Universidad Nacional Mayor de San Marcos

Universidad del Perú. Decana de América

Facultad de Ingeniería de Sistemas e Informática

Escuela Profesional de Ingeniería de Software

Desarrollo de una API REST para la integración de CRM con el sistema de gestión de colas en empresa de fondos colectivos

TRABAJO DE SUFICIENCIA PROFESIONAL

Para optar el Título Profesional de Ingeniero de Software

AUTOR

Kevin Arthur MENESES RIVERA

ASESOR

Javier Arturo GAMBOA CRUZADO

Lima, Perú

2018

Referencia bibliográfica

Meneses, K. (2018). *Desarrollo de una API REST para la integración de CRM con el sistema de gestión de colas en empresa de fondos colectivos*. [Trabajo de Suficiencia Profesional, Universidad Nacional Mayor de San Marcos, Facultad de Ingeniería de Sistemas e Informática, Escuela Profesional de Ingeniería de Software]. Repositorio institucional Cybertesis UNMSM.

Hoja de metadatos complementarios

Código ORCID del autor	—
DNI o pasaporte del autor	71819234
Código ORCID del asesor	https://orcid.org/0000-0002-0461-4152
DNI o pasaporte del asesor	17906323
Grupo de investigación	—
Agencia financiadora	Autofinanciado
Ubicación geográfica donde se desarrolló la investigación	Lugar: País: Perú Departamento: Lima Provincia: Lima Distrito: Miraflores Calle: Av. Arequipa 4598 Coordenadas geográficas: -12.10574, -77.0768046
Año o rango de años en que se realizó la investigación	2018
Disciplinas OCDE	Ingeniería de sistemas y comunicaciones https://purl.org/pe-repo/ocde/ford#2.02.04

UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS
FACULTAD DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
Escuela Profesional de Ingeniería de Software

Acta de Sustentación del Trabajo de Suficiencia Profesional

Siendo las 18:30 horas del día 21 de noviembre del año 2018, se reunieron los docentes designados como Miembros de Jurado del Trabajo de Suficiencia Profesional, presidido por el Ing. Cámara Figueroa Mario Adegundo (Presidente), Lic. Chávez Soto Jorge Luis (Miembro) y el Dr. Gamboa Cruzado Javier Arturo (Miembro Asesor) para la sustentación del Trabajo de Suficiencia Profesional Intitulado: ***"DESARROLLO DE UNA API REST PARA LA INTEGRACIÓN DE CRM CON EL SISTEMA DE GESTIÓN DE COLAS EN EMPRESA DE FONDOS COLECTIVOS",*** *por el Bachiller:* ***Meneses Rivera Kevin Arthur****; para obtener el Título Profesional de Ingeniero de Software.*

Acto seguido de la exposición del Trabajo de Suficiencia Profesional, el Presidente invitó al Bachiller a dar las respuestas a las preguntas establecida por los miembros del Jurado.

El Bachiller en el curso de sus intervenciones demostró pleno dominio del tema, al responder con acierto y fluidez a las observaciones y preguntas formuladas por los señores miembros del Jurado.

Finalmente habiéndose efectuado la calificación correspondiente por los miembros del Jurado, el Bachiller obtuvo la nota de ... 17 ... (En letras) ... diecisiete ...

A continuación el presidente de jurados el Ing. Cámara Figueroa Mario Adegundo, declara al Bachiller Ingeniero de Software.

Siendo las 19:00 horas, se levantó la sesión.

Presidente
Ing. Cámara Figueroa Mario Adegundo

Miembro
Lic. Chávez Soto Jorge Luis

Miembro Asesor
Dr. Gamboa Cruzado Javier Arturo

FICHA CATALOGRÁFICA

DESARROLLO DE UNA API REST PARA LA INTEGRACIÓN DE CRM CON EL SISTEMA DE GESTIÓN DE COLAS EN EMPRESA DE FONDOS COLECTIVOS

AUTOR: MENESES RIVERA, KEVIN ARTHUR

ASESOR: GAMBOA CRUZADO, JAVIER ARTURO

LIMA- PERÚ, 2018

Título Profesional/Grado Académico: Título Profesional de Ingeniero de Software

Área/Programa/Línea de Investigación: Ingenierías/Tecnologías de Información y Comunicación/Ingeniería de Software

Pregrado: Universidad Nacional Mayor de San Marcos – Facultad de Ingeniería de Sistemas e Informática – Escuela Profesional de Ingeniería de Software.

Formato 28 x 20 cm Páginas: xii, 125

DEDICATORIA

Este trabajo va dedicado a mis padres, por haberme forjado como persona, brindándome la oportunidad de estudiar y haberme apoyado en cada decisión en mi vida.

AGRADECIMIENTOS

Agradezco a la empresa Hiper S.A. por haberme brindado la oportunidad de formar parte de su staff profesional, a la Universidad Nacional Mayor de San Marcos por impartirme los conocimientos necesarios y herramientas para enfrentar el mundo laboral y a mi asesor por compartirme sus conocimientos con paciencia y dedicación.

UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS
FACULTAD DE INGENIERÍA DE SISTEMAS E INFORMATICA
ESCUELA PROFESIONAL DE INGENIERIA DE SOFTWARE

DESARROLLO DE UNA API REST PARA LA INTEGRACIÓN DE CRM CON EL SISTEMA DE GESTIÓN DE COLAS EN EMPRESA DE FONDOS COLECTIVOS.

Autor: MENESES RIVERA, KEVIN ARTHUR

Asesor: GAMBOA CRUZADO, JAVIER ARTURO

Título: Trabajo de Suficiencia Profesional para optar el Título Profesional de Ingeniero de Software

Fecha: Diciembre del 2018

RESUMEN

El volumen de clientes de la empresa de fondos colectivos Maquisistema ha incrementado considerablemente en los últimos años, así como también la cantidad de aplicativos utilizados por sus empleados, surgiendo así la necesidad de una solución que permita la integración del Sistema de Gestión de Colas al CRM. La solución desarrollada abstrajo las principales funcionalidades del sistema de gestión de colas, con especificaciones claras e intuitivas totalmente accesible para cualquier desarrollador. Durante el desarrollo del proyecto se realizaron ajustes a la web de generación de tickets del sistema gestión de colas, con la finalidad de actualizar en línea el estado de atención del ticket atendido desde el CRM, así como también el desarrollo de un servicio web que permita realizar la encuesta de atención desde una tablet conectada por USB. Los resultados esperados de la solución fueron exitosos, ya que actualmente cuentan con un nuevo canal de generación y atención de tickets que permite la atención del cliente desde el CRM.

Palabras claves: API, REST, CRM, servicio web, sistema de gestión de colas.

MAJOR NATIONAL UNIVERSITY OF SAN MARCOS
FACULTY OF SYSTEMS AND INFORMATICS ENGINEERING
PROFESSIONAL SCHOOL OF SOFTWARE ENGINEERING

DEVELOPMENT OF A REST API FOR INTEGRATION OF CRM AND QUEUE MANAGEMENT SYSTEM IN MUTUAL FUNDS COMPANY

Author: MENESES RIVERA, KEVIN ARTHUR
Advisor: GAMBOA CRUZADO, JAVIER ARTURO
Title: Work of Professional Sufficiency to opt for the Professional Title of Software Engineer
Date: December 2018

ABSTRACT

The volume of the company's customers of the collective funds Maquisistema has increased in recent years, as well as the amount of the results of their employees, thus emerging the need for a solution that allows the integration of the Queue Management System to the CRM The solution developed abstracted the main functionalities of the network management system, with the clear and intuitive ones totally accessible for any developer. During the development of the project adjustments are adjusted to the web of the generation of queue management system entries, with the purpose of updating online the status of the attention of the ticket attended from the CRM, as well as the development of a web service that allows you to conduct the service survey from a tablet connected via USB. The expected results of the solution were successful, as they now have a new ticket generation and service channel that allows customer attention from the CRM.

Key Words: REST, API, CRM, web service, queue management system.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

En el marco de mejorar la experiencia del cliente y agilizar su atención, la empresa de fondos mutuos Maquisistema identifica la necesidad de integrar las principales funcionalidades de su sistema de gestión de colas al CRM.

Para hacer frente a esta problemática, se plantea la implementación de una API REST que posibilite la apertura de un nuevo canal de atención y generación de tickets que replique las principales características del sistema de gestión de colas BMATIC.

El presente informe consta de 5 Capítulos:

En el Capítulo I - Trayectoria Profesional, especifica cronológicamente la experiencia profesional y formación académica del autor del presente informe.

En el Capítulo II - Contexto en el que se Desarrolla la Experiencia, se describe la empresa HIPER S.A. y las actividades desempeñadas en el proyecto.

En el Capítulo III - Actividades Desarrolladas, se define la situación problemática identificada por el cliente Maquisistema, la solución planteada, así como también el detalle de las etapas del proyecto, marco teórico utilizado para desarrollar la solución y los resultados.

En el Capítulo IV - Reflexión Crítica de la Experiencia, se describe el aporte del autor bajo el rol desempeñado, la experiencia adquirida y el desarrollo profesional.

En el Capítulo V - Conclusiones y Recomendaciones, se detallan las conclusiones de la implementación, así como también recomendaciones de mejora que se podrían llegar a implementar con la experiencia adquirida posterior al desarrollo de la solución.

Finalmente, se especifica información complementaria al informe mediante especificación de las fuentes de información, el glosario de términos y los anexos.

CAPÍTULO I: TRAYECTORIA PROFESIONAL

El autor del presente informe, se ha desempeñado a lo largo de su trayectoria profesional como desarrollador de software basado en tecnologías web. En un inicio, realizando el análisis y desarrollo de nuevas funcionalidades de aplicaciones web y aplicaciones de escritorio del Banco BCP. Actualmente se desempeña como programador en una empresa peruana de tecnologías de información, participando en el desarrollo de nuevas funcionalidades de un sistema de gestión de colas utilizando metodología SCRUM. Obtuvo el grado de Bachiller en Ingeniería de Software en la Universidad Nacional Mayor de San Marcos en el año 2016. Posee un nivel avanzado de inglés certificado por el Instituto Peruano Norteamericano, ha participado en cursos de extensión profesional de programación y es Scrum Master certificado por las organizaciones Scrum Alliance y ScrumStudy.

EXPERIENCIA PROFESIONAL	
Programador Java HIPER – TECNOLOGÍAS DE LA INFORMACION Desarrollador de nuevas funcionalidades, correcciones y mantenimiento de los módulos de un sistema de gestión de colas, utilizando metodología SCRUM y tecnología Java.	Diciembre 2016 – Presente
EVERIS PERU S. A. **Analista Programador** Asignado al dominio de Transactions and Collections del Proyecto BCP, realizando análisis y desarrollo de requerimientos para aplicativos desarrollados bajo la metodología de desarrollo PAR del Banco BCP y tecnología .NET.	Octubre 2015- Diciembre 2016

FORMACIÓN	
Grado Académico de Bachiller en Ingeniería de Software Escuela Académico Profesional de Ingeniería de Software Facultad de Ingeniería de Sistemas e Informática Universidad Nacional Mayor de San Marcos	Abril 2010- Diciembre 2016

IDIOMAS	
Idiomas – Ingles Avanzado Instituto Cultural Peruano Norteamericano	Marzo 2016- Diciembre 2018

CURSOS	
ANDROID ADVANCED DEVELOPER **Cibertec**	Octubre 2017 - Enero 2018
SCRUM MASTER CERTIFIED **Scrum Alliance**	Septiembre- Octubre 2016
SCRUM MASTER CERTIFIED **ScrumStudy**	Agosto- Setiembre 2016
ANDROID DEVELOPER **Cibertec**	Octubre 2015 - Enero 2016
NET 4.5.1 WEB APPLICATION DEVELOPER **Cibertec**	Noviembre 2014- Enero 2015

CERTIFICACIONES	
SCRUM MASTER CERTIFIED, Licencia 564203 **Scrum Alliance**	**Vigencia:** Sep. 2016 - Sep. 2020
SCRUM MASTER CERTIFIED, Licencia 550898 **ScrumStudy**	**Vigencia:** Ago. 2016 - Ago. 2020

OTROS CONOCIMIENTOS	
Lenguajes de Programación	Java, Javascript C, C#, VB, C++, SQL.
Sistemas Operativos	Windows, Ubuntu, Centos, macOS.
Manejadores de Base de Datos	SQL Server, PostgreSQL, MySQL y Oracle.

CAPÍTULO II: CONTEXTO EN EL QUE SE DESARROLLA LA EXPERIENCIA

2.1. EMPRESA – ACTIVIDAD QUE REALIZA

RESEÑA HISTÓRICA

Hiper es una empresa peruana que fue fundada en el año 1983, dedicada al diseño, desarrollo e implementación de soluciones tecnológicas para empresas con grandes volúmenes de clientes, estas soluciones tecnológicas están relacionadas con la atención de clientes y gestión de transacciones respondiendo eficazmente a las necesidades de sus clientes.

Hiper actualmente cuenta con las certificaciones IT-MARK y CMMI.v3, mostrando el gran interés de implementar la mejora continua en sus productos, servicios y operaciones.

La actividad que la empresa desarrolla es brindar productos y servicios de alta calidad a 17 países de nuestra región (más de 120 clientes B2B), aportando una vasta experiencia en procesos transaccionales financieros y no financieros. Hiper es representante oficial de Verifone en Perú desde el año 1991, desarrolla soluciones y brinda servicios de reparación para POS/Pinpad a sus clientes. (HIPER, 2018)

Datos de la Empresa (*):

Tabla 1. Datos de la empresa Hiper S.A.

(SUNAT, 2018)

Información de la empresa	
Razón Social	HIPER S.A.
Urbanización	Calle Beta Nº 181
Distrito/Ciudad	Callao
Departamento	Provincia Constitucional del Callao
RUC	20122882048
Principales Clientes	- Banco BBVA Continental - Banco Interbank - Banco de Crédito del Perú - Banco Scotiabank Perú - Banco GNB - Bancard Paraguay - Banco Popular - Costa Rica - Banco BBVA Colombia - ESSALUD - RENIEC - SUNAT - HERBALIFE - Caja Municipal Cuzco - Caja Municipal de Sullana - Caja Municipal Huancayo
Principales Proveedores	- Verifone - Tarjetas GEMALTO - HP - EPSON - SAP

2.2. VISIÓN

- Ser principales gestores tecnológicos en nuestro país con proyección a Latinoamérica**.**

- Ser socios tecnológicos de nuestros clientes, brindando calidad de servicio.

2.3. MISIÓN

- Con el cliente: Satisfacerlo con proyectos integrales y exitosos**.**

- Con el personal: Asumir nuevos retos tecnológicos que ayuden al logro del desarrollo integral.

- Con la sociedad: Facilitar las actividades de las personas e instituciones con soluciones creativas. (HIPER, 2014)

2.4. ORGANIZACIÓN DE LA EMPRESA

La organización de la empresa durante el periodo 2018 se estructura de la siguiente manera:

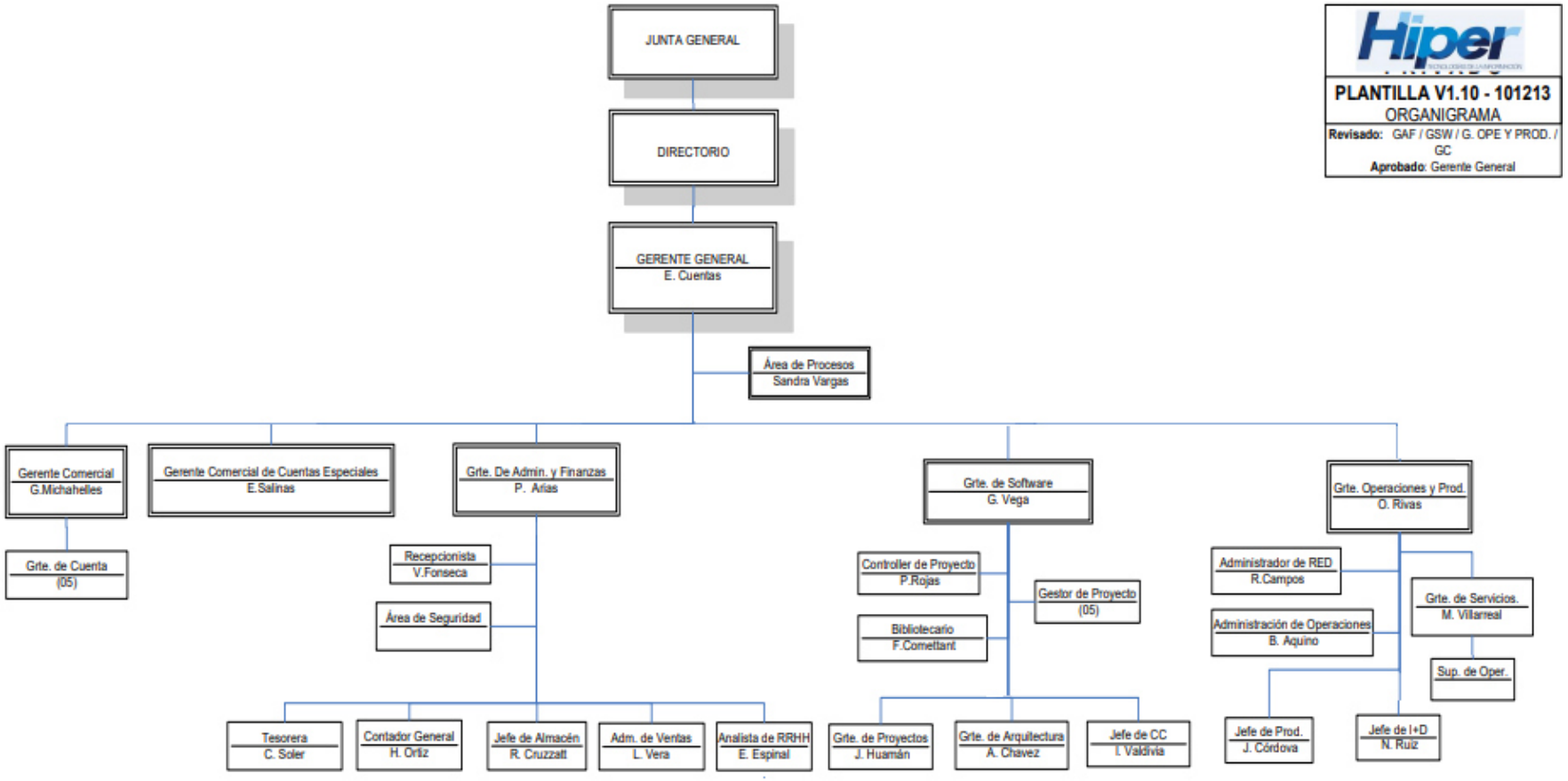


Figura 1. Organigrama de Hiper S.A 2018 (Intranet Hiper, 2018)

2.5. ÁREA, CARGO Y FUNCIONES DESEMPEÑADAS

2.5.1. ÁREA

El área en la cual se desempeñaron las funciones asignadas al cargo fue el área de Desarrollo de Aplicaciones de la empresa HIPER.

2.5.2. CARGO

Programador Java, bajo la Jefatura del área de Arquitectura de Software, como miembro del equipo asignado al producto BMATIC, constituido por 6 personas; un gestor de proyectos, un arquitecto de software, dos desarrolladores y un analista técnico.

2.5.3. FUNCIONES DESEMPEÑADAS

- Plantear soluciones de diseño y arquitectura en equipo.
- Levantamiento de información.
- Realizar pruebas de integración y capacitación del producto.
- Documentar manuales y especificaciones funcionales.
- Codificar funcionalidades para los productos en desarrollo.
- Capacitar al área de desarrollo, operaciones y clientes.

2.6. EXPERIENCIA PROFESIONAL REALIZADA EN LA ORGANIZACIÓN

PROYECTO BMATIC, ENCARGADO DEL MANTENIMIENTO, DESARROLLO DE NUEVAS FUNCIONALIDADES Y MODULOS PARA EL SISTEMA DE GESTIÓN DE COLAS.

- Análisis, diseño, implementación, mantenimiento de aplicativos Web y Servicios Web, utilizando tecnología java, específicamente Spring, Spring Boot, Struts, así como también Hibernate y JPA, SQL Server y Oracle para la capa de datos.

- Reuniones programadas con el Analista de Sistemas de la empresa de fondos colectivos para definir las funcionalidades y coordinaciones de pruebas para las pruebas de integración del proyecto.

- Codificación de los requerimientos.

- Generación de casos de prueba para los servicios REST a través de la herramienta POSTMAN, que posee una interfaz para generar peticiones para los servicios y almacenarlos como suite de pruebas.

- Capacitación del sistema de gestión de colas a miembros del área de desarrollo, partners y clientes.

- Coordinación con los miembros del equipo.

- Instalación del sistema de gestión de colas en la nube para las pruebas de integración con CRM de la empresa de fondos colectivos.

- Apoyo en las pruebas de integración.

- Documentación de las nuevas funcionalidades.

- Especificación de métodos de la API REST con Swagger.

CAPÍTULO III: ACTIVIDADES DESARROLLADAS

3.1. SITUACIÓN PROBLEMÁTICA

3.1.1. DEFINICIÓN DEL PROBLEMA

ANTECEDENTES

Maquisistema es una empresa de fondos colectivos cuya cantidad de asociados aumenta cada año, estos clientes visitan diariamente sus oficinas con la finalidad de realizar consultas para acceder a un financiamiento para la compra de vehículos e inmuebles.

Actualmente el público objetivo de la empresa, tanto clientes y no clientes son afectados por el largo tiempo de atención para ser atendidos debido que actualmente la empresa no utiliza el sistema de gestión de colas recientemente adquirido, pues desea integrarlo con el CRM, esto les acarrea tiempos de espera elevados para ser atendidos por largas colas, mermando la calidad de servicio y generando insatisfacción del público objetivo.

Adicional a lo mencionado, la empresa tiene un horario de atención poco flexible esto dificulta a las personas con horario de oficina a poder acercarse y realizar sus operaciones o consultas.

PROBLEMA PRINCIPAL

Ausencia de integración entre el CRM y el sistema de gestión de colas que permita la atención centralizada de los clientes, ante la necesidad de ejecutar las principales funcionalidades del sistema de gestión de desde el CRM.

PROBLEMAS ESPECIFICOS

- Ausencia de una solución innovadora que permita disminuir los tiempos de atención al cliente.

- Ausencia de un canal que permita generar y atender tickets de forma centralizada cuyas funcionalidades puedan utilizadas por nuevas futuras aplicaciones multiplataforma desarrolladas por la empresa.

- Ausencia de un sistema de gestión de colas que permita ser integrado con el CRM.

3.2. SOLUCIÓN

3.2.1. OBJETIVO GENERAL

Analizar, desarrollar e implementar una Interfaz de Programación de Aplicaciones (API) REST que abstraiga las principales funcionalidades del Sistema de Gestión de Colas y permita la integración del CRM.

3.2.2. OBJETIVOS ESPECÍFICOS

- Recuperar la imagen y presencia en el mercado de fondos colectivos del país, mejorando los tiempos de atención y hacer frente a la competencia.

- Generar un canal de emisión y atención de tickets que conserve las principales funcionalidades del Sistema de Gestión de Colas y pueda ser invocado desde aplicaciones multiplataforma.

- Mantener la consistencia de datos de la información generada a través del sistema de gestión de colas y el nuevo canal desarrollado.

3.2.3. ALCANCE

El alcance de la solución implementada engloba las funciones vinculadas a los procesos de Desarrollo de Software (Gerente de Proyecto, Gestor de Proyecto, Analista Funcional y Programador) y Control de calidad (Analista de Calidad y Tester).

ALCANCE FUNCIONAL

El alcance del proyecto comprende replicar las principales funcionalidades del aplicativo de gestión de colas hacia una API REST, así como también realizar adecuaciones en los módulos de ventanilla y encuesta del Sistema de Gestión de Colas para facilitar la integración con el CRM.

Los principales métodos de la API REST deben permitir:

- Realizar la creación de un ticket.
- Registro de operaciones asociadas.
- Finalizar la atención de un ticket desde un sistema externo.
- Derivar un ticket en atención.
- Generar un ticket interno.
- Envío de número de DNI y código de ticket a servicio web de empresa de fondos mutuos.

No está definido dentro del alcance del proyecto realizar las modificaciones al CRM para la invocación de los métodos de la API REST.

ALCANCE ORGANIZACIONAL

Comprende al área de Administración y Finanzas, que a su vez abarca la subgerencia de Sistemas.

ALCANCE GEOGRÁFICO

Empresa MAQUISISTEMA en Lima, distrito de Miraflores, en Av. Arequipa 4598.

3.2.4. ETAPAS Y METODOLOGÍAS

3.2.4.1. INICIO DEL PROYECTO.

El proyecto inició con la aprobación del Plan de Proyecto por parte de la Jefatura de la empresa de fondos colectivos, este plan es generado luego de haberse realizado reuniones con el cliente.

A continuación se adjunta un fragmento del documento de Plan de Proyecto para el proyecto.

Hiper

PLANTILLA – V2.41 – 151014
PLAN DE PROYECTO – T2
Revisado: APC | Aprobado: EPC
Página 2 de 8

PLAN DEL PROYECTO:

Integración de BMatic y CRM

Nombre del Cliente: MAQUISISTEMAS

Nro. Propuesta: 17671

VERSIÓN 1.0

Resumen :

Planificar las actividades a realizar durante el proyecto 17671 que implica la implementación de modificaciones a BMatic 5 con la finalidad de integrarse al CRM actual de Maquisistemas en el proceso de atención de sus clientes.

1 ALCANCE

1.1 CONTEXTO DEL PRODUCTO

El presente proyecto consiste en realizar modificaciones a la ventanilla web, creación de nuevos de componentes (API REST y Servicio de Encuesta) en la versión 5 del producto BMatic a fin de lograr la integración con el CRM del cliente.

1.2 FUNCIONES DEL PRODUCTO

Figura 2. Documento de Plan de Proyecto MASIS. (Plan-MASIS-BMAT5-Proyecto 17671, 2018)

Para la materialización del proyecto se utilizó la metodología ágil SCRUM, este enfoque ágil brinda un conjunto de principios, buenas prácticas y herramientas que ayudan a los

equipos a entregar productos en ciclos cortos, esto permite visualizar resultados anticipados, flexibilidad, adaptación y una rápida retroalimentación.

La implementación de la metodología SCRUM al presente proyecto se realizó de la siguiente manera.

ROLES

Dueño del Producto: Jefe de tecnología de Maquisistema.

Equipo de Desarrollo: Un arquitecto de software, dos desarrolladores y un analista técnico.

Scrum Master: Analista de Sistemas de Maquisistema.

ARTEFACTOS SCRUM

Pila de Producto: Lista priorizadas de requerimientos establecida por el dueño de producto.

Pila de Sprint: Lista de requerimientos escogida de la pila de producto para ser desarrollada en el sprint.

Incremento del producto: Lista de requerimientos completados durante el sprint.

EVENTOS DE SCRUM

Sprint: Se acordó una duración de dos semanas por sprint, en los cuales se presentaban funcionalidades visibles y no visibles para el usuario.

Planeamiento del sprint: Reunión realizada al comienzo cada sprint, aquí se define los elementos de la pila del producto se desarrollaran durante el ciclo y se entregarán al final de este.

Revisión del Sprint: Reunión desarrollada al finalizar cada Sprint., en este punto se realiza la inspección del incremento desarrollado, esta fase reúne a los miembros de equipo, dueño del producto y stakeholders.

Scrum Diario: Reunión realizada al inicio del día, generalmente 9:00 am, la duración de cada reunión fue de 15 minutos.

Retrospectiva del Sprint: No implementada.

El siguiente gráfico muestra las actividades y artefactos Scrum y su relación a través del flujo de mejora continua.

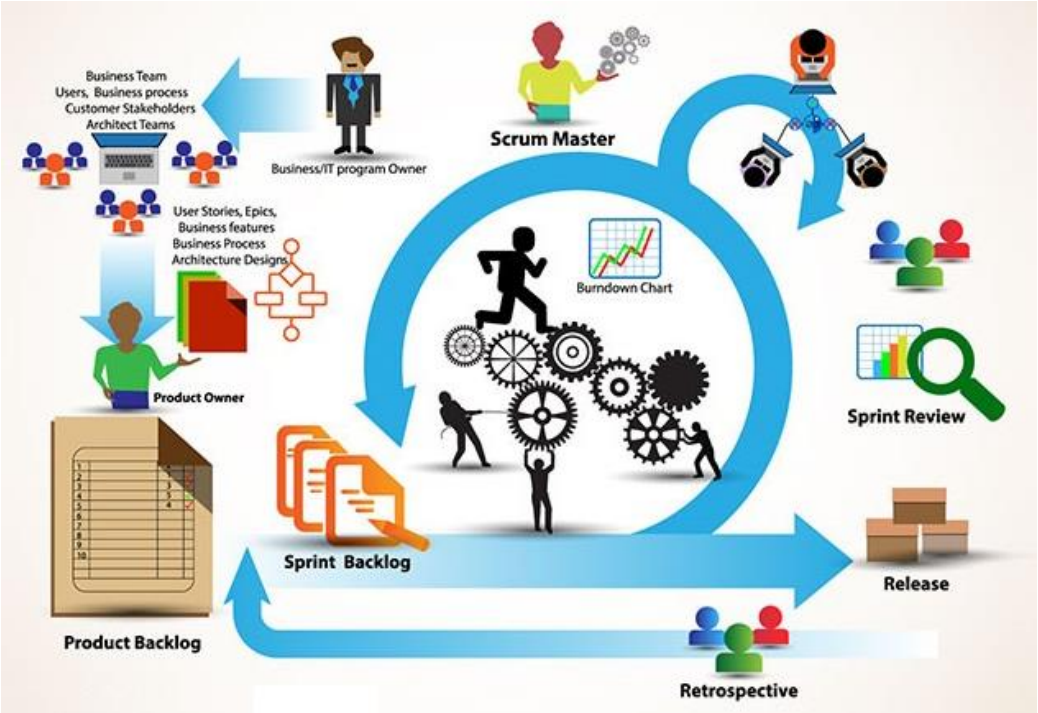


Figura 3. Ciclo Scrum. (A Guide to the Scrum Body Of Knowledge, 2018)

Para el presente proyecto, el incremento realizado al finalizar el sprint fue enviado al área de Control de Calidad de la empresa Hiper para realizar las certificación correspondiente, en la forma tradicional de implementación de la metodología ágil SCRUM el incremento es enviado a producción y el testing es realizado por un tester que pertenece al equipo de desarrollo.

Mientras el equipo de Calidad de Software de Hiper realizaba las pruebas, el equipo de desarrollo continuaba con la codificación del siguiente sprint, de encontrarse alguna incidencia esta era agregada a la pila del sprint para su pronta asignación.

3.2.4.2. ANÁLISIS DE LA SOLUCIÓN.

Para solucionar la problemática del cliente se tuvo como opciones las siguientes alternativas.

1- Realizar la implementación de las funcionalidades del sistema de gestión en el CRM.

2- Desarrollar una API REST que abstraiga las funcionalidades del Sistema de Gestión de Colas.

Escogiéndose la opción 2, que es óptima y escalable, pues la solución permite ser utilizada por futuros desarrollos de la empresa.

La solución seleccionada utiliza como formato de intercambio de datos JSON y cumple con los siguientes requisitos de arquitectura para ser considerada una API REST:

Arquitectura cliente-servidor: La independencia entre cliente y servidor posibilita una flexibilidad muy alta.

Stateless: Utiliza el protocolo sin estado HTTP, por lo que no almacena datos del cliente para mantener un estado del mismo.

Cacheable: Almacena en cache peticiones, con la finalidad de aumentar el rendimiento y escalabilidad.

Sistema por capas: No fuerza al cliente a identificar mediante que capas se tramita la información, esto permite la conservación de independencia del cliente con respecto a dichas capas.

Interfaz uniforme: La interfaz de comunicación establecida por el cliente y servidor no depende del servidor en el que se ejecuta las solicitudes, menos aún del cliente, de tal forma se certifica que no importa el origen de las solicitudes ni quien las admita, siempre

y cuando se cumpla con la interfaz previamente establecida. Esta regla hace más simple el protocolo incrementando el desempeño y escalabilidad sistema.

La solución cumple con los requisitos no funcionales de interoperabilidad, seguridad, optimización de recursos, facilidad de análisis y pruebas, facilidad de instalación y reemplazo gracias al uso de tokens y cumplimiento de los requisitos de la arquitectura Rest.

Para el análisis de la solución se realizaron reuniones de levantamiento de información en la empresa de fondos colectivos, de las cuales se identificaron los siguientes requerimientos:

Nº 1 – Envío de número de DNI y código de ticket a servicio web de Maquisistema

Se requiere que el aplicativo BMatic envíe el código del ticket y el número de DNI asociado al ticket a un web service de Maquisistema.

El envío de estos datos se realizará cuando el operador de la ventanilla inicie la atención de ticket mediante el botón *Iniciar* de la ventanilla.

Consideraciones:

- El envío de los datos debe ser interno y no interrumpir las funciones de la ventanilla.

- Maquisistema desarrollará un web service que recibirá los datos enviados por Bmatic.

- El contrato del web service será definido por Hiper en coordinación con Maquisistema.

- Se modificará la ventanilla para que genere una notificación de envío del DNI y código del ticket ante el evento de inicio de atención del ticket.

- La dirección del web service de Maquisistema a donde Bmatic enviará los datos, se leerá de un archivo de configuración.

Nº 2 – Registro de operaciones asociadas al ticket en Bmatic desde un sistema externo

Durante la atención del ticket, el operador de la ventanilla selecciona tipificaciones en el CRM de Maquisistema. Se requiere que estas tipificaciones se registren como operaciones asociadas al ticket que se está atendiendo en el sistema Bmatic.

Consideraciones:

- Se desarrollará un método en la API REST de Bmatic que recibirá las tipificaciones enviadas por el CRM de Maquisistema y las asociará al ticket en atención.

- El desarrollo de las modificaciones del CRM están a cargo de Maquisistema.

- Las tipificaciones del CRM deben estar previamente creadas como operaciones en Bmatic, mediante su registro en la web administrativa.

Nº 3 – Finalizar la atención de un ticket desde un sistema externo

Se requiere que se pueda finalizar la atención de un ticket desde la pantalla del CRM de Maquisistema.

Consideraciones:

- Se desarrollará método en la API REST de Bmatic con la funcionalidad de finalizar la atención de un ticket en atención, el cual será consumido por el CRM de Maquisistema.

- El desarrollo de las modificaciones del CRM están a cargo de Maquisistema.

- Se modificará la ventanilla para que la finalización externa del ticket se refleje en la ventanilla y quede en el mismo estado como si se hubiese culminado la atención con el botón Finalizar de la ventanilla.

- En el caso se tenga configurada la encuesta, la finalización externa del ticket debe desencadenar el inicio del registro de la encuesta en la tablet manteniendo el mismo comportamiento como si se hubiese culminado la atención con el botón Finalizar de la ventanilla.

Nº 4 - Derivar un ticket en atención desde un sistema externo a Bmatic

Se requiere que se pueda derivar la atención de un ticket desde la pantalla del CRM de Maquisistema.

Consideraciones:

- Se desarrollará un método en la API REST de Bmatic que permitan derivar un ticket a un tipo de ventanilla especificado, el cual será consumido por el CRM de Maquisistema.

- El desarrollo de las modificaciones del CRM están a cargo de Maquisistema.

- Se modificará la ventanilla para que la derivación externa del ticket se refleje en la ventanilla y quede en el mismo estado como si se hubiese derivado el ticket desde la ventanilla y permita luego seguir atendiendo nuevos tickets.

- En caso esté configurada la encuesta, esta solo se desencadenará cuando se finalice la atención del ticket derivado, en la última ventanilla que lo atienda.

Nº 5 - Generar un ticket interno desde un sistema externo a Bmatic

Se requiere que se pueda generar un ticket interno desde la pantalla del CRM de Maquisistema.

Consideraciones:

- Se desarrollará un método en la API REST de Bmatic que permita generar un ticket interno el cual será consumido por el CRM de Maquisistema.

- El desarrollo de las modificaciones del CRM están a cargo de Maquisistema.

- Se modificará la ventanilla para que inicie la atención del ticket interno.

- Este requerimiento mantiene el funcionamiento actual del ticket interno de la versión estándar de Bmatic.

Nº 6 - Acceso a bases de datos Operativa de Bmatic

El cliente requiere acceso a las bases de datos operativas del sistema con la finalidad de poder generar sus reportes personalizados.

Consideraciones:

- Las consultas que realice el cliente pueden causar bloqueo de tablas.

- La creación de consultas a las bases de datos están a cargo del cliente.

- Las nuevas versiones del sistema pueden incluir cambios en el diseño de la base de datos por lo que el cliente debería implementar cambios sus consultas para soportar las nuevas versiones.

- Se requiere una explicación al cliente del diseño de la base de datos operativa.

A partir de los requerimientos mencionados se especificaron las siguientes historias de usuario:

Tabla 2. Lista de Historias de Usuario del proyecto.

(Confluence - MASIS-BMAT5-Proyecto 17671, 2018)

H0101	Definición de las funcionalidades del sistema de gestión de colas a implementar en la API REST
H0102	Implementar arquitectura inicial de la API REST
H0201	Implementar arquitectura de autenticación JWT
H0202	Envío de información del ticket desde ventanilla a CRM/simulador
H0301	Crear método en la API REST para registrar tipificaciones
H0302	Modificación de servicio de encuesta para ser invocado por sistema externo
H0401	Crear método en la API REST para listar los sectores de una agencia
H0402	Crear método en la API REST para listar los tipos de ventanilla
H0403	Crear método en la API REST para listar los tipos de ticket que pueden ser generados por un tipo de ventanilla
H0404	Crear método en la API REST para obtener datos de la ventanilla en base al código del operador
H0501	Crear método en la API REST para finalizar ticket
H0502	Crear método en la API REST para derivar ticket.
H0503	Crear método en la API REST para crear ticket interno.
H0601	Actualizar el estado de la ventanilla al realizar la derivación del ticket.
H0602	Actualizar el estado de la ventanilla al finalizar la atención del ticket.
H0603	Actualización estado de ventanilla al iniciar atención.

A continuación se detallan las historias de usuario más relevantes en el desarrollo del proyecto.

H0501- Crear método en la API REST para finalizar ticket

Tabla 3. Historia de Usuario - Crear método en la API REST para finalizar ticket.
(Confluence - MASIS-BMAT5-Proyecto 17671, 2018)

Req	3
ID	H0501
Iteración	5
Status	**CERTIFICADO**
Historia	Como sistema/aplicación externo quiero finalizar la atención de un ticket.
Criterios de Aceptación	Sólo debe permitirse la finalización de tickets que se encuentren **en atención.** En caso de tener **activado** el uso de operaciones para el sistema, sólo debe permitirse la finalización de aquellos tickets **en atención** que tengan registrado como mínimo **una operación**. En caso de tener **desactivado** el uso de operaciones para el sistema, se permitirá finalizar la atención de tickets **en atención** sin que tengan registrado operación alguna. La hora de finalización del ticket se actualiza según la hora del sistema (base de datos) El ticket deberá **cambiar su estado** a nivel de base de datos. Pasar de 'A' a 'Y'. Deberá verificarse a nivel de base de datos el **registro correcto** de calificación realizada en la encuesta.

Diagrama de Secuencias

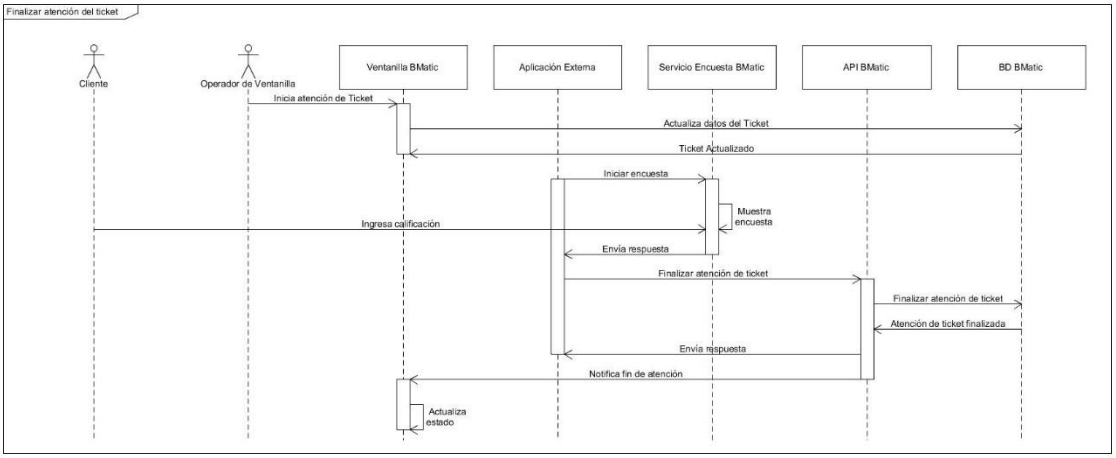


Figura 4. Crear método en la API REST para finalizar ticket. (Confluence - MASIS-BMAT5-Proyecto 17671, 2018)

Secuencia:

1. El operador inicia la atención del ticket.
2. La ventanilla actualiza los datos de inicio de atención del ticket en el repositorio de datos.
3. El aplicativo externo invoca al servicio de Encuesta de Bmatic.
4. Dicho servicio desencadena la encuesta.
5. El cliente ingresa la calificación.
6. El servicio de encuesta envía la respuesta a la aplicación externa.
7. La aplicación externa invoca al servicio "Finalizar atención de ticket" de la API de BMatic.
8. La API REST finaliza la atención del ticket en el repositorio de datos.
9. La API REST envía una respuesta a la aplicación externa.
10. La API REST notifica a la ventanilla la finalización del ticket.
11. La ventanilla actualiza su estado.

H0502 - Crear método en la API REST para derivar ticket.

Tabla 4. Historia de Usuario - Crear método en la API REST para derivar ticket.
(Confluence - MASIS-BMAT5-Proyecto 17671, 2018)

Req	4
ID	H0502
Iteración	1
Status	**CERTIFICADO**
Historia	Como sistema/aplicación externo quiero derivar la atención de un ticket.
Criterios de Aceptación	Sólo debe permitirse la derivación de tickets que se encuentren **en atención.** Debe verificarse que el nuevo ticket generado (derivado) se encuentre **en espera** (estado 'E'). El ticket original deberá actualizar su hora de fin según la hora del sistema (base de datos) y cambiar a atendido (estado 'Y')

Diagrama de Secuencias

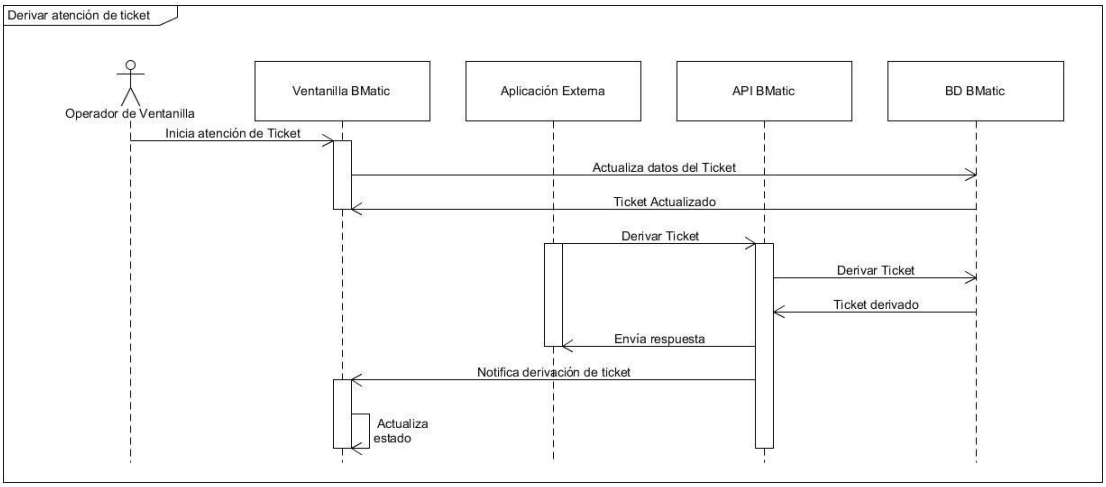


Figura 5. Crear método en la API REST para derivar ticket. (Confluence - MASIS-BMAT5-Proyecto 17671, 2018)

Secuencia:

1. El operador inicia la atención del ticket.
2. La ventanilla actualiza los datos de inicio de atención del ticket en el repositorio de datos
3. El aplicativo externa invoca al servicio "Derivar atención de ticket" de la API REST de BMatic.
4. La API REST finaliza la atención del ticket en el repositorio de datos.
5. La API REST envía una respuesta a la aplicación externa.
6. La API REST notifica a la ventanilla la derivación del ticket.
7. La ventanilla actualiza su estado.

H0503 - Crear método en la API REST para generar ticket interno.

Tabla 5. Historia de Usuario -Crear método en la API REST para generar ticket interno**.**
(Confluence - MASIS-BMAT5-Proyecto 17671, 2018)

Req	5
ID	H0503
Iteración	5
Status	**CERTIFICADO**
Historia	Como sistema/aplicación externo quiero generar un ticket interno
Criterios de Aceptación	La ventanilla para la cual se generará el ticket interno debe estar **libre** La ventanilla debe estar **autorizada** para generar ticket interno (en caso esté activada esta configuración) La ventanilla debe estar **configurada** para poder atender el tipo de ticket interno que se desea crear. El operador (código) enviado como parámetro debe tener una sesión **activa** actual en la ventanilla (código) enviado también como parámetro Debe verificarse que la ventanilla **automáticamente se asigna** el ticket interno luego de generarse. El ticket cambia a estado 'A' (en atención)

Diagrama de Secuencias

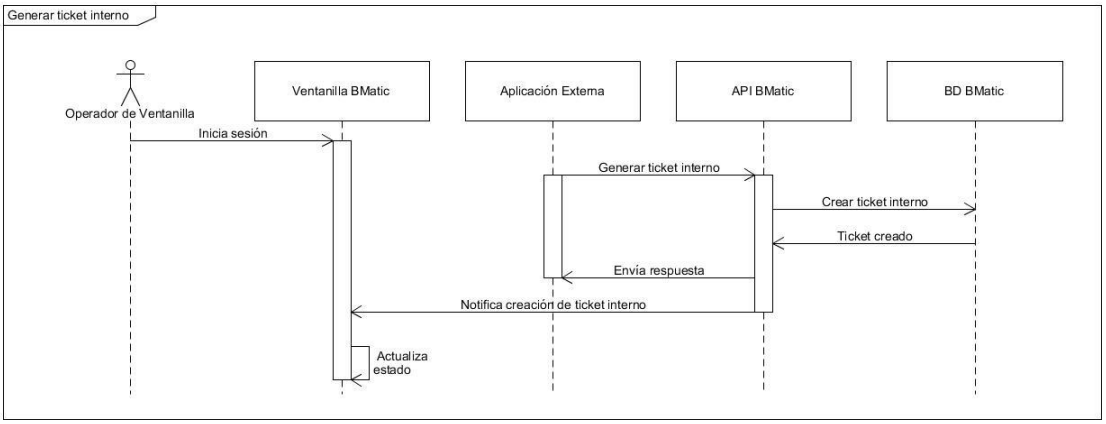


Figura 6. Crear método en la API REST para generar ticket interno. (Confluence - MASIS-BMAT5-Proyecto 17671, 2018)

Secuencia:

1. El operador inicia sesión en la ventanilla.
2. La aplicación externa invoca al servicio "Generar Ticket Interno" de la API de BMatic.
3. La API REST genera el ticket interno.
4. La API REST envía una respuesta a la aplicación externa.
5. La API REST notifica a la ventanilla sobre la creación del ticket interno
6. La ventanilla actualiza su estado (se asigna el ticket automáticamente)

H0301-Crear método en la API REST para registrar tipificaciones

Tabla 6. Historia de Usuario - crear método en la API REST para registrar tipificaciones. (Confluence - MASIS-BMAT5-Proyecto 17671, 2018)

Req	2
ID	H0301
Iteración	3
Status	**CERTIFICADO**
Historia	Como sistema/aplicación externo quiero agregar distintas operaciones a un ticket en atención
Criterios de Aceptación	Sólo debe permitir el registro de operaciones asociadas a un ticket si la opción "Activar operaciones" del módulo "Parámetros Generales" se encuentra **activada** Sólo debe permitir el registro de operaciones asociadas a un ticket que se encuentre **en atención** Al registrar una operación se debe considerar: El código del tipo de operación a registrar debe encontrarse configurado en BMatic. La hora de inicio de dicha operación deberá ser obtenida de la hora del sistema (base de datos) La hora de fin de dicha operación se actualizará recién al registrar la **siguiente** operación para el ticket. La hora del fin de la última operación registrada se actualizará al **finalizar** la atención del ticket.

Diagrama de Secuencias

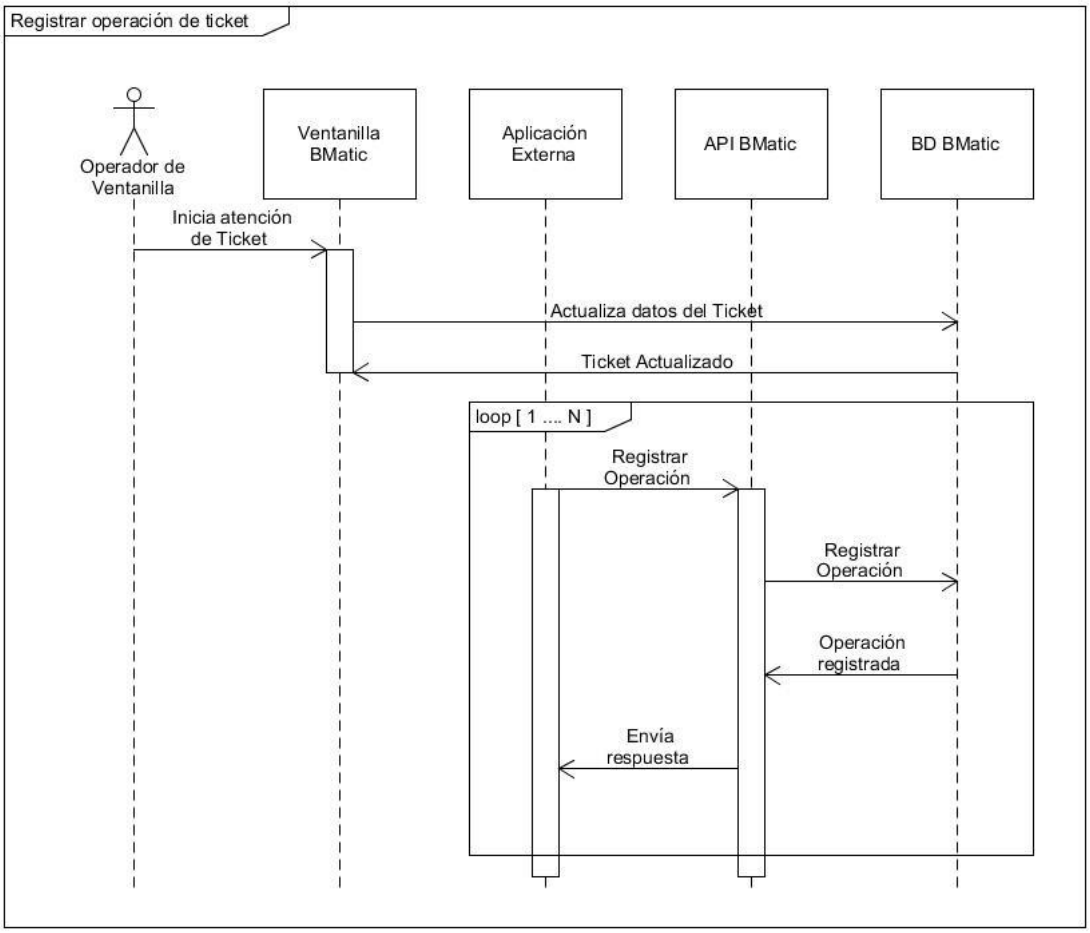


Figura 7. Crear método en la API REST para registrar tipificaciones.

(Confluence - MASIS-BMAT5-Proyecto 17671, 2018)

Secuencia:

1. El operador inicia la atención del ticket.
2. La ventanilla actualiza los datos de inicio de atención del ticket en el repositorio de datos.
3. El aplicativo externo invoca al servicio "Registrar Operación" de la API de BMatic.
4. La API REST registra la operación para el ticket en el repositorio de datos.
5. La API REST envía una respuesta a la aplicación externa

Considerar:

- Puede registrarse más de una operación asociada al ticket.

H0202-Envío de información del ticket desde ventanilla a CRM

Tabla 7. Historia de Usuario - Envío de información del ticket desde ventanilla a CRM. (Confluence - MASIS-BMAT5-Proyecto 17671, 2018)

Req	1
ID	H0202
Iteración	1
Status	**CERTIFICADO**
Historia	Como sistema/aplicación externo quiero recibir los datos de un ticket al momento de iniciar su atención en la ventanilla web
Criterios de Aceptación	El ticket deberá iniciar su atención en la ventanilla **independientemente** del resultado del envío de datos al servicio web externo.

Diagrama de Secuencias

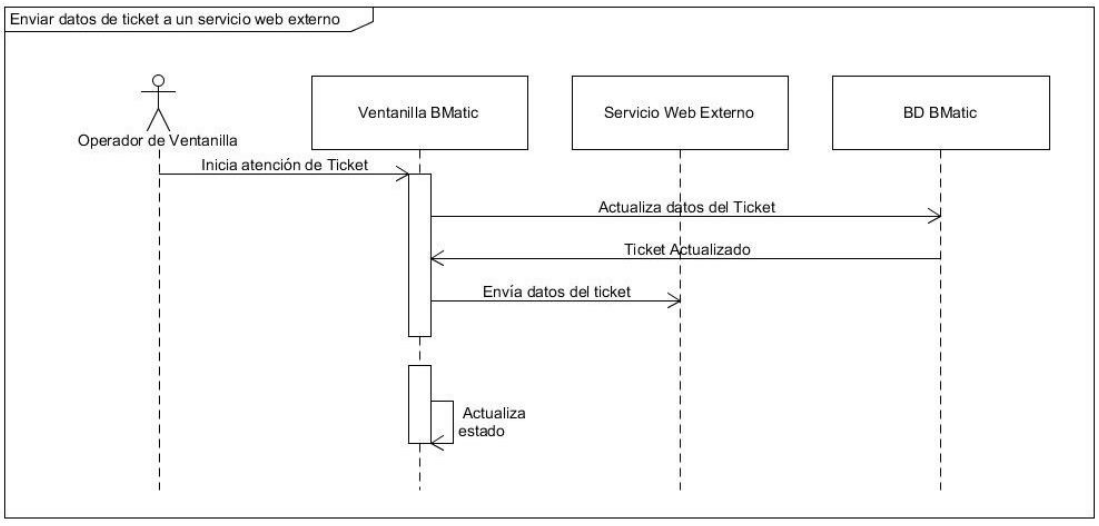


Figura 8. Envío de información del ticket desde ventanilla a CRM. (Confluence - MASIS-BMAT5-Proyecto 17671, 2018)

Secuencia:

1. El operador inicia la atención del ticket en la ventanilla.
2. La ventanilla actualiza los datos del ticket en la base de datos.
3. La ventanilla envía los datos del ticket al servicio web externo.
4. La ventanilla actualiza su estado.

Considerar:

Él envió de datos del ticket al servicio web externo se realizará de forma **asíncrona**.

Arquitectura de la solución

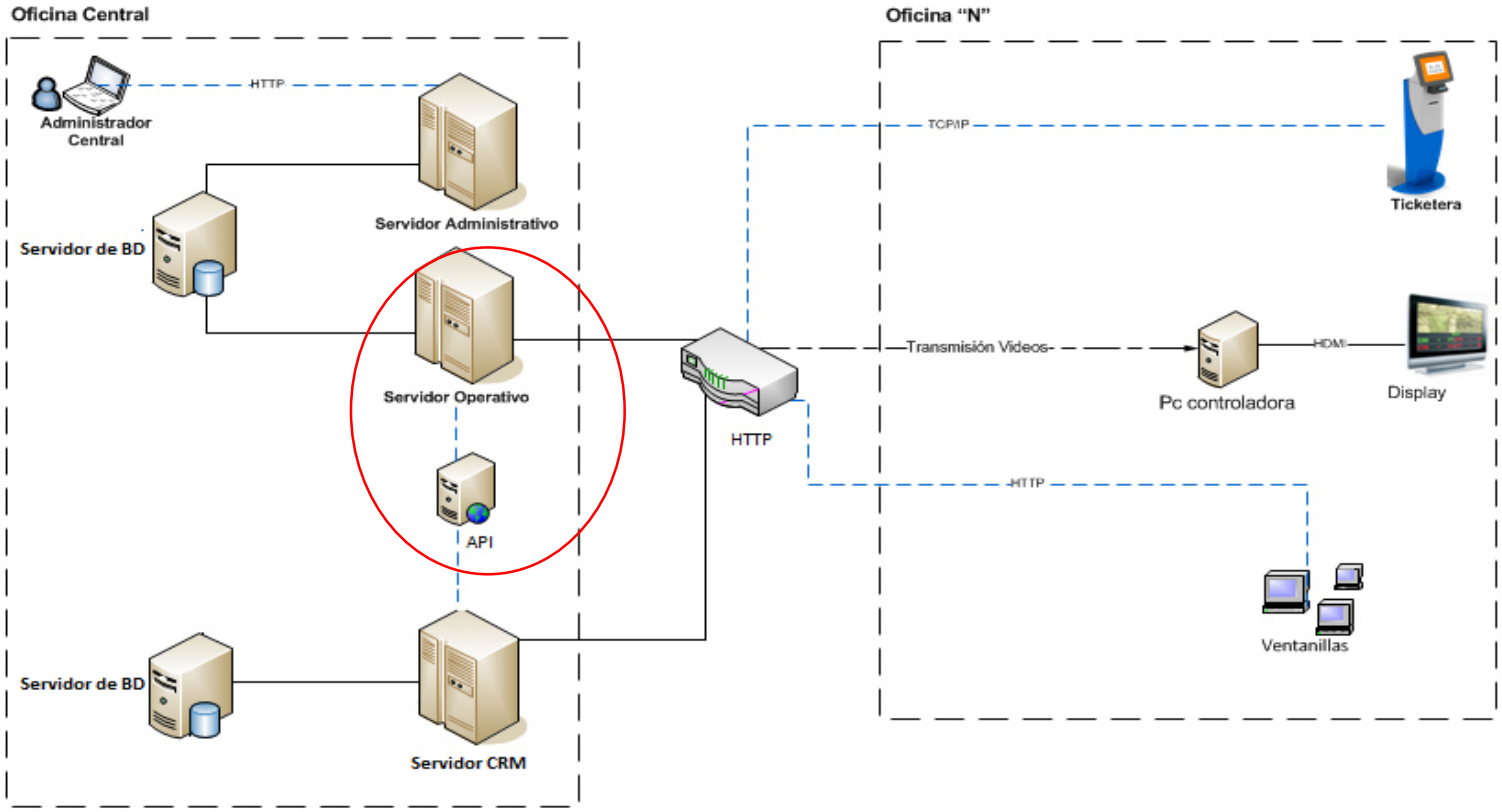


Figura 9. Arquitectura final de la solución implementada para Maquisistema. (Confluence - MASIS-BMAT5-Proyecto 17671, 2018)

En donde:

Tabla 8. Descripción de los elementos de la arquitectura. (Confluence - MASIS-BMAT5-Proyecto 17671, 2018)

SERVIDOR	**DESCRIPCIÓN**
Servidor Administrativo	El servidor central que permite realizar configuraciones y consultas al sistema, es un computador cuyos componentes principales son: **Servidor de aplicaciones:** Es el servicio donde residen las aplicaciones que permiten gestionar la red de agencias desde un punto central. **Servidor de videos:** Es el servidor que atiende los requerimientos de transmisión de videos y cambios en la programación por parte de los displays en las oficinas. **Reportador BMatic:** Reportes basados en los modelos Datamart que permitan consultar la situación operativa funcional de las oficinas. **Monitoreo:** Se cuenta con los Tableros de Control (Dashboard) Gerencial y de Oficina necesarios para la toma acertada de decisiones.
Servidor Operativo	El servidor central que gestiona todas las transacciones de las ventanillas, tiqueteras y display. Es un computador cuyos componentes principales son: **Servidor de aplicaciones:** Es el servicio donde residen los servicios y aplicaciones necesarios para operar las ventanillas, ticketeras y otros. **Servicio API para ticket virtual y atención de ventanillas**: Es el servicio que atiende a aplicaciones externas que se

	integren a Bmatic como Smartphones, CRM, ERP, otras aplicaciones desktop, Web o servicios de integración.
Servidor de BD BMATIC	Servidor de Base de Datos del aplicativo BMATIC que es utilizado por los diversos módulos, componentes web, servicios web y API del aplicativo BMATIC.
Servidor CRM	Servidor de Aplicaciones que aloja el CRM de la empresa de fondos mutuos.
Servidor de BD CRM	Servidor de Base de Datos del CRM de la empresa de fondos mutuos.

3.2.3.3. DISEÑO.

La etapa de diseño consistió en realizar las especificaciones de la API.

Cada especificación detalla las llamadas que se pueden ejecutar, así como también las respuestas que retornan. Además, el formato de la descripción de un método está formado mediante la estructura de datos JSON que contiene claves especificadas en la tabla detallada a continuación.

Tabla 9. Estructura de métodos de la API.

(Confluence - MASIS-BMAT5-Proyecto 17671, 2018)

Estructura de la descripción de un método de la API			
Clave	Tipo	Ejemplo	Definición
id	Entero	1	Identificador único del método. Es generado de forma automática en el servidor
nombre	Cadena	Internal	Denominación del método
método	Cadena	POST	Especifica el método Http del endpoint.
descripción	Cadena	El método genera un ticket interno para el usuario en sesión de la ventanilla especificada en la trama	La descripción del método
endpoint	Cadena	/api/v1/tickets/internal	Plantilla ocupada con parámetros cuyo punto final establece la ubicación del recurso.

parámetros	Lista	{
		"counter_id": "W000030",
		"teller_id": "swaner4",
		"queuetype_id": "R0001"
		}
			Especifica los parámetros mediante una lista, donde cada uno es un JSON.

En la siguiente tabla se detalla la estructura de los parámetros:

Tabla 10. Estructura de parámetros de la API.

(Confluence - MASIS-BMAT5-Proyecto 17671, 2018)

Estructura de un parámetro		
Clave	**Valor de ejemplo**	**Descripción**
Nombre	counter_id	Nombre de usuario de ventanilla del sistema.
Descripción	Nombre de usuario con sesión activa en la ventanilla.	Describe el significado del parámetro.
Requerido	Sí	Establece la obligatoriedad de los parámetros. Posibilidades: Sí o No.
Ubicación	Path	Especifica la ubicación del parámetro. Existen tres opciones: Path, Payload o Parameters
Opciones válidas	-	Indica los valores aceptables por el tipo de dato definido.
Valor por defectos	-	Indica el valor por defecto que toma el parámetro.
Tipo de dato	Cadena	Define de tipo es el parámetro.

A continuación se detallan las principales peticiones admitidas por la API REST.

3.5.1. Crear método en la API REST para finalizar ticket

Método: POST

Endpoint: /v1/tickets/{ticket}/close

Descripción: Finaliza la atención de un ticket.

Parámetros de la petición:

- ticket_id**:**

  Descripción: Identificador único, autogenerado y encriptado del ticket.
  Requerido: Sí.
  Ubicación: Path.
  Tipo de dato: String.

- phone:

  Descripción: Teléfono del cliente para las notificaciones.
  Requerido: No.
  Ubicación: Parameters.
  Tipo de dato: String.
  Opciones válidas: [1-3]
  Valor por defecto Nulo.

- qualification_value: valor de calificación de la atención del ticket.

  Descripción: Valor de calificación de la atención del ticket.
  Requerido: No.
  Ubicación: Parameters.
  Tipo de dato: String.
  Opciones válidas: +[1-9]{1}[0-9]{3,13}
  Valor por defecto Nulo.

Ejemplo de petición:

```
POST /v1/tickets/89m32b0/close
{
        "phone": "+51947812025",
```

"qualification_value": 3
}

Respuestas:

- **Código 200:** Ticket finalizado con éxito. Devuelve el ticket.

  Ejemplo de respuesta:

{
"id": "89m32b0",
"number": 2,
"status": "ATTENDED",
"position": 10,
.
.
.
"status": "ATTENDED",
"started_at": "2017-02-20T10:10:55.000Z",
"finished_at": "2017-02-20T10:11:41.000Z"
}

- **Código 400:** Uno o más parámetros no son válidos, devuelve una descripción de validación fallida.

- **Código 404:** El recurso solicitado no encontrado, devuelve un mensaje de error simple.

- **Código 400:**

  Uno o más parámetros no son válidos, devuelve una descripción de validación fallida

- **Código 500:** Se ha producido un error inesperado, devuelve un mensaje de error simple.

3.5.2. Crear método en la API REST para derivar ticket

Método: POST

Endpoint: /v1/tickets/{ticket}/transfer

Descripción: Realiza la derivación de un ticket hacia otro tipo de ventanilla.

Parámetros de la petición:

- ticket_id:

  Descripción: Identificador único, autogenerado y encriptado del ticket.
  Requerido: Sí.
  Ubicación: Path.
  Tipo de dato: String.

- sector_id:

  Descripción: Identificador único del sector de la agencia.
  Requerido: Sí.
  Ubicación: Parameters.
  Tipo de dato: String.

- countertype_id:

  Descripción: Identificador único del tipo de ventanilla.
  Requerido: Sí.
  Ubicación: Parameters.
  Tipo de dato: String.

- queuetype_id:

  Descripción: : Identificador único del tipo de ticket.
  Requerido: Sí.
  Ubicación: Parameters.
  Tipo de dato: String.

Ejemplo de petición:

POST /v1/tickets/89m32b0/transfer

```
{
        "sector_id": "s51n5",
        "countertype_id": "v56a5",
        "queuetype_id": "q5s53"
}
```

Respuestas:

- **Código 200:** Ticket derivado con éxito. Devuelve el nuevo ticket generado.

  Ejemplo de respuesta:

```
{
        "id": "12x81z1",
        "number": 2,
        "status": "WAITING",
        "internal": 1,
        "temporal": 0,
        "derived": 1,

        .
        .
        .
        "created_at": "2017-02-20T10:00:00.000Z"
}
```

- **Código 400:** Uno o más parámetros no son válidos, devuelve una descripción de validación fallida.

- **Código 404:** El recurso solicitado no encontrado, devuelve un mensaje de error simple.

- **Código 400:**

  Uno o más parámetros no son válidos, devuelve una descripción de validación fallida

- **Código 500:** Se ha producido un error inesperado, devuelve un mensaje de error simple.

3.5.3. Crear método en la API REST para generar ticket interno

Método: POST

Endpoint: /v1/tickets/internal

Descripción: Creación de un ticket interno cuando el usuario se encuentra con una sesión activa en la ventanilla.

Parámetros de la petición:

- counter_id:

  Descripción: Identificador único de la ventanilla.
  Requerido: Sí.
  Ubicación: Parameters.
  Tipo de dato: String.

- teller_id:

  Descripción: Identificador único del usuario con sesión activa en la ventanilla.
  Requerido: Sí.
  Ubicación: Parameters.
  Tipo de dato: String.

- queuetype_id:

  Descripción: Identificador único del tipo de ticket.

Requerido: Sí.

Ubicación: Parameters.

Tipo de dato: String.

Ejemplo de petición:

POST /v1/tickets/internal

{
"counter_id": "v045a",
"teller_id": "ui405",
"queuetype_id": "q5s53"
}

Respuestas:

- **Código 200:** Ticket derivado con éxito. Devuelve el ticket interno creado.

Ejemplo de respuesta:

{
"id": "89m32b0",
"number": 2,
"status": "WAITING",
"internal": 1,
"temporal": 0,
"derived": 0,
.
.
.
"created_at": "2017-02-20T10:00:00.000Z"
}

- **Código 404:** El recurso solicitado no encontrado, devuelve un mensaje de error simple.

- **Código 400:**

  Uno o más parámetros no son válidos, devuelve una descripción de validación fallida

- **Código 500:** Se ha producido un error inesperado, devuelve un mensaje de error simple.

3.5.4. Crear método en la API REST para registrar tipificaciones

Método: POST

Endpoint: /v1/tickets/{ticket}/operation

Descripción: Realiza el registro de operaciones realizadas en la atención del ticket.

Parámetros de la petición:

- ticket_id:

Descripción:	Identificador único, autogenerado y encriptado del ticket.
Requerido:	Sí.
Ubicación:	Path.
Tipo de dato:	String.

- operation_id:

Descripción:	Único identificador de las operaciones del realizadas por el CRM.
Requerido:	Sí.
Ubicación:	Parameters.
Tipo de dato:	String.

Ejemplo de petición:

```
POST /v1/tickets/89m32b0/operation HTTP/1.1
{
        "operation_id": "s84a5"
}
```

Respuestas:

- **Código 201:** Se creó la operación para el ticket de manera correcta con éxito. Devuelve el ticket interno creado.

  Ejemplo de respuesta:

```
{
        "id": "89m32b0",
        "number": 2,
        "created_at": "2018-02-24T12:30:00.000Z",
        "operationtype": [{
                "id": "o9",
                "name": "Pago de servicios",
                "level": "2"
        }]
}
```

- **Código 404:** El recurso solicitado no encontrado, devuelve un mensaje de error simple.

- **Código 400:**

  Uno o más parámetros no son válidos, devuelve una descripción de validación fallida

- **Código 500:** Se ha producido un error inesperado, devuelve un mensaje de error simple.

En la sección ANEXOS, se detalla la documentación completa de los métodos de la API.

3.2.3.4. CONSTRUCCIÓN.

La etapa de construcción consistió en la codificación de las historias de usuario, para ello se dividió la construcción en 6 sprints de 2 semanas cada uno.

A continuación se detallan los sprints realizados en el proyecto.

Tabla 11. Sprints ejecutados durante el desarrollo del proyecto.

(Confluence - MASIS-BMAT5-Proyecto 17671, 2018)

Sprint 1	H0101	Definición de las funcionalidades del sistema de gestión de colas a implementar en la API
	H0102	Implementar arquitectura inicial de la API
Sprint 2	H0201	Implementar arquitectura de autenticación JWT
	H0202	Envío de información del ticket desde ventanilla a CRM/simulador
Sprint 3	H0301	Crear método para registrar tipificaciones
	H0302	Creación de servicio de encuesta para ser invocado por sistema externo
Sprint 4	H0401	Crear método para listar los sectores de una agencia
	H0402	Crear método para listar los tipos de ventanilla
	H0403	Crear método de api para listar los tipos de ticket que pueden ser generados por un tipo de ventanilla
	H0404	Crear método de api para obtener datos de la ventanilla en base al código del operador
Sprint 5	H0501	Crear método de api para finalizar ticket
	H0502	Crear método de api para derivar ticket.

	H0503	Crear método de api para crear ticket interno.
Sprint 6	H0601	Actualizar el estado de la ventanilla al realizar la derivación del ticket
	H0602	Actualizar el estado de la ventanilla al finalizar la atención del ticket
	H0603	Actualización estado de ventanilla al iniciar atención

3.2.3.5. CERTIFICACIÓN.

Para realizar la evaluación de la solución se utilizó como referencia el informe de validación de casos de prueba del Área de Certificación de HIPER, esta documentación detalla los 3 ciclos de prueba realizados; en los cuales se validó los casos de pruebas asociados a los componentes del sistema de gestión de colas:

- Ventanilla Web
- API
- Servicio de Botonera
- Aplicación de encuesta BMATIC.

3.2.3.5.1. PRIMER CICLO DE PRUEBAS DEL PRODUCTO

En las pruebas del primer incremento de producto, el equipo de Calidad de Software procedió a realizar las pruebas correspondientes a los Sprints 1 y 2, en esta primera certificación no se evidenciaron incidencias.

A continuación, se describe el resultado de la certificación, donde se detalla el total de casos de prueba (CP) que se ejecutaron, en donde:

OK: Los CP que se ejecutaron con éxito.

Error: Los CP que se ejecutaron sin éxito.

Total Ejecutados: La cantidad total de CP que se deben ejecutar para el producto.

Debe Ejecutarse: La cantidad total de CP que se deben ejecutar para esa certificación.

Tabla 12. Resultado de certificación del primer incremento de producto.

(Informe de Validación de Control de Calidad de Hiper, 2018)

Casos de prueba - Primer Release	**Si se realizó**	**No se realizó**		
OK	37	0		
Error	0	0		
No soportados	0	0		
Total Ejecutados	37	0	37	**Total CP del Producto**
Debe ejecutarse	0	0	37	**Total CP para esta certificación**

El reporte final de incidencias del primer incremento de producto reportado por el Equipo de Calidad de Hiper fue el siguiente:

Se presenta el cuadro de Incidencias final terminada certificación donde, la fila de Prioridad se refieren a la importancia que tiene la incidencia de ser resuelta a la brevedad, por ello se define la prioridad del 1 al 5, donde la Prioridad 1 es la cantidad de incidencias que deben ser resueltas cuanto antes por ser de mayor impacto al producto y la Prioridad 5 son la de menor impacto.

El término de este reléase se considera exitoso porque la intersección de la fila total con la columna Total es igual a cero.

Tabla 13. Cuadro resumen del primer ciclo de certificación.

(Informe de Validación de Control de Calidad de Hiper, 2018)

	En Curso	**Nueva/Pendiente**	**Total**
TOTAL	0	0	0
Prioridad 1	0	0	0
Prioridad 2	0	0	0
Prioridad 3	0	0	0
Prioridad 4	0	0	0
Prioridad 5	0	0	0

3.2.3.5.2. SEGUNDO CICLO DE PRUEBAS DEL PRODUCTO

En las pruebas del primer incremento de producto, el equipo de Calidad de Software procedió a realizar las pruebas correspondientes a los Sprints 3 y 4, en esta segunda certificación se evidenciaron algunas incidencias.

A continuación, se describe el resultado de la certificación, donde se detalla el total de casos de prueba (CP) que se ejecutaron, en donde:

OK: Los CP que se ejecutaron con éxito.

Error: Los CP que se ejecutaron sin éxito.

Total Ejecutados: La cantidad total de CP que se deben ejecutar para el producto.

Debe Ejecutarse: La cantidad total de CP que se deben ejecutar para esa certificación.

Tabla 14. Resultado de Certificación del segundo incremento de producto.

(Informe de Validación de Control de Calidad de Hiper, 2018)

Casos de prueba - Segundo Release	**Si se realizó**	**No se realizó**		
OK	50	0		
Error	2	0		
No soportados	0	0		
Total Ejecutados	89	0	89	**Total CP del Producto**
Debe ejecutarse	52	0	37	**Total CP para esta certificación**

El reporte final de incidencias del primer incremento de producto reportado por el Equipo de Calidad de Hiper fue el siguiente:

Se presenta el cuadro de Incidencias final terminada certificación donde, la fila de Prioridad se refieren a la importancia que tiene la incidencia de ser resuelta a la brevedad, por ello se define la prioridad del 1 al 5, donde la Prioridad 1 es la cantidad de incidencias que deben ser resueltas cuanto antes por ser de mayor impacto al producto y la Prioridad 5 son la de menor impacto.

El término de este reléase no se considera exitoso debido a que se encontraron 2 incidencias de prioridad 3.

Tabla 15. Cuadro resumen del segundo ciclo de certificación.

(Informe de Validación de Control de Calidad de Hiper, 2018)

	En Curso	**Nueva/Pendiente**	**Total**
TOTAL	0	0	0
Prioridad 1	0	0	0
Prioridad 2	0	0	2
Prioridad 3	0	0	0
Prioridad 4	0	0	0
Prioridad 5	0	0	0

3.2.3.5.3. TERCER CICLO DE PRUEBAS DEL PRODUCTO

En las pruebas del primer incremento de producto, el equipo de Calidad de Software procedió a realizar las pruebas correspondientes a los Sprints 5 y 6, en esta última certificación no se evidenciaron incidencias.

A continuación, se describe el resultado de la certificación, donde se detalla el total de casos de prueba (CP) que se ejecutaron, en donde:

OK: Los CP que se ejecutaron con éxito.

Error: Los CP que se ejecutaron sin éxito.

Total Ejecutados: La cantidad total de CP que se deben ejecutar para el producto.

Debe Ejecutarse: La cantidad total de CP que se deben ejecutar para esa certificación.

Tabla 16. Resultado de certificación del tercer incremento de producto

(Informe de Validación de Control de Calidad de Hiper, 2018)

Casos de prueba - Tercer Release	**Si se realizó**	**No se realizó**		
OK	50	0		
Error	0	0		
No soportados	0	0		
Total Ejecutados	139	0	139	**Total CP del Producto**
Debe ejecutarse	50	0	0	**Total CP para esta certificación**

El reporte final de incidencias del tercer incremento de producto reportado por el Equipo de Calidad de Hiper fue el siguiente:

Se presenta el cuadro de incidencias final terminada certificación donde, la fila de Prioridad se refieren a la importancia que tiene la incidencia de ser resuelta a la brevedad, por ello se define la prioridad del 1 al 5, donde la Prioridad 1 es la cantidad de incidencias que deben ser resueltas cuanto antes por ser de mayor impacto al producto y la Prioridad 5 son la de menor impacto.

El término de este reléase se considera exitoso porque la intersección de la fila total con la columna Total es igual a cero.

Tabla 17. Cuadro resumen del tercer ciclo de certificación.

(Informe de Validación de Control de Calidad de Hiper, 2018)

	En Curso	**Nueva/Pendiente**	**Total**
TOTAL	0	0	0
Prioridad 1	0	0	0
Prioridad 2	0	0	0
Prioridad 3	0	0	0
Prioridad 4	0	0	0
Prioridad 5	0	0	0

3.2.5. FUNDAMENTOS UTILIZADOS

3.2.5.1. Representational State Transfer (REST).

Es un estilo arquitectónico para sistemas de hipermedia distribuidos que deriva de otros estilos arquitectónicos basados en una red y combina restricciones adicionales que definen a una interfaz de conexión uniforme.

Los servicios web REST permiten que los sistemas solicitantes accedan y manipulen representaciones textuales de los recursos web empleando un conjunto uniforme y predefinido de operaciones que no tienen estado.

Las aplicaciones cliente de un servicio web REST realizan solicitudes a través de la URI de un recurso con el objetivo de realizar operaciones HTTP (GET, POST, PUT, DELETE, etc) y obtener una respuesta en formato XML, HTML, JSON o algún otro formato definido.

En el año 2000, después haberse evitado la crisis de escalabilidad de la Web, Fielding nombró y describió este estilo arquitectónico web en su tesis doctoral "Representational State Transfer", REST se compone de las siguientes restricciones descritas a continuación. (Massé, 2012)

CLIENT SERVER:

La delegación de responsabilidades es el tema central de las restricciones de cliente-servidor de la Web. La web es un sistema netamente cliente –servidor, en el que cada uno tiene distinto rol que jugar y son implementados independientemente.

UNIFORM INTERFASE:

Las interacciones entre los componentes de la Web, es decir, sus clientes, servidores e intermediarios basados en la red, dependen de la uniformidad de sus interfaces. Si alguno de los componentes se desvía de los estándares establecidos, el sistema de comunicación de la web se descompone.

Existen cuatro restricciones dentro de esta categoría:

- Identificación de recursos: Cada concepto basado en la web es un recurso que puede ser identificado por un valor único.
- Modificación de recursos mediante representaciones: Las representaciones de los recursos son manipuladas por los clientes y cada recurso puede ser representado de distintas formas en distintos clientes.
- Mensajes autos descriptivos: El estado que se desea de un recurso puede estar descrito en el mensaje de solicitud del cliente.
- Hipermedia como motor de estados de la aplicación: La representación de los estados de cada recurso deben de englobar los enlaces a los recursos relacionados.

LAYERED SYSTEM:

Permite contar con capas jerárquicas y cada una de ellas tiene una obligación específica.

CACHÉ:

Esta restricción permite etiquetar como cacheable la respuesta de la solicitud, permitiendo al cliente reutilizar esta respuesta en subsecuentes interacciones disminuyendo la latencia entre solicitudes.

STATELESS

Los servidores web no están obligados a memorizar el estado de sus aplicaciones cliente, cada cliente es responsable de administrar la complejidad de comunicar su estado e incluir todo el contexto que considere relevante en cada interacción con el servidor web.

CODE-ON-DEMAND

Es la única restricción del estilo arquitectónico de la Web que se considera opcional que permite a los servidores web transferir temporalmente programas ejecutables, como secuencias de comandos o complementos a los clientes.

3.2.5.2. API.

"Application Programming Interface, es un conjunto de rutinas, protocolos y herramientas que permiten construir aplicaciones de software. Una API especifica cómo los deben de interactuar los componentes de software y ofrece los bloques de construcción para desarrollar un programa informático". (Dumont, 2016, pág. 17)

3.2.5.3. API REST.

Los servicios web son servidores web, especialmente diseñados para soportar las necesidades de un sitio web o cualquier otra aplicación. Estos servicios utilizan interfaces de programación de aplicaciones, API por sus siglas en inglés, para comunicarse con los servicios web. Una API expone un conjunto de datos y funciones las cuales facilitan las

interacciones entre los programas de computadora y permiten el intercambio de información, respondiendo directamente a las solicitudes de los clientes. (Massé, 2012)

.

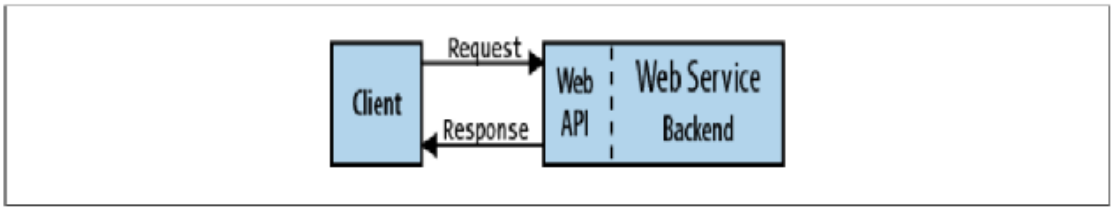


Figura 10: Web API.(Rest API Desing Rulebook, 2012)

3.2.5.4. CRM.

CRM es un sistema integrado empleado con la finalidad de realizar la planificación, programación y verificación de las funciones de preventa y postventa en la organización. CRM engloba en su totalidad los temas de trato con prospectos y clientes, incluye también el centro de contacto, mercadotecnia, fuerza de ventas, asistencia técnica y servicio fuera de la oficina central de una organización.

En general, CRM es una estrategia comercial que tiene como objetivo obtener nuevos clientes y mantener a los clientes actuales para aumentar las ventajas competitivas. CRM ayuda a las empresas a mantener relaciones con clientes que están directamente conectados con compensaciones competitivas. (Seyed Hossein, Harihodin Bin, Rasimah Che, & Mohsen Malekalketab, 2016).

El objetivo principal de CRM es mejorar el crecimiento y la rentabilidad a largo plazo a través de una mejor comprensión del comportamiento del cliente.

3.2.5.5. Servicios Web.

Son aplicativos independientes y modulares que permiten ser descritos, publicados, localizados e invocados mediante una red.

“Los servicios web se pueden definir como el mecanismo o el medio de comunicación a través del cual dos aplicaciones intercambiarán los datos independientemente de su arquitectura de subrayado y la tecnología.” (Singh, Kumar Sahu, & Singh, 2018, pág. 451)

Los servicios web cuentan con interfaces que esconden los detalles de implementación con la finalidad de ser ejecutados de forma independiente a las plataformas de software o hardware en los que se implementan, esto sin depender del lenguaje de programación en el que fue desarrollado. Este escenario alienta a los aplicativos basados en servicios web a ser implementaciones de tecnología cruzada, orientados hacia componentes y enlazados libremente.

El interés en los servicios web se incrementa rápidamente desde el inicio de su uso. Para intercambiar información entre los la aplicación de manera estándar es el objetivo principal de los servicios web. Esta comunicación entre las aplicaciones se basa en el principio SOAP y REST. (Mumbaikar & Puja, 2013)

Los servicios RESTful y basados en SOAP son los dos principales servicios de aprovisionamiento web en la actualidad. Los servicios RESTful se están volviendo muy populares con el paso del tiempo y están reemplazando los servicios web basados en SOAP en muchos campos debido a su facilidad de uso, implementación y ligereza. Por lo tanto, surge una cuestión vital de interés con respecto a un mejor rendimiento entre los servicios RESTful y basados en SOAP. (Malik & Do-Hyeun , 2017)

3.2.5.6. Sistema de Gestión de Colas.

Los sistemas de gestión de colas ayudan a organizar las colas de espera proporcionando a los visitantes un método de ordenación, citas programadas. Generalmente proporcionan medios audiovisuales para dar instrucciones a los clientes, indican a los clientes que su turno es el siguiente, y brindan herramientas útiles que les permitan controlar el nivel de servicio y generar estadísticas de rendimiento.

El sistema de gestión de colas es un sistema que ayuda al proveedor de servicios a administrar clientes de manera eficiente. El sistema puede facilitar la gestión del flujo de clientes que es útil para el administrador del proveedor del servicio. (Sheikh, 2016)

3.2.5.7. Bmatic.

Bmatic es una solución integral para la gestión de redes de atención al cliente desarrollada por la empresa HIPER S.A., que además de ser un sistema que gestiona colas permite

mesurar la situación operativa de las agencias, generando valiosa información estadística de la afluencia de cliente, días y horas pico, rendimiento de los operadores, entre otros. BMatic cuenta con reportes estadísticos y tableros de control que optimizan la supervisión y monitoreo en tiempo real del movimiento en las diferentes sucursales con la finalidad de tomar decisiones en tiempo real. (HIPER, 2014).

3.2.5.8. Scrum.

Scrum es un marco de trabajo para el desarrollo y el mantenimiento de productos que consiste equipos Scrum, roles, eventos, artefactos y reglas asociadas.

> Scrum es un marco de trabajo de procesos que ha sido usado para gestionar el trabajo en productos complejos desde principios de los años 90. Scrum no es un proceso, una técnica o método definitivo. En lugar de eso, es un marco de trabajo dentro del cual se pueden emplear varios procesos y técnicas. Scrum muestra la eficacia relativa de las técnicas de gestión de producto y las técnicas de trabajo de modo que podamos mejorar continuamente el producto, el equipo y el entorno de trabajo. El marco de trabajo Scrum consiste en los Equipos Scrum y sus roles, eventos, artefactos y reglas asociadas. Cada componente dentro del marco de trabajo sirve a un propósito específico y es esencial para el éxito de Scrum y para su uso. (Sutherland & Schwaber, 2016, pág. 4)

De acuerdo a Martín Alaimo y Martin Sálias, en su libro “Proyectos Ágiles con Scrum”, en Scrum hay un equipo identificado como el Equipo Scrum que se divide en tres partes: el Equipo de Desarrollo, el Scrum Master y el Product Owner. El Scrum Master es el responsable de que el Equipo de Desarrollo Scrum comprenda y utilice Scrum, de liderar de forma servicial al Equipo Scrum y asistir a todas las personas que se relacionan con el Equipo Scrum a comprender cuales de esas relaciones son beneficiosas y cuáles no. Puede ser considerado un coach o facilitador encargado de acompañar al Equipo Scrum y a las personas externas para generar relaciones e interacciones que maximicen el valor entregado por el Equipo Scrum. El Product Owner es quien representa el negocio, stakeholder, cliente y usuarios finales. Tiene la responsabilidad de maximizar el valor del producto y del trabajo del Equipo de desarrollo. (Alaimo & Sálias, 2015)

3.2.5.9. JSON WEB TOKEN (JWT).

El token web JSON (JWT) es un estándar abierto (RFC 7519) que define una forma compacta y autónoma para transmitir de forma segura información mediante un objeto JSON en espacios de entornos restringidos. JWT ha encontrado su camino en los principales marcos de trabajo web, cuyas características clave de su arquitectura son simplicidad, compacidad y usabilidad. (Peyrott & Auth0, 2017).

Estructura:

Están compuestos generalmente por tres secciones: un encabezado (header), un contenido (payload) y una firma (signature).

a) Encabezado: Establece el algoritmo utilizado para crear la firma y muestra de la siguiente forma:

```
header = '{"alg":"HS256","typ":"JWT"}'
```

Donde HS256 indica que este token está firmado utilizando el algoritmo de encriptación HMAC-SHA256.

```
payload= '{"loggedInAs":"admin","iat":1422779638}'{"alg":"HS256","typ":"JWT"}'
```

b) Contenido: Provee detalles de los privilegios. también conocido como claims del token.

El estándar menciona abarcar una marca temporal denominado iat con la finalidad de especificar el instante en el que el token ha sido generado.

c) Firma: Es generada codificando el encabezado y el contenido en base64url, concatenándolos con un punto como delimitador:

```
clave          = 'clavesecreta'
tokensinsigno  = encodeBase64Url(encabezado) + '.' + encodeBase64Url(contenido)
firma          = HMAC-SHA256(clave, tokensinsigno)
```

Se detalla una muestra de token cuyo encabezado, contenido y firma son concatenados con puntos:

```
UyJhbGciOiJIUzI1NiIaInR5cCI6IkpXVCJ9.eyJxb2dnZWRJbkFzIjoiYWRtaW4iLCJpYXQiOjE0M
jI3Nzk2Mzh9.gzSbaSYS8EXBxLN_oWnFSRgCzc1JmMjLiuyu5CSpyHI
```

El token es enviado a través de HTML y HTTP, siendo más consistente que estándares apoyados en XML, por ejemplo SAML. Emplea algoritmos de cifrado HMAC con SHA-256 y SHA-256 con firma digital.

Alternativas de encriptación y autenticación son provistas por los estándar algoritmos web JSON (RFC 7518).

Ciclo de vida de un Token JWT

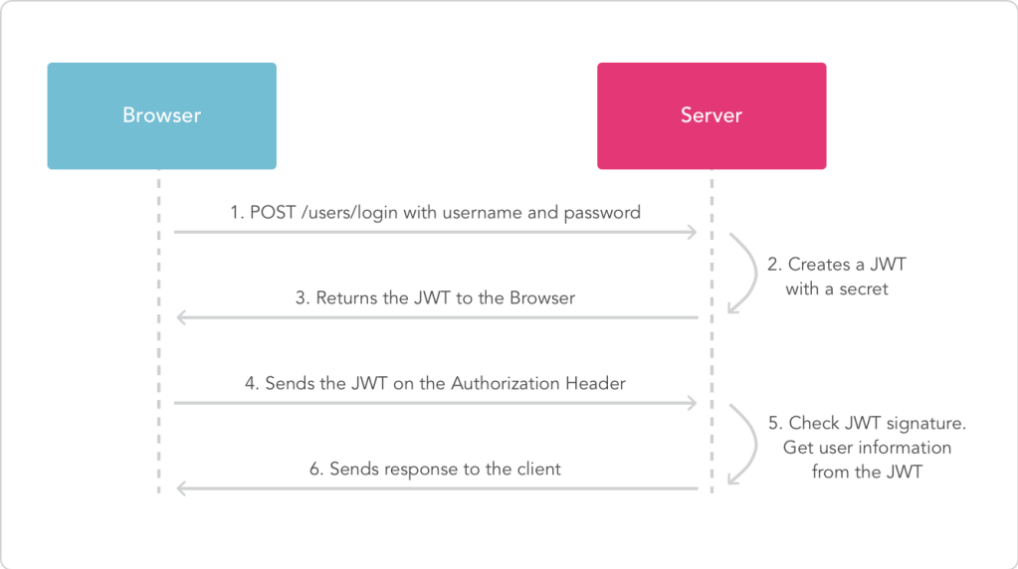


Figura 11: Ciclo de vida de un Token JWT. (The JWT Handbook, 2018)

El flujo completo de JSON Web Token tiene los siguientes pasos:

1. El usuario de algún aplicativo de tipo desktop, móvil o web realiza el login con sus credenciales en el servidor que tiene publicada la API.

2. Mediante la validación del usuario en el servidor y el uso de la llave secreta se crea un nuevo JSON Web Token para retornarlo al usuario.
3. El JSON Web Token firmado almacena los datos del usuario, así como también la caducidad del token y es retornado por el servidor.
4. El navegador guarda el JSON Web Token y lo transmite en cada una de las peticiones a través del "Authorization: Bearer".
5. El servidor realiza la validación de la firma y vencimiento del token, verifica también si el usuario cuenta con acceso al recurso a través de la información contenida en el payload.
6. Al finalizar la confirmación del token y validar que los permisos del usuario son válidos el servidor retorna la petición

3.2.5.10. POSTMAN.

Postman es el único entorno de desarrollo de API completo utilizado por 5 millones de desarrolladores**.** Postman fue creado inicialmente como extensión para el aplicativo Google Chrome; sin embargo, en la actualidad cuenta con aplicativos para distintas plataformas como MAC y Windows y Linux.

Postman está compuesto por diferentes herramientas y utilidades gratuitas que posibilitan realizar diversas actividades en el mundo API REST : elaboración de una biblioteca de peticiones de APIs para miembros internos o terceros, creación de escenarios de prueba con la finalidad de evaluar las funcionalidades de las APIs, hace posible también la creación de diversos entornos de trabajo (con variables locales y globales), todo ello con la posibilidad de ser distribuido entre los miembros del equipo de forma gratuita (exportación información a través de una URL de tipo JSON). (Postdot Technologies, 2018)

Postman cuenta también con una versión de pago que facilita un ambiente cloud colaborativo en el que los equipos pueden elaborar en conjunto colecciones para APIs de forma sincronizada en la nube.

3.2.5.11. SWAGGER.

Es un software de código abierto que ayuda a los desarrolladores a diseñar, construir, documentar y consumir servicios web RESTful.

Swagger facilita la documentación manual y documentación automática de las APIs por medio de anotaciones en el código fuente, así como también la generación automática de bibliotecas de clientes y generación de casos de pruebas para pruebas automatizadas. (SmartBear Software, 2016)

3.2.6. IMPLEMENTACIÓN DE LAS ÁREAS, PROCESOS, SISTEMAS Y BUENAS PRÁCTICAS

3.2.6.1. METODOLOGÍA PARA EL DESARROLLO DEL PRODUCTO.

La empresa Híper tiene como política el uso de una metodología para el desarrollo de software que tiene definido los siguientes procesos:

1. Inicio

2. Planificación

3. Análisis y Diseño

4. Construcción

5. Certificación

6. Despliegue

7. Cierre

7. Post Venta

El proceso de la metodología del desarrollo referido en el informe es la Construcción, la cual engloba a todas las actividades correspondientes a este proceso, se realizaron también las siguientes actividades adicionales.

- Participar en las reuniones correspondientes al levantamiento.

- Participar de forma activa en la reunión de estimación de horas para la codificación los requerimientos.
- Preparar los ambientes correspondientes a las pruebas de integración.
- Realizar pruebas de integración de los requerimientos.
- Realizar pruebas de integración del CRM y sistema de gestión de colas.
- Capacitación a los miembros del equipo de desarrollo, equipo de operaciones y cliente sobre las nuevas funcionalidades implementadas.
- Brindar soporte al Área de Control de Calidad en las Configuración del producto y explicaciones correspondientes a las funcionalidades de los requerimientos.

3.2.6.2. BUENAS PRÁCTICAS APLICADAS AL DESARROLLO DEL SOFTWARE.

Durante el desarrollo de software se utilizó las buenas prácticas SOLID, así como también el uso de patrones de diseño de software.

Los principios SOLID no son ni leyes ni reglas; son principios destinados a ayudarnos a escribir código.

El termino S.O.L.I.D. es un acrónimo que fue enunciados por Robert C. Martin alrededor del año 2000, el cual representa cinco principios básicos del diseño orientado a objetos también denominado ODD. El uso estos principios no siguen un orden prioritario, pero al combínalos hacen que el código sea limpio, fácil de mantener o extender.

Los principios de SOLID son los siguientes:

S (Single Responsibility Principle o Principio de Responsabilidad Única)

Una clase debe tener solo una responsabilidad. Digamos que nuestra clase es responsable de guardar datos. Eso significa que no debería ser responsable de recuperar datos o cualquier otra tarea.

O (Principio abierto/cerrado u open/close principle)

Una vez que se ha escrito una clase, no debe permitir que nadie haga cambios. Nadie debería poder volver y modificar el código de clase para implementar nuevas funcionalidades.

L (Principio de sustitución de Liskov)

Un objeto dependiente debería poder usar cualquier objeto de tipo objeto principal.

I (Principio de Segregación de interface)

Se recomienda una interfaz más pequeña. Si una interfaz tiene un método, solo habrá un lugar para cambiar si necesitamos cambiar el código. Sin embargo, en el caso de las interfaces con más métodos, podría haber más razones para el cambio.

D (Inversión de dependencias)

Este principio aborda el acoplamiento flojo especifica que con la ayuda de Inyección de dependencias, podemos escribir código que no depende de clases concretas.

Para el desarrollo de los requerimientos se utilizaron los siguientes patrones de diseño:

Patrones Creacionales

Builder: Separa la creación de un objeto complejo de sus representaciones. Centraliza la creación de objetos desde una clase constructora omitiendo la creación de la clase dentro de la misma clase.

Factory Method: Crea un método abstracto y delega su implementación a las subclases devolviendo instancias de una clase en particular mediante un identificador.

Patrones Estructurales:

Singleton: Delimita la creación de una única instancia posible de una clase durante la ejecución de la aplicación.

Patrones de Comportamiento

Command: Encapsulan una acción y los parámetros necesarios para ejecutarse.

Template Method: Describe la estructura de un algoritmo, facilitando a las subclases la libertad para establecer la implementación del comportamiento real.

Observer: Los objetos se suscriben a una serie de eventos y cuando el objeto modifica su estado notifica a los objetos suscritos este cambio.

Estos patrones de diseños hicieron el desarrollo nuestro código más sencillo. La mayor complejidad radica en saber cuándo utilizar cada una de ellas.

3.3 EVALUACIÓN

3.3.1 EVALUACIÓN ECONÓMICA

La evaluación económica del proyecto incluye los siguientes ítems

- Costos del Proyecto
- Personal Requerido por Mes
- Flujo de Pagos
- Flujo de Caja
- VAN, TIR

Costos Asociados al Proyecto (Duración de 4 meses)

Personal	**Monto (S/.)**
Maquisistema	4,000.00
Personal Externo	12,125.00

	Monto (S/.)
Software Licenciado	
Windows Server 2016 (1)	1,600.00
SQL SERVER 2014 (1)	700.00

Otros Costos	**Monto (S/.)**
Teléfono / Mes	120.00
Luz / Mes	700.00
Agua / Mes	120.00
Mantenimiento y Limpieza/Mes	80.00
Seguridad / Mes	50.00

Flujo de Pagos

Tabla 18. Flujo de Pagos

	Mes 1 (S/.)	Mes 2 (S/.)	Mes 3 (S/.)	Mes 4 (S/.)
Personal Maquisistema	4,000,00	4,000,00	4,000,00	4,000,00
Personal Externo	11,000,00	11,000,00	11,000,00	11,000,00
Windows Server 2016(1)	1,600.00	-	-	-
SQL Server 2014 (1)	700.00	-	-	-
Teléfono/Mes	120.00	120.00	120.00	120.00
Luz/Mes	700.00	700.00	700.00	700.00
Agua/Mes	120.00	120.00	120.00	120.00
Limpieza y Mantenimiento/Mes	80.00	80.00	80.00	80.00
Seguridad/Mes	50.00	50.00	50.00	50.00

Flujo de Caja

Tabla 19. Flujo de Egresos

	Mes 1 (S/.)	Mes 2 (S/.)	Mes 3 (S/.)	Mes 4 (S/.)
Ingresos	-	-	-	-
Egresos	- 18,370.00	- 16,070.00	-16,070.00	- 16,070.00
Flujo de Caja	- 18,370.00	- 16,070.00	-16,070.00	- 16,070.00

Ahorro por recurso humano operativo (productividad), licencias de software y uso de los equipos del proveedor de software para el desarrollo.

Tabla 20. Flujo de Caja de Ingresos

	Mes 1 (S/.)	Mes 2 (S/.)	Mes 3 (S/.)	Mes 4 (S/.)
Ingresos	25,800.00	25,800.00	25,800.00	25,800.00
Egresos	-	-	-	-
Flujo de Caja	25,800.00	25,800.00	25,800.00	25,800.00

Existe un ahorro mensual de MAQUISISTEMA de S/.25,800.00 desde que inicia hasta que termina el proyecto.

Cálculo del TIR

La empresa de fondos colectivos Maquisistema asume una inversión de S/ 66,580.00 para el desarrollo del proyecto, el cual generará un flujo de caja para el primer año de S/ 48,048.00 y para el segundo año de S/ 28,700.00, con lo cual se tiene la siguiente ecuación.

$$0 = -66{,}580 + \frac{48{,}048.00}{(1+TIR)} + \frac{28{,}700.00}{(1+TIR)^2}$$

$$TIR = 11.00\%$$

Cálculo del VAN

La empresa de fondos colectivos Maquisistema asume una tasa de descuento del 10.00% armándose la siguiente ecuación.

$$VAN = -66{,}580 + \frac{41{,}600.00}{(1+TIR)} + \frac{31{,}600.00}{(1+TIR)^2}$$

$$VAN = -66{,}580 + \frac{41{,}600.00}{(1+0.01)} + \frac{31{,}600.00}{(1+0.01)^2}$$

$VAN = S/.\ 3{,}190.90$

3.3.2 INTERPRETACIÓN DEL VAN Y DEL TIR

Como es sabido, para invertir en un proyecto es necesario analizar la rentabilidad y factibilidad del mismo, por ello se usa el VAN (Valor Anual Neto) y el TIR (Tasa Interna de Retorno), en el cálculo vemos que el VAN es superior a cero, por tanto, el proyecto es factible, dicho de otro modo, si el VAN es mayor a cero entonces hay que invertir. En cuanto al cálculo del TIR es de 11.00% siendo superior a la tasa de descuento del 10.00% que Maquisistema asume, por tanto, se garantiza un retorno positivo, dicho de otro modo, si tu tasa de descuento es inferior al TIR entonces hay que invertir.

CAPÍTULO IV: REFLEXIÓN CRÍTICA DE LA EXPERIENCIA

4.1 APORTES EN EL ÁREA DE DESARROLLO Y RESPONSABILIDADES

a) El rol desempeñado por del autor del informe fue de programador para el área de desarrollo de software de la empresa Hiper, participé personalización del Sistema de Gestión de Colas BMATIC y el desarrollo de la API REST para la integración con el CRM de la empresa Maquisistema.
b) Los requerimientos que se me asignaron fueron desarrollados bajo las buenas prácticas de programación, utilizando estándares de código limpio para poder facilitar el mantenimiento de futuras correcciones de incidencias o el desarrollo de nuevas funcionalidades.
c) El equipo estuvo conformado por un gestor de proyecto, un arquitecto de software, un analista funcional y tres programadores, quienes se encargaron de realizar la codificación de los requerimientos considerando impactar lo menos posible en las funcionalidades preexistentes del sistema de gestión de cola .
d) Luego de finalizar la fase de construcción de los requerimientos, el equipo de desarrollo realizó pruebas integrales, al finalizar las pruebas integrales se inició el congelamiento, en este punto del proceso de desarrollo el área de calidad de software realiza la certificación correspondiente, una vez culminada la certificación, el área de calidad reporta las incidencias encontradas, estas incidencias tienen con un código de identificación y se destina la corrección en los siguientes sprints del proyecto.
e) Al finalizar la fase de certificación, se realizó la asignación de incidencias para su respectiva corrección, es aquí donde se evidencia la efectiva comunicación entre el analista funcional, analista de calidad, gestor del proyecto y los programadores, pues se asignó las incidencias de manera rápida, se resolvió de manera satisfactoria las dudas existentes correspondientes a las incidencias encontradas, esto permitió replicar las incidencias de manera satisfactoria para su inmediata corrección.

f) Luego de culminar con la corrección de incidencias, se procedió a realizar el congelamiento respectivo y se coordinó con el equipo del Área de Calidad para dar inicio un nuevo ciclo de certificación.

g) El autor se benefició porque adquirió conocimientos funcionales del manejo colas y adquirió conocimientos técnicos de las tecnologías utilizadas para el desarrollo de las funcionalidades, conoció a profesionales competentes que lo capacitaron durante la etapa de implementación, y desarrolló habilidades blandas para asumir nuevos retos en otros proyectos desarrollados por Hiper, con mayor seguridad y confianza.

CAPÍTULO V. CONCLUSIONES Y RECOMENDACIONES

5.1 CONCLUSIONES

a) Se analizó, desarrolló e implementó la API REST de forma satisfactoria, logrando mediante la solución la integración del Sistema de Gestión de Colas al CRM.

b) La integración del CRM y sistema de gestión de colas recuperó significativamente la presencia e imagen en el mercado mejorando la atención y disminuyendo el tiempo de atención del cliente.

c) Se generó un nuevo canal de generación de tickets que permite ser utilizado por futuros desarrollos de aplicaciones móviles o web.

d) Se mantiene la consistencia de información generada por la API REST y Sistema Gestión del Sistema utilizando los mismos registros y campos de base de datos.

e) Se brindó documentación los métodos de la API con especificaciones claras y concisas fáciles de entender para cualquier desarrollador de software.

f) La solución se desplegó exitosamente en el mes de mayo del año 2018 en la sede central de Maquisistema en la ciudad de Lima.

g) La documentación especificada en la etapa de análisis fue deficiente debido a que se desconocía todas las funcionalidades del módulo de atención de tickets; por lo que en el transcurso del desarrollo se tuvieron que realizar modificaciones en la especificaciones y realizar validaciones no contempladas.

5.2 RECOMENDACIONES

a) Culminar el desarrollo de las funcionalidades restantes del módulo de Ventanilla en la API REST y modificar el módulo de ventanilla de manera utilice los nuevos métodos expuestos.

b) Estandarizar y aplicar las buenas prácticas el desarrollo del software y patrones de diseño en todos los productos desarrollados.

c) Continuar con la migración de los aplicativos contenidos en el repositorio local SVN hacia repositorios de GIT para todos los módulos del sistema de gestión de colas, así como también para los otros productos de HIPER; ya que con el desarrollo de este proyecto se evidenció mejoras en la productividad del equipo.

d) Realizar capacitaciones sobre las funcionalidades del sistema de gestión de colas así como también las tecnologías utilizadas para el desarrollo de la API REST de manera que los miembros del área de desarrollo estén prevenidos técnica y profesionalmente, ya que esto evitará pérdidas de tiempo y complicaciones con los clientes.

5.3 FUENTES DE INFORMACIÓN

Postdot Technologies. (2018). *POSTMAN Learing Center*. Obtenido de https://learning.getpostman.com/docs/

Sheikh, A. (2016). Automated Queue Management System. *Global Journal of Management and Business Research*.

SmartBear Software. (2016). *Swagger SmartBear* . Obtenido de https://swagger.io/docs/specification/about/.

Alaimo, M., & Sálias, M. (2015). *Proyectos Ágiles con Scrum.* Buenos Aires: Kleer.

Dumont, P. F. (2016). Sistema integrado con servicios web que brinde soporte a los procesos de gestión de proyectos de la empresa desarrolla de software TAU. Lima, Perú.

HIPER. (2014). Obtenido de http://www.hiper.com.pe

HIPER. (2018). Obtenido de https://www.linkedin.com/company/hiper-s-a-

Malik, S., & Do-Hyeun , K. (2017). A comparison of RESTful vs. SOAP web services in actuator networks. Jeju-si, South Korea .

Massé, M. (2012). *REST API Design Rulebook.* California: O'Reilly Media.

Mumbaikar, S., & Puja, P. (2013). Web Services Based On SOAP and REST Principles. *International Journal of Scientific and Research Publication*.

Peyrott, S. E., & Auth0. (2017). *JWT HandBook.* Washinton. Obtenido de https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-Spanish-European.pdf

Satpath, T. (2017). *A Guide to the Scrum Body Of Knowledge (SBOK™Guide).* Arizona: SCRUMstudy.

Seyed Hossein, N., Harihodin Bin, S., Rasimah Che, M. Y., & Mohsen Malekalketab, K. (12 de 2016). lectronic Customer Relationship Management, Customer Satisfaction, and Customer Loyalty: A Comprehensive Review Study. *International journal of management and economics invention*.

Singh, J., Kumar Sahu, S., & Singh, A. (2018). *Smart Computing and Informatics: Proceedings of the First International Conference on SCI 2016,Volume 2.* Singapur: Springer.

SUNAT. (2018). *SUNAT*. Obtenido de https://e-consultaruc.sunat.gob.pe

Sutherland, J., & Schwaber, K. (2016). *La Guia Definitiva de Scrum: Las Reglas de Juego.* Illinois.

5.4 GLOSARIO

Control De Calidad: Conjunto de técnicas y actividades de carácter operativo utilizadas para verificar los requisitos relativos a la calidad de software.

Confluence: Software de colaboración desarrollado y comercializado por Atlassian, utilizado principalmente en entornos corporativos.

Framework: Marco de Trabajo para desarrollo de una aplicación.

HTTP: Es el protocolo de comunicación que permite las transferencias de información en la World Wide Web.

HTTPS: Es un protocolo de aplicación basado en el protocolo HTTP, destinado a la transferencia segura de datos de Hipertexto, es decir, es la versión segura de HTTP.

Java: lenguaje de programación multiplataforma para el desarrollo de aplicaciones web y de escritorio.

OAuth: Es un estándar abierto que permite la autorización de manera confiable de una API de forma sencilla y estándar para aplicaciones multiplataforma.

Restful: Arquitectura de webservice para transmisión a través de http.

SAML: Es un estándar abierto que define un esquema XML para el intercambio de datos de autenticación y autorización , es también denominado El Lenguaje de Marcado para Confirmaciones de Seguridad,

Servidor: equipo computacional dedicado a procesar información durante períodos largos de tiempo, a menudo las 24 horas del día los 365 días del año.

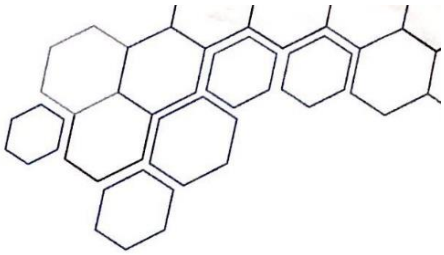
SOAP: Es un protocolo estándar utilizados en los servicios Web, define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML.

Token: Es una cadena de caracteres que tiene un significado coherente en cierto lenguaje de programación.

XML: Es un meta-lenguaje que permite definir lenguajes de marcas desarrollado por el World Wide Web Consortium (W3C) utilizado para almacenar datos en forma legible y proviene en inglés de Extensible Markup Language o "Lenguaje de Marcado Extensible”.

ANEXOS

ANEXO 1: CONSTANCIA DE PARTICIPACIÓN EN EL PROYECTO

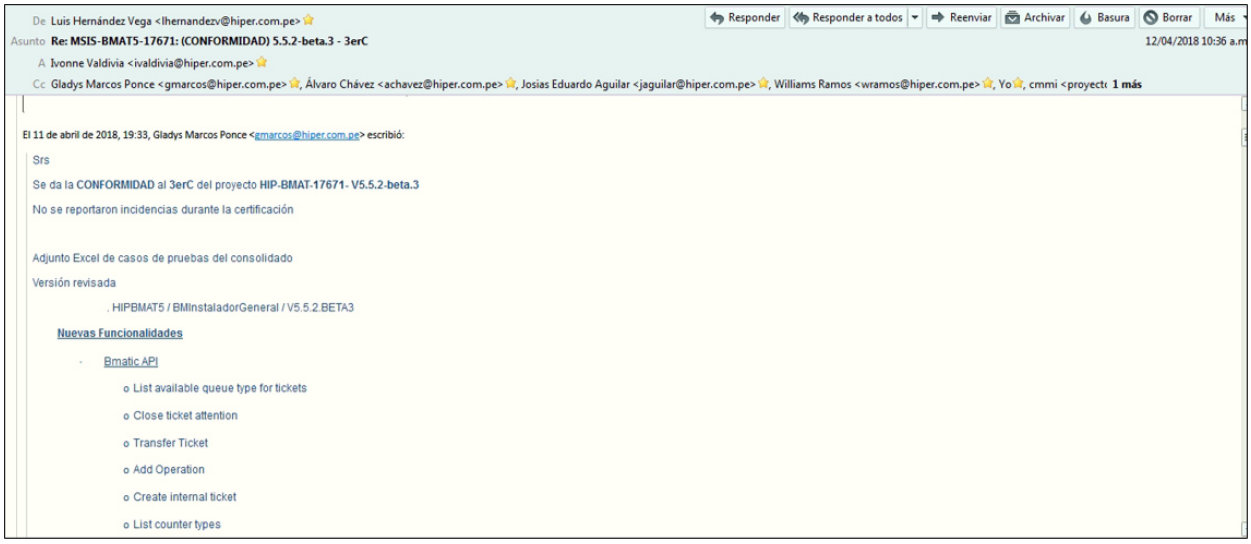


Constancia de Participación

Por medio del presente documento, se da fe que el señor Kevin Arthur Meneses Rivera, participó en el proyecto 17671 de Híper S.A. para la empresa de Fondos Colectivos Maquisistema de Perú con el rol de programador.

GABRIEL VEGA CALLE
Gerente de Desarrollo
Hiper S.A.

ANEXO 2: CONFORMIDAD DE CERTIFICACIÓN DEL ÁREA DE CALIDAD



De Luis Hernández Vega <lhernandezv@hiper.com.pe>
Responder | Responder a todos | Reenviar | Archivar | Basura | Borrar | Más
Asunto **Re: MSIS-BMAT5-17671: (CONFORMIDAD) 5.5.2-beta.3 - 3erC** 12/04/2018 10:36 a.m
A Ivonne Valdivia <ivaldivia@hiper.com.pe>
Cc Gladys Marcos Ponce <gmarcos@hiper.com.pe>, Álvaro Chávez <achavez@hiper.com.pe>, Josias Eduardo Aguilar <jaguilar@hiper.com.pe>, Williams Ramos <wramos@hiper.com.pe>, Yo, cmmi <proyect> **1 más**

El 11 de abril de 2018, 19:33, Gladys Marcos Ponce <gmarcos@hiper.com.pe> escribió:

Srs

Se da la **CONFORMIDAD** al **3erC** del proyecto **HIP-BMAT-17671- V5.5.2-beta.3**

No se reportaron incidencias durante la certificación

Adjunto Excel de casos de pruebas del consolidado

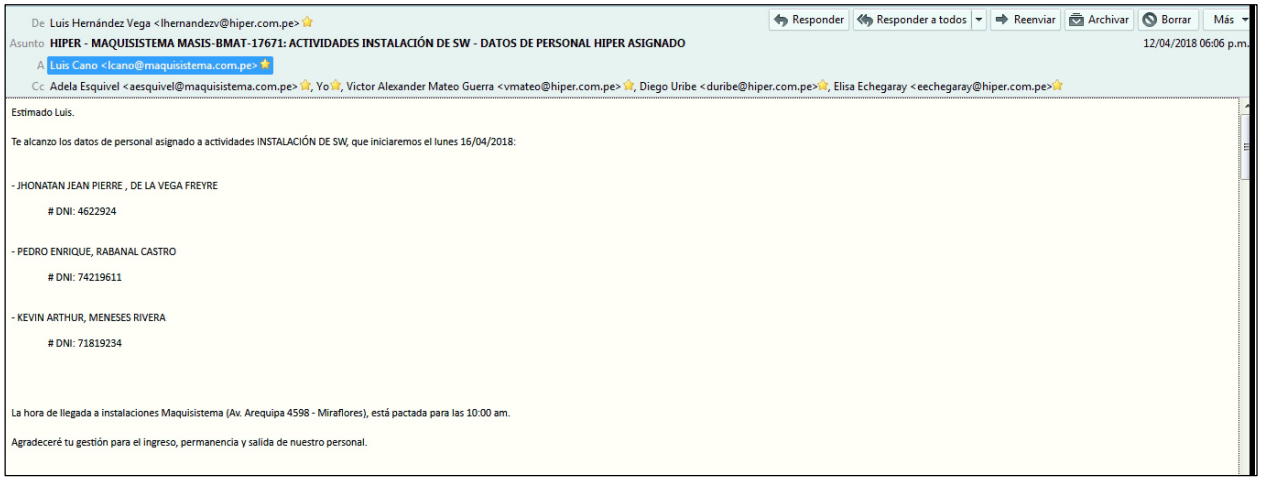
Versión revisada

. HIPBMAT5 / BMInstaladorGeneral / V5.5.2.BETA3

Nuevas Funcionalidades

- Bmatic API
  - o List available queue type for tickets
  - o Close ticket attention
  - o Transfer Ticket
  - o Add Operation
  - o Create internal ticket
  - o List counter types

ANEXO 3: COORDINACIONES PASE A PRODUCCIÓN.



De Luis Hernández Vega <lhernandezv@hiper.com.pe>
Responder | Responder a todos | Reenviar | Archivar | Borrar | Más
Asunto **HIPER - MAQUISISTEMA MASIS-BMAT-17671: ACTIVIDADES INSTALACIÓN DE SW - DATOS DE PERSONAL HIPER ASIGNADO** 12/04/2018 06:06 p.m.
A Luis Cano <lcano@maquisistema.com.pe>
Cc Adela Esquivel <aesquivel@maquisistema.com.pe>, Yo, Victor Alexander Mateo Guerra <vmateo@hiper.com.pe>, Diego Uribe <duribe@hiper.com.pe>, Elisa Echegaray <eechegaray@hiper.com.pe>

Estimado Luis.

Te alcanzo los datos de personal asignado a actividades INSTALACIÓN DE SW, que iniciaremos el lunes 16/04/2018:

- JHONATAN JEAN PIERRE , DE LA VEGA FREYRE
  # DNI: 4622924

- PEDRO ENRIQUE, RABANAL CASTRO
  # DNI: 74219611

- KEVIN ARTHUR, MENESES RIVERA
  # DNI: 71819234

La hora de llegada a instalaciones Maquisistema (Av. Arequipa 4598 - Miraflores), está pactada para las 10:00 am.

Agradeceré tu gestión para el ingreso, permanencia y salida de nuestro personal.

ANEXO 4: GUÍA DE REFERENCIA DE OBJECTO ENVIADO A CRM

Bmatic Objects

Common Object Definitions

Collection of common objects used by BMatic.

Event

Sample object:


```
{
    "id": "UUID",
    "type": "4dg95",
    "published_at": "2018-02-20T10:00:00.000Z",
    "request":{
        "transaction_id": "c9kas95",
        "producer_id": "rp951",
        "event_id": "ev25a5"
    },
    "data":{
        "object":{
            "id": "89m32b0",
            "number": 2,
            "status": "INATTENTION",
            "internal": 1,
            "temporal": 0,
            "derived": 0,
            "label": "CC2",
            "messages": [
                "We have a promotion for you!"
            ],
            "channel": {
                "id": "c10",
                "name": "Mobile"
            },
            "branch": {
                "id": "27b6",
                "name": "Hiper Central",
                "short_name": "Hiper",
                "address": "Calle Beta 181 - 195, Callao",
                "latitude": -12.049919,
```

```
            "longitude": -77.0845193,
            "status": "ACTIVE"
        },
        "sector": {
            "id": "m9",
            "name": "Tower A",
            "short_name": "A",
            "status": "ACTIVE"
        },
        "service": {
            "id": "s8",
            "name": "Plataforma",
            "short_name": "Plataforma"
        },
        "queue_type": {
            "id": "R0009",
            "code": "CC",
            "name": "Comercial"
        },
        "counter": {
            "id": "v10",
            "name": "V10",
            "teller": "pchavez",
            "type": {
                "id": "v2002",
                "name": "Ventanilla"
            }
        },
        "customer": {
            "doc_type": "D",
            "doc_number": "74693766",
            "name": "Juan Perez",
            "phone": "+51987776576",
            "category": {
                "id": "I000004",
                "name": "VIP"
            }
        },
        "created_at": "2017-02-20T10:00:00.000Z",
        "called_at": "2017-02-20T10:05:00.000Z",
        "started_at": "2017-02-20T10:05:18.000Z"
    }
  }
}
```


Model of event object.

FIELDS

id *String*

Unique event identifier (Auto generated by BMatic).

type *String*

Unique type event identifier (Auto generated by BMatic).

published_at *DateTime*

Datetime published of event.

Standard: `ISO 8601`

request *Request*

Request.

data *Data*

Object data.

Request

Sample object:

```
{
    "transaction_id": "c9kas95",
    "producer_id": "rp951",
    "event_id": "ev25a5"
}
```

Model of request.

transaction_id *String*

Transaction unique identifier (Auto generated by BMatic).

producer_id *String*

Producer identifier.

event_id *String*

Event unique identifier.

Data

Sample object:


```
{
    "object":{
            "id": "89m32b0",
            ...
            ...
            ...
            "waiting_time": 0
    }
}
```


Model of data.

FIELDS

object *Ticket*

Model of ticket object.

Ticket

Sample object:


```
{
    "id": "89m32b0",
    "number": 2,
    "status": "INATTENTION",
    "position": 10,
    "low_estimated_time": 10,
    "high_estimated_time": 15,
    "internal": 1,
    "temporal": 0,
    "derived": 0,
    "label": "CC2",
    "messages": [
        "We have a promotion for you!"
    ],
    "channel": {
        "id": "c10",
        "name": "Mobile"
    },
    "branch": {
        "id": "27b6",
        "name": "Hiper Central",
        "short_name": "Hiper",
        "address": "Calle Beta 181 - 195, Callao",
```

```
        "latitude": -12.049919,
        "longitude": -77.0845193,
        "status": "ACTIVE"
    },
     "sector": {
        "id": "m9",
        "name": "Tower A",
        "short_name": "A",
        "status": "ACTIVE"
    },
    "service": {
        "id": "s8",
        "name": "Plataforma",
        "short_name": "Plataforma"
    },
     "queue_type": {
        "id": "R0009",
        "code": "CC",
        "name": "Comercial"
    },
    "counter": {
        "id": "v10",
        "name": "V10",
        "teller": "pchavez",
        "type": {
             "id": "v2002",
             "name": "Ventanilla"
        }
    },
    "customer": {
        "doc_type": "D",
        "doc_number": "74693766",
        "name": "Juan Perez",
        "phone": "+51987776576",
        "category": {
             "id": "I000004",
             "name": "VIP"
        }
    },
    "created_at": "2017-02-20T10:00:00.000Z",
    "called_at": "2017-02-20T10:05:00.000Z",
    "started_at": "2017-02-20T10:05:18.000Z",
    "finished_at": "2017-02-20T10:11:41.000Z",
    "print": [
        {
            "align": "CENTER",
            "font": "A_REGULAR",
            "previous": "Welcome to",
            "content": "Hiper",
            "next": "!"
        }
    ]
}
```


Model of ticket object.

FIELDS

id *String*

Ticket unique identifier (Auto generated by Bmatic). This value is encrypted.

---

number *Integer*

Number in queue assigned to the ticket.

---

status *Enum*

Current status of the ticket.

Possible values:

- `WAITING`: When the ticket is waiting for teller call.
- `CALLED`: When the ticket is called by teller.
- `INATTENTION`: When the ticket is in attetion.
- `ATTENDED`: When the attention of ticket is finished.

---

position *Integer*

Ticket position in the queue.

---

low_estimated_time *Integer*

Lower bound of the estimated waiting time. It's equal to the average waiting time minus the configured desviation.

---

high_estimated_time *Integer*

Upper bound of the estimated waiting time. It's equal to the average waiting time plus the configured desviation.

---

internal *Integer*

Indicates if the ticket is internal or not.

Possible values:

- `0`: When the ticket is not internal.
- `1`: When the ticket is internal.

---

temporal *Integer*

Indicates if the ticket is temporal or not.

Possible values:

- `0`: When the ticket is not temporal.
- `1`: When the ticket is temporal.

---

derived *Integer*

Indicates if the ticket is derived or not.

Possible values:

- `0`: When the ticket is not derived.
- `1`: When the ticket is derived.

---

label *String*

Printable ticket code. If the ticket is derived this value will be the original ticket code.

---

messages *List<String>*

Messages to print.

---

channel *Channel*

Channel where the ticket was created.

---

branch *Branch*

Branch where the ticket is valid.

---

sector *Sector*

Sector where the ticket is valid.

---

service *Service*

Service requested by the customer.

---

queue_type *QueueType*

Assigned queue type.

---

counter *Counter*

Assigned counter.

---

customer *Customer*

Customer

---

created_at *DateTime*

Creation date of ticket.

Standard: `ISO 8601`

---

called_at *DateTime*

Call date of ticket.

Standard: `ISO 8601`

---

started_at *DateTime*

Start date of ticket attention.

Standard: `ISO 8601`

Channel

Sample object:

```
{
    "id": "c10",
    "name": "Mobile",
}
```

Model of channel object.

FIELDS

id *String*

Channel unique identifier (Configured from Bmatic configuration panel).

name *String*

Complete name of the channel.

Maximum length: `20`

Branch

Sample object:


```
{
    "id": "27b6",
    "name": "Hiper Central",
    "short_name": "Hiper",
    "address": "Calle Beta 181 - 195, Callao",
    "latitude": -12.049919,
    "longitude": -77.0845193,
    "status": "ACTIVE",
    "congestion": "MEDIUM",
    "low_estimated_time": 10,
    "high_estimated_time": 14
}
```


Model of branch object.

Caution: The `congestion`, `low_estimated_time` and `high_estimated_time` are calculated using the configuration of the associated branch type. You need configure the next:

- Time range to take for calculate the waiting time.
- Desviation of estimated waiting time.
- Minimal waiting time of medium congestion.
- Minimal waiting time of high congestion.

If the branch is `INACTIVE` these fields will not return.

FIELDS

id *String*

Branch unique identifier (Configured from Bmatic configuration panel).

name *String*

Complete name of the branch.

Maximum length: `50`

---

short_name *String*

Name to display in small devices.

Maximum length: `20`

---

address *String*

Branch address.

Maximum length: `100`

---

latitude *Float*

Latitude component of branch location.

---

longitude *Float*

Longitude component of branch location.

---

status *Enum*

Current status of branch.

Possible values:

- `ACTIVE`: At least one sector is `ACTIVE`
- `INACTIVE`: All sectors are `INACTIVE`

---

congestion *String*

Flag of branch congestion.

Possible values:

- `LOW`: The average waiting time is lower to the minimal waiting time of medium congestion.
- `MEDIUM`: The average waiting time is lower to the minimal waiting time of high congestion but equal or upper to the minimal waiting time of medium congestion.
- `HIGH`: The average waiting time is equal or upper to the minimal waiting time of high congestion.

---

low_estimated_time *Integer*

Lower bound of the estimated waiting time. It's equal to the average waiting time minus the configured desviation.

high_estimated_time *Integer*

Upper bound of the estimated waiting time. It's equal to the average waiting time plus the configured desviation.

Sector

Model of sector object.

Sample object:

```
{
    "id": "m9",
    "name": "Tower A",
    "short_name": "A",
    "status": "OPEN"
}
```

FIELDS

id *String*

Sector unique identifier (Auto generated by Bmatic).

name *String*

Complete name of the sector.

Maximum length: `50`

short_name *String*

Name to display in small devices.

Maximum length: `20`

status *Enum*

Current status of the sector.

Possible values:

- `ACTIVE`: At least one counter in the sector is `ACTIVE` or `SUSPENDED`.
- `INACTIVE`: All counters in the sector are `INACTIVE`.

Service

Sample object:

```
{
    "id": "s8",
    "name": "Plataforma",
    "short_name": "Plataforma"
}
```

Model of service object.

FIELDS

id *String*

Service unique identifier (Auto generated by Bmatic).

name *String*

Complete name of the service.

Maximum length: `50`

short_name *String*

Name to display in small devices.

Maximum length: `20`

Queue Type

Sample object:


```
{
    "id": "R0009",
    "code": "CC",
    "name": "Comercial"
}
```


Model of queue type object.

FIELDS

id *String*

Unique identifier of the queue type(Auto generated by Bmatic).

code *String*

Code of the queue type.

Maximum length: `5`

name *String*

Complete name of the queue type.

Maximum length: `50`

Counter

Sample object:


```
{
    "id": "v10",
    "name": "V10",
    "teller": "achavez",
    "type": {
        "id": "v2002",
        "name": "Ventanilla"
    }
}
```


Model of counter object.

FIELDS

id *String*

Counter unique identifier (Configured from Bmatic configuration panel).

name *String*

Name of the counter.

Maximum length: `20`

teller *String*

Account name of teller in counter.

Maximum length: `30`

type *CounterType*

Associated counter type.

Customer

Sample object:

```
{
    "doc_type": "D",
    "doc_number": "74693766",
    "name": "Juan Perez",
    "phone": "+51978512025",
    "category": {
        "id": "I000004",
        "name": "VIP"
    }
}
```

Model of customer object.

FIELDS

doc_type *String*

Customer document type code.

Maximum length: `5`

doc_number *String*

Customer document number.

Maximum length: `20`

name *String*

Customer name.

phone *String*

Customer phone.

category *Category*

Customer category.

Category

Sample object:

```
{
    "id": "I000004",
    "name": "VIP",
}
```

Model of customer category.

FIELDS

id *String*

Category unique identifier.

name *String*

Complete name of the category.

ANEXO 5: GUÍA DE REFERENCIA DE SERVICIO DE ENCUESTA

BMatic Survey

Introduction

Improve your customers' experience!

Schemes: `http`

Base path: `/survey`

Survey Service

You should consider using the recommended parameters for better performance.

Show poll

Sample request:

```
POST /survey/poll HTTP/1.1
Content-Type: application/json

{
    "message": "Rate my attention please",
    "timeout": 5
}
```

Sample response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "score": 4
}
```

Returns the survey rating.

> **Caution:** You must consider the next:
>
> - When any problem that prevents the qualification occurs, the service returns 500.

ENDPOINT

`POST /survey/poll`

REQUEST

message *String* REQUIRED

The question that will appear in the poll.

timeout *Integer* REQUIRED

Maximum waiting time of survey. This value is in seconds.

Maximum length: 2

RESPONSES

200 *Ok* *Qualification*

Survey qualification.

400 *Bad Request* *Error*

One or more parameters are not valid, returns a description of validation failed.

404 *Not Found* *Error*

The resource requested not found, returns a simple error message.

Common Object Definitions

Collection of common objects used by the survey service.

Qualification

Model of qualification object.

Sample object:


```
{
   "score": 2
}
```


FIELDS

score *Integer*

Qualification value provided by the customer.

Possible values:

- 2: Bad rating
- 3: Regular rating
- 4: Good rating

Error

Sample object:


```
{
   "type": "SurveyDeviceCustomException",
   "message": "An unexpected error has occurred on survey device",
}
```


Model of error object.

FIELDS

type *String* Name of type error.

message *String* Simple description of the error.

ANEXO 6: GUÍA DE REFERENCIA DE API REST BMATIC

Introduction

Welcome to the Bmatic API! You can use our API to integrate your application with Bmatic System. Now, trought our endpoints you can access BMATIC branches services from any application.

You can download our specification and generate your own client in your prefered language trought the swagger code generator and speed up your integration process.

Improve your customers' experience!

Schemes: `http, https`

Host: `api.bmatic.com`

Base path: `/v1`

Tickets API

Your customers don't need anymore to be at branch for create tickets. You can use the endpoints of this section to allow them create ticket from anywhere easily.

List available services for tickets

Sample request:

```
GET /v1/tickets/services?channel_id=c10 HTTP/1.1
```

Sample response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": "s8",
        "name": "Plataforma",
        "short_name": "Plataforma"
    }
]
```

Returns available services for the specified channel.

Caution: You must consider the following:

- An empty list is returned when not available services exist from the specified channel.

ENDPOINT

`GET /v1/tickets/services`

QUERY PARAMS

channel_id *String* REQUIRED

Unique identifier of the channel used. This value must be set in the client application.

RESPONSES

200 *Ok* *List<Service>*

A list of services.

400 *Bad Request* *Error*

One or more parameters are not valid, returns a description of validation failed.

500 *Internal Server Error* *Error*

An unexpected error has occurred, returns a simple error message.

List available queue type for tickets

Sample request:

```
GET /v1/tickets/queuetype?typecounter_id=V001 HTTP/1.1
```

Sample response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": "r8",
        "code": "PC",
        "name": "Plataforma Cliente",
        "internal": 1
    }
]
```

Returns the list of queue type that the type counter can attend.

Caution: You must consider the following:

- An empty list is returned when not available queue type exist from the specified type counter.

ENDPOINT

`GET /v1/tickets/queuetype`

QUERY PARAMS

typecounter_id *String* REQUIRED

Unique identifier of the type conter used.

RESPONSES

200 *Ok* *List<Queue Type>*

A list of queue types.

400 *Bad Request* *Error*

One or more parameters are not valid, returns a description of validation failed.

404 *Not Found* *Error*

The resource requested not found, returns a simple error message.

500 *Internal Server Error* *Error*

An unexpected error has occurred, returns a simple error message.

Show ticket details

Sample request:

```
GET /v1/tickets/89m32b0 HTTP/1.1
```

Sample response:

```
HTTP/1.1 200 OK
Content-Type: application/json


{
    "id": "89m32b0",
    "number": 2,
    "status": "ATTENDED",
    "position": 10,
    "low_estimated_time": 10,
    "high_estimated_time": 15,
    "phone": "+51987776576",
    "messages": [
        "We have a promotion for you!"
    ],
    "channel": {
        "id": "c10",
        "name": "Mobile"
```

```
    },
    "branch": {
        "id": "27b6",
        "name": "Hiper Central",
        "short_name": "Hiper",
        "address": "Calle Beta 181 - 195, Callao",
        "latitude": -12.049919,
        "longitude": -77.0845193,
        "status": "ACTIVE"
    },
    "sector": {
        "id": "m9",
        "name": "Tower A",
        "short_name": "A",
        "status": "ACTIVE"
    },
    "service": {
        "id": "s8",
        "name": "Plataforma",
        "short_name": "Plataforma"
    },
    "queue_type": {
        "id": "R0009",
        "code": "CC",
        "name": "Comercial"
    },
    "counter": {
        "id": "v10",
        "name": "V10",
        "teller": "achavez",
        "type": {
            "id": "v2002",
            "name": "Ventanilla"
        }
    },
    "created_at": "2017-02-20T10:00:00.000Z",
    "called_at": "2017-02-20T10:05:00.000Z",
    "started_at": "2017-02-20T10:05:18.000Z",
    "print": [
        {
            "align": "CENTER",
            "font": "A_REGULAR",
            "previous": "Welcome to",
            "content": "Hiper",
            "next": "!"
        }
    ]
}
```


Returns a customer ticket by ID.

ENDPOINT

`GET /v1/tickets/{ticket_id}`

PATH PARAMS

ticket_id *String* REQUIRED

Ticket unique identifier (Generated in the ticket creation).

QUERY PARAMS

fields *List<String>* RECOMENDED

Entity fields that will included in the response(See available fields in the ticket object definition). For example: `fields=id,queue_type,number`.

RESPONSES

200 *OK* *Ticket*

A ticket.

---

400 *Bad Request* *Error*

One or more parameters are not valid, returns a description of validation failed.

---

404 *Not Found* *Error*

The resource requested not found, returns a simple error message.

---

500 *Internal Server Error* *Error*

An unexpected error has occurred, returns a simple error message.

Create ticket

Sample request:

```
POST /v1/tickets/ HTTP/1.1
Content-Type: application/json

{
    "channel_id": "c10",
    "branch_id": "27b6",
    "sector_id": "m9",
    "service_id": "s8",
    "customer_id": "c9was9js",
    "doc_type": "D",
    "doc_number": "90452291",
    "phone": "+51987882567"
}
```

Sample response:

```
HTTP/1.1 201 Created
Content-Type: application/json

{
    "id": "89m32b0",
    "number": 2,
    "status": "WAITING",
    "position": 10,
    "low_estimated_time": 10,
    "high_estimated_time": 15,
    "phone": "+51987882567",
    "messages": [
        "We have a promotion for you!"
    ],
    "channel": {
        "id": "c10",
        "name": "Mobile"
    },
    "branch": {
        "id": "27b6",
        "name": "Hiper Central",
```

```
        "short_name": "Hiper",
        "address": "Calle Beta 181 - 195, Callao",
        "latitude": -12.049910,
        "longitude": -77.0845193,
        "status": "ACTIVE"
    },
    "sector": {
        "id": "m9",
        "name": "Tower A",
        "short_name": "A",
        "status": "ACTIVE"
    },
    "service": {
        "id": "s8",
        "name": "Plataforma",
        "short_name": "Plataforma"
    },
    "queue_type": {
        "id": "R0009",
        "code": "CC",
        "name": "Comercial"
    },
    "counter": {
        "id": "v10",
        "name": "V10",
        "teller": "achavez",
        "type": {
            "id": "v2002",
            "name": "Ventanilla"
        }
    },
    "created_at": "2017-02-20T10:00:00.000Z",
    "print": [
        {
            "align": "CENTER",
            "font": "A_REGULAR",
            "previous": "Welcome to",
            "content": "Hiper",
            "next": "!"
        }
    ]
}
```


Create a new customer ticket and returns it.

Caution: You must consider the following before create a ticket:

- The referenced channel must support the requested service.
- The referenced branch must contain the requested sector.
- The referenced sector must support the requested service.
- The customer must has no other ticket in waiting for the same service.

ENDPOINT

`POST /v1/tickets/`

REQUEST

channel_id *String* REQUIRED

Unique identifier of the channel used. This value must be set in the client application.

branch_id *String* REQUIRED

Unique identifier of the requested branch.

sector_id *String* REQUIRED

Unique identifier of the requested sector.

service_id *String* REQUIRED

Unique identifier of the requested service.

customer_id *String*

Customer unique identifier.

doc_type *String*

Type of document used to identify the client.

Maximum length: `5`

doc_number *String*

Number of document used to identify the client.

Maximum length: `20`

phone *String*

Customer phone for notifications associated to the ticket.

Validation pattern: `^\+[1-9]{1}[0-9]{3,14}$`

RESPONSES

201 *Created* *Ticket*

Successful creation, returns the new ticket.

400 *Bad Request* *Error*

One or more parameters are not valid, returns a description of validation failed.

500 *Internal Server Error* *Error*

An unexpected error has occurred, returns a simple error message.

Close ticket attention

Sample request:

```
POST /v1/tickets/89m32b0/close HTTP/1.1
Content-Type: application/json

{
    "phone": "+51947812025",
    "qualification_value": 3
}
```

Sample response:

```
HTTP/1.1 200 Ok
Content-Type: application/json

{
    "id": "89m32b0",
    "number": 2,
    "status": "ATTENDED",
    "position": 10,
    "low_estimated_time": 10,
    "high_estimated_time": 15,
    "phone": "+51947812025",
    "messages": [
        "We have a promotion for you!"
    ],
    "channel": {
        "id": "c10",
        "name": "Mobile"
    },
    "branch": {
        "id": "27b6",
        "name": "Hiper Central",
        "short_name": "Hiper",
        "address": "Calle Beta 181 - 195, Callao",
        "latitude": -12.049919,
        "longitude": -77.0845193,
        "status": "ACTIVE"
    },
    "sector": {
        "id": "m9",
        "name": "Tower A",
        "short_name": "A",
        "status": "ACTIVE"
    },
    "service": {
        "id": "s8",
        "name": "Plataforma",
        "short_name": "Plataforma"
    },
    "queue_type": {
        "id": "R0009",
        "code": "CC",
        "name": "Comercial"
    },
    "counter": {
        "id": "v10",
        "name": "V10",
        "teller": "achavez",
        "type": {
            "id": "v2002",
            "name": "Ventanilla"
        }
    },
    "created_at": "2017-02-20T10:00:00.000Z",
    "called_at": "2017-02-20T10:10:50.000Z",
    "started_at": "2017-02-20T10:10:55.000Z",
    "finished_at": "2017-02-20T10:11:41.000Z",
    "print": [
        {
            "align": "CENTER",
            "font": "A_REGULAR",
            "previous": "Welcome to",
```

```
            "content": "Hiper",
            "next": "!"
        }
    ]
}
```


Finish ticket attention.

> **Caution:** You must consider the following before finish ticket attention:
>
> - The referenced ticket must be in attention.

ENDPOINT

`POST /v1/tickets/{ticket_id}/close`

PATH PARAMS

ticket_id *String* REQUIRED

Ticket unique identifier (Generated in the ticket creation). This identifier is encrypted.

REQUEST

phone *String*

Customer phone for notifications associated to the ticket.

Validation pattern: `^\+[1-9]{1}[0-9]{3,14}$`

qualification_value *Integer*

Ticket qualification value by the customer before transfer.

Possible values:

- `2`: Bad rating
- `3`: Regular rating
- `4`: Good rating

RESPONSES

200 *Ok* *Ticket*

Ticket successfully finalized. Returns the ticket.

400 *Bad Request* *Error*

One or more parameters are not valid, returns a description of validation failed.

404 *Not Found* *Error*

The resource requested not found, returns a simple error message.

500 *Internal Server Error* *Error*

An unexpected error has occurred, returns a simple error message.

Transfer Ticket

Sample request:

```
POST /v1/tickets/89m32b0/transfer HTTP/1.1
Content-Type: application/json

{
    "sector_id": "s51n5",
    "countertype_id": "v56a5",
    "queuetype_id": "q5s53"
}
```

Sample response:

```
HTTP/1.1 200 Ok
Content-Type: application/json

{
    "id": "89m32b0",
    "number": 2,
    "status": "WAITING",
    "internal": 1,
    "temporal": 0,
    "derived": 1,
    "messages": [
        "We have a promotion for you!"
    ],
    "channel": {
        "id": "c10",
        "name": "Mobile"
    },
    "branch": {
        "id": "27b6",
        "name": "Hiper Central",
        "short_name": "Hiper",
        "address": "Calle Beta 181 - 195, Callao",
        "latitude": -12.049919,
        "longitude": -77.0845193,
        "status": "ACTIVE"
    },
     "sector": {
        "id": "m9",
        "name": "Tower A",
        "short_name": "A",
        "status": "ACTIVE"
    },
    "service": {
        "id": "s8",
        "name": "Plataforma",
        "short_name": "Plataforma"
    },
     "queue_type": {
        "id": "R0009",
        "code": "CC",
        "name": "Comercial"
    },
    "counter": {
        "id": "v10",
        "name": "V10",
```

```
        "teller": "pchavez",
        "type": {
            "id": "v2002",
            "name": "Ventanilla"
        }
    },
    "customer": {
        "doc_type": "D",
        "doc_number": "74693766",
        "name": "Juan Perez",
        "phone": "+51987776576",
        "category": {
            "id": "I000004",
            "name": "VIP"
        }
    },
    "created_at": "2017-02-28T10:00:00.000Z"
}
```


Transfer ticket to another counter

> **Caution:** You must consider the following before transfer a ticket:
>
> - The referenced ticket must be in attention.

ENDPOINT

`POST /v1/tickets/{ticket_id}/transfer`

PATH PARAMS

ticket_id *String* REQUIRED

Ticket unique identifier (Generated in the ticket creation). This identifier is encrypted.

REQUEST

sector_id *String* REQUIRED

Unique identifier of the sector to which the ticket will be transfered.

countertype_id *String* REQUIRED

Unique identifier of the counter type to which the ticket will be transfered.

queuetype_id *String* REQUIRED

Unique identifier of the type ticket to which the ticket will be transfered.

RESPONSES

200 *Ok* *Ticket*

Successful derivation. Returns the new transfered ticket.

400 *Bad Request* *Error*

One or more parameters are not valid, returns a description of validation failed.

404 *Not Found* *Error*

The resource requested not found, returns a simple error message.

500 *Internal Server Error* *Error*

An unexpected error has occurred, returns a simple error message.

Add Operation

Sample request:

```
POST /v1/tickets/89m32b0/operation HTTP/1.1
Content-Type: application/json

{
    "operation_id": "s84a5"
}
```

Sample response:

```
HTTP/1.1 201 Created
Content-Type: application/json

{
    "id": "89m32b0",
    "number": 2,
    "created_at": "2018-02-24T12:30:00.000Z",
    "operationtype": [
        {
            "id": "o9",
            "name": "Pago de servicios",
            "level": "2"
        }
    ]
}
```

Add a new operation for the ticket

Caution: You must consider the following before create a operation:

- The use of operations must be configured at BMatic
- The referenced operation must be configured at BMatic.
- The referenced ticket must be in attention.

ENDPOINT

`POST /v1/tickets/{ticket_id}/operation`

PATH PARAMS

ticket_id *String* REQUIRED

Ticket unique identifier (Generated in the ticket creation). This identifier is encrypted.

REQUEST

operation_id *String* REQUIRED

Unique identifier of the requested operation.

RESPONSES

201 *Created* *Operation*

Successful creation. Returns the new operation.

400 *Bad Request* *Error*

One or more parameters are not valid, returns a description of validation failed.

404 *Not Found* *Error*

The resource requested not found, returns a simple error message.

500 *Internal Server Error* *Error*

An unexpected error has occurred, returns a simple error message.

Create internal ticket

Sample request:

```
POST /v1/tickets/internal HTTP/1.1
Content-Type: application/json

{
    "counter_id": "v045a",
    "teller_id": "ui405",
    "queuetype_id": "q5s53"
}
```

Sample response:

```
HTTP/1.1 201 Created
Content-Type: application/json

{
    "id": "89m32b0",
    "number": 2,
    "status": "WAITING",
    "internal": 1,
    "temporal": 0,
    "derived": 0,
    "messages": [
        "We have a promotion for you!"
    ],
    "channel": {
        "id": "c10",
        "name": "Mobile"
    },
```

```
    "branch": {
        "id": "27b6",
        "name": "Hiper Central",
        "short_name": "Hiper",
        "address": "Calle Beta 181 - 195, Callao",
        "latitude": -12.049919,
        "longitude": -77.0845193,
        "status": "ACTIVE"
    },
     "sector": {
        "id": "m9",
        "name": "Tower A",
        "short_name": "A",
        "status": "ACTIVE"
    },
    "service": {
        "id": "s8",
        "name": "Plataforma",
        "short_name": "Plataforma"
    },
     "queue_type": {
        "id": "R0009",
        "code": "CC",
        "name": "Comercial"
    },
    "counter": {
        "id": "v10",
        "name": "V10",
        "teller": "pchavez",
        "type": {
             "id": "v2002",
             "name": "Ventanilla"
        }
    },
    "customer": {
        "doc_type": "D",
        "doc_number": "74693766",
        "name": "Juan Perez",
        "phone": "+51987776576",
        "category": {
             "id": "I000004",
             "name": "VIP"
        }
    },
    "created_at": "2017-02-20T18:00:00.000Z"
}
```


Create internal ticket

> **Caution:** You must consider the following before create a internal ticket:
>
> - The referenced counter must be free
> - The referenced counter must be authorized to generate internal ticket
> - The referenced type of ticket must allow the generation of internal tickets

ENDPOINT

`POST /v1/tickets/internal`

REQUEST

counter_id *String* REQUIRED

Unique identifier of the requested counter.

teller_id *String* REQUIRED

Unique identifier of the requested teller.

queuetype_id *String* REQUIRED

Unique identifier of the requested type ticket.

RESPONSES

201 *Created* *Ticket*

Successful creation, returns the new internal ticket.

400 *Bad Request* *Error*

One or more parameters are not valid, returns a description of validation failed.

500 *Internal Server Error* *Error*

An unexpected error has occurred, returns a simple error message.

Counters API

You should consider using the recommended parameters for better performance.

List counter types

Sample request:

```
GET /v1/counters/type HTTP/1.1
```

Sample response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    {
        "id": "v2002",
        "name": "Ventanilla"
    }
}
```

Returns the counter types configured at BMatic.

ENDPOINT

`GET /v1/counters/types`

RESPONSES

200 *Ok* *List<CounterType>*

A list of counter types.

404 *Not Found* *Error*

The resource requested not found, returns a simple error message.

500 *Internal Server Error* *Error*

An unexpected error has occurred, returns a simple error message.

Show counter details by teller

Sample request:

```
GET /v1/counters/users?user_id=pchavez HTTP/1.1
```

Sample response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": "W0001",
    "name": "V1",
    "teller": "pchavez",
    "type": {
        "id": "V0010",
        "name": "Comercial",
    },
    "queuetypes": [
        {
            "id": "R0001",
            "code": "PR",
            "name": "Preferencial",
            "internal": 0
        },
        {
            "id": "R0002",
            "code": "NC",
            "name": "No cliente",
            "internal": 1
        }
    ]
}
```

Returns the counter details according to the operator logged in.

ENDPOINT

`GET /v1/counters/users?user_id={username}`

QUERY PARAMS

user_id *String* REQUIRED

User unique identifier.

RESPONSES

200 *Ok* *Counter*

A counter.

400 *Bad Request* *Error*

One or more parameters are not valid, returns a description of validation failed.

404 *Not Found* *Error*

The resource requested not found, returns a simple error message.

500 *Internal Server Error* *Error*

An unexpected error has occurred, returns a simple error message.

Common Object Definitions

Collection of common objects used by the API.

Branch

Sample object:


```
{
    "id": "27b6",
    "name": "Hiper Central",
    "short_name": "Hiper",
    "address": "Calle Beta 181 - 195, Callao",
    "latitude": -12.049919,
    "longitude": -77.0845193,
    "status": "ACTIVE",
    "congestion": "MEDIUM",
    "low_estimated_time": 10,
    "high_estimated_time": 14,
    "sectors": [
        {
            "id": "m9",
            "name": "Tower A",
            "short_name": "A",
            "status": "ACTIVE"
        }
    ]
}
```

Model of branch object.

Caution: The `congestion`, `low_estimated_time` and `high_estimated_time` are calculated using the configuration of the associated branch type. You need configure the next:

- Time range to take for calculate the waiting time.
- Desviation of estimated waiting time.
- Minimal waiting time of medium congestion.
- Minimal waiting time of high congestion.

If the branch is `INACTIVE` these fields will not return.

FIELDS

id *String*

Branch unique identifier (Configured from Bmatic configuration panel).

name *String*

Complete name of the branch.

Maximum length: `50`

short_name *String*

Name to display in small devices.

Maximum length: `20`

address *String*

Branch address.

Maximum length: `100`

latitude *Float*

Latitude component of branch location.

longitude *Float*

Longitude component of branch location.

status *Enum*

Current status of branch.

Possible values:

- `ACTIVE`: At least one sector is `ACTIVE`
- `INACTIVE`: All sectors are `INACTIVE`

congestion *String*

Flag of branch congestion.

Possible values:

- `LOW`: The average waiting time is lower to the minimal waiting time of medium congestion.
- `MEDIUM`: The average waiting time is lower to the minimal waiting time of high congestion but equal or upper to the minimal waiting time of medium congestion.
- `HIGH`: The average waiting time is equal or upper to the minimal waiting time of high congestion.

---

low_estimated_time *Integer*

Lower bound of the estimated waiting time. It's equal to the average waiting time minus the configured desviation.

---

high_estimated_time *Integer*

Upper bound of the estimated waiting time. It's equal to the average waiting time plus the configured desviation.

---

sectors *List<Sector>*

List of the branch's sectors. A correct configuration include at least one sector per branch.

Unique items: `true`
Minimum items: `1`

Channel

Sample object:

```
{
    "id": "c10",
    "name": "Mobile",
}
```

Model of channel object.

FIELDS

---

id *String*

Channel unique identifier (Configured from Bmatic configuration panel).

---

name *String*

Complete name of the channel.

Maximum length: `20`

Counter

Sample object:


```
{
    "id": "v10",
    "name": "V10",
    "teller": "achavez",
    "type": {
        "id": "v2002",
        "name": "Ventanilla"
    },
    "queuetypes": [
        {
            "id": "R0001",
            "code": "PR",
            "name": "Preferencial",
            "internal": 0
        },
        {
            "id": "R0002",
            "text": "NC",
            "name": "No cliente",
            "internal": 1
        }
    ]
}
```


Model of counter object.

FIELDS

id *String*

Counter unique identifier (Configured from Bmatic configuration panel).

name *String*

Name of the counter.

Maximum length: `20`

teller *String*

Account name of teller in counter.

Maximum length: `30`

type *CounterType*

Associated counter type.

queuetypes *List<QueueType>*

List of queue types that the counter can attend.

Counter Type

Sample object:


```
{
    "id": "v2002",
    "name": "Ventanilla"
}
```


Model of counter type object.

FIELDS

id *String*

Counter type unique identifier (Auto generated by Bmatic).

name *String*

Complete name of the counter type.

Maximum length: `50`

Day

Sample object:


```
{
    "day": "2017-02-20T00:00:00.000Z",
    "ranges": [
        {
            "start_time": "2017-02-20T14:00:00.000Z",
            "end_time": "2017-02-20T15:00:00.000Z"
        }
    ]
}
```


Model of day object.

FIELDS

day *Date*

Date with available ranges.

Standard: `ISO 8601`

ranges *List<Range>*

List of available time ranges.

Error

Sample object:


```
{
    "type": "ResourceNotFoundException",
    "message": "Resource not found for the request GET /v2",
}
```


Model of error object.

FIELDS

type *String* Name of type error.

message *String* Simple description of the error.

Operation

Sample object:


```
{
    "id": "89m32b0",
    "number": 2,
    "created_at": "2018-02-24T12:30:00.000Z",
    "operationtype":{
        "id": "o8",
        "name": "Pago de servicios",
        "level": "2"
        }
}
```


Model of operation object.

FIELDS

id *String*

Operation unique identifier.

number *Integer*

Number of operation.

created_at *DateTime*

Creation date of operation.

Standard: `ISO 8601`

type *OperationType*

Associated operation type.

Operation Type

Sample object:

```
{
    "id": "o8",
    "name": "Pago de servicios",
    "level": "2"
}
```

Model of operation type object.

FIELDS

id *String*

Operation type unique identifier.

name *String*

Name of operation type.

Maximum length: `40`

level *Integer*

Level of operation.

Maximum length: `1`

Print Line

Sample object:

```
{
    "align": "CENTER",
    "font": "A_REGULAR",
    "previous": "Welcome to",
    "content": "Hiper",
    "next": "!"
}
```

Model of print line object.

FIELDS

align *Enum*

Alignment of the line.

Possible values:

- `LEFT`
- `CENTER`
- `RIGHT`

---

font *Enum*

Font of the line.

Possible values:

- `A_REGULAR`
- `B_REGULAR`
- `A_DOUBLE_HIGH`
- `B_DOUBLE_HIGH`
- `A_DOUBLE_WIDTH`
- `B_DOUBLE_WIDTH`
- `A_QUADRUPLE`
- `B_QUADRUPLE`
- `C_DOUBLE_HIGH`
- `C_DOUBLE_WIDTH`
- `C_QUADRUPLE`

---

previous *String*

Message to print before the content.

Maximum length: `50`

---

content *String*

Message to print as content.

Maximum length: `50`

---

next *String*

Message to print after the content.

Maximum length: `50`

Queue Type

Sample object:

```
{
    "id": "R0009",
    "code": "CC",
    "name": "Comercial",
```

```
    "internal": 1
}
```

Model of queue type object.

FIELDS

id *String*

Unique identifier of the queue type(Auto generated by Bmatic).

code *String*

Code of the queue type.

Maximum length: `5`

name *String*

Complete name of the queue type.

Maximum length: `50`

internal *Integer*

Flag for internal ticket.

Possible values:

- `0`: The queue type can not generate internal tickets
- `1`: The queue type can generate internal tickets

Sector

Model of sector object.

Sample object:

```
{
    "id": "m9",
    "name": "Tower A",
    "short_name": "A",
    "status": "OPEN"
}
```

FIELDS

id *String*

Sector unique identifier (Auto generated by Bmatic).

name *String*

Complete name of the sector.

Maximum length: `50`

---

short_name *String*

Name to display in small devices.

Maximum length: `20`

---

status *Enum*

Current status of the sector.

Possible values:

- `ACTIVE`: At least one counter in the sector is `ACTIVE` or `SUSPENDED`.
- `INACTIVE`: All counters in the sector are `INACTIVE`.

Ticket

Sample object:


```
{
    "id": "89m32b0",
    "number": 2,
    "status": "BLOCKED",
    "position": 10,
    "low_estimated_time": 10,
    "high_estimated_time": 15,
    "phone": "+51987776576",
    "internal": 1,
    "temporal": 0,
    "derived": 0,
    "messages": [
        "We have a promotion for you!"
    ],
    "channel": {
        "id": "c10",
        "name": "Mobile"
    },
    "branch": {
        "id": "27b6",
        "name": "Hiper Central",
        "short_name": "Hiper",
        "address": "Calle Beta 181 - 195, Callao",
        "latitude": -12.049919,
        "longitude": -77.0845193,
        "status": "ACTIVE"
    },
    "sector": {
        "id": "m0",
        "name": "Tower A",
        "short_name": "A",
        "status": "ACTIVE"
    },
    "service": {
        "id": "s8",
        "name": "Plataforma",
```

```
        "short_name": "Plataforma"
    },
    "queue_type": {
        "id": "R0009",
        "code": "CC",
        "name": "Comercial"
    },
    "counter": {
        "id": "v10",
        "name": "V10",
        "teller": "achavez",
        "type": {
            "id": "v2002",
            "name": "Ventanilla"
        }
    },
    "created_at": "2017-02-20T10:00:00.000Z",
    "called_at": "2017-02-20T10:05:00.000Z",
    "started_at": "2017-02-20T10:05:18.000Z",
    "finished_at": "2017-02-20T10:11:41.000Z",
    "print": [
        {
            "align": "CENTER",
            "font": "A_REGULAR",
            "previous": "Welcome to",
            "content": "Hiper",
            "next": "!"
        }
    ]
}
```


Model of ticket object.

Caution: The entities associated to ticket not provide all their fields for performance reasons. If you need more information, you can use others endpoints to get this information using the ids returned in the ticket. You can to see the default information provided in the sample object.

The `low_estimated_time` and `high_estimated_time` are calculated using the configuration of the associated branch type. You need configure the next:

- Time range to consider to calculate the waiting time.
- Desviation of waiting time.

If the branch associated is not in `WAITING`, these fields are not returned including position field.

FIELDS

id *String*

Ticket unique identifier (Auto generated by Bmatic). This value is encrypted.

number *Integer*

Number in queue assigned to the ticket.

status *Enum*

Current status of the ticket.

Possible values:

- `WAITING`: When the ticket is waiting for teller call.
- `CALLED`: When the ticket is called by teller.

- `INATTENTION`: When the ticket is in attetion.
- `ATTENDED`: When the attention of ticket is finished.

---

position *Integer*

Ticket position in the queue.

---

low_estimated_time *Integer*

Lower bound of the estimated waiting time. It's equal to the average waiting time minus the configured desviation.

---

high_estimated_time *Integer*

Upper bound of the estimated waiting time. It's equal to the average waiting time plus the configured desviation.

---

phone *String*

Customer phone for notifications associated to the ticket.

---

internal *Integer*

Indicates if the ticket is internal or not.

Possible values:

- `0`: When the ticket is not internal.
- `1`: When the ticket is internal.

---

temporal *Integer*

Indicates if the ticket is temporal or not.

Possible values:

- `0`: When the ticket is not temporal.
- `1`: When the ticket is temporal.

---

derived *Integer*

Indicates if the ticket is derived or not.

Possible values:

- `0`: When the ticket is not derived.
- `1`: When the ticket is derived.

---

messages *List<String>*

Messages to print.

---

channel *Channel*

Channel where the ticket was created.

---

branch *Branch*

Branch where the ticket is valid.

---

sector *Sector*

Sector where the ticket is valid.

---

service *Service*

Service requested by the customer.

---

queue_type *QueueType*

Assigned queue type.

---

counter *Counter*

Assigned counter.

---

created_at *DateTime*

Creation date of ticket.

Standard: `ISO 8601`

---

called_at *DateTime*

Call date of ticket.

Standard: `ISO 8601`

---

started_at *DateTime*

Start date of ticket attention.

Standard: `ISO 8601`

---

finish_at *DateTime*

Finish date of ticket attention.

Standard: `ISO 8601`

---

print *List<PrintLine>* List of lines to print.
