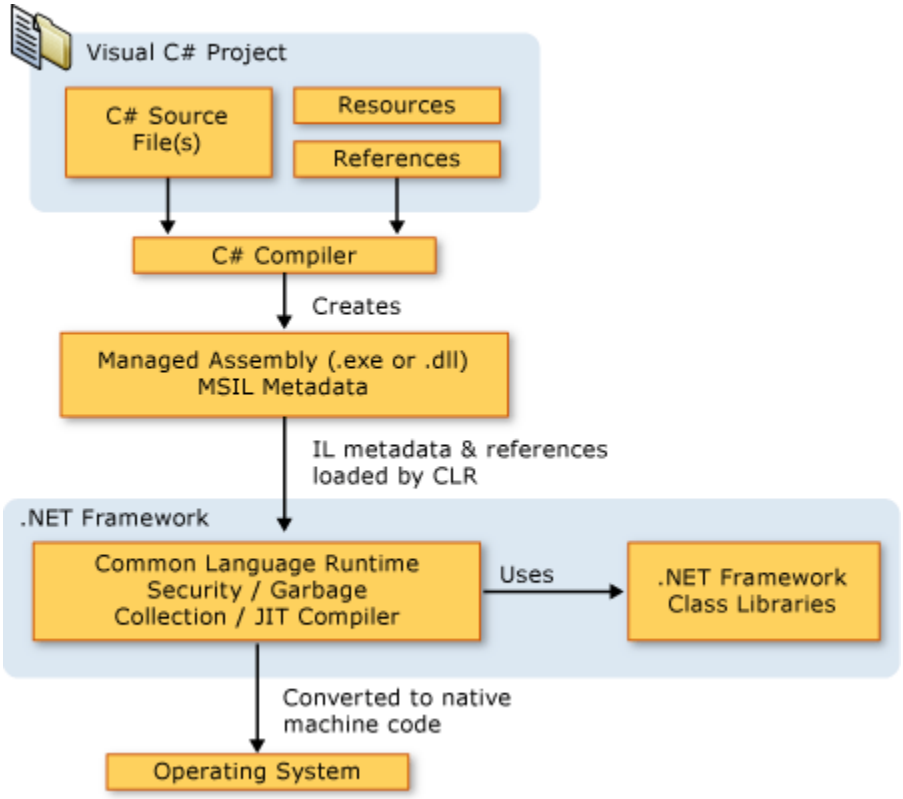


Figura 1. Estructura de un proyecto .NET framework

La interoperabilidad del lenguaje es una característica de .NET Framework, dado que el código IL producido por el compilador C# cumple con la especificación de tipo común (CTS), el código IL generado desde C3 puede interactuar con el código generado a partir de las versiones .NET de Visual Basic, Visual C++ o cualquiera de los otros 20 lenguajes compatibles con CTS. Un único ensamblado puede contener múltiples módulos escritos en diferentes lenguajes .NET, y los tipos pueden referirse entre sí como si estuvieran escritos en el mismo idioma.

Además de los servicios de tiempo de ejecución, .NET Framework también incluye una extensa biblioteca de más de 4000 clases organizadas en espacios de nombres que proporcionan una amplia variedad de funcionalidades útiles para todo, desde la entrada y salida de archivos hasta la manipulación de cadenas. La aplicación típica utiliza ampliamente la biblioteca de clases .NET Framework para gestionar las tareas comunes.

5.3 Entity Framework y Entity Framework Core

Entity Framework (EF) es un conjunto de tecnologías que soportan el desarrollo de aplicativos de software orientados a datos. El EF permite a los desarrolladores trabajar con información en la forma de objetos y propiedades específicos al dominio, sin la necesidad de preocuparse con las tablas de la base de datos y las columnas donde se almacena esta información. Con EF, los desarrolladores pueden trabajar con un nivel de abstracción mayor que cuando están lidiando con datos, y pueden crear y mantener aplicaciones orientadas a datos con menos código que en aplicaciones tradicionales.

EF es un mapeador de objeto-relacional (ORM por su nombre en inglés) que reduce la fricción entre un mundo orientado a objetos de desarrolladores del .NET Framework y el mundo de las bases de datos relacionales. Permite a los desarrolladores primariamente interactuar con el modelo conceptual de la aplicación, usando conocidas técnicas orientadas a objetos. que permite a los desarrolladores .NET trabajar con una base de datos usando objetos .NET. Elimina la necesidad de la mayoría de los códigos de acceso a datos que los desarrolladores normalmente necesitan escribir.

Hay 2 capas principales en una aplicación EF:

- La capa de modelado
- La capa de objetos

La capa de modelados contiene 3 componentes:

1. Un modelo conceptual que consiste de tipos de entidades y relaciones de dominio específico.
2. Un esquema de base de datos que define las tablas y relaciones
3. Un mapeo entre el modelo conceptual y el esquema de la base de datos

EF usa el componente de mapeo para transformar operaciones contra objetos de entidades, tales como crear, leer, actualizar y borrar, en operaciones equivalentes de la base de datos.

La capa de objetos del EF contiene objetos comunes del CLR que reflejan las entidades y relaciones definidas en los modelos conceptuales. Estos objetos pueden ser consumidos por lenguajes de programación. El formato exacto de los tipos es controlado por opciones que se proveen al EF.

EF Core es una versión ligera, extensible y multi-plataforma de la popular tecnología de acceso a datos Entity Framework. EF Core soporta muchos motores de base de datos, como sql server, sqlite, PostgreSQL, Mysql, entre otros.

Con EF Core, el acceso a los datos es realizado por medio de un modelo. Un modelo es una entidad de clases y un contexto derivado que representa una sesión con la base de datos, permitiéndonos hacer consultas y guardar información. Se puede generar un modelo de una base de datos existente, hacerlo manualmente para que coincida con la base de datos, o utilizar EF Migrations para crear una base de datos a partir del modelo (y evolucionarlo a la par con los cambios del modelo a través del tiempo).

```
using Microsoft.EntityFrameworkCore;
using System.Collections.Generic;

namespace Intro
{
    public class BloggingContext : DbContext
    {
        public DbSet<Blog> Blogs { get; set; }
        public DbSet<Post> Posts { get; set; }

        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseSqlServer(@"Server=(localdb)\mssqllocaldb;Database=MyDatabase;Trusted_Cc
        }
    }

    public class Blog
    {
        public int BlogId { get; set; }
        public string Url { get; set; }
        public int Rating { get; set; }
        public List<Post> Posts { get; set; }
    }

    public class Post
    {
        public int PostId { get; set; }
        public string Title { get; set; }
        public string Content { get; set; }

        public int BlogId { get; set; }
        public Blog Blog { get; set; }
    }
}
```

Figura 2. Ejemplo de un modelo de EF

La instancia de tu clase de entidades son obtenidas de la base de datos usando el Language Integrated Query (LINQ).

```
using (var db = new BloggingContext())
{
    var blogs = db.Blogs
        .Where(b => b.Rating > 3)
        .OrderBy(b => b.Url)
        .ToList();
}
```

Figura3. Ejemplo de una query en LINQ de EF

La información es creada, eliminada y modificada en la base de datos usando instancias de tu clase de entidades

```
using (var db = new BloggingContext())
{
    var blog = new Blog { Url = "http://sample.com" };
    db.Blogs.Add(blog);
    db.SaveChanges();
}
```

Figura 4. Ejemplo de una inserción a una base de datos desde EF

5.4 Web services (Servicios Web)

Los web services son aplicaciones Cliente-Servidor que se comunican por medio del protocolo de transferencia de hipertexto (HTTP), pudiendo o no pasar por internet. Estos proporcionan un medio estándar de interoperar entre aplicaciones que se ejecutan en una variedad de sistemas. El desarrollo o consumo de estos servicios no está enlazado con ningún sistema operativo o lenguaje de programación.

Debido a que los servicios web utilizan http y este es un protocolo sin estados, los servicios web usualmente no tienen estados (aunque es posible simularlos haciendo uso herramientas adicionales, como cookies). Esto quiere decir que el servicio trata cada petición como una independiente de la anterior.

Existen principalmente 2 tipos de web services:

- SOAP: en este tipo de servicios el lenguaje de comunicación es XML y obedece a las especificaciones definidas por SOAP, encargado de definir la estructura, reglas de codificación y las conversiones para representar las invocaciones y respuestas del servicio web.

  RESTful: En este caso el lenguaje de comunicación es JSON y es adecuado para escenarios básicos que pueden trabajar con la utilización mínima de herramientas. Este tipo de servicios web cada día gana más popularidad y se está convirtiendo en el nuevo estándar debido a la fácil comprensión del formato JSON.

5.5 SCRUM

Scrum es una metodología ágil para administración de proyectos, principalmente proyectos de software. Scrum es un framework iterativo e incremental. Está diseñado para equipos de 3 a 9 miembros, que dividen su trabajo en acciones que pueden ser completadas en iteraciones con tiempos definidos llamados sprints, nunca mayor a un mes y normalmente de dos semanas; después se analiza el progreso y el plan en las reuniones de seguimiento conocidas como daily stand-up meetings o daily scrums.

Un principio clave de Scrum es el reconocimiento de que un cliente puede cambiar su mente sobre lo que quiere o necesita (normalmente llamado volatilidad de requerimientos) y que habrá retos impredecibles, por lo tanto Scrum adopta un acercamiento empírico: aceptar que el problema no puede ser enteramente descrito o definido al inicio y en lugar de eso enfocarse en cómo maximizar la habilidad del equipo para desarrollar rápidamente, para responder a requerimientos emergentes y para adaptarse a la evolución de las tecnologías y cambios en el mercado.

Existen 3 roles clave en cualquier proyecto de Scrum:

- Product Owner: Es el representante de la empresa que desea la creación del producto y es el rol más importante para el proyecto, ya que es el encargado de transmitir al equipo los requerimientos del proyecto, así como de dar su aprobación a los entregables.
- Scrum Master: Es el equivalente al líder de proyecto en la metodología de PMO o similares, sin embargo, en este caso no se asume la función de líder desde una posición de poder, si no que se considera como un integrante más del equipo cuya función es garantizar que se cumplan los tiempos y requerimientos del cliente. Adicionalmente, sensibiliza al product owner de los retos del proyecto y lo ayuda a entender por qué las cosas toman tanto tiempo y el impacto que tienen los cambios solicitados.
- Scrum developer team: Es el equipo de desarrollo que participa en el proyecto y atiende a las daily standup meetings. Son los encargados de escribir y probar el código.

5.6 Máquinas de estado

Una máquina de estados es un concepto utilizado en diseño de programas informáticos o lógica digital. Existen 2 tipos de máquinas de estado: Finitas e infinitas. Las primeras están formadas por un número finito de estados, transiciones y acciones que pueden ser modeladas con un flujo de grafos, donde el camino puede ser seguido cuando las condiciones de este se cumplen. Las segundas no se utilizan en la práctica.

Una máquina de estados es un dispositivo que almacena el estado de algo en un momento determinado. El estado cambia con base en las entradas, proveyendo la salida esperada para el cambio implementado. Una máquina de estados finita tiene memoria interna finita. Los parámetros de entrada se leen en secuencia y producen una respuesta en la forma de interface de usuario.

Las máquinas de estado son representadas usando diagramas de estados. La salida de una máquina de estados está en función a la entrada y al estado actual. Las máquinas de estado juegan un rol significativo en áreas tal como ingeniería eléctrica, lingüística, ciencias de la computación, filosofía, biología, matemáticas y lógica. Son más usadas en el modelado de comportamiento de aplicaciones, ingeniería de software, diseño de sistemas digitales, protocolos de redes, compiladores y en el estudio de la computación y lenguajes.

6. Antecedentes

6.1 Antecedente de la tecnología

Las tecnologías inalámbricas han tenido un auge y desarrollo impresionante en los últimos años, una de las mayores representantes de esto ha sido la telefonía celular. Desde su inicio a finales de los 60's ha revolucionado de una manera impresionante las actividades que realizamos día con día. Los celulares se han convertido en una herramienta necesaria tanto para la gente común como para los negocios, aumentando la seguridad personal y productividad. Aunque originalmente la telefonía celular fue diseñada originalmente con el único objetivo de transmitir voz, al día de hoy nos brinda otro tipo de servicios como datos, audio y video.

El servicio de mensajes cortos o SMS (Short Message Service), como su nombre lo dice, es un mecanismo de entrega de mensajes cortos sobre redes móviles. El funcionamiento básico de esto es que el mensaje del originador es almacenado en una central de SMS para posteriormente ser direccionado al dispositivo receptor. Esto quiere decir que cuando el receptor no está disponible, el mensaje puede ser guardado para continuar con la segunda parte del ciclo posteriormente. Los SMS tienen una limitante actual a 160 caracteres alfanuméricos.

Debido a que los SMS hacen uso de canales señalizados en lugar de canales dedicados, los mensajes pueden ser enviados y recibidos simultáneamente por medio del servicio de voz y datos de una red GSM (Global System for Mobile). SMS funciona tanto nacional como internacionalmente por medio de roaming.

La telefonía celular en México dejó de ser un servicio exclusivo para un sector privilegiado de la población y hoy en día muchas personas son usuarios intensivos de esta tecnología. Según un informe del Instituto Federal de Telecomunicaciones del tercer trimestre de 2017, en México existen 111,880,621 líneas de telefonía móvil, es decir que prácticamente 90 de cada 100 habitantes cuenta con este servicio.

A pesar del intento de la reforma en telecomunicaciones, estas líneas están conformadas por 65.2% de América Móvil (Telcel), 21.4% Telefónica Movistar, 12.2% AT&T y solamente el 1.2% son operadores móviles virtuales (OMV). Como se puede observar, el emporio del ingeniero Carlos Slim sigue representando más de la mitad del mercado.

A pesar de que según estos estudios la telefonía móvil tiene una penetración del 90%, no todas estas líneas están siendo utilizadas en un Smartphone, pero lo que es seguro es que esos números celulares tienen la capacidad para recibir SMS. Sumado a esto, a pesar de que contaran con acceso a un dispositivo móvil capaz de utilizar internet, el 80% de la población no cuenta con acceso a internet desde su red de datos.

Por otro lado, el 90% de los SMS son leídos en menos de 3 minutos, y el 95% de los SMS enviados son leídos. Todo esto ayuda a entender la importancia y el potencial que tiene el envío de mensajes SMS con diferentes fines. Aprovechando este canal, es posible lograr los siguientes beneficios:

1. Incrementar eficiencia operativa
2. Reducir costos
3. Incrementar ingresos
4. Automatizar alertas en tu proceso
5. Asegurar e incrementar niveles de servicio

Los SMS pueden ser utilizados en casi cualquier ámbito, por ejemplo:

1. Político
    a. Campañas
    b. Comunicación
    c. Avisos inmediatos
    d. Reuniones
    e. Presencia
    f. Encuestas
2. Bancario
    a. Saldos
    b. Movimientos
    c. Cambios
    d. Cobros
    e. Recordatorios
    f. Transferencias
3. Entretenimiento
    a. Menús
    b. Promociones
    c. Invitaciones especiales
    d. Reservaciones / Cancelaciones
    e. Planes de lealtad
    f. Felicitaciones
    g. Cupones
4. Salud
    a. Citas
    b. Recordatorios
    c. Cancelaciones
    d. Información
    e. Medicamentos
5. Educativo
    a. Avisos
    b. Calificaciones
    c. Fechas importantes
    d. Pagos pendientes
    e. Tareas
    f. Juntas
6. Deportivo
    a. Horarios
    b. Promociones
    c. Precios
    d. Concursos
7. Comercio/Agencias/Talleres
    a. Cobranza
    b. Entrega
    c. Seguimiento
    d. Ofertas
    e. Cupones
    f. Encuestas
    g. Posicionamiento
8. Seguridad
    a. Códigos de activación
    b. Cambios de contraseña
    c. Compras seguras
    d. Validación de identidad
    e. Avisos

Todo esto puede ayudarte a generar ahorros en:

- Llamadas
- Call centers
- Visitas
- Otros medios

E incrementar tus ingresos por:

- Ventas
- Promociones
- Capacitación
- Fidelización

6.2 Antecedentes del proyecto

La empresa identificó todas estas oportunidades de negocio en el mercado y decidió realizar la inversión para desarrollar su propia plataforma SMS, permitiendo proporcionar este servicio de diversas maneras: de forma integral (con una página web para utilizarlo) o bien como un aditamento a plataformas ya funcionales como el caso de *onBoardings*, *ERPs*, *CRMs*, etc. por medio de una API; mismas que se detallarán en un apartado más adelante.

Al iniciar el proyecto, se tenía conocimiento de que existían principalmente 3 vertientes para poder realizar el envío de mensajes SMS:

1. Conectarnos a una API de otra empresa: Esto implica construir sobre la construcción de alguien más, lo cual podía resultar en un tema riesgoso, ya que, si había problemas con la otra empresa o se rompía la relación comercial, nos quedaríamos sin sistema. Evidentemente a esta limitante se puede evitar desarrollando un sistema que se conecte a múltiples proveedores, o bien uno que esté preparado para realizar el cambio de una plataforma a otra por medio de parámetros y prácticamente sin tiempo fuera de servicio (*downtime*). Sin embargo, el principal motivo para no seleccionar esta opción fue el hecho del costo. El hecho de conectarnos a la plataforma de alguien más implicaba que nuestro costo siempre iba a ser dependiente de alguien más, tomando en cuenta sus costos administrativos, técnicos y de hardware. Debido a que en el envío masivo de SMS el margen de ganancia por cada uno es mínimo, se decidió maximizar la ganancia (aunque esto implicara aumentar la inversión y el tiempo de desarrollo) pero nos daría más autonomía, mejor control sobre el aplicativo y mayor margen.
2. Utilizar números virtuales: Antes de que se formara la empresa, uno de los socios ya había intentado entrar al negocio del envío masivo de SMS por medio de números virtuales. Gracias a esta experiencia previa, se optó no tomar esta opción ya que no daba seguridad del envío o la recepción de los mensajes (un tema esencial en caso de utilizar mensajes 2 vías) y dependíamos de otra herramienta/plataforma que virtualizara los números debido que al no contar con suficiente volumen los operadores de telefonía móvil (o carriers) no nos permitían utilizar esta opción.
3. Utilizar hardware y chips físicos: Finalmente se decidió esta opción para implementar el desarrollo por diversos motivos
    a. El margen de ganancia era mayor: Una vez descontando la inversión inicial del equipo físico y amortizando la del siguiente, el costo en hardware de cada SMS (con un ajuste de crecimiento adecuado) se vuelve muy bajo.
    b. Nos otorgaba mayor control (prácticamente total) en general sobre el sistema: Si había algún problema con algún chip de telefonía (ya sea por el chip físico o problemas con la red del operador que controlaba ese chip) podíamos simplemente remplazarlo (con un costo marginal comparado con el costo de cambiar de herramienta), o bien si teníamos

problemas con el hardware podíamos contactar directamente al fabricante para que nos ayudara a solucionar los problemas o hacer válida la garantía del producto. El hecho de poder comunicarnos sin intermediarios agiliza el proceso de resolución de un problema.

c. Contábamos con alguien con experiencia previa en el tema: La empresa cuenta con un aliado comercial que ya había incursionado en este negocio y ofreció vender su conocimiento tanto de su tabla comparativa de fabricantes de hardware, información de contacto, así como una plataforma muy básica pero funcional que hiciera uso de ese mismo hardware para iniciar el negocio en el menor tiempo posible debido a su experiencia previa y a código previamente desarrollado y reutilizado o portado.
d. Minimizaba el tiempo para salir al mercado: Gracias al punto anterior no era necesario invertir tiempo ni dinero en la curva de aprendizaje para el desarrollo de la primera versión del producto, lo cual minimizaba los tiempos para comenzar a vender el producto

Una vez tomada la decisión de desarrollar la plataforma utilizando hardware y chips físicos, se realizó todo el proceso requerido y se recibió el primer desarrollo previamente comentado. Este desarrollo constaba de una API REST que contenía lo siguiente:

1. Un método para envío de mensajes 1 vía: Como su nombre lo indica, este permitía recibir peticiones para envío de mensajes (por medio del protocolo HTTP en formato JSON debido a ser un servicio REST) y realizaba unas validaciones básicas de saldos y permisos.
2. Alta y consulta de usuarios: Un método que hacía posible la creación de nuevos usuarios (en caso de que un usuario con permisos de administrador los solicitara) o bien cualquier usuario pudiera consultar sus datos y saldos.

Es importante destacar que la persona que realizó este desarrollo no contaba con experiencia en el lenguaje de C# o en el .NET Framework aunque sí tenía un antecedente de programador. Una vez aclarado esto, el desarrollo recibido constaba de una aplicación de consola desarrollado en .NET Framework 4.5 que utilizaba un paquete NuGet (nombre del gestor de paquetes de asp.net) llamado OWIN para convertir una aplicación de consola en una API RESTful y otro llamado Topshelf para convertir una aplicación de consola en un servicio de Windows.

Para saber porqué es importante recalcar que inicialmente se utilizaron estas bibliotecas es importante explicar que en el .NET Framework existen “plantillas” (o templates) que permiten desarrollar esta funcionalidad de manera nativa y por lo tanto de manera más eficiente y con mayor rendimiento. Adicional a esto, se utilizaba una versión de la librería OWIN que contenía una falla que ocasionaba una fuga de memoria (memory leak) en el aplicativo. Es precisamente en este punto donde comienza mi participación profesional en el proyecto.

7. Contexto de la participación profesional

El objetivo inicial del proyecto era el desarrollo de una plataforma de envío de mensajería SMS basada en arquitectura de servicios de tal manera que los servicios fueran independientes y pudieran ser utilizados en caso de que alguien quisiera incorporar nuestro sistema de mensajería en una plataforma ya existente. Todo esto pensando en el ideal de las empresas pequeñas de un producto mínimo viable, para reducir el tiempo en llegar al mercado. En cuanto al tema técnico, se decidió utilizar una arquitectura de servicios porque permitía una mayor escalabilidad para la plataforma, así como un mayor porcentaje de reutilización de código, facilidad de lectura y eficiencia.

Como se comentó previamente, para iniciar este proyecto se compró hardware y un desarrollo muy elemental que nos permitió iniciar en negocio con mayor agilidad, pero inmediatamente nos dimos cuenta de que este desarrollo era demasiado básico, ineficiente y que no cumpliría con nuestras necesidades de seguridad, eficiencia, escalabilidad y reutilización de código, así que se propusieron un par de modificaciones (tanto en arquitectura como las herramientas en las que estaba desarrollado) que se detallarán en el siguiente apartado.

Para tener más claro el objetivo y el alcance de este proyecto en particular se definió el siguiente roadmap planeado sin fecha de término debido a la participación multiproyecto de todos los programadores en la empresa, así como por el cambio de prioridades según el surgimiento de proyectos internos o externos.

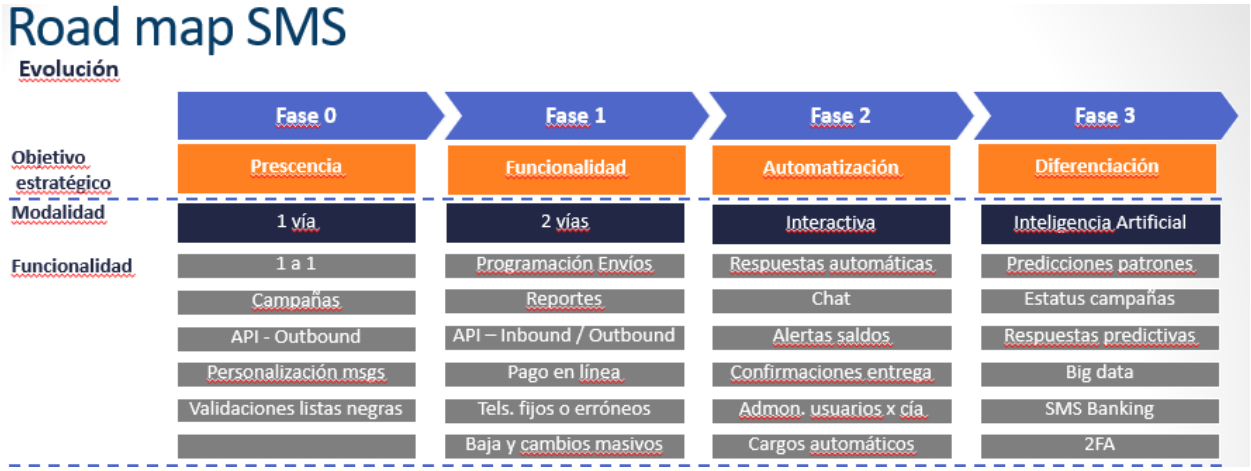


Figura 5. Road map del proyecto

Como se puede observar en la fila de objetivo estratégico, cada fase tiene un objetivo muy concreto que en ocasiones define la parte técnica y en otras la de negocio:

- En la fase 0 etiquetada como presencia, se enfoca en el acercamiento inicial al mercado, es decir en generar un producto mínimo viable (MVP por sus siglas en inglés) para entrar en el negocio. Más en específico en tener la parte base para el envío de SMS: mensajes 1 vía o también conocidos como únicamente de salida. Debido a que hay maneras más complejas de hacerlas, en ese apartado únicamente se consideran los mensajes uno a uno, es decir una petición por

mensaje a enviar. Una vez construido esto, se habla de campañas y personalización de mensajes, ambos apartados se refieren al desarrollo de una página web (o front) donde un usuario no técnico pueda hacer uso de nuestro servicio de mensajes. En el caso de la personalización de mensajes, es un aditamento a la creación de campañas para permitir la personalización del mensaje con base en los datos almacenados del usuario, como el nombre, apellidos o fecha de nacimiento. Finalmente se menciona la validación de listas negras, esto es simplemente la integración de una base de datos con números inscritos para no recibir mensajes. Es importante aclarar que este apartado no se realizó, principalmente decidido por el área administrativa debido al poco impacto del mismo y a la relación de confianza con nuestros clientes y de nuestros clientes con los suyos.

- En la fase 1 identificada como funcionalidad, define una etapa que se enfoca en construir funcionalidad adicional a la básica que puede ser de utilidad para nuestros clientes o que pueda ayudarnos a captar más clientes. Empezando por el apartado de 2 vías, API -Inbound / outbound o mensajes con respuesta, que permiten generar un canal de comunicación entre nuestros clientes y los suyos. Automatización de reportes, enfocándonos principalmente en porcentajes de éxito de las campañas 1 y 2 vías, así como un reporte de las respuestas de los usuarios, incluyendo el éxito de la campaña en cuestión. El apartado de teléfonos fijos o erróneos habla de que el API pueda identificar si se está recibiendo la petición para realizar el envío de algún SMS con un número que no puede existir (debido a que, por ejemplo, la lada no es correcta) o bien que existe, pero se trata de una línea asignada a teléfonos fijos y no móviles. Finalmente se implementó también un método para permitir a nuestros clientes de la plataforma realizar una baja o cambios masivos de la información de sus contactos, aunque por el momento únicamente se ha implementado el apartado de bajas masivo. En esta etapa se consideraba también el pago en línea, pero no ha sido implementado debido a actividades de mayor prioridad, así como debido a un conflicto interno con nuestro proveedor de pasarela de pagos.
- La fase 2 se enfoca en la Automatización, cuyo objetivo es dar más independencia a nuestros clientes y aumentar la efectividad y los canales para interactuar con sus propios clientes. De este apartado únicamente se han desarrollado las respuestas automáticas, el cual trata de un flujo automático de preguntas y respuestas, mismo que puede ser utilizado para realizar encuestas entre otras cosas. Esta fase también incluye la creación de un chat vía SMS, en este punto con interacción humana como otro canal de comunicación o de interacción con atención al cliente como un call center. También se mencionan alertas de saldo automatizadas para notificar que si se quiere seguir utilizando el servicio se necesitaría una nueva recarga o bien automatizarla por medio de cargos automáticos. Se consideró incluir confirmaciones de entrega, pero esta información únicamente está disponible para los operadores, así que se descartó por completo.
- La fase 3 y la última de este roadmap, habla de la diferenciación. En nuestro caso pensamos hacerlo por medio de inteligencia artificial (IA) para poder automatizar y predecir respuestas, así como incluir un esquema de seguridad robusto para entrar en la banca por SMS, sin embargo, aún no se ha detallado mucho esta fase debido a la lejanía de la misma y a los posibles cambios en el mercado. Se mencionaron las respuestas predictivas como una evolución a las respuestas automáticas y con ayuda de IA y Big Data, así como aumentar el esquema de seguridad para

entrar en SMS Banking (transferencias, consultas de saldo y demás operaciones por medio de SMS) y autenticación de 2 factores.

Actualmente nos encontramos en un híbrido entre la fase 1 y la fase 2 por temas de requerimientos del negocio y que algunas prioridades tanto del negocio como de la empresa han cambiado, acelerando algunas tareas y retrasando otras. Hasta la fecha, se han desarrollado los siguientes módulos:

- Mensajes 1 vía programables: mensajes únicamente de salida con la posibilidad de programar el envío para un día y hora en específico. Disponible tanto en la API como en el sitio web.
- Mensajes 2 vías programables: similar al punto anterior, pero en este caso para mensajes 2 vías, es decir con respuestas.
- Personalización de mensajes y campañas: en el caso del sitio web, permitir la personalización de un mensaje por medio de la utilización de comodines como {Nombre} que sea traducido por el nombre almacenado en la base de datos de ese número telefónico.
- Validación de números: revisar que el número recibido en la petición sea válido y pertenezca a una red móvil y no fija, rechazando aquellas peticiones que no cumplan con esto, con su debido mensaje de error al cliente
- Automatización de reportes interactivos: Incluir un apartado en el sitio web para que el usuario pueda realizar filtros sobre los reportes que desea generar.
- Respuestas automáticas: tipo de mensaje de 2 vías que con base en la respuesta del cliente puede tomar decisiones sobre qué mensaje enviar después. En este punto se incluyó un mensaje default para cada campaña en caso de que se necesite.

Durante el camino del desarrollo de este sistema, se identificó que la empresa no utilizaba ninguna metodología de desarrollo de software, así que se propuso y se comenzó el proceso para implementar SCRUM como metodología de desarrollo para poder dar seguimiento y una mejor planeación a los proyectos, convirtiéndome en Scrum Master de la empresa y desempeñando las tareas derivadas de esta nueva responsabilidad.

En cuanto al sistema, debido a que se recibió una versión alfa de la plataforma y para poder cumplir los requerimientos del negocio y el alcance del roadmap, fue necesario modificar la arquitectura y el código casi totalmente en al menos 2 ocasiones, algunas adecuaciones menores, otras mayores para poder seguir funcionando de la manera más óptima y permitir una mayor escalabilidad a esta plataforma que no se esperaba que fuera de alta transaccionabilidad (es decir, que ocurren muchas transacciones por minuto o paralelamente).

Además, en algún punto del desarrollo se detectó que el rendimiento del aplicativo era muy deficiente, así que se realizaron optimizaciones por medio de un par de mediciones y herramientas para este propósito que ayudaron a identificar las fallas que posteriormente fueron corregidas.

Finalmente, se probó el aplicativo para asegurarse de que tuviera la escalabilidad y personalización necesaria. Se decidió optar por un método de instancias adicionales para peticiones particulares de nuestros clientes y para permitir poder atender más clientes en caso de que un sistema se saturara,

mismo que permite también que se elabore, en un futuro, una API para manejar las diversas instancias que pueden correr en diversos servidores si el negocio así lo requiere. Para garantizar que una nueva instancia siempre pudiera ser montada, se generó la documentación pertinente con su debido nivel de detalle para facilitar que inclusive una persona que no forma parte del proyecto, pero contara con los permisos necesarios en los servidores, pudiera ejecutar el cambio. Adicionalmente, se generó también la documentación requerida tanto para el sitio web como para el API Restful. En el caso del sitio web, se creó un tutorial interactivo con la herramienta de javascript EnjoyHint, mientras que para el API se generó la documentación en github marcando claramente datos de entrada, de salida, códigos de error e incluso se incluyeron ejemplos en diversos lenguajes de programación para facilitar la migración o integración.

8. Metodología utilizada

Debido a que la empresa es pequeña, joven y no cuenta con un equipo robusto de desarrolladores, al inicio del proyecto no se utilizó ninguna metodología para dar seguimiento al estado del proyecto. Esto evidentemente ocasionó repercusiones en el mismo y en ocasiones generaba que no se conocía el avance real del mismo, lo cual ocasionó muchas veces que los tiempos se disparaban y no se pudieran ajustar a las planeaciones previas. Se intentaron implementar un par de metodologías, pero no había quien se comprometiera a implementarlas ya que el programador senior trabajaba principalmente vía remota y al no tratarse de metodologías estándar o contar con documentación de las mismas, yo no tenía suficiente conocimiento de ellas ni material didáctico lo suficientemente robusto y completo como para implementarlas, además de la identificación de algunas deficiencias.

Por esto mismo, decidí llevar la propuesta de implementar SCRUM al desarrollo de nuestros aplicativos para poder darle más certeza, agilidad y trazabilidad a nuestros proyectos. Fue difícil el comenzar a implementar una metodología de desarrollo de proyectos en una empresa que no tenía la cultura de ellos, en especial cuando los proyectos ya se encontraban a medio camino y sin documentación pertinente para lo mismo, pero sin lugar a duda el equipo y el área administrativa pudo ver el beneficio de seguir uno.

El primer paso para implementar la metodología fue una capacitación en cuanto a los conceptos básicos de SCRUM, asegurarse que quedaran claros los conceptos de sprint, daily stand up meeting, backlog, scrum master, scrum team, product owner, etc. Para cumplir este objetivo preparé una presentación basándome en el manual de Scrum conocido como Scrum Booko SBOK (Scrum Body of knowledge) y de certificaciones en línea, así como de un curso presencial. Adicional a la presentación, se generó material complementario como un diccionario de conceptos en Excel que sirviera como referencia para los roles de esta metodología, así como los conceptos específicos de la misma. Se compartieron también notas personales para las personas que quisieran profundizar más en el tema, así como el manual de la metodología en formato pdf y con anotaciones personales y resaltado de las partes esenciales para la metodología.

Después fue importante recalcar estos conceptos mientras iban ocurrieron en la implementación de la metodología en el día a día, es decir el convertirlos en hábitos y crear una cultura de metodología ágil dentro de la empresa. En aproximadamente un mes, el equipo ya dominaba estos conceptos y empezaban a respetarlos y ver la utilidad en los mismos. El mismo equipo pudo notar personalmente y en conjunto que utilizar esta metodología era útil para ellos y daba orden, así como claridad a las tareas que tenían que realizar, tanto que en alguna ocasión llegaron a comentar que esta metodología era muy buena para los desarrolladores ya que les permitía enfocarse en el desarrollo de código en lugar de cambiar a tareas más administrativas o de recopilación de información o vistos buenos, de las cuales el Scrum master se responsabiliza dándole seguimiento al sprint.

Otro punto importante pero un poco ajeno a la implementación de SCRUM, es que para respetar la metodología se ayudó a reforzar el uso de al menos 3 servidores durante el desarrollo de cualquier aplicativo para aumentar la calidad y disminuir los errores:

- **Servidor de desarrollo**: En este servidor era básicamente un servidor donde se probaban los aplicativos en el sistema operativo en el cual iban a correr (Windows server) y verificar que todo funcionara en ese sistema operativo y en una máquina con una RAM, ROM y procesador similares a aquel en el cual se iba a montar productivamente. Además, se tenía una base de datos alojada en este servidor para que se pudieran realizar pruebas de la misma y de la interacción con el aplicativo, eliminando la necesidad de que cada desarrollador necesitara tener su base de datos local.
- **Servidor de pruebas**: Este servidor era un servidor idéntico al de producción en casi todos los sentidos (misma RAM, misma ROM, mismos aplicativos, mismos controladores, etc.) exceptuando evidentemente la carga de un aplicativo en uso (debido a que no había usuarios utilizando este sistema, únicamente para pruebas). El objetivo de este equipo era que una vez que se hubieran probado los cambios en el servidor de desarrollo, se tenían que aplicar esos cambios aquí también. Como la base de datos de este servidor era diferente a la de desarrollo, era necesario tener preparados los scripts a ser aplicados, preparando y en algunos casos generando un manual completo para poder realizar estos cambios, sirviendo también como confirmación de que se tenían todos los requerimientos necesarios de base de datos, instalación de controladores, etc. para que el sistema funcionara correctamente en el servidor productivo. Este es el servidor donde obteníamos los vistos buenos de los clientes que nos permitían actualizar sus aplicativos. Cabe aclarar que no todos los desarrolladores tenían acceso a este servidor para cuidar la integridad del mismo. Este servidor es un servidor virtualizado privado (VPS por sus siglas en inglés) que se contrata de una compañía de hosting norteamericana que tiene un porcentaje de uptime del 99.9%. Se decidió contratar esta empresa debido a sus costos, su fiabilidad y a la cercanía con México y las ventajas en cuanto a latencia que eso implica.
- **Servidor productivo**: En este servidor estaban alojadas las versiones finales de los aplicativos que nuestros clientes usaban. Únicamente el desarrollador sr. y yo (como administrador auxiliar del servidor) teníamos acceso a este servidor para hacer las liberaciones, además de que por regla únicamente podíamos copiar archivos que estuvieran previamente en el servidor de pruebas, para garantizar que fueran los correctos y previamente aprobados por los clientes. Evidentemente este servidor también tenía una base de datos independiente de las anteriores. Cabe aclarar que se instauró una regla para que las liberaciones a productivo únicamente fueran realizadas en la noche o muy temprano para que no afectaran el periodo de operación de nuestros usuarios, así como para tener oportunidad de solucionar algún contratiempo en caso de ser necesario y minimizar al máximo el tiempo en que el aplicativo no estaba disponible. Este servido también es un VPS y se encuentra alojado con la misma empresa, pero no es el mismo servidor físico.

Estos dos fueron los principales cambios en la cultura empresarial que se implementaron durante mi participación en el proyecto.

9. Desarrollo del sistema

9.1. Tecnología utilizada

Antes de iniciar a detallar el proyecto es importante hacer la aclaración del porqué de la tecnología utilizada. Hoy en día uno de los lenguajes más utilizados para desarrollar este tipo de aplicativos es java debido a su compatibilidad en diversos ambientes, y el hecho de no utilizarlo es cuestionado en algunas ocasiones. En este caso su servidor no tomó la decisión de qué plataforma o lenguaje utilizar, sin embargo, sí puedo compartirles cómo se tomó la decisión para esto, y cabe aclarar que antes de entrar a laborar en la empresa y en el proyecto, yo no tenía experiencia previa con el lenguaje más que un par de tutoriales y códigos básicos.

La decisión de utilizar ASP.NET, el .NET y Entity Framework y una base de datos SQL Server (opción privativa de Microsoft) se tomó principalmente por 2 factores que de cierta manera están entrelazados:

- El único programador senior de la empresa (uno de los fundadores y actualmente director asociado) tenía amplia experiencia en el lenguaje, el framework y manejo de servidores Windows. Debido a la experiencia en el mantenimiento, seguridad e instalación del servidor, era natural el paso de elegir Windows Server como tecnología para hospedar nuestros aplicativos. La toma de esta primera decisión tomaría un gran peso en las consecuentes. Además, como el programador senior tenía muchos años de experiencia con el mismo y debía de asumir un rol de líder en la empresa, utilizar la tecnología con la que estaba más familiarizado no era una decisión difícil
- El segundo motivo para utilizar la tecnología de Microsoft en lugar de otra fue que como ya se contaba con un servidor virtual contratado con Windows server, era poco probable que el aplicativo fuera a migrar a otro tipo de servidor. Debido a esto se decidió desarrollar en .Net para aprovechar la compatibilidad, fácil liberación y optimización de nuevos aplicativos a este tipo de servidores debido a la optimización de .NET para plataformas Windows.

9.2. Definición de la arquitectura del sistema

Regresando al tema del aplicativo y como se mencionó previamente, se recibió código de una aplicación que únicamente enviaba mensajes de 1 vía. Para poder atacar y cumplir con lo ambicioso del roadmap, se inició por definir la arquitectura que cumpliera con nuestro alcance del proyecto. Para esto se tomó la decisión de continuar con una arquitectura de servicios. En algún momento de la re-arquitectura (misma que se hablará más adelante) también se consideró el tema de micro servicios, pero debido a la complejidad, curva de aprendizaje, costo del aplicativo y su margen de ganancia tan pequeño, se decidió que la granularidad de servicios era la adecuada para nuestros propósitos. Al final, la arquitectura quedó de la siguiente manera:

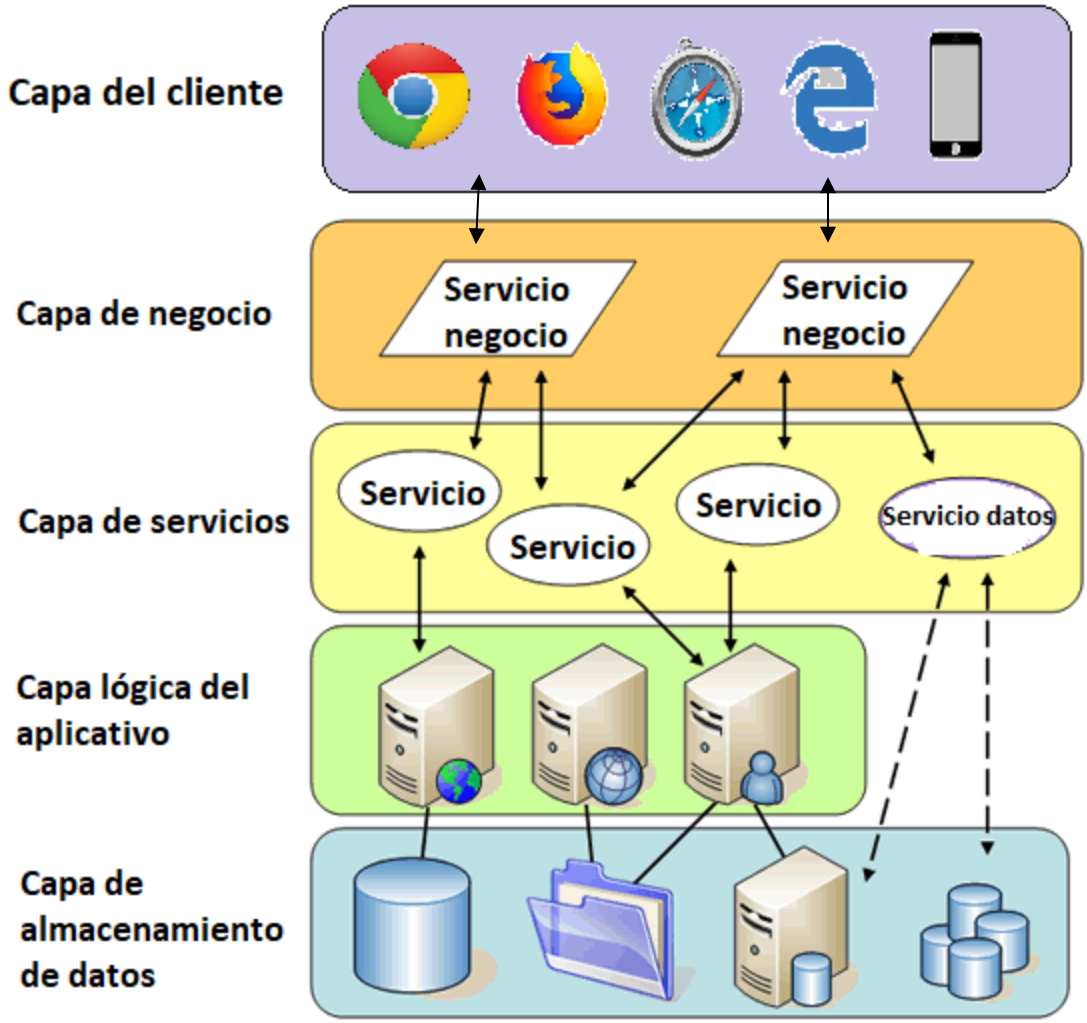


Figura 6. Arquitectura de servicios

Como se puede apreciar en la imagen, es una arquitectura bastante estándar de servicios, donde se separa el aplicativo en diversas capas:

- **Capa de datos**. Como su nombre lo indica, es el lugar donde se almacena la información. Esta información puede estar almacenada de diversas maneras como lo son una base de datos, archivos, un servidor con otro tipo de datos acompañado de una base de datos, un conjunto de bases de datos (en el caso de microservicios o datos distribuidos). En el caso de este aplicativo en particular, se utilizó únicamente una base de datos SQL Server debido a que en nuestra plataforma de SMS en la actualidad únicamente se puede hacer el envío de texto.
- **Capa de lógica del aplicativo**. Esta es la capa del software, que es decir dónde está alojado el aplicativo y es la conexión entre los datos y los servicios interno. En esta capa se incluyen temas innatos de la plataforma como las variables de sesión, los archivos de configuración, entre otras cosas.

- **Capa de servicios**. Aquí es donde se exponen los servicios de acceso únicamente interno, aunque en algunas ocasiones existe un servicio sobre estos mismos en la capa de negocio, pero los permisos para consumir cada uno de estos es diferente.
- **Capa de negocio**. En esta capa se encuentra toda la lógica del backend (API Restful) y del front end). Estos son los servicios a los que tiene acceso el cliente e incluye toda la lógica de negocio, las restricciones, permisos, cobros, personalizaciones, etc.
- **Capa del cliente**. En este caso solo se incluye el front, es decir el sitio web en el cual los usuarios pueden utilizar nuestros servicios sin conocimientos técnicos y es todo lo que es accesible por medio de un navegador web.

9.3. Modificación y optimización en primera etapa.

Una vez definida y descrita la arquitectura (tomando en cuenta que inicialmente solo contábamos con una versión atómica de las 3 capas más bajas) se comenzó a trabajar en la modificación y optimización de la plataforma. Primero se hicieron unas modificaciones menores, agregar un par de funcionalidades, modificar otra que estaba causando conflictos, etc. Una de estas funcionalidades que estaba ocasionando conflictos era qué no se estaba haciendo un correcto tratamiento del “status report”. En el caso de SMS este “status report” es un pequeño reporte que nos entrega el hardware encargado de hacer el envío del mensaje. En este reporte se incluye qué sucedió con el mensaje, es decir si fue aceptado o rechazado por el operador móvil al intentar realizar el envío. En la versión recibida se intentó integrarla (había evidencia en funciones no utilizadas) pero al no lograr hacerla funcionar se ignoró completamente esta funcionalidad. Para implementarla correctamente, se tuvo que dar lectura a los manuales de operación y del API del hardware, inclusive tuvo que tenerse una reunión virtual con el fabricante debido a la desactualización y falta de información de la documentación recibida. Una vez implementada, se tomó la decisión administrativa de únicamente cobrar aquellos mensajes que pudieron ser enviados exitosamente, como medida de diferenciación respecto a la competencia.

Una vez que se logró la estabilización de la plataforma y la corrección de algunos errores, comenzamos a observar que la aplicación estaba consumiendo demasiados recursos de memoria (RAM), lo cual me llevó a hacer un análisis más profundo para encontrar la causa de esto. Fue en este punto que después de mucha investigación y análisis de rendimiento, se descubrió que un causante de esto era que se hacía uso de una librería llamada Topshelf que se encargaba de convertir una aplicación de consola en una API Restful, misma que contenía un memory leak. Primero se intentó actualizar la librería a la última versión en la cual según la información recabada ya no ocurría este error, sin embargo, esta actualización no generó una mejora significativa en cuanto al rendimiento y uso de memoria.

Después de hacer un análisis de las alternativas entre si seguir invirtiendo tiempo en encontrar un error muy específico de un sistema muy particular y que no estaba diseñado adecuado a los requerimientos o migrar el aplicativo a otro frameworky considerar el tiempo que nos tomaría cada una de estas opciones, se decidió migrar la herramienta de un programa de consola con librerías para transformarlo en una API a una API REST nativa del framework .NET. Esto principalmente para empezar a hacer un correcto uso de las opciones que nos ofrece el framework, y por otro lado para evitar en la medida de lo

posible, el uso de herramientas externas y minimizar la posibilidad de errores tanto en manejo de memoria y recursos como en vulnerabilidades cibernéticas.

Para aprovechar este cambio de plataforma, en esta etapa también se decidió dividir la capa lógica en 2, ya que al ser un aplicativo de alta transaccionabilidad el hecho de recibir las peticiones para enviar mensajes y generar el envío en ese mismo momento estaban retrasando la recepción y procesamiento de más peticiones nuevas, inclusive en algunas ocasiones y debido a que se trataba de peticiones asíncronas se estaban ocasionando algunos deadlocks. El acercamiento que se tomó fue dividir en 2 diferentes el envío de mensajes 1 vía, la parte que nuestros clientes ven, que es el API que se encarga de recibir los mensajes, realizar las validaciones pertinentes y guardarlos en una base de datos, y en la segunda un servicio de Windows que se encarga de obtener la información de la base de datos, procesarla enviándola al hardware Gateway para su posterior envío y actualización del estatus del mensaje. Es importante destacar que, aunque se realizó la separación de estos procesos, se siguió respetando la arquitectura de servicios. El diagrama de esta nueva distribución de la arquitectura quedó de la siguiente manera:

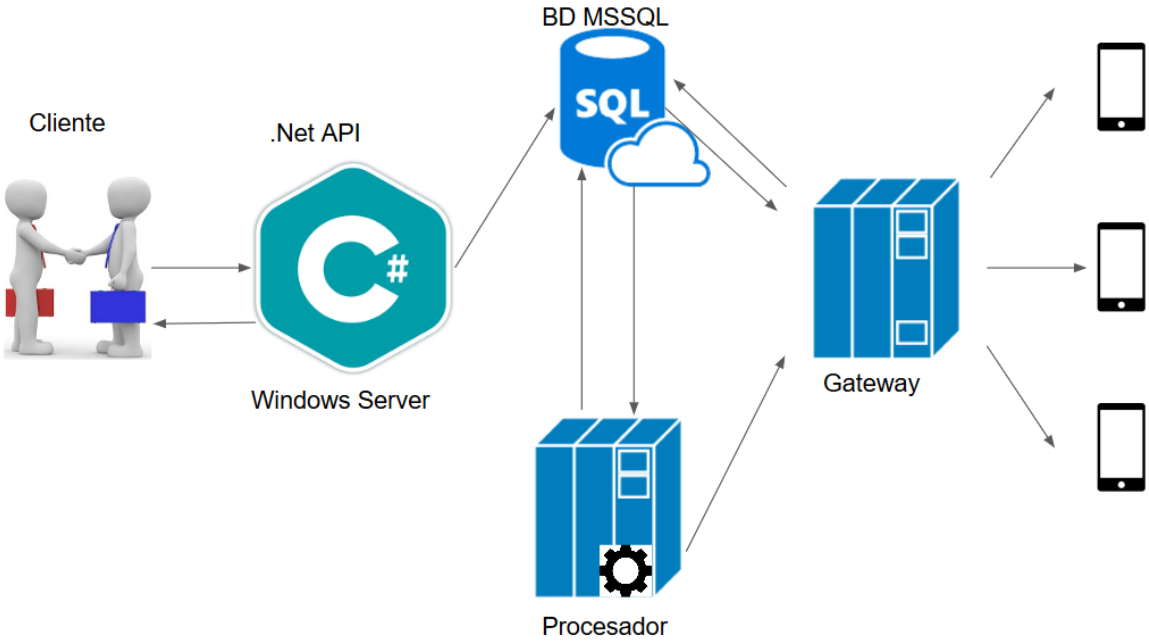


Figura 7. Nueva arquitectura del proyecto

Con esta nueva versión, se logró optimizar el tiempo de respuesta a nuestros clientes ya que no es necesario que esperen la respuesta del Gateway de que el mensaje fue encolado. Esta modificación tuvo un ligero impacto en cuanto a los tiempos de salida de los mensajes, sin embargo, como el verdadero envío del SMS es realizado por el Gateway y puede llegar a tomar hasta 15 segundos por mensaje, al trabajar con un volumen elevado de transacciones este tiempo se vuelve despreciable debido a que la principal limitante de tiempo para el envío de los mensajes es el Gateway, derivada de los tiempos de respuesta del operador móvil. A pesar de esta ligera desventaja, los beneficios fueron mucho mayores

ya que ayudó reducir cuellos de botella e inclusive el problema de los dead locks ya que a pesar de que las peticiones seguían siendo asíncronas únicamente se trata de las inserciones ya que el procesamiento de los mensajes era realizado por el segundo programa.

En este punto también observamos que el Gateway con el que estábamos trabajando tenía un grave error, en ocasiones cuando recibía más de 100 peticiones por número a enviar, el hardware (probablemente ocasionado por un problema de memoria) fallaba y se reiniciaba, perdiendo registro de los mensajes que aún no enviaba y ocasionando problemas con la continuidad del servicio. Por esto mismo y dado que el fabricante no nos ofreció una respuesta afirmativa y satisfactoria, se decidió limitar la cantidad de peticiones que se enviaban al Gateway, de tal manera que este mismo nunca contaba con más de 20 mensajes encolados por cada número de salida. Además, esto nos ofrecía un mayor control para saber qué mensaje realmente estaba cerca de ser enviado y cuál no.

9.4. Desarrollo de un portal Web

Una vez que se validó que esta versión era estable y efectivamente tenía un aumento en el rendimiento tanto de memoria como de velocidad, se comenzó a construir nueva funcionalidad. Debido a que todavía nos encontrábamos en la etapa de presencia, la decisión natural a tomar era el desarrollar un portal web para poder llegar también a personas no técnicas que estuvieran interesados en el envío masivo de SMS.

Para poder desarrollar esta parte, debido a que aún no teníamos clientes pero queríamos seguir la filosofía del mínimo producto viable, decidimos seleccionar los módulos que consideramos eran los más básicos para minimizar el tiempo de salida al mercado. Para seleccionar los módulos se realizó una labor de análisis de la competencia. Se seleccionaron a los competidores más grandes del mercado tanto nacional como internacionalmente para poder analizar qué tenían todos en común, sus oportunidades y decidir lo más básico que necesitaba una plataforma de esta índole. Algunos de los competidores analizados fueron Nexmo, Twilio, sms masivos, entre otros. Después de realizar este análisis se concluyó que los módulos esenciales para estos sistemas son los siguientes:

- **Un dashboard con numerología básica**. Esto simplemente es un tablero sencillo que incluye la numeralia histórica del uso de la plataforma. En nuestro caso optamos por mostrar únicamente 6 apartados: mensajes enviados, mensajes no enviados (o fallidos), saldo restante, campañas lanzadas exitosamente, campañas creadas pero nunca enviadas y el número total de contactos en el sistema. Adicionalmente se creó una sección en esta misma pantalla que nos permitiera realizar el envío de un mensaje individual sin la necesidad de dar de alta un contacto, agilizando las pruebas para nuevos usuarios pero principalmente con el objetivo de ser como una

herramienta de ventas y para demostraciones.

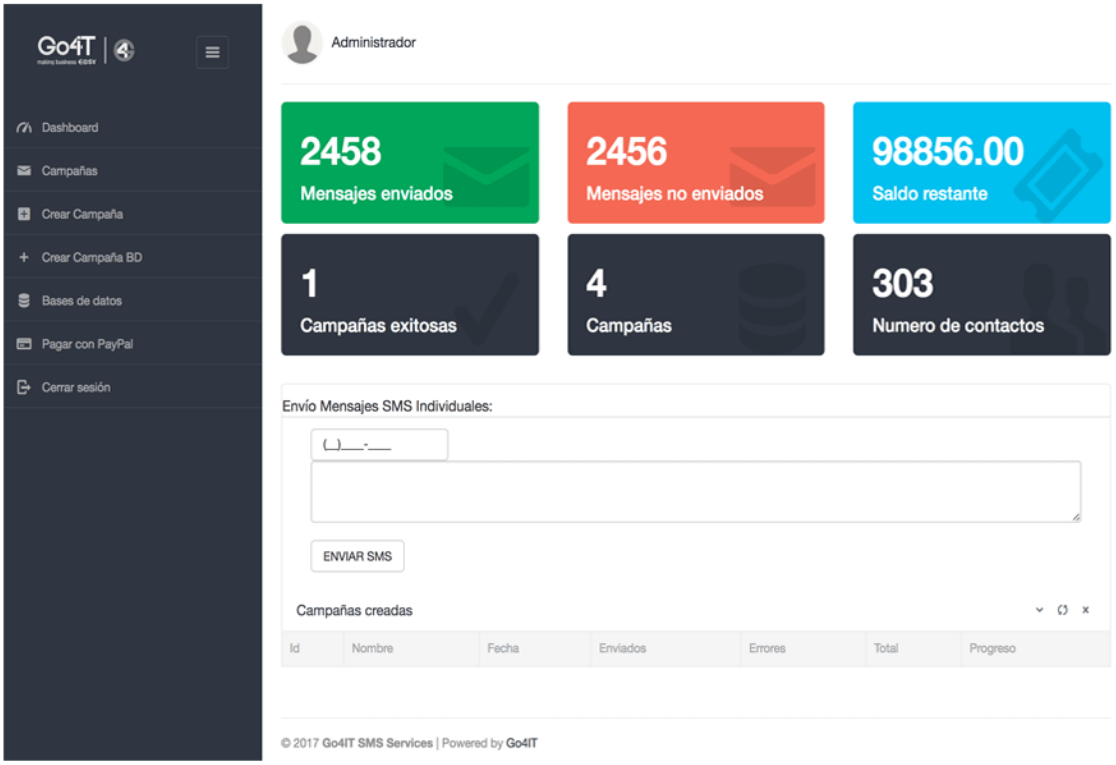


Figura 8. Dashboard del portal web

- **Un apartado para cargar y editar contactos.** Esta sección es una pequeña base de datos para cada cliente en donde puede almacenar sus contactos, cargándolos uno a uno o bien por medio de un archivo Excel para agilizar el proceso. Evidentemente al realizar la carga de contactos se realiza una validación de los datos. En este caso se valida que el número de celular sea realmente un número y no una cadena y que tenga una longitud de 10 números, este es el único dato obligatorio para el sistema. También se genera una validación de las fechas, asegurarnos de que realmente sean formato fecha en el archivo de Excel. Como se puede observar entre la rotulación de las columnas y los datos, existen unas pequeñas cajas de texto que sirven para realizar filtros, esto se logró con ayuda del framework de Telerik devcraft para .NET. Estos filtros sirven para encontrar un contacto en específico, o bien un conjunto de contactos que cumplieran con las características del filtro, esto podía utilizarse para confirmar que un número se encontrara dentro del sistema, o bien para actualizar un registro en automático. Debido a la naturaleza del mínimo producto viable, no se desarrolló la funcionalidad para editar o borrar algunos contactos en bloques o desde un archivo Excel. Sin embargo, sí se desarrolló el apartado para exportar toda la base a un archivo Excel. Combinando esto con el botón de borrar todos los registros, era posible realizar “ediciones” en bloques sin invertir demasiado tiempo en desarrollar esta funcionalidad, así como todas sus validaciones necesarias. Un último detalle que se desarrolló en este apartado (aunque fue en un momento más adelante en el tiempo) fue la validación para garantizar que un número no existiera 2 o más veces en el sistema, evitando

dobles costos al usuario por cargar una base de datos que no se encontrase depurada.

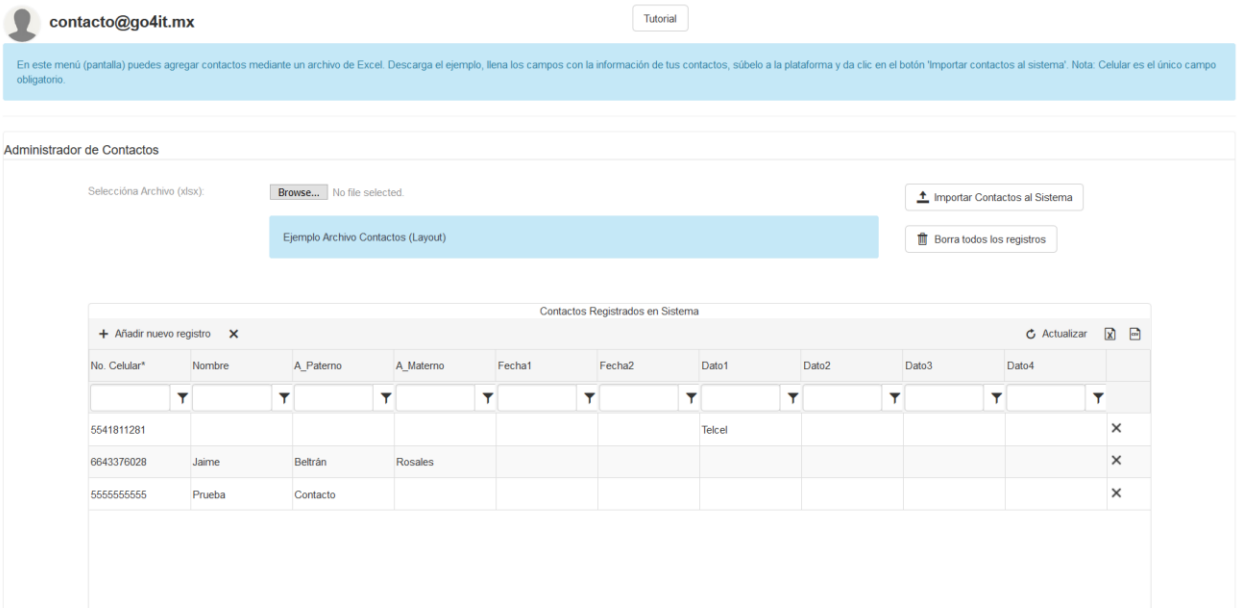


Figura 9. Agenda del portal web

- **Envío de campañas masivas con base en los contactos**. Finalmente, el apartado más importante en una plataforma de envío masivo de SMS es precisamente en el cual es posible realizar esta funcionalidad. Esta pantalla es realmente un “dos en uno”, ya que en una parte se selecciona a los contactos y el mensaje, y en una segunda se visualiza la campaña antes de ser enviada, así como habilitando una posible edición. Como se puede ver en la primera imagen (cabe aclarar que hay apartados de esta pantalla que se desarrollaron después de este acercamiento inicial, sin embargo para evitar regresar a hablar de ese apartado, se han incluido las imágenes actuales y se explicará la funcionalidad de hoy en día), comenzamos por asignarle un nombre y un mensaje a la campaña, así como decidir si se trata de un mensaje flash (o de clase 0) y si es necesario que el mensaje conserve los acentos (estos dos últimos puntos desarrollados posteriormente debido a un tema que se explicará posteriormente). Adicional a esto, se observa el mismo grid que se apreciaba en el apartado de los contactos. En este caso se adicionó una columna del lado izquierdo para poder seleccionar los contactos a los cuales se desea que se realice el envío. Es posible (y era uno de nuestros objetivos principales) el permitir que nuestros usuarios pudieran simular el uso de listas por medio de filtros. Adicionalmente, es posible seleccionar todos los contactos del grid (incluyendo cuando se realizó un filtro) por medio del checkbox que se encuentra junto a los encabezados de cada columna, esto para facilitar la selección de algunos o todos los contactos, según aplique.

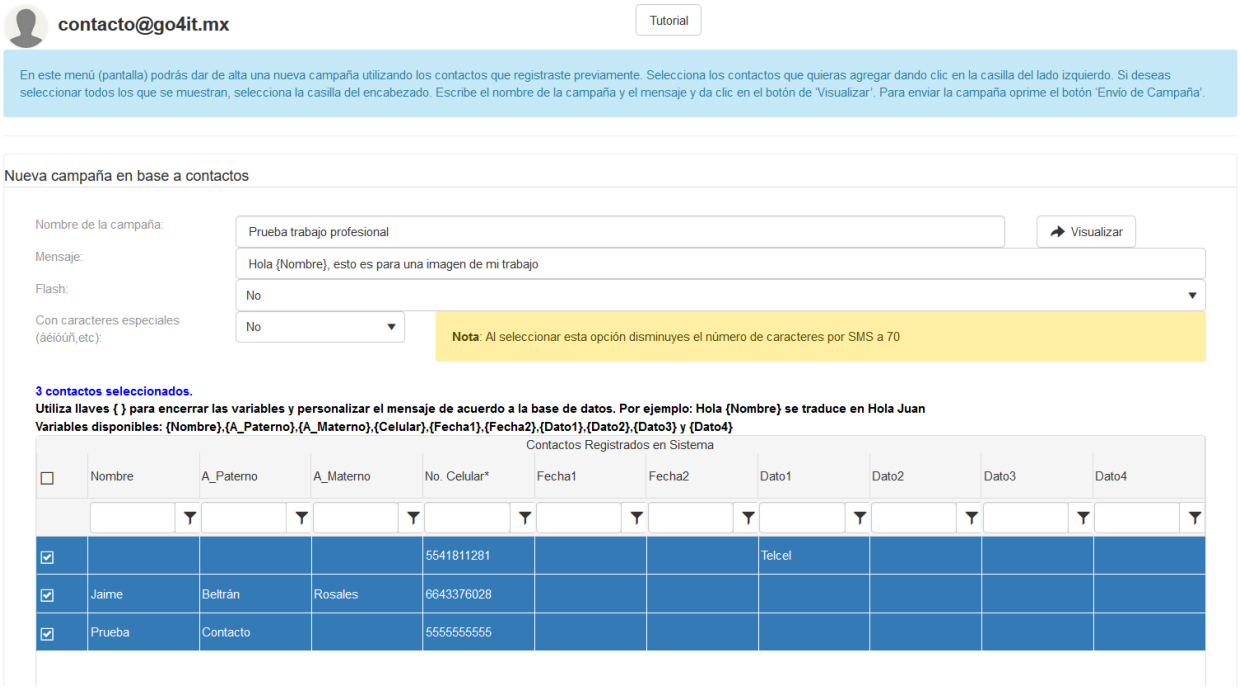


Figura 10. Pantalla para envío de campañas masivas.

Una funcionalidad muy interesante y que no se observó en la competencia es el hecho de poder usar variables para personalizar el mensaje. Como se puede analizar en la siguiente imagen, el mensaje generado coincide con los datos del usuario al que pertenece ese número. Esta es una funcionalidad que se observa mucho en plataformas de mailing pero en su momento no la habíamos visto en otros sitios de envío masivo de SMS. También posteriormente y para hacer la plataforma más transparente y visualmente atractiva, se agregaron las tarjetas que le indican al usuario el saldo actual, el costo de la campaña y su saldo posterior al envío, esto con la finalidad de que se pudiera tomar una decisión informada y totalmente clara sin la necesidad de regresar a la pantalla inicial a ver el saldo.

Es importante destacar que para hacer el cambio de una pantalla a otra no es necesario recargar toda la página; al igual que en html, asp.net tiene algo llamado postback que es el equivalente a Ajax, es decir que nos permite recargar únicamente una fracción de la página, en este caso hacemos un cambio a la información que contiene el grid. Esto se realizó con ayuda de elementos de telerik conocidos como RadAjaxManager y la configuración de Autopostback=”true” al dar click en el botón de visualizar.

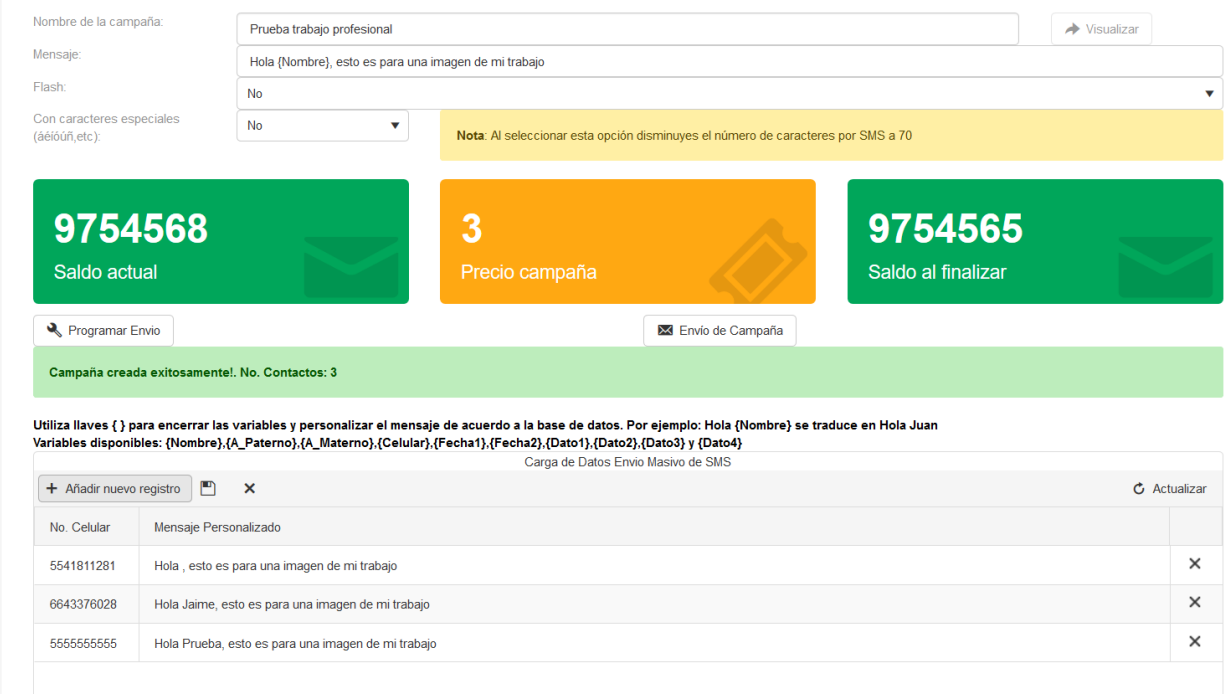


Figura 11. Pantalla para envío de campañas masivas con visualizadores de saldo y precios.

Tomando en cuenta que quizás hubo algún error en el mensaje escrito y para no tener que realizar la corrección de cada contacto individualmente, se incluyó la funcionalidad para editar el mensaje para todos. Una vez que se modifica el texto del mensaje, se desactiva el botón para enviar la campaña y es necesario dar click en el botón de actualizar (cambiando el texto del botón de visualizar y reactivándolo).

Finalmente, como se puede ver en todas las pantallas, en la parte superior se tiene una leyenda en color azul para indicar una pequeña guía sobre cuál es el objetivo de esta pantalla y un botón para un tutorial rápido (que explicaré más adelante).

Además de la parte visual, es importante destacar una funcionalidad que no es visible para el usuario pero fue uno de los pilares al construir esta plataforma. Se trata de la manera en que se realiza el envío de la campaña. En lugar de realizar el tradicional envío desde el código del cliente (por medio de javascript) o inclusive del lado del servidor, pero con la necesidad de mantener la sesión iniciada y tener una conexión estable de internet se decidió el priorizar la seguridad del envío de los mensajes. Para esto, se desarrolló una pequeña aplicación de consola que es invocada cuando el usuario da click en enviar campaña. Esta aplicación corre en el servidor que se encuentra hospedando el aplicativo. De esta manera garantizamos que, aunque el cliente haya perdido la conexión a internet o tenga intermitencia en el servicio, el envío de la campaña puede continuar sin problemas, un punto muy importante en nuestro país donde la conexión a internet en una oficina o en los hogares no es muy estable o rápida. Gracias a esta medida podemos asegurar que la campaña siempre sea enviada y que la velocidad de envío no depende del ancho de banda del cliente.

Una vez que se tenían tanto la API (enfocada en clientes técnicos que quisieran integrarnos a su sistema) y el portal web (enfocado en clientes sin conocimientos técnicos) podíamos afirmar que ya teníamos un producto mínimo viable para salir a ofrecer nuestra plataforma al mercado, es decir, habíamos cumplido con el objetivo de la fase que era la presencia. Fue en este punto que se pudo invertir más tiempo en desarrollar nueva funcionalidad para la API (que posteriormente sería integrada al sitio web).

9.5. Incorporación de mensajes 2 vías

La primera funcionalidad en desarrollarse fueron los mensajes 2 vías. Debido a que el envío de mensajes sale desde diversos números para agilizar el envío, surge el problema de cómo enlazar un mensaje enviado con su respectiva respuesta. Para desarrollar esta funcionalidad hubo diferentes puntos de vista y la discusión se enfocó principalmente en 2 diferentes opciones:

- **Utilizar palabras reservadas por campaña**. Esta opción consistía en solicitar al cliente que asignara una palabra clave a su campaña y solicitara a las personas interesadas en responder primero utilizando esa palabra. Por ejemplo:
  Mensaje: "Responde a este mensaje con tu partido político favorito de la siguiente manera: VOTO PAN, VOTO PRI, VOTO MORENA, ETC."
  Respuesta válida: VOTO PRD
  respuesta inválida: PRD
  respuesta inválida: PRD VOTO
  Esta opción facilita el labor de programación ya que nos da la herramienta de un identificador de campaña, de tal manera que no importa el número al que respondan seríamos capaces de identificar a qué campaña pertenece, aunque se tendría la limitante de que no podrían usarse la misma palabra reservada para dos campañas mientras ambas siguen vigentes. La principal desventaja de esta opción es que obligamos al usuario y a nuestros clientes a gastar más caracteres en explicar cómo se tiene que responder, así como que dificultamos al usuario la comprensión y facilidad para responder.
- **Enlazarlos por medio del número del que salió y al cual se respondió**. Esta alternativa tiene la gran ventaja de que no requiere de ninguna indicación especial para los clientes o usuarios, facilitando su uso, pero descargando esa complejidad en la labor de programación. Para permitir que esto ocurra es necesario mantener una relación de los números que realizaron el envío hacia qué números y enlazarlo con la respuesta haciendo esto mismo pero en sentido contrario. Sin embargo, esta opción tiene la desventaja de que la relación número de salida A con número de envío B no puede ocurrir para ninguna otra campaña. Es decir, si el cliente 1 realizó un envío al número B y este salió por número A, es importante que, si algún cliente 2 quiere realizar un envío al mismo número B, este no debería de ser enviado por el número A, de lo contrario no podríamos diferenciar hacia qué campaña está respondiendo ese usuario. Esto implica que para poder utilizar este método son primordial 2 cosas:
  1. **Es indispensable realizar esta validación.** Es decir, antes de poderle aceptar el mensaje a nuestro cliente, es necesario el revisar si cumple con que tengamos algún chip/números disponible A que aún no haya enviado un mensaje hacia ese número B. Para poder cumplir adecuadamente con esta validación, es necesario realizar un cálculo

de nuestros números disponibles para hacer envíos dos vías y el volumen promedio que se está manejando.

2. **Las campañas deben de tener vigencia.** Si no se cumple con esta regla, estaríamos bloqueando permanentemente la relación de mensajes de un número A hacia un número B. Esta limitante no nos pareció tan crítica ya que es natural que las respuestas de los usuarios pierdan valor después de cierto tiempo. Por ejemplo, si alguno de nuestros clientes está enviando una campaña para conocer por qué candidato a presidente votarían, la respuesta de los usuarios no tiene ningún valor una vez pasada la elección, o bien después de que se cierren las cifras para sacar las siguientes encuestas. Para que la plataforma pueda funcionar en ambos escenarios sin problemas, se decidió que este pueda ser un parámetro variable de cada campaña

Una vez habiendo descrito el panorama, se pensó muy detalladamente sobre las implicaciones de cada una de estas opciones y se pusieron las ventajas y desventajas de cada una de estas en una balanza, inclinándonos por la segunda opción de enlazar el número que realizó el envío con la respuesta, esto principalmente para facilitar a nuestros clientes y a los clientes de nuestros clientes el uso de la herramienta y evitar limitar el número de caracteres que pueden usar en los SMS imponiéndoles una obligación, pero también porque tanto en la empresa como yo personalmente, consideramos que la tecnología y en general la ingeniería siempre debe de facilitar y/o mejorar la vida (o los procesos) de nuestros clientes.

Una vez tomada la decisión, se procedió a diseñar el sistema. Para esto uno de los primeros pasos fue hacer un diagrama de la arquitectura. Para la primera parte del proceso (el envío del mensaje) se decidió utilizar la misma arquitectura que para los mensajes una vía, solamente que guardando la información en una tabla diferente ya que era necesario almacenar más información de cada mensaje y para evitar que el elevado volumen y tamaño de mensajes 1 vía interfiriera con 2 vías. Una vez decidido esto, se realizó el diagrama de la arquitectura que quedó de la siguiente manera:

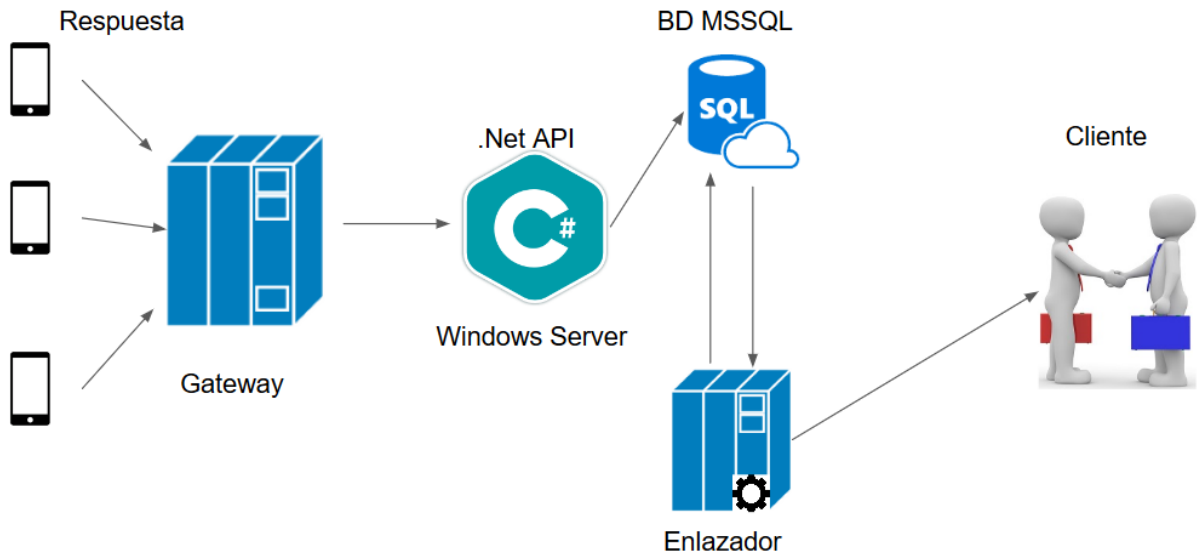


Figura 12. Diagrama de la nueva arquitectura

Como se puede observar, la arquitectura es muy similar a la de una vía, pero en sentido inverso. El flujo ahora inicia por la respuesta de los usuarios, misma que es capturada por el Gateway de SMS. El siguiente paso fue un poco más interesante. En la revisión inicial de la API uno de mis compañeros laborales había encontrado un método para poder leer los mensajes que había recibido el Gateway, sin embargo, nos parecía un método ineficiente ya que requería tener otro programa corriendo en todo momento, solicitando los últimos *n* mensajes recibidos de cada chip/número en el modem. Esto implicaba realizar una comparación con la base de datos cada vez que se realizaba un ciclo del proceso para revisar si se trataba de un mensaje nuevo o era uno previamente guardado, la cual era altamente ineficiente ya que el Gateway no nos otorgaba un identificador único para cada mensaje y se tenían que hacer comparaciones del número del que recibió el mensaje, el número del usuario que lo envío y el mensaje que contenía.

En este punto nos volvimos a poner en contacto con el fabricante para que nos explicara si existía otra manera para poder guardar los mensajes recibidos en cuanto llegaran al Gateway, a lo que nos comentaron y compartieron nueva documentación en la cual indicaban una configuración para realizar en el Gateway de tal manera que realizara un callback a una URL con cierta información cada vez que se recibía un nuevo mensaje.

Esto último nos permitió desarrollar una nueva API para uso únicamente interno y separada de la anterior para poder garantizar que el tráfico hacía la recepción de nuevas peticiones de envío de mensajes no iba a retrasar la recepción de mensajes recibidos en esos chips. Al igual que la anterior, se decidió que esta API únicamente guardara información en la base de datos para su futuro procesamiento, esto para evitar saturaciones del aplicativo en caso de recibir muchos mensajes y peticiones de nuevas recepciones al mismo tiempo.

Una vez que se tenía la información almacenada en la base de datos, se creó otro servicio de Windows que obtuviera esa información y la comparara contra la tabla de dos vías, para verificar si esa respuesta correspondía a un mensaje esperado de una campaña que continuara vigente, en caso de ser afirmativo enlazaba el mensaje con la respuesta y posteriormente enviaba una petición POST con un JSON a una URL que el cliente nos proporcionó para recibir el log de respuestas al momento, y detonar acciones en caso de ser necesario.

Para este periodo en el tiempo, estábamos relativamente cerca al periodo electoral y surgió la idea de crear una herramienta que nos ayudara a automatizar las encuestas y sus respectivas respuestas, así como contabilizar cuántas personas habían respondido cada una de las opciones, o bien ninguna de ellas. Terminamos bautizando este producto como respuestas automáticas, y terminó siendo un producto muy interesante por la tecnología que utilizamos para implementarlo.

El producto consistía en un listado de todos los mensajes y su relación entre los mismos, de tal manera que de un mensaje 1 con cierta respuesta te llevara a un mensaje 2, con otra respuesta te lleve al mensaje 3 y si no cumple con ninguna de las opciones te lleve al mensaje 4, y lo mismo para el siguiente mensaje, en caso de ser necesario.

9.6. Uso de máquinas de estado

Inicialmente no estábamos seguros de cómo resolver esta problemática, pero después de ver una representación visual de nuestra idea y recordando un poco algunas de mis clases en la facultad fue una decisión natural el seleccionar las máquinas de estado como tecnología para implementar este producto. Al escuchar esta opción todos estuvimos inmediatamente de acuerdo en que era la mejor opción para implementar esta funcionalidad. Por ejemplo, una representación visual de una encuesta relacionada con el medio de transporte:

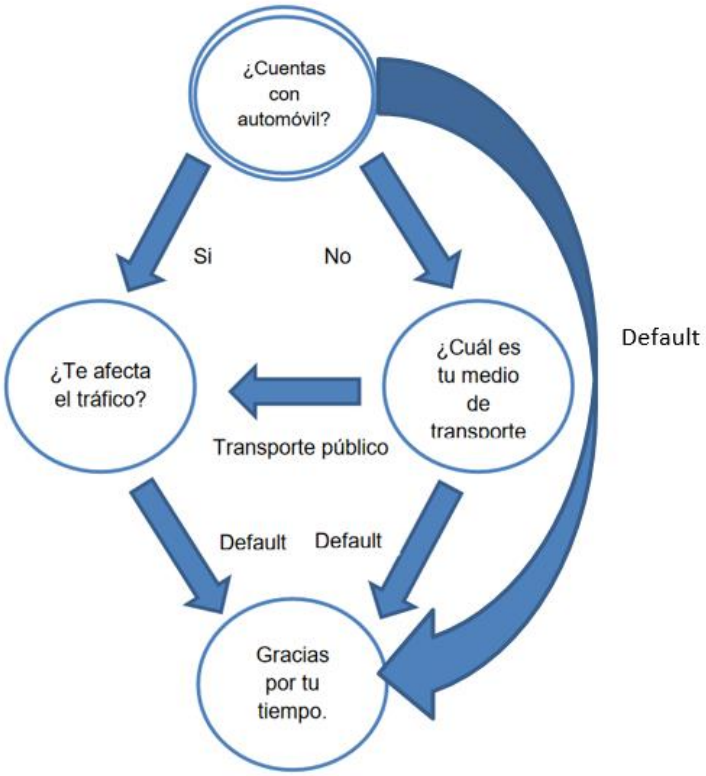


Figura 13. Representación visual de una encuesta como máquina de estados

Tal como en las máquinas de estado tradicionales, era necesario tener un apartado con todos los estados (en este caso mensajes) y otro con todas las transiciones, este último debe de tener una relación entre el estado actual, la respuesta para utilizar esta transición y el estado siguiente. Nosotros decidimos usar la palabra clave Default para determinar el siguiente estado a seguir en caso de que se recibiera una respuesta diferente a las solicitadas. En este ejemplo, si nos encontráramos en el primer estado (“¿Cuentas con automóvil?”) y el usuario respondiera algo diferente de Si o No, automáticamente se cambiaría al estado final de “Gracias por tu tiempo” y se le enviaría ese último mensaje, contabilizándolo como una respuesta diferente.

Debido a que estábamos implementando esta funcionalidad para nuestra API Rest, era necesario que capturáramos toda la información por medio de una cadena en formato JSON, así que tal y como lo

requieren las máquinas de estados, era necesario solicitar los estados y las transiciones, un ejemplo de cómo recibíamos esta información es la transformación en JSON de la imagen anterior:


```
{
  "campania":"Encuesta",
  "estados":[
    {
      "idEstado":0,
      "mensaje":"¿Cuentas con automóvil?"
    },
    {
      "idEstado":1,
      "mensaje":"¿Te afecta el tráfico?"
    },
    {
      "idEstado":2,
      "mensaje":"¿Cuál es tu medio de transporte?"
    },
    {
      "idEstado":3,
      "mensaje":"Gracias por tu tiempo."
    }
  ],
  "transiciones":[
    {
      "idT":0,
      "input":"Si",
      "idNext":1
    },
    {
      "idT":0,
      "input":"No",
      "idNext":2
    },
    {
      "idT":0,
      "input":"Default",
      "idNext":3
    },
    {
```

```
      "idT":1,
      "input":"Default",
      "idNext":3
    },

    {
      "idT":2,
      "input":"Transporte público",
      "idNext":1
    },
    {
      "idT":2,
      "input":"Default",
      "idNext":3
    }
  ]
}
```


Como se puede observar, el JSON consiste de 3 partes:

- El nombre de la campaña, únicamente utilizado para nuestros clientes pudieran identificar su campaña
- Un arreglo con los estados, junto con su respectivo identificador. Estos son todos los diferentes mensajes a enviar
- Otro arreglo con las transiciones, estas constan de estado actual, condición para cambio de estado (o mensaje de respuesta) e identificador del siguiente estado.

Una vez definida la tecnología a utilizar, el formato de los datos de entrada y sabiendo que utilizaríamos los métodos previamente desarrollados para 2 vías, el siguiente paso era definir cómo realizar el cambio automático de estado con base en la respuesta del usuario.

Para solucionar esta incógnita se optó por montar otro método no público, que utilizaba la funcionalidad de 2 vías de mandar un post a una URL que el cliente nos compartía, solo que lo apuntábamos a este nuevo método privado con el identificador de la campaña y con base en el mensaje enviado podíamos calcular el estado actual, y con ayuda de la respuesta del usuario, lo enlazábamos con las transiciones y lanzábamos el segundo mensaje, todo completamente automatizado.

Un último tema a aclarar de esta funcionalidad es que tal y como se observa en la petición para crear la máquina de estados, no se solicitan los números a los cuales hacer el envío, esto es debido a que también para agilizar los tiempos de respuesta y además no solicitar en múltiples ocasiones la misma máquina de estados, se separó la funcionalidad en dos. El otro método únicamente recibe el id de la

campaña (que se le comparte al usuario como respuesta a la petición de la máquina de estados) y el número de teléfono a agregar a la campaña, con sus respectivas validaciones.

Para este punto del proyecto ya se encontraban desarrollados todos los métodos deseados para poder tener una buena oferta hacia el mercado, incluyendo el diferenciador de respuestas automáticas. Sin embargo, aún hacía falta mejorar un poco la funcionalidad de los métodos del API con los ya se contaba. Lo siguiente desarrollado cae en esta última categoría.

Por solicitud de uno de nuestros clientes y porque también nosotros veíamos su beneficio, el siguiente enfoque de este proyecto fue el agregar la posibilidad para la programación de envíos. Implementar esta funcionalidad fue bastante directo, ya que contábamos con la ventaja de que habíamos separado el receptor de mensajes del procesador encargado de realizar el envío. Para poder habilitar esta funcionalidad únicamente se agregó el campo en la base de datos de la fecha y hora del envío (datetime) y se modificó el procedimiento almacenado (stored procedure) que obtiene los mensajes que deben de ser enviados para que únicamente regresara aquellos que cumplían con ambas características de que aún no estaban procesados y que su fecha y hora de envío fuera menor o igual a la fecha y hora actual. Esta funcionalidad se implementó tanto para una vía como para mensajes con respuesta y respuestas automáticas ya que va de la mano con este último. Es importante destacar que también fue necesario agregar nuevas validaciones a estos métodos para garantizar la integridad de los datos y el correcto funcionamiento de todo, por ejemplo, no es posible programar mensajes antes de la fecha actual.

Finalmente, la última funcionalidad de la API de cara al cliente que se desarrolló fue la validación de números celulares. Como comenté previamente, ya se realizaba una validación básica para comprobar que el número recibido fuera de 10 dígitos y que no contuviera caracteres, sin embargo, esta nueva optimización estaba basada en una lista de rangos telefónicos que publica el Instituto Federal de Telecomunicaciones (IFT) que contiene las claves de estado, series y rangos de teléfono, mismos que están identificados por compañía y si se trata de una serie de números fijos o móviles. Con esta nueva información, se implementó una función que nos ayudará a identificar si el número recibido efectivamente se trata de una línea móvil o de un número con clave de marcación válida. Esto fue de gran utilidad para nuestros clientes ya que los ayudó a limpiar sus bases de datos e identificar números falsos, así como evitar cobros a números que jamás iban a recibir ese SMS.

Junto con esta validación de números y para garantizar el correcto uso de nuestra plataforma, se implementó también un sistema de palabras bloqueadas y URLS permitidas. Esto con la finalidad de proteger a la plataforma de un uso indebido para temas ilegales o de ataques cibernéticos como phishing (suplantación de identidad) o spam. El objetivo de utilizar palabras bloqueadas es manejar un blacklist ya que es más sencillo bloquear algunas palabras que conocemos utilizadas principalmente para actividades ilícitas que tener un diccionario gigantesco de palabras permitidas en un whitelist. Sin embargo, en cuanto a las URL es mejor que nos soliciten el alta de la misma para nosotros poder entrar personalmente y verificar que el sitio no se hace pasar por alguien más, en lugar de permitir todos los portales y bloquear solo algunos que ya están identificados como fraudulentos, ya que esta medida sería

ineficiente ocasionada del hecho de que siempre pueden ser creados nuevos portales maliciosos, mientras que un portal “limpio” es muy raro que se convierta en dañino.

Una vez que se tenía desarrollada esta validación de números plus, se utilizó para segmentar nuestro envío de mensajes y aumentar nuestro porcentaje de efectividad. Lo que se hizo fue que al identificar a qué operador móvil pertenecía el número telefónico, se realizaba el envío por un chip de ese mismo operador móvil, aumentando la posibilidad de que el SMS llegará sin problemas a su destinatario debido a que nunca cambiaba de operador de red.

En este punto nuestros clientes empezaron a aumentar y empezamos a notar en los logs unos errores muy particulares, había momentos inclusive donde la API tomaba muchísimo tiempo para responder. Según el log, en ocasiones la base de datos nos estaba respondiendo con un timeout. Primero procedimos a analizar las querys y observamos que había unos procedimientos almacenados que por estar dentro de una transacción estaban generando deadlocks. Se corrigió este tema limitando las querys que estaban dentro de las transacciones.

9.7. Optimización de la base de datos

Aprovechando esta revisión de la base de datos, decidimos optimizar la base de datos para las modificaciones que se habían realizado debido a los nuevos métodos. Para lograr esto, utilizamos la herramienta incluida dentro del SQL Server Management Studio llamada SQL Profiler.

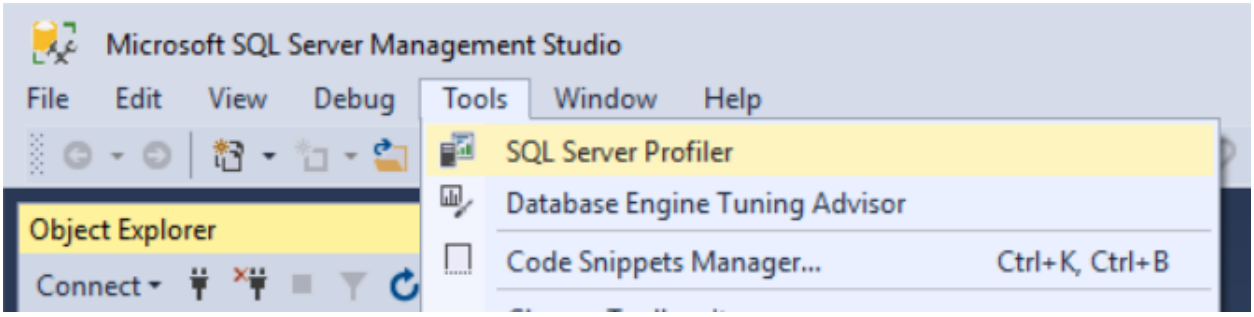


Figura 14. Imagen de cómo acceder a SQL Profiler.

Para optimizar querys, utilizamos la opción de Tunning

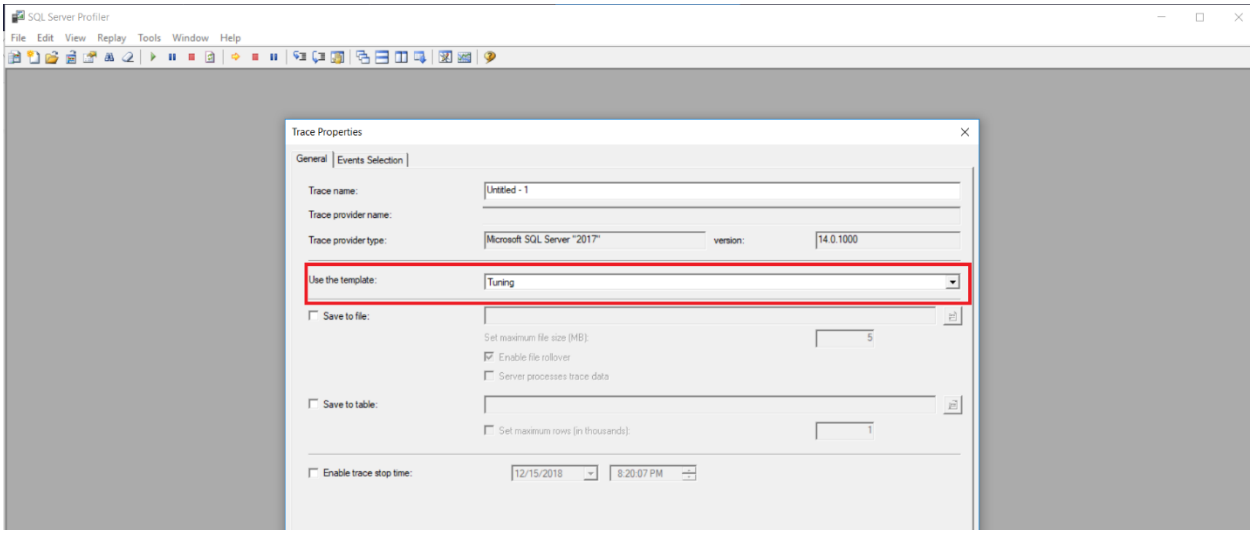


Figura 15. Pantalla de propiedades de SQL Profiler

Y para que este optimizador no capture todas las querys de todas las bases de datos del sistema, se le introduce un filtro para que capture solo aquellas de la base de datos que queremos monitorear.

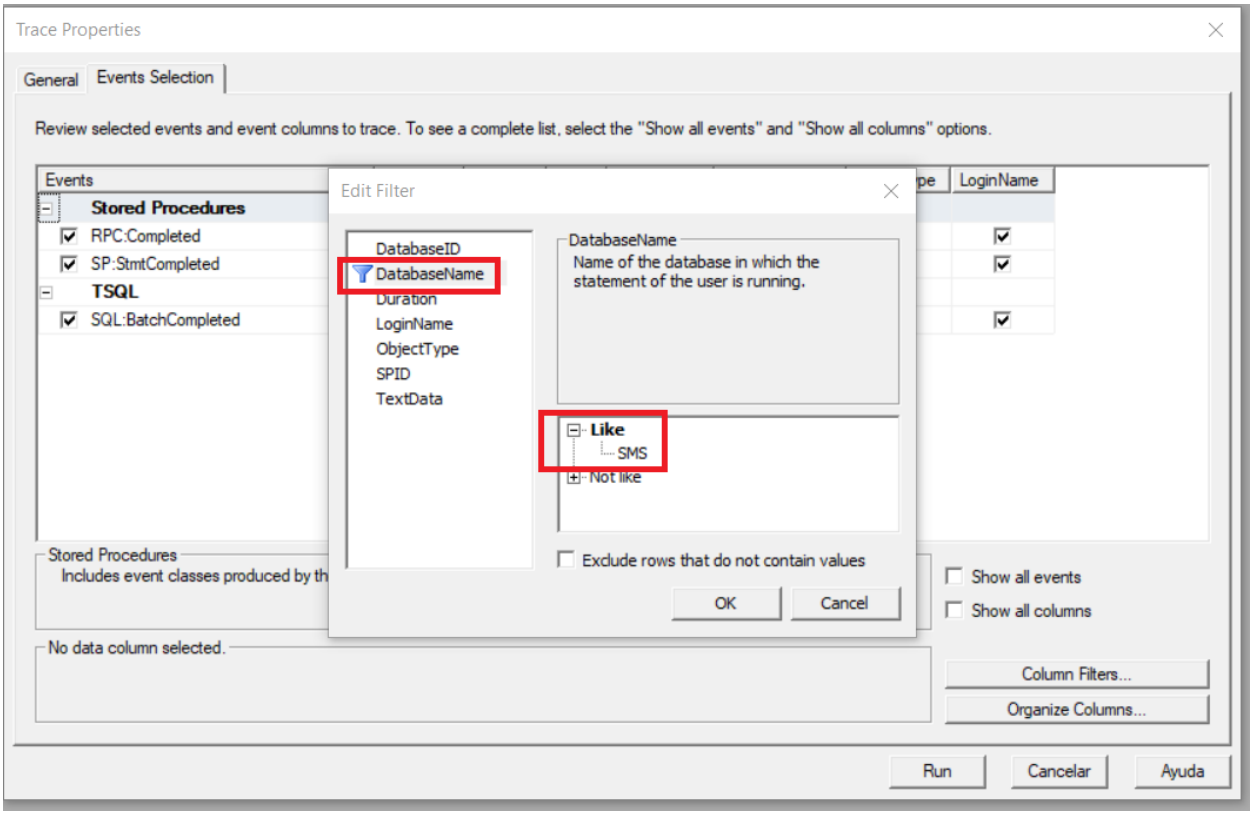


Figura 15. Pantalla sobre como introducir un filtro en SQL Profiler

Una vez que ocurre esto, se le da click en el botón de run y se deja correr por un par de segundos. Es importante aclarar que este tipo de operaciones únicamente deben de correrse en ambientes de pruebas o bien en horarios donde la plataforma no tenga mucho tráfico, ya que todo esto consume

muchos recursos y puede llegar a alentar este u otros aplicativos que se encuentren en el mismo servidor. Posteriormente se detiene el análisis y se exporta como un Trace File

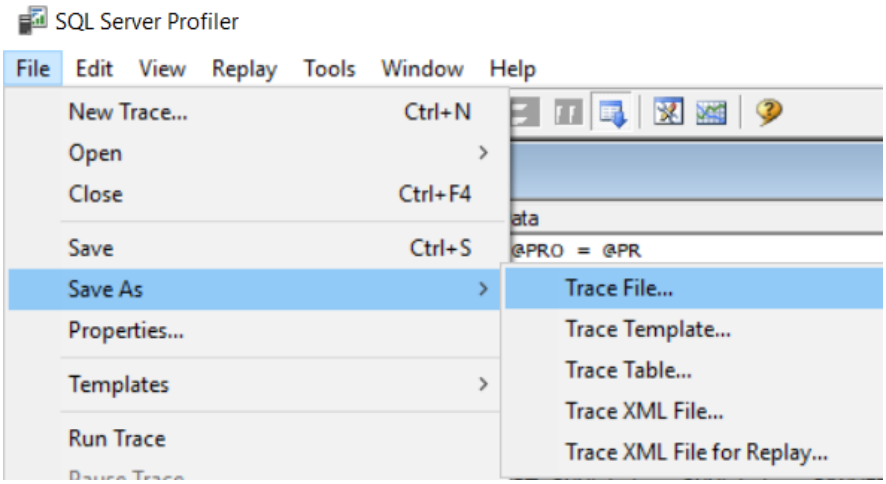


Figura 16. Pantalla indicando como guardar un análisis de SQL Profiler como Trace File

El siguiente paso es utilizar este archivo generado junto con el Tunning Advisor y se selecciona la base de datos que se desea optimizar.

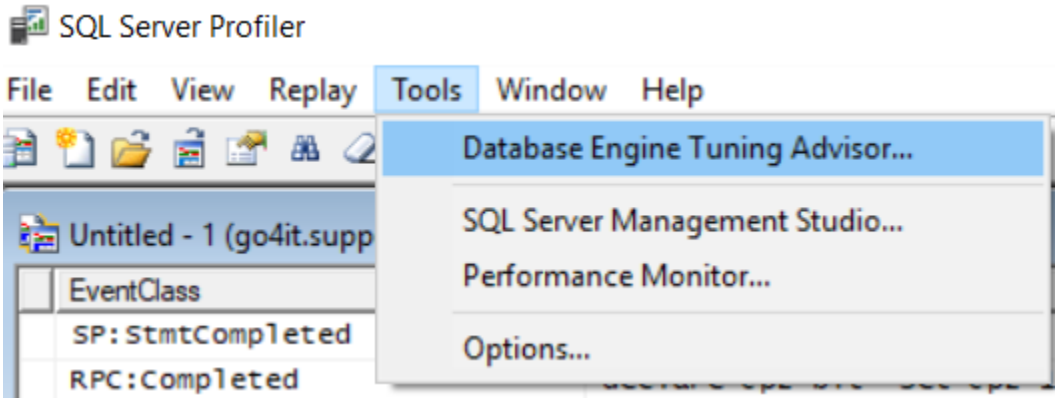


Figura 17. Pantalla ilustrativa sobre como abrir Tunning Advisor

Después solamente se da click en Start Analysis y en unos minutos el sistema nos arrojará sus recomendaciones, así como el porcentaje de optimización que se espera de estas.

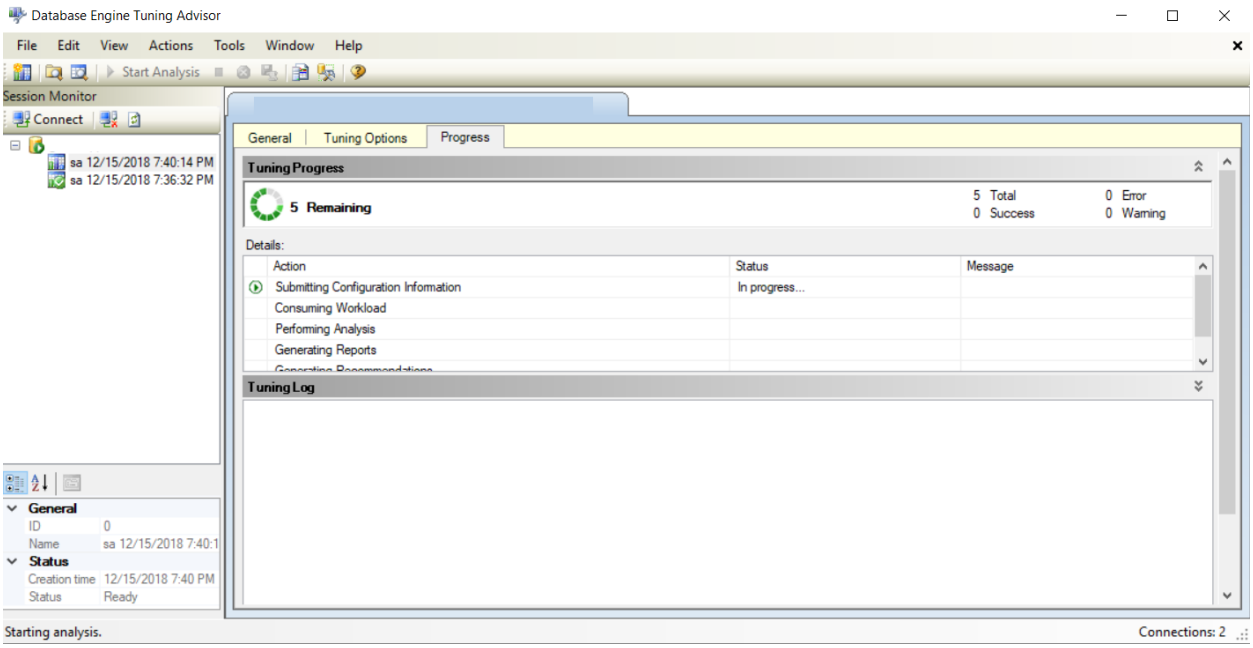


Figura 18. Pantalla de un proceso de análisis de Tunning Advisor

En nuestro caso, una vez que terminó de hacer el análisis, nos recomendó generar un par de índices compuestos y otras estadísticas. El porcentaje de optimización de este proceso fue de un 59%

Figura 19. Porcentaje de optimización de Tunning Advisor

9.8. Optimización de memoria y desempeño

A pesar de que ya no había problemas con el aplicativo, notamos que los recursos que este consumía en el servidor eran considerablemente elevados, lo cual podía impactar nuestra capacidad futura de montar nuevos aplicativos en ese servidor. Debido a esto se decidió el volver a analizar opciones para optimizar tanto la API como los procesadores.

En este punto, la empresa empezó a montar otros aplicativos en nuestro servidor y empezamos a saturarlo, por lo que nos pusimos a analizar qué era lo que estaba ocupando más recursos y vimos que los aplicativos de SMS estaban consumiendo más recursos de los que considerábamos necesarios. Justo en estos momentos en el tiempo, Microsoft liberó la segunda versión otra manera para desarrollar APIS REST llamada .NET CORE, el cual además de permitir desarrollar aplicativos cross-platform (que corren en varias plataformas como Windows, Linux y Mac) genera aplicaciones con un rendimiento mucho mejor que las tradicionales.

Habíamos explorado previamente la opción de .NET CORE anteriormente pero no se había implementado debido a que la comunidad documentaba que la tecnología todavía tenía muchos errores y bugs, además de que implicaba un cambio de paradigma de .NET Framework a Entity Framework, que

es cambiar de un paradigma tradicional orientado a código por otro orientado a datos (comúnmente bases de datos).

Para poder ajustarnos a este nuevo paradigma, fue necesario crear todos los modelos de los datos como clases y crear otras clases con los modelos que replicaban las respuestas de las tablas de los procedimientos almacenados de la base de datos, por lo tanto fue necesario hacer modificaciones tanto en el código como en la base de datos.

Sin embargo, todo este trabajo valió la inversión de tiempo, ya que en este caso un aplicativo completo en .NET CORE corre en aproximadamente 6MB de RAM, mientras que en .NET Framework 4.6 corría en casi 300MB, como se puede observar en las siguientes 2 imágenes, poniendo atención en el proceso APIv2_Control vs APIv3_Control y su columna Private Bytes.

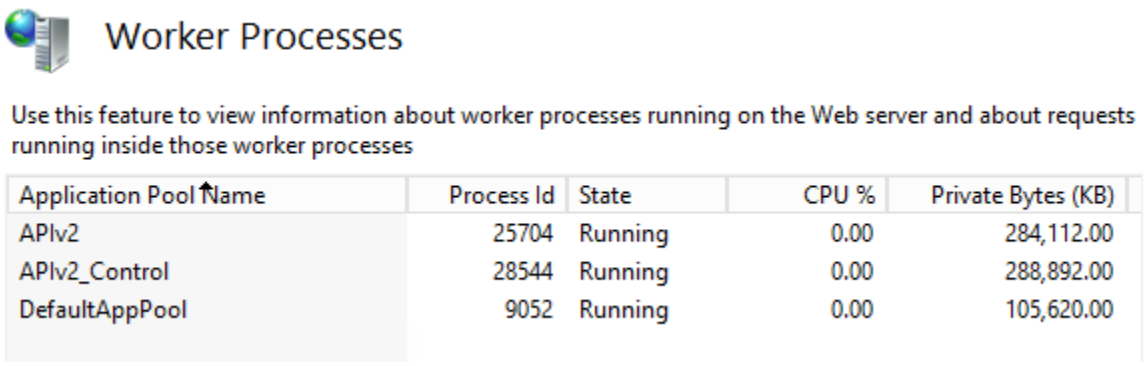


Figura 20. Consumo de memoria y procesador de los aplicativos previo a la migración.

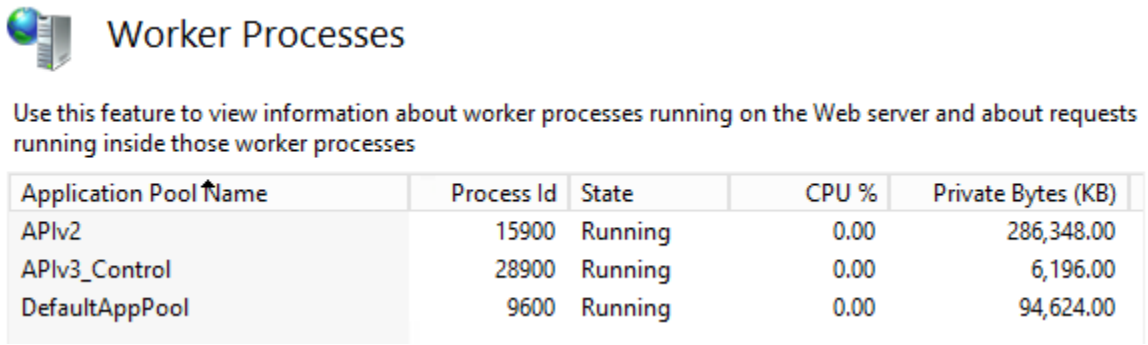
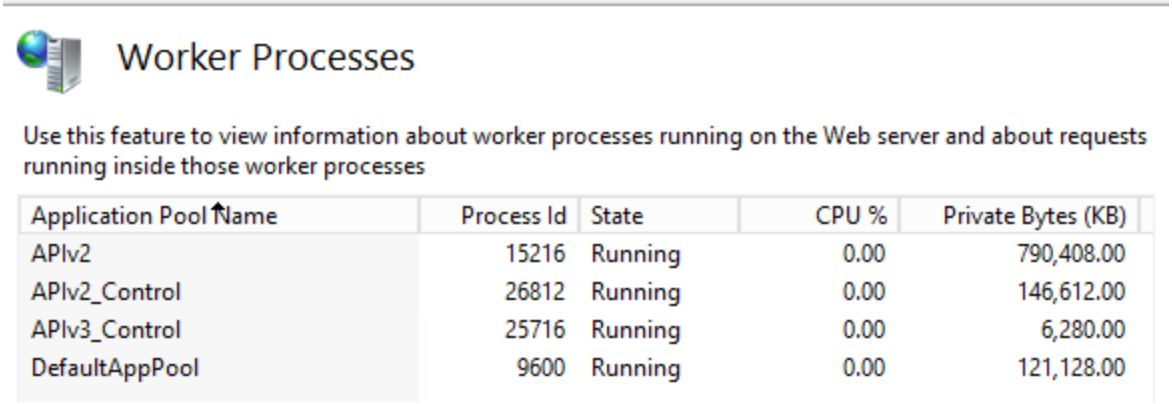


Figura 21. Comparativa del consumo de recursos de los aplicativos. Viejo vs nuevo

Con esta diferencia de 300 a 6 megabytes, estamos hablando de una mejora del 5,000%, y esto simplemente cambiando en qué se desarrolló, sin mover aspectos específicos de la lógica. Aunque estos números suenan impresionantes, en otras ocasiones daba menos diferencia, pero de todos modos significativa:



Worker Processes

Use this feature to view information about worker processes running on the Web server and about requests running inside those worker processes

Application Pool Name	Process Id	State	CPU %	Private Bytes (KB)
APIv2	15216	Running	0.00	790,408.00
APIv2_Control	26812	Running	0.00	146,612.00
APIv3_Control	25716	Running	0.00	6,280.00
DefaultAppPool	9600	Running	0.00	121,128.00

Figura 22. Segunda comparativa del consumo de recurso de los aplicativos

En este caso estamos comparando 145 megabytes vs 6. Haciendo el cálculo esto es una mejora del 2,400% que sigue siendo gigantesca. Ahora, esto solo soluciona la parte del API REST, pero faltaba optimizar el servicio procesador de los mensajes. Se intentó migrar también a .NET CORE 2.0, pero no se encontró como generar un servicio de esta manera, así que se tuvo que ir más a detalle en el código y se corrió un analizador del proceso. Descubriendo que una de las librerías que utilizábamos estaba consumiendo más de la mitad de la RAM del proceso. Esta librería se llama Newtonsoft y es utilizada para manejar objetos de tipo JSON. Una vez identificado el error, se sustituyó esta librería por una manera más nativa de .NET y se mejoró el rendimiento en un 50%. Lo que se realizó fue convertir todos los JSON como objetos de clases de .NET que tuvieran la misma estructura y pudieran ser utilizados de la misma manera en los procesos y funciones.

9.9. Integración de funcionalidades

Una vez migrado, optimizado y probado el esqueleto del sistema (API) se procedió a agregar todas estas funcionalidades al sitio web para el envío de SMS masivos. La primera funcionalidad en ser integrada fueron los mensajes con respuesta o dos vías. La creación de campañas de este tipo de mensajes es igual a la creación de una campaña 1 vía, como se puede observar en la imagen siguiente. Sin embargo, aprovecho la oportunidad para mostrar la funcionalidad de programación de campañas:

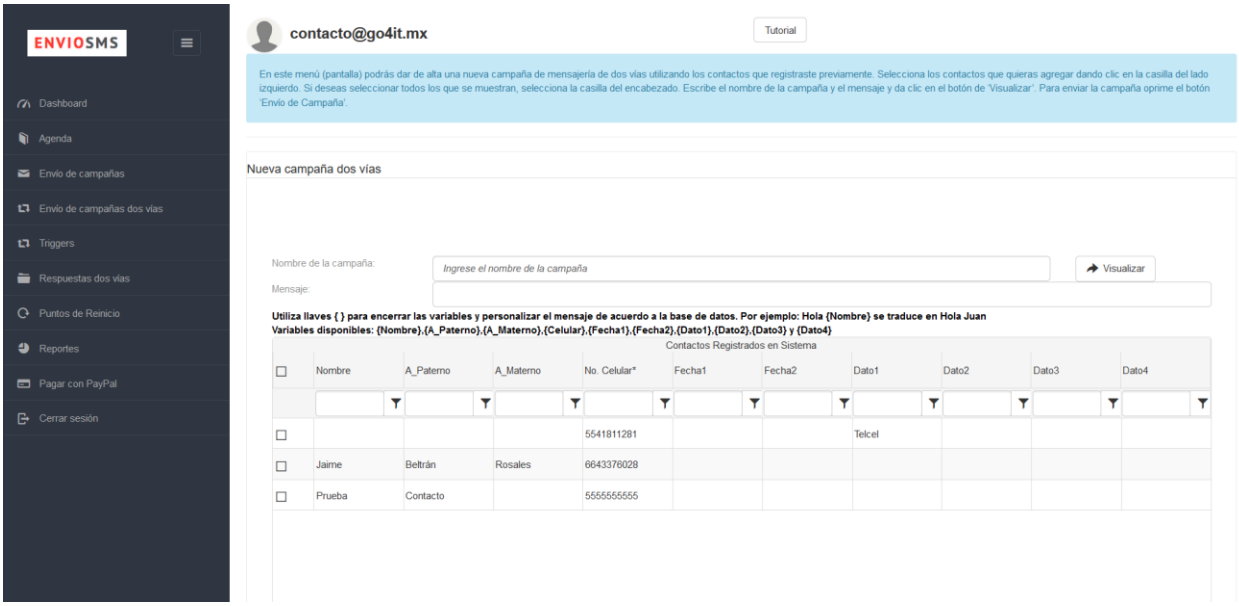


Figura 23. Pantalla de nueva campaña 2 vías del portal web

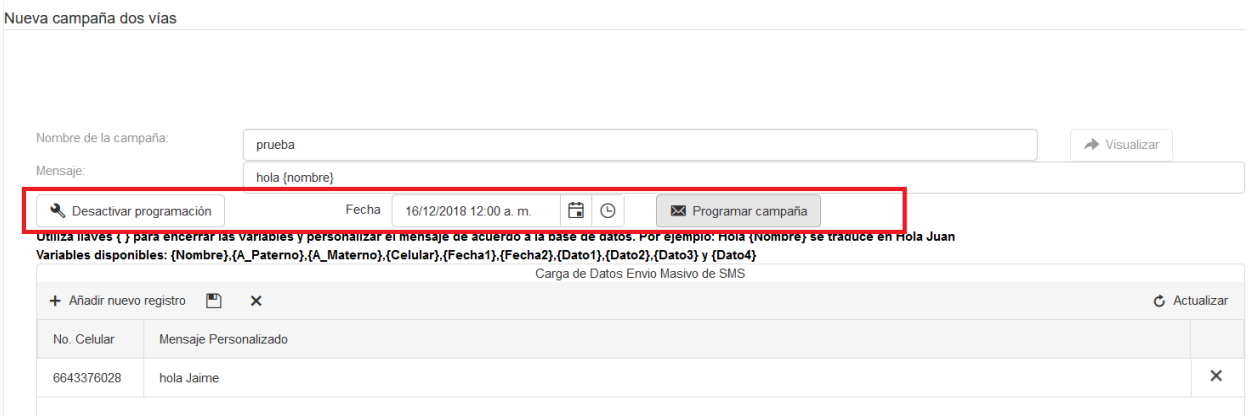


Figura 24. Pantalla mostrando la programación de campañas

La verdadera diferencia de este método se encuentra en las respuestas dos vías. En este apartado, mostramos una gráfica con el porcentaje de participación de la campaña, así como una tabla con todas las respuestas recibidas.

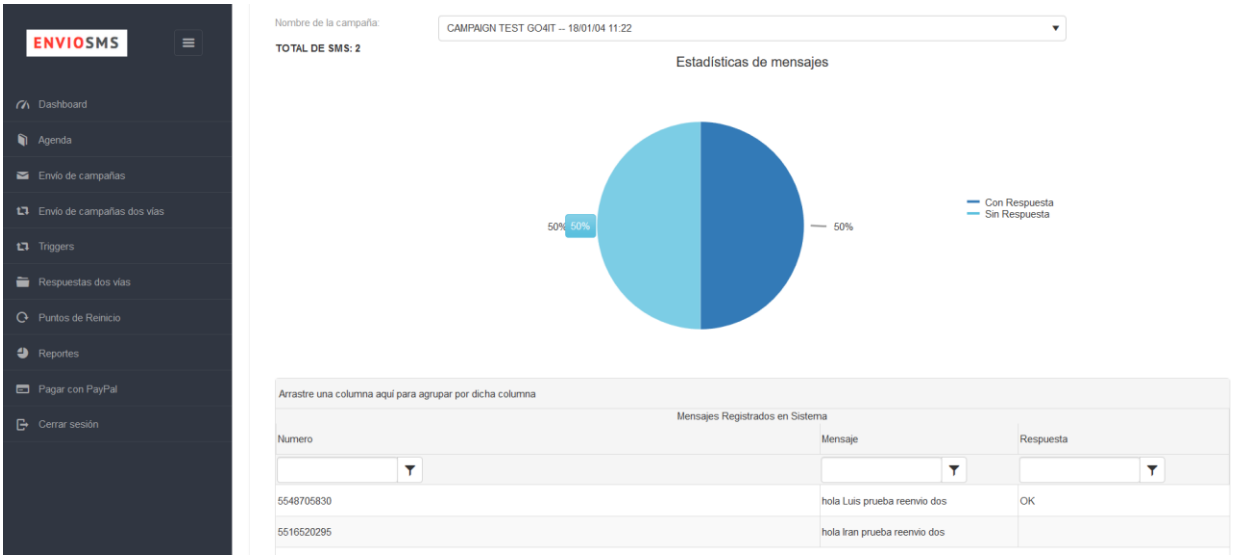


Figura 25. Pantalla de respuestas 2 vías del portal web.

Después de la funcionalidad 2 vías, quisimos integrar respuestas automáticas. Como se observa en la imagen, para crear una nueva campaña hay que dar click en agregar mensaje, donde cada uno de estos mensajes representa un estado de la máquina de estados. Al agregar un nuevo mensaje se crea un botón también para agregar respuestas, las cuales representan las transiciones, por eso se solicita la respuesta a recibir y el siguiente mensaje a enviar. El resto de la creación de la campaña es igual a los anteriores.

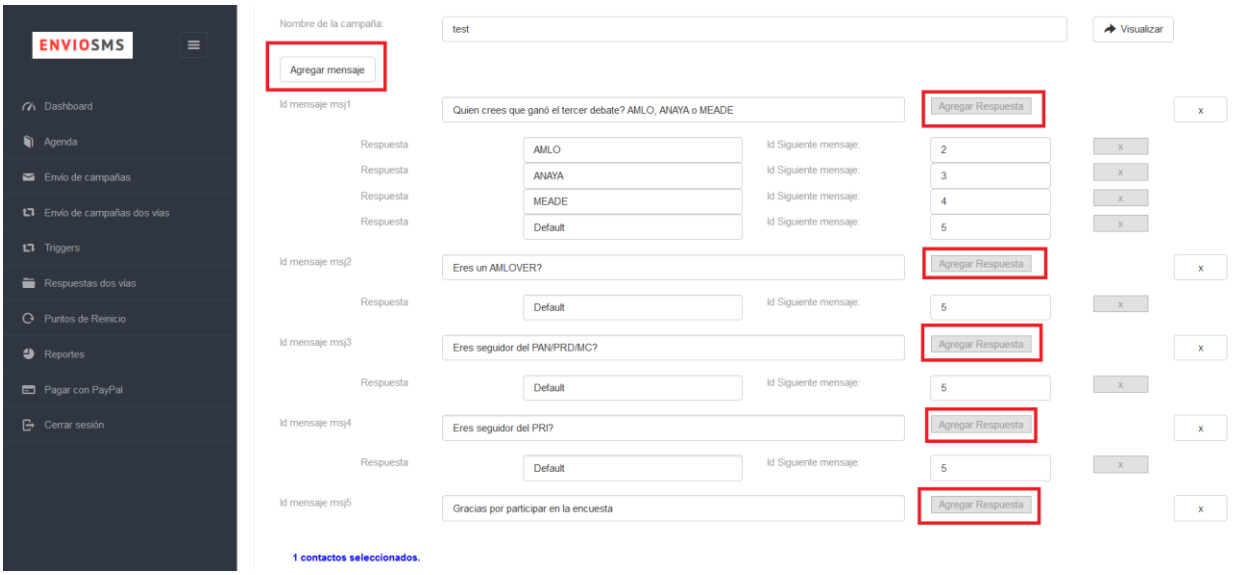


Figura 26. Pantalla de respuestas automáticas del portal web.

9.10. Generador de reportes

Para poder dar más autonomía a nuestros clientes, se integró un apartado de reportería sencilla. En este caso todos los reportes están en la misma página, pero divididos en diferentes pestañas según el tipo de mensajes enviados. Comenzaré por explicar los reportes para mensajes 1 vía.

Estos reportes son los más sencillos de todos. Consisten de 4 apartados:

- Una barra para filtrar con base en una fecha de inicio y una fecha de término
- Una gráfica de pastel indicando el porcentaje de mensajes exitosos y fallidos
- Otra gráfica de barras que muestra el número de mensajes exitosos y fallidos
- Una tabla con cada mensaje enviado, el precio, la fecha y hora en la que se realizó el envío, así como el estatus del mismo. Esta tabla cuenta también con una barra para realizar filtros dentro de la misma.

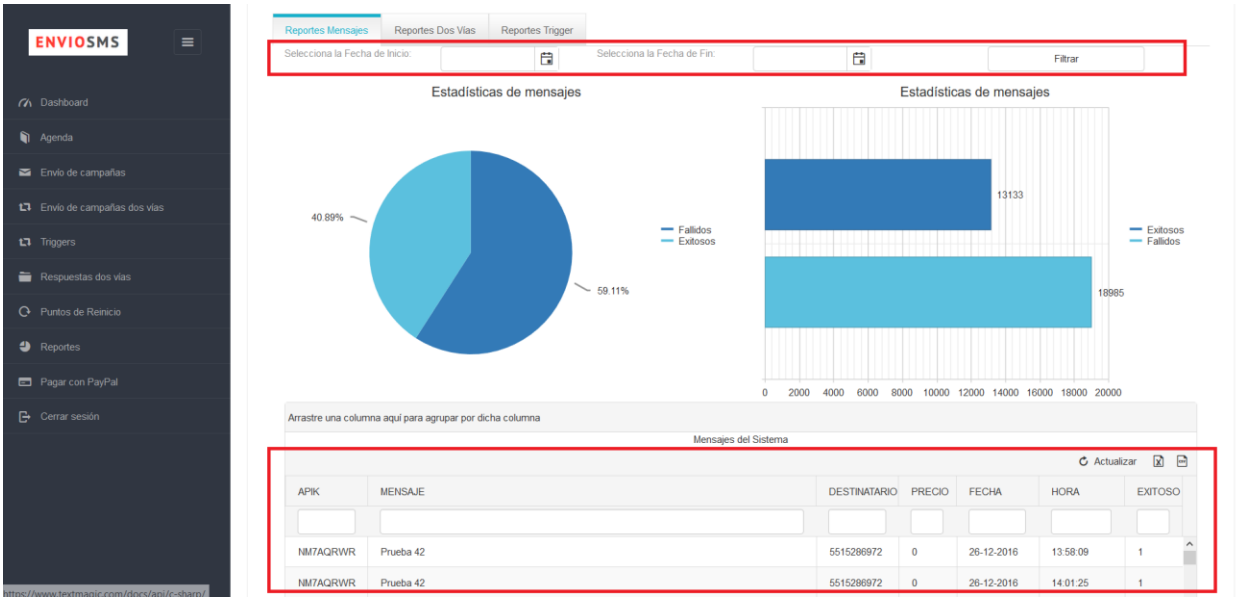


Figura 27. Pantalla de reportes 1 vía.

La siguiente pestaña de los reportes son los mensajes dos vías. Este reporte contiene los mismos apartados que los reportes de 1 vía con una gráfica adicional que representa los porcentajes de participación de la campaña, es decir las respuestas de los usuarios. Es importante destacar que las gráficas y la tabla también se actualiza al realizar los filtros.

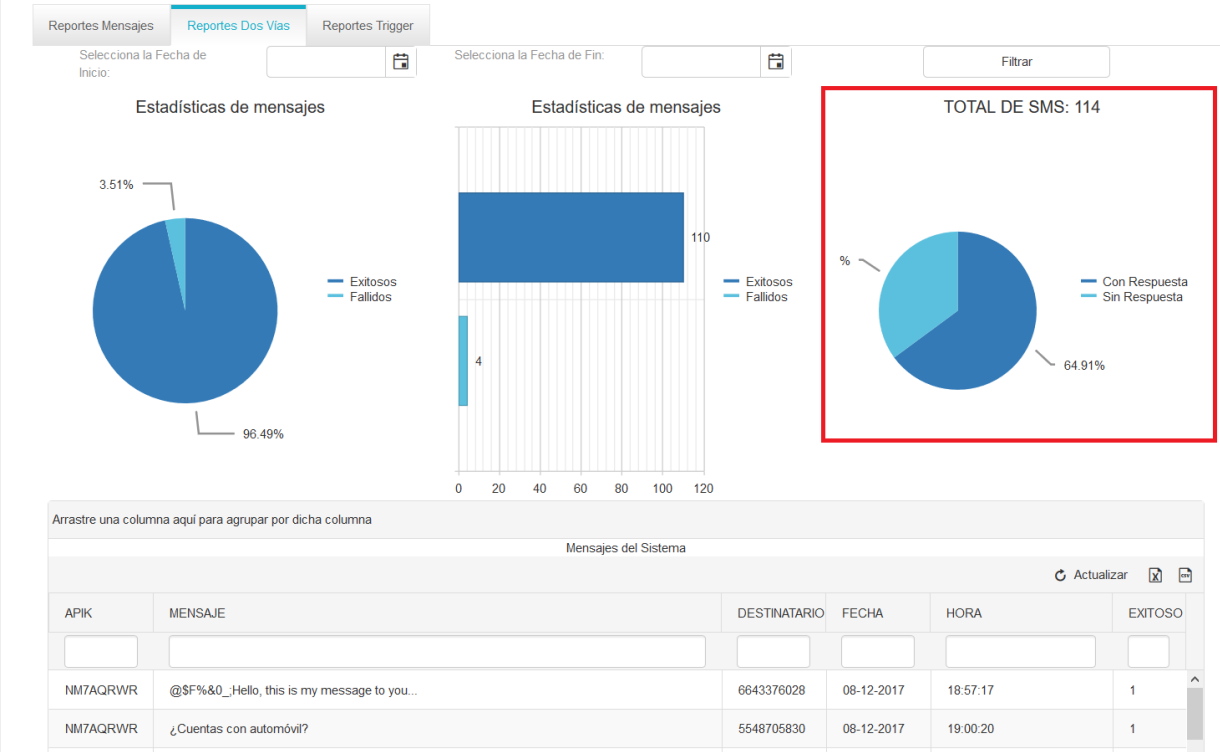


Figura 28. Pantalla de reportes 2 vías.

La última pestaña de este apartado son los reportes de respuestas enlazadas. Estos son los más completos de todos y fue todo un reto mostrar la información, ya que se tuvieron que generar procedimientos almacenados que cambiaran como se muestra la información nativamente para que se pudieran generar los reportes correctos.

Estos reportes contienen 3 apartados:

- Un histograma donde cada columna representa un mensaje enviado (o un estado de la máquina de estados) y sobre ese histograma se apilan las diferentes respuestas (transiciones ejecutadas) para ese estado en específico. En ese apartado se manejan los números totales de cada respuesta
- Debajo de ese histograma se encuentra un menú desplegable para seleccionar el estado que se quiere analizar más a detalle
- Una gráfica de pastel que representa los porcentajes de cada respuesta para ese estado.

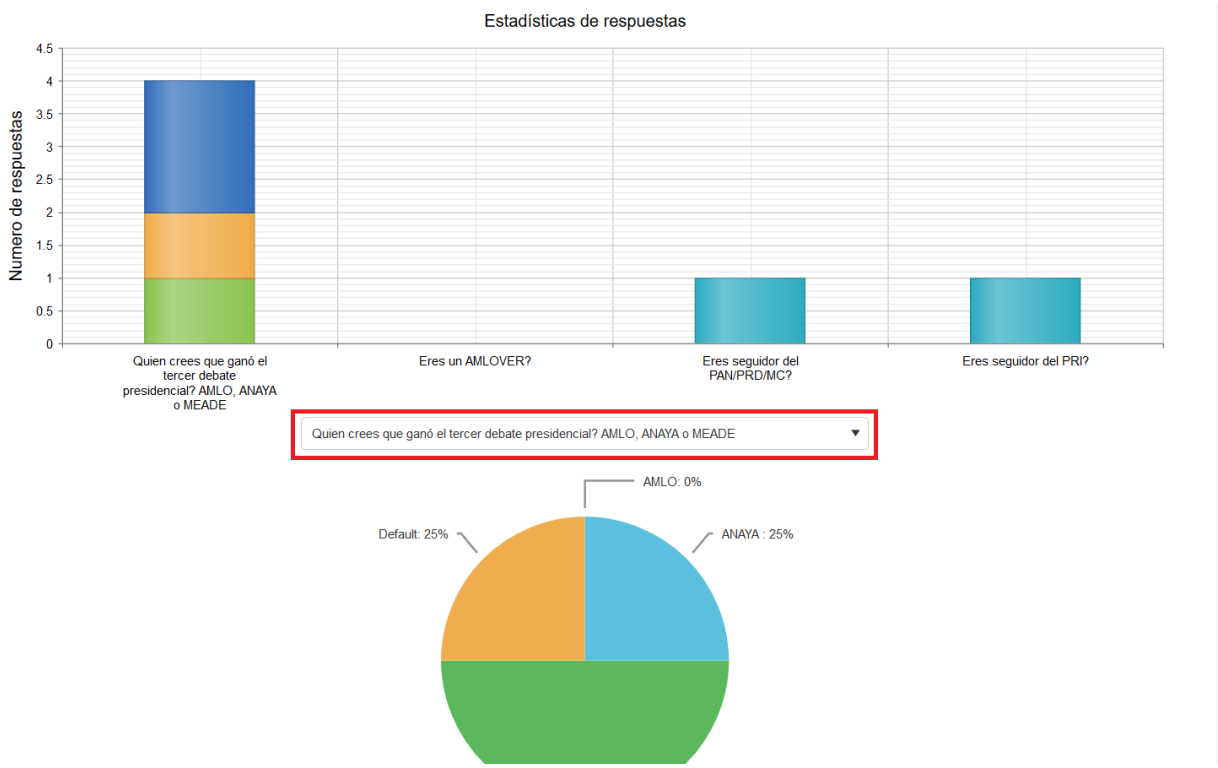


Figura 29. Pantalla de reportes de respuestas automáticas.

Finalmente, el último desarrollo que se codificó para el sitio web fue un breve tutorial en cada pantalla indicando a los usuarios en dónde dar click para hacer un flujo completo de cada tipo de envío o qué información podían visualizar en cada pantalla. Este tutorial es automáticamente activado durante el primer login del usuario y se desactiva para ocasiones posteriores al terminar el tutorial; aún desactivado siempre es posible acceder a él por medio del botón en la parte superior derecha de las pantallas que dice “Tutorial”. Para realizar esto se utilizó la librería de javascript llamada EnjoyHint que genera tutoriales interactivos. Para ejemplificar esto lo haremos con el tutorial de la agenda:

1. 

Figura 30. Pantalla de tutorial para encontrar el formato de ejemplo

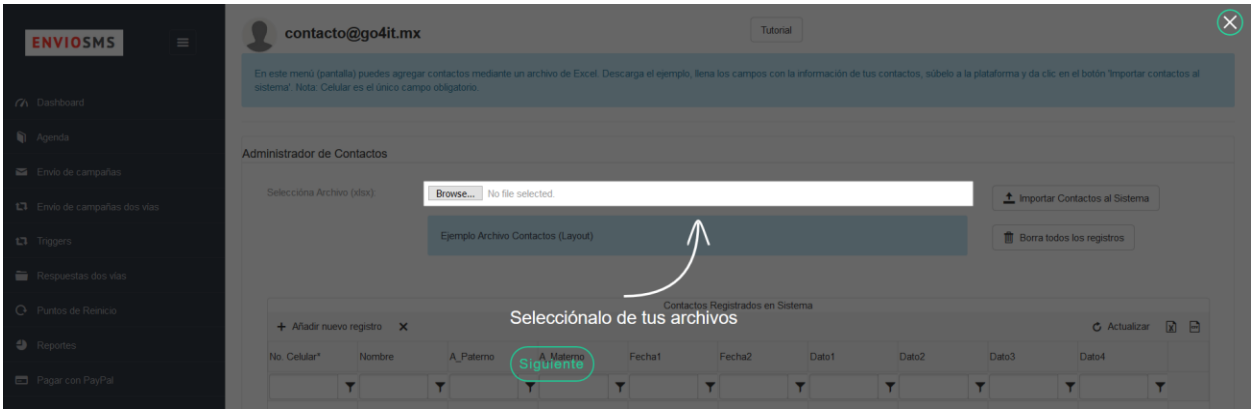
2. 

Figura 31. Pantalla de tutorial para seleccionar un archivo

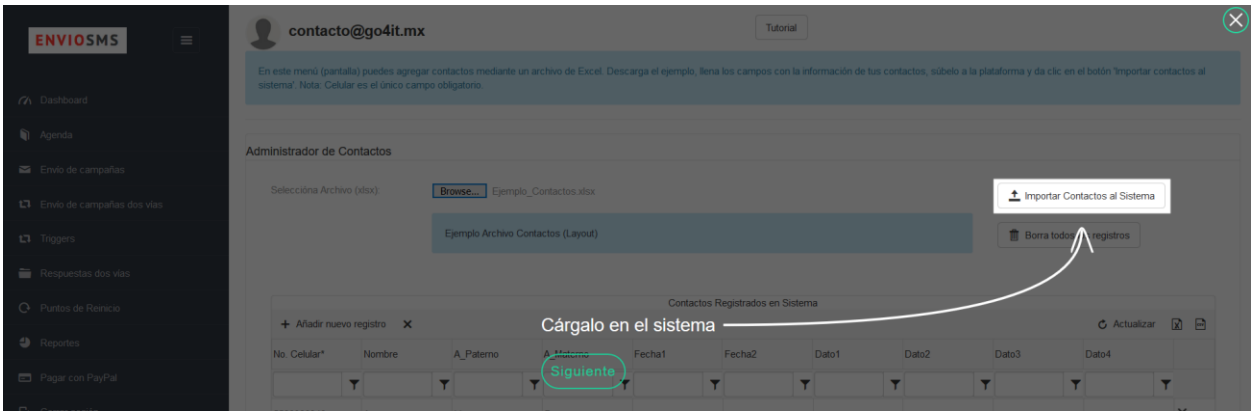
3. 

Figura 32. Pantalla de tutorial para cargar el archivo al sistema

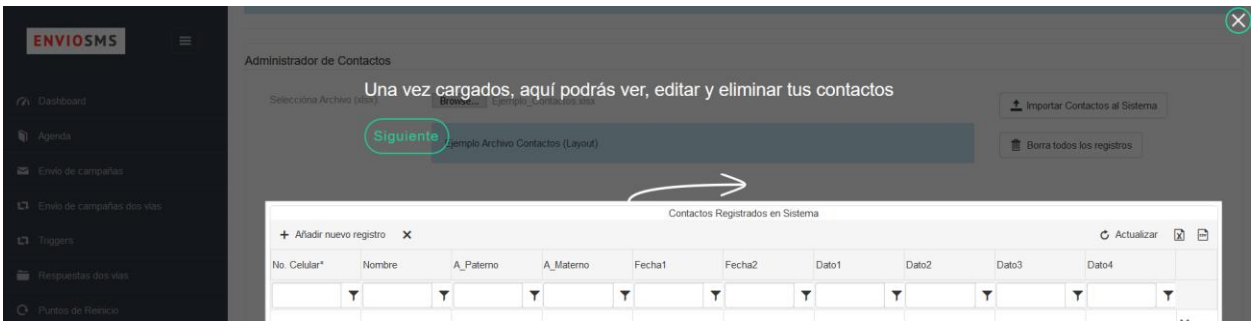
4. 

Figura 33. Pantalla de tutorial para visualizar contactos

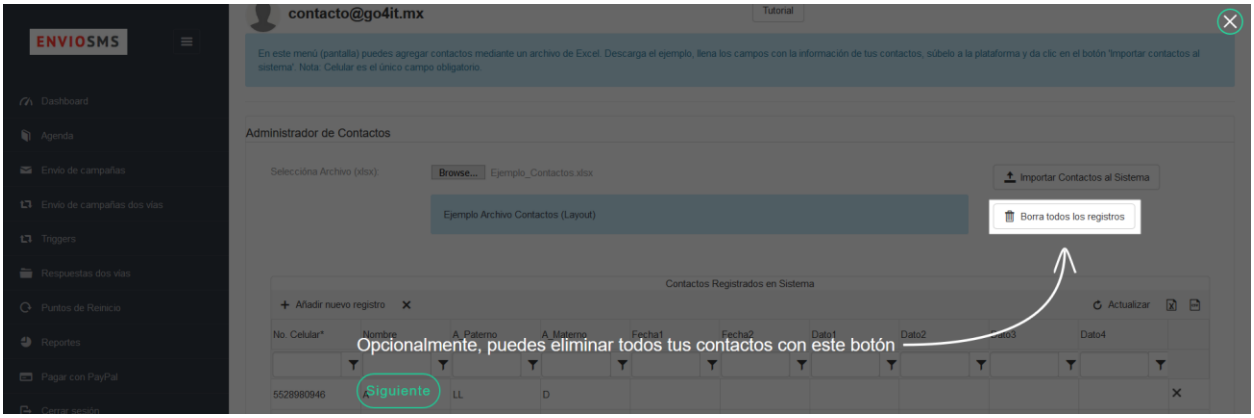
5. 

Figura 34. Pantalla de tutorial para eliminar todos los contactos

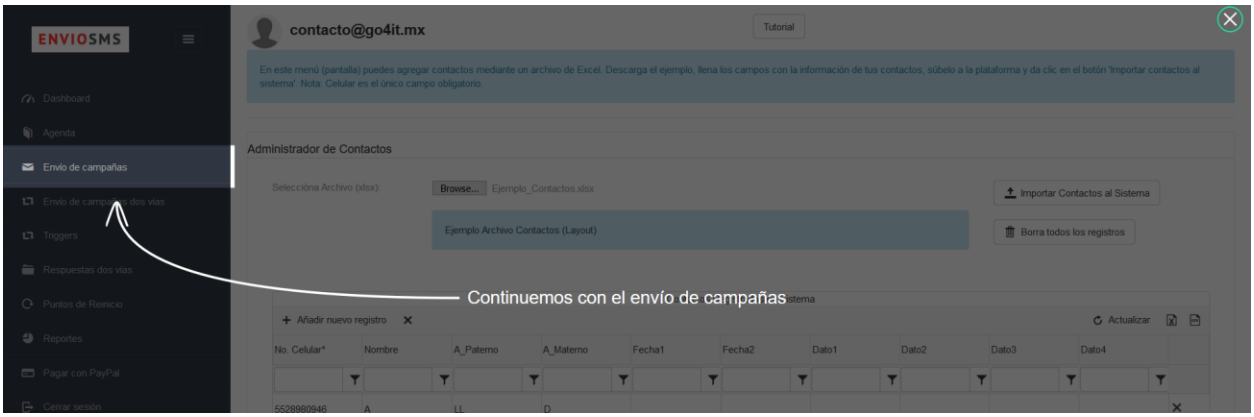
6. 

Figura 35. Pantalla de tutorial para continuar.

Como se puede observar en las imágenes, el tutorial es bastante intuitivo y amigable para cualquier tipo de usuario y para implementarlo no se requiere de muchas líneas de código, aunque para que funcionara correctamente, sí fue necesario modificar ligeramente el código fuente por temas de diferentes resoluciones.

Es importante destacar que todos los métodos desarrollados tanto para la API como para la página web se realizaron con su debida implementación de logs de errores y en algunos casos hasta exitosos. En estos logs almacenábamos principalmente 4 cosas:

- **Tipo de log**. Esto simplemente describía si se trataba de un log exitoso, de error o una advertencia.
- **Código de error/éxito**. Cada log (exitoso o fallido) tenía asociado un código de error que se encontraba en nuestro catálogo de errores, mismo que se guardaba con cada nuevo log
- **Dónde ocurrió**. La página, procedimiento almacenado o tabla en dónde se había detonado el log.
- **Descripción del mismo**. Una descripción más detallada proporcionada por .NET o SQL Server según correspondiera.

Este manejo de logs resultó muy importante ya que nos permitía dar trazabilidad a las operaciones del aplicativo. Esta característica disminuyó el tiempo que necesitábamos para poder solucionar un problema o bien optimizar alguna sección del aplicativo. Además de trazabilidad, esto permite que el aplicativo sea auditable y podamos compartir información detallada de cómo se ha comportado el sistema.

Otro aspecto que vale la pena comentar y normalmente no se le da la importancia debida es el decir explícitamente que todas las querys, procedimientos almacenados o cualquier tipo de interacción que había entre el código y la base de datos, ya fuera entre usuario y la base de datos o el sistema y la base, se ejecutaba por medio de querys sanitizadas, es decir que se utilizaban parámetros para proteger al sistema contra uno de los ataques cibernéticos más comunes y sencillos conocido como inyección de SQL o fallas en el sistema ocasionadas porque el usuario requería enviar algún carácter especial o restringido.

Relacionado con esto último, es el tema de que las cadenas de conexión de los aplicativos desarrollados nunca se encontraron dentro del código, si no en un archivo nativo del Framework identificado como web.config para sitios web o app.config para aplicaciones de .NET Core, mismo que es encriptado para su uso en el servidor. En este mismo archivo de configuraciones también se almacenaban un par de parámetros como endpoints de los servicios a los cuales tenían que comunicarse, un par de banderas de configuración y una lista de correos a los cuales notificar en caso de que el sistema detectara un error, mientras que otros que estaban más propensos a modificaciones o que su alteración era crítica para la operación del sistema y no era posible reiniciar el aplicativo para aplicarlos (requerido al realizar un cambio en el web.config), se encontraban almacenados en la base de datos.

Debido a que esta plataforma tiene una gran cantidad de operaciones ocurriendo al mismo tiempo, era importante optimizarla en cada pequeño detalle, por esto mismo se tomó en cuenta que cualquier operación de entrada y/o salida (I/O) es muy costosa en cuanto en rendimiento para la misma, por lo tanto se redujeron las interacciones lo más posible con la ayuda de los procedimientos almacenados.

Todo este proceso de nuevos desarrollos, implementación de funciones en la plataforma web y optimización de código se realizó actualizando la documentación de los métodos que existían previamente, así como generando la nueva documentación para la funcionalidad más reciente, de tal manera que estuviera accesible y fuera comprensible para cualquier persona que necesitara consumir la API o bien desarrollar nueva funcionalidad. Durante la duración del proyecto se desarrollaron los siguientes manuales:

- **Manuales de usuario sobre cómo consumir la API con ejemplos en diversos idiomas.** Este manual básicamente consistía en describir cada método, sus datos de entrada, mensajes de respuesta y códigos de error, así como sus parámetros obligatorios y otros opcionales. Esta se desarrolló una parte en github, otra en Word y una en postman.
- **Manual de instalación y administración del Gateway**. Este es un manual con las lecciones aprendidas de las interacciones con el fabricante del mismo y sobre cómo dar de alta un nuevo Gateway dentro de la misma API.
- **Manual de mantenimiento y administración de los chips**. Este principalmente describe procesos de mantenimiento para verificar que las cosas sigan funcionando correctamente y cómo accionar si no es el caso.
- **Manual de instalación de una nueva instancia de la API**. Este es un manual completo que describe cómo instalar absolutamente todo para montar una nueva instancia, desde la base de datos hasta el aplicativo y las pruebas a realizar para garantizar que esté funcionando al 100%.

Como ejemplo y evidencia de esto último se agrega la siguiente imagen que representa una fracción de un apartado de un manual para instalar una nueva instancia del API.

2 Montar la base de datos y crear los usuarios

En este apartado se describe a detalle cada uno de los pasos que se identificaron como necesarios para montar una nueva base de datos necesaria para la nueva instancia del api deseada.

1. Respaldar la base de datos productiva de SMS desde SQL Server Management Studio
   a. Dar click derecho sobre la base de datos Task > Back Up…

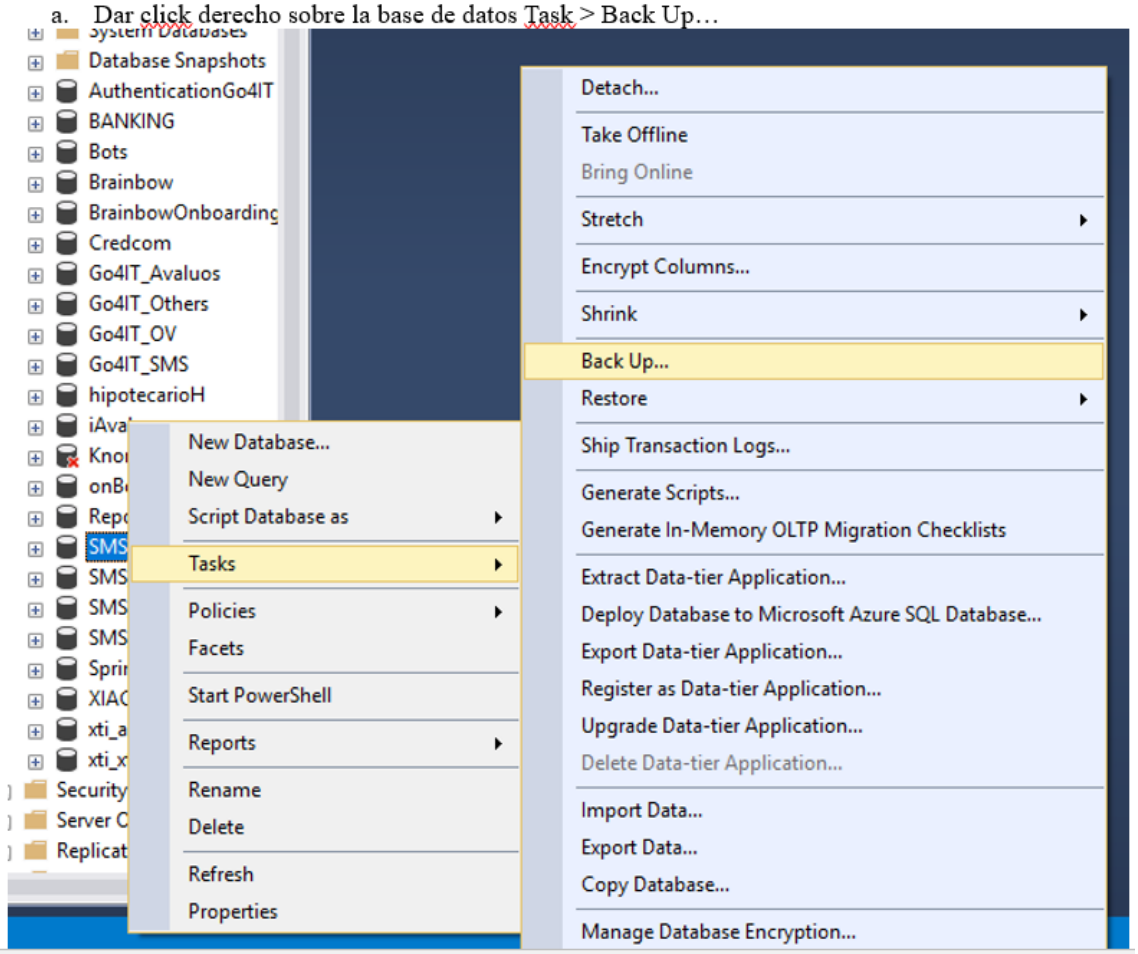


Figura 36. Pantalla del manual de instalación.

Finalmente, se tuvieron reuniones de trabajo muy intensas para poder decidir cómo íbamos a garantizar la escalabilidad de este aplicativo. Una vez que ya sabíamos que su rendimiento era el adecuado, decidimos la opción de levantar varias instancias en uno o más servidores. Una vez tomada la decisión, se generó la documentación pertinente para que cualquier persona con acceso al servidor en el cual se montaría la nueva instancia pudiera realizar la instalación del sistema sin problema alguno.

10. Resultados

El resultado de este proyecto fue el desarrollo, re-arquitectura y optimización de una plataforma de mensajería SMS, con lo cual se logró desarrollar una herramienta estable y optimizada. Gracias a este aplicativo, hoy la empresa tiene un producto más que ofrecer al mercado, así como que nuestros clientes cuentan con un canal más de comunicación con sus afiliados. Adicional a lo anterior, esta plataforma nos ha permitido incursionar en una nueva área por medio de un login único, apoyado en autenticación de dos factores, aumentando la seguridad de los aplicativos de la empresa y de nuestros clientes.

Como se observó previamente en otro apartado, gracias al trabajo y tiempo invertido en la optimización del aplicativo se logró una mejora en consumo de memoria RAM del 5000%, mientras que en las transacciones de SQL se observó un aumento en la velocidad de las transacciones en un 59%, permitiendo a esta aplicativo ser lo suficientemente ligero y parametrizable como para cumplir nuestras necesidades de escalabilidad y personalización.

Durante el desarrollo del aplicativo se tuvieron muy presentes las medidas básicas de seguridad informática y se dieron recomendaciones sobre como accionar con base en la experiencia profesional con la que se contaba, así como la labor de investigación necesaria para garantizarla.

No hay que olvidar que el desarrollo de la documentación generada durante todo el proyecto fue crucial para garantizar tanto la persistencia del negocio como para minimizar el tiempo que nuestros clientes necesitaban para poder utilizar el aplicativo, ya que inicialmente recibimos la versión atómica del mismo sin ninguna documentación de ningún tipo.

Gracias a la documentación del proyecto, se volvió más fácil la incorporación de nuevos integrantes al equipo de desarrollo de este aplicativo, ayudándoles a entender el funcionamiento general del sistema y de los módulos personalizables correspondientes. Mostrando los componentes y documentos a revisar en caso de que algún cliente reportara una incidencia. Incluso surgió la oportunidad de realizar una presentación comercial acerca de los diferentes componentes del sistema a clientes potenciales interesados en utilizar y adaptar nuestra plataforma.

Sin embargo, yo personalmente considero que el mayor resultado de este proyecto fue la implementación de SCRUM como metodología de desarrollo. Gracias a ello, actualmente en nuestros proyectos se respetan mejor los tiempos de cierre de un sprint. Hay una frase muy importante que aprendí durante mi tiempo en la empresa que es “lo que no se mide, no se puede mejorar”. A pesar de que hasta la fecha actual no se ha eliminado completamente el retraso de los proyectos (aunque si ha existido una mejoría notable tanto por el equipo como por el área administrativa), ahora tenemos una manera de poder medirlo, corregirlo y optimizarlo. Todavía nos encontramos en la etapa de optimización, pero durante cada iteración, cada sprint, el equipo aprende a estimar mejor los tiempos, riesgos y a tomar medidas para cumplir con los tiempos.

11. Conclusiones

La participación durante todo el desarrollo del proyecto me ha ayudado mucho en mi crecimiento personal y profesional. Siendo esta la primera empresa en la que laboro y mi primera experiencia profesional, el aprendizaje ha sido vasto tanto en aspectos técnicos como en los administrativos, desarrollando mi capacidad de análisis y resolución de problemas.

Este proyecto fue una gran experiencia para mí debido a que fue un proyecto que aparentemente era muy sencillo, pero conforme se iba desarrollando observábamos nuevas limitantes que hubo que sobrellevar para lograr finalizar un producto estable como lo es hoy en día. Aprendí muchísimas cosas y pude reafirmar mis conocimientos aprendidos durante mis estudios en la Facultad de Ingeniería.

Como lo es en general con el aprendizaje, fue un proceso largo y difícil, dado que se tenía la responsabilidad de que el sistema funcionara a la perfección antes, durante y después de cada nueva liberación, siendo una tarea nada fácil. Durante todo el proyecto se generaron algunos *bugs* que tuvieron que ser solucionados y en algunas ocasiones me llevaron a noches largas en la oficina debido a la dificultad para encontrar la causa del problema.

Uno de mis mayores aprendizajes no fue en el área técnica, si no en el área administrativa. El hecho de tratar directamente con los clientes no lo convierte en una labor sencilla, colaborando con ellos y coordinando esfuerzos para que pudieran utilizar el sistema sin problemas, inclusive cuando los errores no eran de nuestra herramienta. Los clientes son bastantes duros y demandantes, y durante este proyecto aprendí que están en su derecho de serlo y que como ingenieros debemos ser más humildes en nuestros desarrollos.

Suena fácil, pero no es sencillo el coordinar un proyecto tanto interno como cara a los clientes. Recuerdo muy claramente mi clase de Ingeniería de Software con el Dr. Daniel Trejo Medina que nos comentaba lo difícil que es tratar con clientes y por detrás coordinar un equipo. Como todo estudiante, cuando recibimos este comentario lo ignoramos o no le pusimos la importancia debida, pero al momento experimentarlo personalmente, las palabras toman un mayor sentido

12. Bibliografía

Bill Wagner. “Introduction to the C# Language and the .NET Framework.” *Microsoft Docs*. N.p., n.d. Web. 3 diciembre 2018. <https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework >

“Entity Framework Documentation”. Microsoft Docs. N.p., n.d. Web. 3 de diciembre 2018< https://docs.microsoft.com/en-us/ef/#pivot=entityfmwk&panel=entityfmwk1>

Rowan Miller. “Entity Framework Core.” *Microsoft Docs*. N.p., n.d. Web. 3 diciembre 2018. < https://docs.microsoft.com/en-us/ef/core/>

Microsoft. “Entity Framework” N.p., n.d. Web. 3 de diciembre 2018. < https://msdn.microsoft.com/en-us/library/gg696172(v=vs.103).aspx>

Rowan Miller. “Database Providers” *Microsoft Docs*. N.p., n.d. Web. 3 diciembre 2018. < https://docs.microsoft.com/en-us/ef/core/providers/index>

Rick Anderson “Introduction to ASP.NET Core” Microsoft Docs. N.o., n.d. Web 3 de diciembre 2018 < https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-2.1 >

Ingeniería del software Roger pressman, McGraw-Hill, 2010

Tridibesh Satpathy “SCRUM BODY OF KNOWLEDGE (SBOK GUIDE) Third Edition” 2017

“State Machine” N.p., n.d. Web 3 de diciembre 2018 < https://www.techopedia.com/definition/16447/state-machine >

Telerik. “Radgrid – Telerik Asp.Net grid” Telerik documentation. N.p., n.d. Web 12 de diciembre 2018. < https://demos.telerik.com/aspnet-ajax/grid/examples/overview/defaultcs.aspx >

Documentación EnjoyHint Web 15 de diciembre de 2018. < https://github.com/xbsoftware/enjoyhint >