Buenos Aires Ciudad

Agencia de Sistemas

Marco Normativo de IT

ES0901 - Estándar de Desarrollo
Agencia de Sistemas de Información

Gobierno de la Ciudad Autónoma de Buenos Aires

Julio 2024

Índice

1 Introducción

El presente documento tiene por finalidad definir los requisitos que deben cumplir todas las aplicaciones del GCABA, como así también establecer la interacción con los distintos actores en el marco del proceso de desarrollo de dichas aplicaciones. En este sentido, corresponde mencionar que el presente Estándar de Desarrollo se complementa con los estándares de Seguridad (ES0902) y de arquitectura (ES0101) el cual se enfoca sobre los aspectos estructurales de las aplicaciones y su entorno de operación.

Se deja constancia que las definiciones utilizadas son aplicables tanto para las aplicaciones desarrolladas según especificaciones de las distintas reparticiones y agencias de la GCABA, como para aquellas aplicaciones de software comerciales o desarrolladas por terceros con o sin adaptaciones a medida, independientemente de su condición comercial (ej. open source, licencia de uso, servicios Cloud, cesión de derechos, etc.).

> ***IMPORTANTE**: Salvo que para un proyecto en particular se especifique por contrato algún acuerdo diferente para alguno de los puntos detallados en este documento, todos los sistemas que se desarrollen en el ámbito de los proyectos de GCABA deben respetar la totalidad de los criterios aquí descriptos. En virtud de ello, la existencia de un contrato en estas condiciones, deberá contar con la aprobación de la Agencia de Sistemas de Información.*

2 Marco Normativo de TI

Toda solución de software deberá cumplir con lo expresado en el Marco Normativo de TI del GCABA, publicado en el boletín oficial del día 08-11-2013, Resolución 177-ASINF-2013, Resolución 239-ASINF/2014 y N° 12/ASINF/17. Dicha documentación se encuentra disponible en http://www.buenosaires.gob.ar/asi/estandares.

3 Principios de Aplicaciones

Existen ciertos principios generales que rigen a todo desarrollo o customización que nos lleva implementar las buenas prácticas de software, orientadas a aplicaciones óptimas en funcionamiento y mantenimiento.

A Nivel Organización:

O1- Se deben respetar los principios y normativas vigentes de TI del GCABA

O2-El control y responsabilidad de la información de la aplicación debe estar a cargo de un organismo perteneciente a GCABA

O3-Para toda funcionalidad nueva debe validarse previamente si existe y está disponible para sumarla a su necesidad, con el objetivo de minimizar costos y reutilizar procesos estándares.

O4-En el caso de adquirir un producto de software (no a medida) debe contemplarse mínimas adaptaciones y costos a corto plazo y largo plazo al momento de implementarse.

A Nivel Calidad:

C1-Toda aplicación deben contar con una autenticación

C2-Toda aplicación debe contar con los 4 ambientes (DEV, QA, HML y PRD)

C3-Debe contar con documentación del alcance funcional, arquitectura, proceso de instalación y registro de control de cambio.

C4-La comunicación entre las aplicaciones debe estar formalizadas y acordadas.

4 Proceso de Gestión de Cambios de Software

El principal objetivo en relación a las aplicaciones de software es la sustentabilidad. Atento a ello, definimos sustentabilidad en términos de:

Calidad

La calidad de las aplicaciones de software es un factor directamente asociado a los costos de mantenimiento, disrupción de servicio, exposición negativa hacia el Ciudadano, etc. El proceso de gestión de cambios es permanentemente ajustado para incorporar todos los controles que sean necesarios a fin de garantizar la calidad total de las aplicaciones instaladas.

Desempeño

El Gobierno de la Ciudad considera la tecnología y el software como pilares para su gestión efectiva. Es por ello que, las aplicaciones de software deben estar a la altura de tales exigencias en términos de tiempos de respuesta, escalabilidad, performance, y rendimiento general. El proceso de gestión de cambios incluye los pasos necesarios para certificar el adecuado desempeño de las aplicaciones.

Seguridad

En un contexto donde los ataques y la búsqueda de vulnerabilidades para explotar son parte de la rutina, las aplicaciones deben implementar mecanismos cada vez mejores y más inteligentes con el objeto de prevenir y mitigar los daños que todo tipo de intrusión externa y/o interna pudieran ocasionar.

El proceso de gestión de cambios refleja la importancia de la sustentabilidad de las aplicaciones, incluyendo actividades específicas para controlar y asegurar cada uno de los aspectos mencionados.

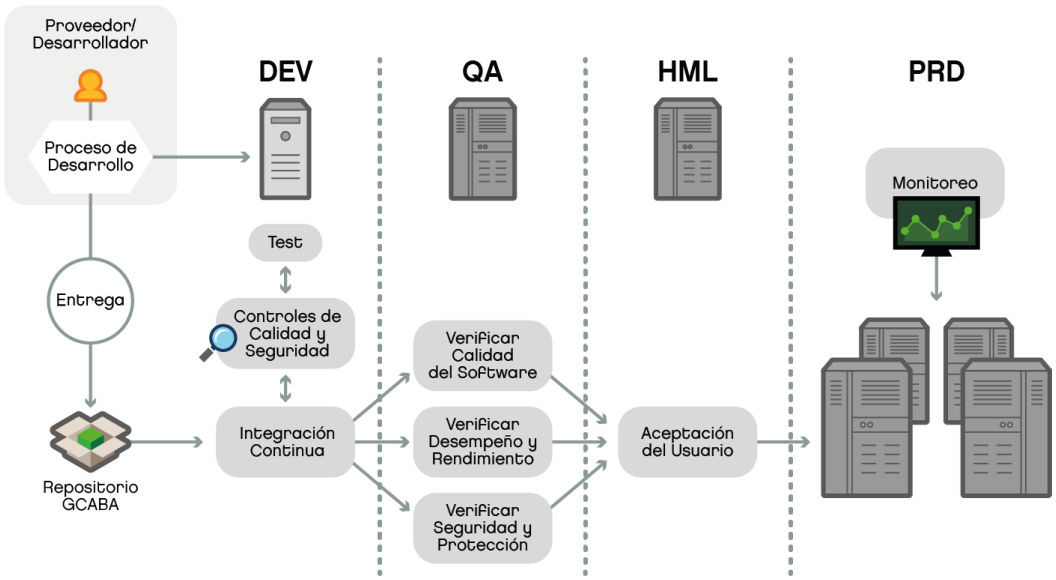


Fig. 1 - Proceso de Gestión de Cambios

El ambiente DEV

Toda iniciativa de software –proyecto, evolución, etc.- debe contar con un ambiente de desarrollo controlado por GCABA como punto inicial del proceso, permitiendo al equipo de Desarrollo implementar la solución con los controles/Verificaciones correspondientes. El ambiente DEV posee la misma infraestructura que el ambiente productivo, con los recursos acorde a un ambiente de Test.

Independientemente de la plataforma utilizada por el Desarrollador para generar su producto, todas las aplicaciones deben desempeñarse correctamente en el entorno DEV provisto a tal efecto.

Repositorio de Fuentes GCABA

El código fuente de las aplicaciones es uno de los “entregables” de cada proyecto y debe ser impactado en el repositorio definido por la Agencia para tal fin. (Ver Anexo Repositorio de Fuentes).

Integración Continua

La actualización en el repositorio inicia el proceso de integración continua coordinado con GCABA, el mismo se compone de:

- Análisis de calidad de Código
- Análisis de detección de vulnerabilidades de seguridad en el código.
- Ejecución de Test unitarios, los cuales deberán cumplir con la premisa de ser totalmente independientes
- Generación del paquete a implementar
- Implementación en Ambiente DEV
- Ejecución de detección de vulnerabilidades de seguridad en la aplicación implementada.
- Ejecución de “smoke tests”
- Ejecución de tests de integraciones básicas.

Los resultados de todos los pasos mencionados deberán reflejarse en la herramienta de Gestión seleccionada para tal fin.

El ambiente de QA

Cuando en DEV se encuentre una versión que haya superado los controles exitosamente, es candidata a implementar en producción y el proceso permite avanzar al ambiente de QA administrado/controlado por recursos del GCABA, donde se verifican automáticamente los mismos controles que en DEV y se suman nuevos como:

- Pruebas exploratorias
- Pruebas de requisitos No Funcionales(Performance, Stress, Escalabilidad, Disponibilidad)
- Pruebas de integración y servicios exhaustivas.
- Assessment de Seguridad Complementario verificando las buenas prácticas(Ver Estándar de Seguridad)

El ambiente de HML

En los casos que los controles en Calidad y Seguridad sean satisfactorios la versión, estará en condiciones de avanzar al ambiente de Homologación/UAT donde los usuarios podrán realizar las tareas de control y verificación respecto de los cambios para aprobar o rechazar.

Con la conformidad del usuario responsable, nos encontramos en condiciones para elevar el cambio a Producción.

El ambiente de PRD

Las aplicaciones productivas contarán con un monitoreo de Infraestructura y chequeos adicionales del comportamiento on-line, contando con controles proactivos y alarmas del funcionamiento.

5 Entregables

Todo proyecto de software deberá proveer la información que necesiten las distintas actividades de los procesos de gestión del cambio, release management y resolución de incidentes. Se detallan a continuación los entregables mínimos requeridos para un proyecto de software y el contenido esperado en cada uno de ellos:

Entregable	Contenido
Documento Plan de Proyecto	• Objetivo y alcance del proyecto. • Cronograma de tareas. • Plan de entregables del proyecto e identificación de hitos. • Organigrama, roles y responsabilidades. • Asignación de recursos a roles. • Estimación de esfuerzo total. • Análisis de riesgos y plan de mitigación de los mismos. • Plan de comunicación y seguimiento del proyecto. • Gestión de la configuración. • Gestión de cambios en requerimientos.
Especificaciones	• Alcance: Visión y objetivos • Historias de Usuario • Criterios de aceptación / Mapping historias de usuario • Requerimientos no funcionales
Documento de Arquitectura	• Descripción de los Servicios • Descripción de Integraciones • Topología y modularización • Modelo lógico y físico de datos • Políticas globales de diseño (concurrencia, almacenamiento de datos, mecanismos de comunicación, mecanismos de seguridad, manejo de errores, etc.) • Arquitectura tecnológica (sistema operativo, software de base, motor de base de datos, etc.) • Dimensionamiento acorde a las especificaciones no funcionales. La arquitectura propuesta debe cumplir con las exigencias del documento Estándar de Arquitectura de la ASI. • Plan de Pruebas con la estrategia de prueba a seguir (pruebas funcionales, no funcionales: de integración, stress, regresión, etc.), cronograma, diseño de pruebas por módulos.
Paquete de Software	Para cada entrega acordada con el proveedor se debe presentar: • Código fuente subido en el repositorio e implementado en DEV. • Documentos de control de cambios. • Cobertura de Test unitarios con piso del 80%. • Manual de Instalación. • Manual de Operación. • Instalación de todos los componentes de software adicionales necesarios para el correcto funcionamiento de lo entregado.

Entregable	Contenido
Documento Manual de Usuario	• Documentación completa del software que incluya todas las funcionalidades para todos los roles y la administración de roles, permisos y seguridad.
Material de Capacitación	• Propuesta con el plan de capacitación. • Documentación que incluya la completitud de la funcionalidad a capacitar.
Minutas e Informes de avances	• Todos las minutas generadas durante el proyecto e informes de avances presentados/enviados.

La documentación del proyecto se debe entregar en la herramienta colaborativa del proyecto seleccionada, con excepción del código fuente que se sube al repositorio.
Adicionalmente, los entregables en su versión final deben ser entregados por el canal definido en el contrato o normativa vigente.

6 Definiciones de Infraestructura

Esta sección tiene por objeto formalizar y especificar el modelo de arquitectura aplicable a los Data Centers operados por la ASI, asegurando mediante una adecuada gestión tecnológica, su escalabilidad, agilidad y alta disponibilidad.

Todos los nuevos sistemas que se instalen en Data Centers operados por la ASI deberán respetar los criterios aquí descriptos. Los criterios establecidos en este documento serán también aplicables para los integradores que implementen todo tipo de sistemas desarrollados por terceros dentro de la infraestructura informática del GCABA en cumplimiento de contratos que así lo soliciten.

Estas implementaciones, deberán efectuarse respetando los criterios de arquitectura y entornos de desarrollo, homologación y producción descriptos en este documento de tal forma que los sistemas producto de la contratación puedan ser implementados en Data Centers del GCABA.

6.1 Arquitectura Física

La arquitectura de los sistemas y la topología de las redes de servidores de todos los ambientes proporcionados por la ASI donde se alojen los sistemas y Bases de Datos en el Data Center de la ASI son iguales.

En estos servidores de aplicaciones residirá la lógica de presentación, es decir la que se comunica con los navegadores utilizados por el Usuario, y parte o toda la lógica de negocio. Por lo tanto, en los navegadores no debe residir ningún dato, tabla, archivo o documento excepto durante el tiempo que dure una transacción. Toda información deberá ser almacenada en servidores específicos.

Los servidores de aplicación serán agrupados bajo el esquema de Granjas, permitiendo la escalabilidad de los sistemas primero en forma horizontal, través del incremento de la cantidad y capacidad de los servidores físicos, y en segundo plano a través del aumento de la capacidad de los servidores en los cuales residan los motores de Bases de Datos.

6.2 DMZs

Una DMZ se usa habitualmente para ubicar servidores que es necesario que sean accedidos desde fuera, como servidores de correo electrónico, Web y DNS. El esquema denominado, Granja de Servidores de Aplicaciones, se localiza dentro de una DMZ

Cada DMZ contiene un balanceador de carga de capa 7 (load balancer), quien será el encargado de distribuir las peticiones entre los distintos servidores de aplicaciones, a fin de poder distribuir la carga de transacciones entre los distintos servidores que atienden a los Usuarios, y al mismo tiempo contar con la facilidad de efectuar el mantenimiento en caliente de las aplicaciones.

La ASI contará con cuatro DMZs:

- **DMZ de Producción para Usuarios Externos**: Utilizada únicamente para alojar aplicaciones en producción que necesiten ser accedidas desde fuera de la Red del GCABA.
- **DMZ de Producción para Usuarios Internos:** Utilizada únicamente para alojar aplicaciones en producción que necesiten ser accedidas desde dentro de la Red del GCABA.
- **DMZ de Homologación para Usuarios Externos:** Utilizada para alojar las aplicaciones que se encuentren en etapa de homologación, y necesiten ser accedidas desde fuera de la Red del GCABA.
- **DMZ de Homologación para Usuarios Internos:** Utilizada para alojar las aplicaciones que se encuentren en etapa de homologación, y necesiten ser accedidas desde dentro de la Red del GCABA.

Esta arquitectura/topología será idéntica tanto para los sistemas que implementan transacciones desde Internet por Usuarios externos al GCABA, como para las aplicaciones accedidas por personal del GCABA desde la red interna del Gobierno.

6.3 Esquema Lógico del Data Center – Política General del Flujo de Tráfico

Las políticas generales de flujos de tráfico pretenden dar las pautas de dirección de los flujos de datos entre dominios.

El dominio de redes públicas sólo podrá realizar peticiones a equipos que se encuentren en el domino de extranet. En ningún caso deben realizar conexiones a equipos que se encuentren en dominios confiables distintos de la extranet.

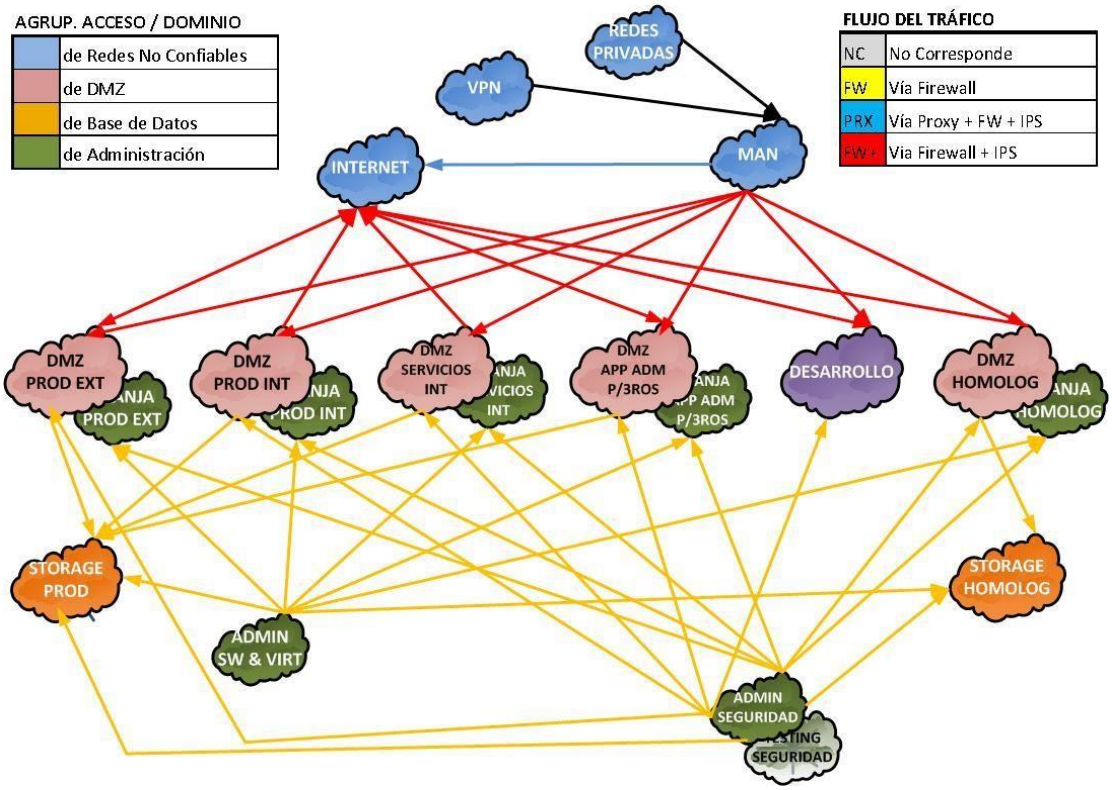
Los servidores del dominio de extranet podrán realizar peticiones exclusivamente hacia los servidores del dominio de servidores que tengan sus bases de datos.

Los servidores de pasarela (gateway) de la extranet tendrán privilegios especiales de acceso, pudiendo acceder a Internet (proxy de salida) y a la Intranet (dispositivos de VPN).

El dominio de Intranet no podrá realizar conexiones hacia los dominios públicos. Para poder acceder a ellos deberán hacerlo a través de las pasarelas de acceso a Internet (proxy de salida).

Los servidores del dominio de Servidores en ningún caso podrán realizar conexiones salientes hacia otros dominios. Podrán recibir conexiones del resto de dominios a través de la interfaz que corresponda

En la siguiente figura se representa gráficamente los diferentes flujos de datos permitidos entre los distintos dominios de la Red del GCABA:



6.4 Entornos

El Data Center de la ASI contará con los siguientes Entornos:

- Entorno de Desarrollo.
- Entorno de Testing.
- Entorno de Homologación.
- Entorno de Producción.

Todos los entornos serán idénticos en términos topológicos y en cuanto a las funcionalidades en ellos implementadas. Todos los ambientes contarán con balanceo de carga.

Ningún Usuario, externo o interno, accederá directamente a servidores que no están en alguna de las DMZ.

La responsabilidad y administración de los entornos mencionados será siempre de la ASI.

6.5 Arquitectura Lógica

Todas las aplicaciones deberán tener claramente separados sus Servidores de Aplicación de sus Servidores de Bases de Datos.

En ningún caso la ASI será responsable por la pérdida de información persistente; es decir archivos temporales, logs, datos no propios del sistema; que las aplicaciones pudiesen generar en los servidores mencionados.

La ASI podrá modificar la configuración del esquema de granjas (cantidad de servidores, mecanismos de persistencia de sesión y balanceo de carga) sin necesidad de dar previo aviso a los Usuarios, a los Desarrolladores o al Personal de Soporte, siempre que esto sea trasparente para los afectados.

Los Servidores de Aplicación no tendrán direcciones IP externas y solo recibirán solicitudes HTTP.

Los Servidores de Aplicación que formen parte de una Granja serán entre sí totalmente independientes, es decir que no emplearán ningún mecanismo de acoplamiento entre ellos.

Como mecanismo de balanceo de carga se utilizará la metodología RoundRobin.

Para establecer la comunicación entre los Servidores de Aplicación, las Bases de Datos, el sistema de gestión de documentos y otros servicios tales como el E-Mail, deberán utilizarse únicamente los puertos estándar definidos para dichos servicios.

En el caso de las Bases de Datos se otorgará a la aplicación un Usuario de Base de Datos con los mínimos permisos necesarios para ejecutar las transacciones. Las Bases de Datos de todos los Entornos serán creados por la ASI.

Todas las aplicaciones que tengan que hacer referencia a un servidor deberán hacerlo por su nombre DNS, jamás por su dirección IP. La relación entre nombres y direcciones IP podrá ser cambiada por la ASI tantas veces como sea necesario sin aviso previo a los Usuarios, a los Desarrolladores o al Personal de Soporte, siempre que esto sea trasparente para los afectados.

Para evitar posibles problemas de compatibilidad entre los diferentes Entornos, la ASI pondrá a disposición de quienes lo requieran, las imágenes virtuales de las configuraciones homologadas.

La ASI será la responsable de administrar las configuraciones de los Servidores de todos los Entornos (Desarrollo, Testing, Homologación y Producción).

El espacio en dónde correrán las aplicaciones será definido por la ASI. El servidor físico o virtual en el que correrá podrá ser un espacio compartido con otras aplicaciones.

Todas las aplicaciones correrán en la modalidad 7x24, sin ventana de mantenimiento. En todos los casos los procesos serán ejecutados sin interrumpir la prestación de servicios interactivos o servicios Web.

6.6 Servicios

El Data Center de la ASI dispondrá de un conjunto de Servicios que los Desarrolladores podrán utilizar para implementar sus aplicaciones.

Los servicios disponibles en el Data Center de la ASI son los siguientes:

- Motor Bases de Datos
- Sistema Operativo
- Gestión Documental
- Correo Electrónico
- Protocolo de Sincronización de Relojes

- Documentos en Formato Portable y Firma Digital
- Mensajería Instantánea
- Servidores de Dominio por Entorno (Servidores Web / Servidores de aplicación Java / Servidores de aplicación .net)

6.7 Logs

Los sistemas a implementar en el Data Center de la ASI deben prever la creación de logs. Los mismos serán generados y serán almacenados remotamente en el repositorio de Logs, contenido en un servidor dedicado exclusivamente a dicha función y administrado por la ASI.

Los sistemas no deberán tener ningún archivo de configuración, ni comando alguno o transacción que permita deshabilitar la existencia de estos Logs.

6.8 Componentes de Aplicaciones y Servidores

Cualquier paquete de software que se instale en los servidores de producción asociado a una aplicación y en adición al software de base provisto por la ASI, será considerado parte de la aplicación, siendo el Desarrollador responsable de su buen funcionamiento y compatibilidad con el software de base definido como estándar por la ASI para sus servidores.

El desarrollador es también responsable de garantizar la compatibilidad de su software con otros aplicativos que pudieran correr simultáneamente en los mismos servidores de producción, sean estos físicos o virtuales.

Los paquetes de software a ser instalados como requerimiento para la ejecución de las aplicaciones, deberán ser siempre versiones estables y que correspondan a la distribución del sistema operativo estándar que se encontrará en los Servidores de Producción. Bajo ningún concepto se aceptarán versiones “beta” de ningún paquete de software.

En el caso que sea necesario instalar paquetes compilados, se deberá declarar y documentar qué paquetes se instalarán, manteniendo el Desarrollador la responsabilidad de actualizar dichos paquetes. En estos casos, los paquetes compilados serán considerados parte de la aplicación debiendo suministrarse pre-compilados e integrados en ella. Al momento de solicitar el pase del sistema al Entorno de Producción, se deberá informar qué librerías externas utiliza la aplicación, de tal forma que puedan integrarlas a una Base de Datos con el nombre de la librería, la versión, el sistema que depende de la librería, y los servidores donde se encuentra instalada.

La ASI cruzará las vulnerabilidades publicadas periódicamente con la matriz de dependencias y alertar qué servidores deben ser actualizados.

Los desarrollos que hayan sido realizados integrando componentes, módulos o sistemas de cualquier tipo sujetos a licenciamiento, deberán ser entregados por el Desarrollador conjuntamente con la documentación que acredite la legítima propiedad o derecho de uso de dichas licencias.

6.9 BackUps

La ASI efectuará un backup de los servidores de aplicaciones cada vez que se modifique un programa (aplicación), el software de base (sistema operativo, etc.) o un archivo de configuración. Solo se realizarán backups de los servidores de aplicaciones si se verifica alguna de estas condiciones.

En función de esta condición, se deberá tener en cuenta que no se debe almacenar ninguna información relevante en los servidores de aplicaciones. Solo podrán almacenarse en el servidor de aplicación los logs de debug y archivos temporarios que pierdan su valor al terminar la transacción que los genera.

Las Bases de Datos y repositorios de documentos se persistirán a través de procesos de backup totales y/o incrementales ejecutados con la periodicidad que la aplicación requiera, y que la disponibilidad de la infraestructura tecnológica existente permita. Estos se efectuarán en caliente, sin ventana de mantenimiento. Esto significa que ningún sistema podrá incluir un programa o procedimiento, sea batch o interactivo, que requiera el uso exclusivo de la Base de Datos o la suspensión de la interacción del sistema con Usuarios físicos (transacciones interactivas) o lógicos (Web Services o accesos directos a la Base de Datos)

En otras palabras, todos los sistemas tienen que ser capaces de operar con todas sus funcionalidades las 24 horas del día, los 7 días de la semana, sin ventana de mantenimiento.

6.10 Versiones y Herramientas aceptadas por la ASI

A continuación, se incluyen los versionados de las herramientas aceptadas para un proceso de Homologación de las aplicaciones en el GCABA.

Lenguaje	Versiones Homologadas
Python	3.9.16 | 3.11.4
Java, OpenJDK	11.0.23 | 17.0.11 | 21.0.3 LTS
NodeJS (1)	18.19.1 | 20.11.1
PHP	8.0.30 | 8.1.27 | 8.2.13
.NET	6.0.129 | 8.0.104
Ruby	3.0.6 | 3.1.4

(1) Se reemplaza a Yarn. por NPM (Node Package Manager) como sistema de gestión de paquetes, integrado a Node.

Motor BD	Versiones Homologadas
Oracle	19c
PostgreSQL	12.18 | 13.14 | 14.11 | 15.6
MariaDB	10.5.24 | 10.6.17 | 10.11.7 | 11.2
MongoDB	6.0.15 | 7.0.9 | 7.2.2
Elastic Search	7.x | 8.x
Redis	Para VM's: 6.2.7-1.module+el8.7.0 +15197+cc495aeb Para Openshift: 6.4.2-4.0 | 7.0.12
Apache Solr	8.x | 9.x
SQLite	3.43.2

Framework Backend	Versiones Homologadas
Nest.js	9.4.0 | 10.0.0
Django	>4.2.9
Laravel	10.16.1 | 11.4.x
Symfony	5.4.31 | 6.4
Express.js	4.19.2
Flask	2.3.2
Spring	>= 6.0.16 | >= 6.1.3
Struts	>= 6.3.0.2
Ruby on Rails	6.1.7.8 | 7.0.8.4 | 7.1.3.4
Codelgniter	4.5.x
ASP.NET MVC	5.3.0
Quartz.NET	>= 3.9.0

Framework Frontend	Versiones Homologadas
React	18.0.0 | 18.1.0 | 18.2.0
Angular	16.2.9 | 17.1
Vue	>= 3.4.28
Next.js	14.1
Nuxt	3.5.3
Ionic	7.8.6 | >= 8.2.4

Librerias	Versiones Homologadas
core	1.8.x | 1.9.x | 1.10.x | 1.11.x | 1.12. x
material-components	1.5.x | 1.6.x | 1.7.x | 1.8.x | 1.9.x | 1.10.x | 1.11.x
Data binding	3.5.0
ViewPager2	1.0.0
Navigation	2.5.x | 2.6.0
WorkManager	2.8.0
RXJava3	3.x.x
Camerax	1.2.x | 1.3.0
Retrofit	2.7.x | 2.8.x | 2.9.x
gson	2.9.x | 2.10.x
okhttp3	4.10.x | 4.11.x | 5.0.x
dagger hilt	2.37.x | 2.38.x | 2.39.x |2.40.x | 2.41.x | 2.42.x | 2.43.x | 2.44.x | 2.45.x | 2.46.x | 2.47.x
Room	2.4.x | 2.5.x | 2.6.x
Livedata	2.4.x | 2.5.x | 2.6.x
paging-runtime	3.0.x | 3.1.x | 3.2.x
coroutines (Kotlin)	1.5.x | 1.6.x | 1.7.x
Maps	18.0.x | 18.1.x

Librerias	Versiones Homologadas
Places	2.6.0 | 2.7.0 | 3.0.0 | 3.1.0 | 3.2.0
Multidex	2.0.1
Lottie	3.7.x | 4.0.x | 4.1.x | 4.2.x | 5.0.x | 5.1.x | 5.2.x | 6.0.x | 6.1.x
Volley	1.2.x
firebase-core	18.0.x | 19.0.x | 20.0.x | 20.1.x | 21.0.x | 21.1.x
Junit	4.13.x
Picasso	2.8
Glide	4.12.x | 4.13.x | 4.14.x | 4.15.x
Zxing	4.3.0
spring-boot	2.5.x - 3.1.x
spring-boot-starter-parent	2.5.x - 3.1.x
spring-boot-starter-web	2.5.x - 3.1.x
spring-boot-starter-test	2.5.x - 3.1.x
spring-boot-starter-web-services	2.5.x - 3.1.x
spring-boot-starter-data-jpa	2.5.x - 3.1.x
spring-boot-starter-validation	2.5.x - 3.1.x
spring-boot-starter-security	2.5.x - 3.1.x
Lombok	1.18.x
springfox-swagger2	2.10.x | 3.0.x
springfox-swagger-ui	2.10.x | 3.0.x
Modelmapper	2.4.x | 3.1.x
commons-codec	1.15
commons-lang3	3.12.x
ojdbc8	21.5.x - 23.2.x

Librerias	Versiones Homologadas
Hsqldb	2.7.1
flyway-core	8.3.x | 9.18.x
dozer-core	6.5.2
Okhttp	5.0.x
unirest-java	4.0.x
mockito-core	4.3.x - 5.3.x
junit-api	5.8.x - 5.10.x
Poi	5.0.x - 5.2.x
poi-ooxml	5.0.x - 5.2.x
aws-java-sdk-core	1.12.x
aws-java-sdk-s3	1.12.x
spring-boot-starter-log4j2	2.5.x - 3.1.x
@angular/animations	16.2.9 | 17.1
@angular/cdk	16.2.9 | 17.1
@angular/common	16.2.9 | 17.1
@angular/compiler	16.2.9 | 17.1
@angular/core	16.2.9 | 17.1
@angular/forms	16.2.9 | 17.1
@angular/platform-browser	16.2.9 | 17.1
@angular/platform-browser-dynamic	16.2.9 | 17.1
@angular/router	16.2.9 | 17.1
@angular/pwa	16.2.9 | 17.1
@fullcalendar/core	5.11.x - 6.1.x
core-js	3.24.x - 3.30.x

Librerias	Versiones Homologadas
dotenv-flow	3.2.0
file-saber	2.0.5
Helmet	4.6.x - 7.x
jwt-decode	3.1.2
ng-swagger-gen	2.3.1
ngx-captcha	10.x - 13.x
ngx-extended-pdf-viewer	11.x - 17.x
ngx-permissions	13.x - 16.0.x
ngx-spinner	13.x - 16.0.x
primeflex	3.1.x - 3.3.x
Matplotlib	3.6.1
Bokeh	2.4.3
NumPy	1.22.x - 1.25.x
SciPy	1.8.x
Pandas	1.1.x
TensorFlow	2.x
PyTorch	2.x
NLTK	3.8.x
Gensim	4.x
SpaCy	3.1.x
Pillow	7.x
Scrapy	2.1.x
Castle Project	5.1.x
Log4net	2.0.x
NHibernate	5.4.x

Librerias	Versiones Homologadas
NUnit	3.13.x
NServicesBus	8.0.x
json.net	13.0.x
AutoMapper	12.0.x
RabbitMQ	3.11.x
Polly	7.2.x
NLog	5.1.x
FluentValidation	11.5.x
MimeKit	3.6.x
Hangfire	1.7.x
Openlayers.js	7.4.0
Moment	2.30.0
Jquery	3.6.4 | 3.7.0
Jquery UI	1.13.2
Jquery YUI	3.18.0
Jquery Migrate	3.4.1
Jquery Validation Plugin	1.19.5
CKEditor	4.21.0 | 5.35.0.1
Lodash	4.17.21
Chart	3.9.1 | 4.3.0
Anime	3.2.x
express-fileupload	1.4.3
Handlebars	4.7.8
Underscore	1.13.2 | 1.13.6
TinyMCE	5.10.8 | 6.7.1

Librerias	Versiones Homologadas
PrimeNg	14.x | 15.x | 16.x
Backbone	1.4.1
Highcharts	9.3.3 | 10.3.3 | 11.1.0
Modernizr	3.13.0
Select2	4.0.13
BusyBox	1.36.1
Bootstrap	5.3.0
Bootstrap-select	1.13.18
react-redux	6.x | 7.x | 8.x
Open CV	4.8.0
Werkzeug	3.0.1
Fancy Box	3.5.7
apexcharts	3.37.0 | 3.37.1 | 3.40.0 | 3.41.0 | 3.41.1
ZK	9.0.1.3 | 9.5.1.4 | 9.6.4
Componente Jose del package.json de Npm	2.0.7 | 4.15.5 | 5.2.3
Libpng	1.6.39
Zlib	1.2.13|1.3.1

CMS	Versiones Homologadas
Wordpress	6.2.2
Drupal	10.0.8
LMS (Learning Management System)	**Versiones Homologadas**
Moodle	4.1.6 | 4.2.3

DMS	Versiones Homologadas
Ckan	2.10.4
GeoServer	2.23.5 | 2.24.2

Herramientas	Versiones Homologadas
RubyGems	3.4.13
Apache Web Server (Aplicaciones Web que no usan httpd24)	2.4.6-45
Apache Web Server (Aplicaciones Web que utilizan httpd24-httpd)	2.4.53-7.el9
JWS (JBoss Web Server)	5.7 SP8 | 6.0 SP1
Apache Tomcat	10.1.11
Nginx	1.22.1
OpenSSL	1.1.1x / 3.0.13 / 3.1.15
PrimeKey EJBCA	7.6.0
Apache Cordova Android	11.0.0

7 Plataformas de desarrollo

El GCABA contempla diferentes tecnologías actuales que se pueden utilizar en función de las características del sistema a construir. Una vez que se cuenta con la claridad de la funcionalidad necesaria, la exposición requerida y la demanda, resulta necesario contar con el tiempo para decidir qué tipo de arquitectura es la indicada para lograr el objetivo.
Se deja constancia que existen principios generales que rigen a todo desarrollo o customización que nos lleva implementar las buenas prácticas de software, orientadas a aplicaciones óptimas en funcionamiento y mantenimiento.

7.1 Principios de Desarrollo

A Nivel General

G1. Todos los desarrollos deben utilizar las herramientas y versiones homologadas por la Agencia (Ver Estándar de Arquitectura).

G2. Aplicar todas las buenas prácticas del mercado aplicable a cada tecnología.

A Nivel Diseño

D1. Todas las aplicaciones que requieran interactuar con el ciudadano deben contar con la autenticación única de ciudadano del GCABA.
D2. Toda autenticación en las aplicaciones del GCABA deben contar como único acceso el Active Directory de la Agencia.
D3. Uso de la programación orientada a objetos (POO) implementando una codificación de alto nivel y buenas prácticas.
D4. El código debe ser Responsive para adecuarse a cualquier tipo de dispositivo que visualice la aplicación.
D5. Toda búsqueda o carga de direcciones en un frontEnd tiene que validarse y normalizarse con la opción catastral del GCABA
D6. Las visualizaciones georeferenciales de objetos deben utilizar el Mapa del GCABA
D7. Si la aplicación gestiona archivos (PDF, DOC, PNG, JPG, etc.), la misma se tiene que almacenar en el Storage standard del GCABA. No está permitido guardar en forma permanente archivos en forma local. Para los casos temporales la destrucción de los mismos debe ser forma inmediata.
D8. Los servicios deben estar segurizados con sistema de token.

A Nivel Programación

P1. Utilizar patrones de diseño estándares, por ejemplo: Singleton, Visitor, Strategy, Adapter, Lazy Loading.
P2. Implementar un estilo de programación unificado y estandarizado, respetando la indentación, el nombrado de clases y variables, declaración de variables y comentarios que describan el código.
P3. Disponer de una capa de servicios core API-REST que permita integrarse con otros sistemas del Ecosistema del GCABA
P4. Las validaciones se realizan con control duplicado, por FrontEnd y BackEnd.
P5. Todo componente adicional utilizado en código debe ser validado y acordado.

A nivel Control

C1. El volumen de información transferida deber estar controlado para asegurar el buen funcionamiento de la aplicación.
C2. Pruebas unitarias para dar cobertura al código
C3. Los procesos de auditorías deben contar con la forma de depurarlos teniendo en cuenta las fechas de registración desde un BackOffice y con accesos restringido.
C4. Los procesos batch deben ser parametrizables desde un BackOffice y deben estar preparados para funcionar con una infraestructura de alta disponibilidad y escalable.

A nivel Mantenimiento

M1. Todo sistema tiene que contar un logueo estándar en base a la tecnología seleccionada con errores, alertas y un registro de transacciones core que permita medir la salud de las mismas. El nivel de logueo debe poder parametrizarse sin requerir un despliegue.
M2. Las configuraciones de las variables relacionadas con el entorno (BD, Servicios externos, urls, protocolos, paths, etc.) de la aplicación deben estar en un archivo externo al código, teniendo en cuenta la tecnología utilizada. Por ejemplo, debe estar preparada para funcionar bajo protocolo https (protocolos relativos)
M3. Toda aplicación debe documentar la arquitectura en detalle (diagrama de contexto, componentes, integraciones detalladas, tecnologías, Procesos, dimensionamiento, etc.),

Alcance funcional, no funcional y toda documentación que complemente y sea útil para el skill transfer y mantenimiento

Atento a ello, a continuación se detalla cada opción aceptada, con su justificación marco de implementación y requisitos.

7.2 Arquitectura orientada a Servicios (SOA)

Para los casos de complejidad media y alta, que requieren integración con otros sistemas del universo GCABA, que demanden escalabilidad, robustez, con alto volumen de transacciones y alta disponibilidad, serán candidatos para este tipo de arquitecturas.
En situaciones a implementar necesidades críticas, se seleccionará esta opción con los requisitos que se definen a continuación:

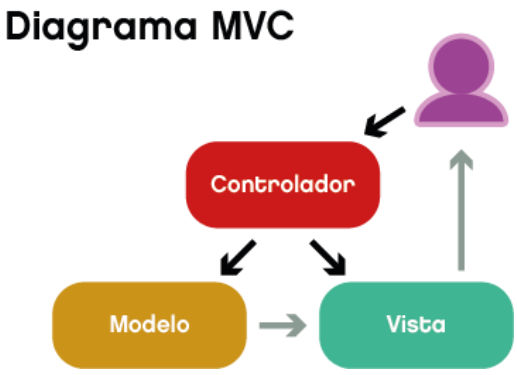
- Cumplir con los principios de diseño de servicios
- Utilizar una arquitectura REST siguiendo el estándar Rest Api URI (Ver anexo "Reglas de ApiRest"), con Mensajes de datos en formato JSON y tienen que documentarse con SWAGGER o RAML.
- Tecnologías:

  JAVA SPRING BOOT: Java JDK, Architecture API Rest, JSON (Message format), Spring suite de tecnologías: BOOT, AOP, Core (IoC), MVC, Test., WebServer Tomcat embebido, Apache Camel con Spring Boot, Hibernate ORM, API JPA, JMS, AMQ, Apache Log4j, logback, Apache Maven TDD, JUnit

  NODE.JS: FrameWork Express, Architecture API Rest, JSON (Message format)
- Para la capa de **FrontEnd** : Angular.js o React.js,HTML5 y CSS3, utilizando el framework BAStrap (https://gcba.github.io/BAstrap/)
- Para La capa de Base de Datos se puede utilizar como gestor de BD ORACLE.

7.3 Arquitectura MVC

Ante una necesidad de baja complejidad, con baja demanda de volumen y de criticidad se puede optar por una arquitectura cliente servidor que cumpla con el patrón MVC.
Para esta selección debe estar específicamente definido que no existe una necesidad de escalar en ninguna de los criterios básicos mencionados.



En esta opción los requisitos definidos son:

- Utilizar el lenguaje de programación PHP con los frameworks Laravel
- Los componentes permitidos para el **FrontEnd** son: JavaScript, JQuery, Ajax, HTML5, CSS3, utilizando el framework BAStrap (https://gcba.github.io/ BAstrap/)
- Para La capa de Base de Datos se puede utilizar MariaDB con el ORM correspondiente al FrameWork.

7.4 Arquitectura CMS

En los casos que exista la necesidad de contar con contenidos, con cierta capacidad de actualización en forma independiente y dinámica es factible utilizar una plataforma CMS y en particular con DRUPAL se satisface la demanda.

Las actualizaciones de software deberán realizarse a través de comandos Drush en todos los casos donde sea factible.

El código de los módulos custom deben seguir los estándares de calidad y estilos de Drupal (https://www.drupal.org/docs/develop/standards). Los módulos instalados deberán ser los estrictamente necesarios para el funcionamiento del sitio no debiendo existir módulos que no cumplan ninguna función o hayan quedado en desuso. Las modificaciones de esquema que se realicen sobre la base de datos deben ser efectuadas a través de las features creadas a tales efectos.

El CMS debe ajustarse para obtener un sitio de alta performance y escalable, utilizando Internal Page Cache module, Content Delivery Network (CDN), Memcache, Varnish y demás funcionalidades recomendadas por la comunidad drupal en https://www.drupal.org/docs

7.5 Plataforma Cloud

En el caso de seleccionar un desarrollo en una plataforma Cloud deberá estar justificada la necesidad a cumplir y los beneficios que tiene esta opción sobre una tradicional. Adicionalmente realizar un análisis a corto y mediano plazo del costo.
En general si necesitan mover o intercambiar información con alguna aplicación existente, que no se encuentre en cloud, requiere un análisis técnico de las integraciones necesarias y debe estar validado y aceptado por la Agencia de Sistemas.
Recomendamos contemplar al momento del diseño la posibilidad de que sea portable a otras plataformas.

Ante una a necesidad cubierta por un producto exclusivo en estas plataformas, el requisito para utilizarlas es que solamente se deba customizar adicionales mínimos y no se pierda la esencia del producto seleccionado en el momento de implementar funcionalidades o procesos.

8 Requisitos de Integraciones Generales

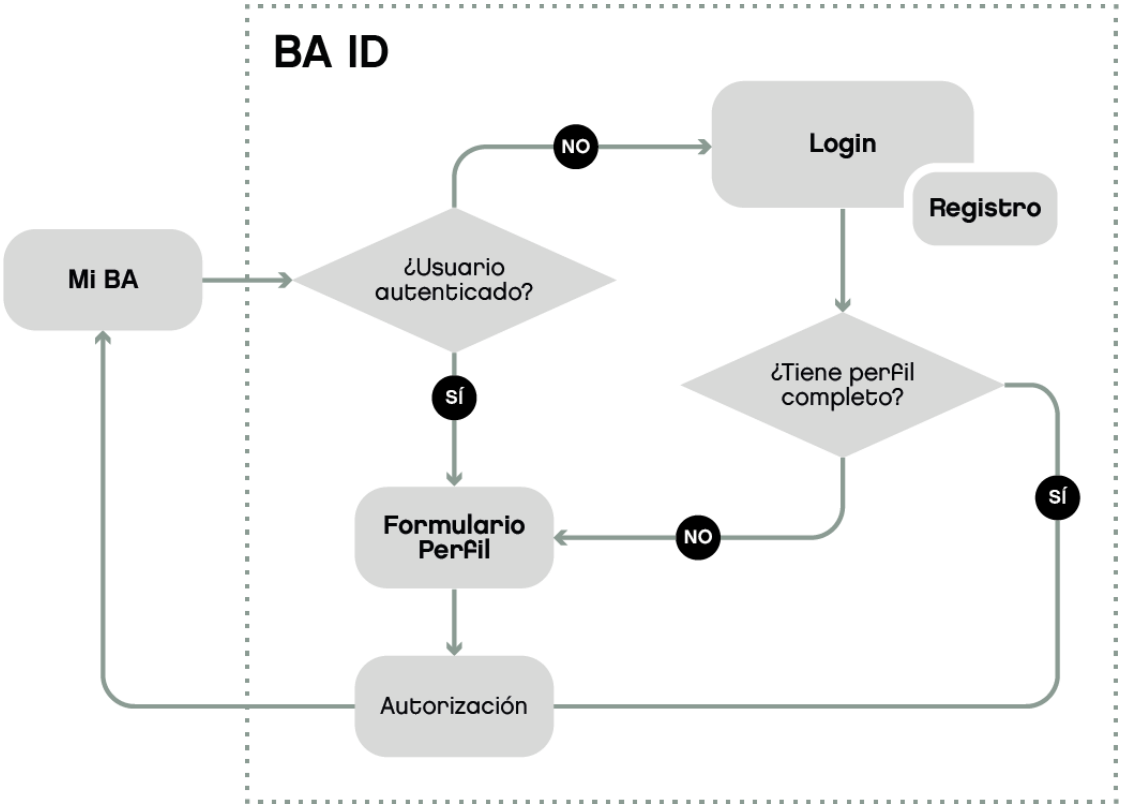
A continuación se definen los requisitos de integraciones generales de las aplicaciones. Ante una nueva necesidad o nuevo concepto a incorporar que no se encuentre incluido en los detallados, deberá

atravesar un proceso de selección y autorización técnica para evaluar la factibilidad de estandarizar o otorgarle un tratamiento individual.

8.1 Autenticación

Toda aplicación que genere una interacción con el ciudadano debe contar con la autenticación con BAID en su Front End.

Mi BA es el panel personal donde se centraliza toda la información de la Ciudad de Buenos Aires, obtenida a través de distintos servicios del Gobierno y facilita al ciudadano las herramientas necesarias para realizar y consultar trámites, reclamos, sacar turnos para hospitales y otros servicios, consulta y pago de patentes, infracciones, ABL, entre otros. . Lo expuesto, es posible gracias a que tiene implementado un sistema de autenticación único para el ciudadano, permitiéndole acceder a múltiples servicios internos del Gobierno con una única identificación. Dicho sistema está basado en el protocolo de OpenID Connect, dado que permite que MIBA sea proveedor del servicio para cualquier aplicación web o móvil de Gobierno que quiera incorporarse. El proceso de identificación que utiliza MIBA está pensado como una estrategia para ser el único proveedor de servicios de autenticación de Gobierno.



MIBA provee el Servicio de Autenticación siguiendo las especificaciones de OpenID Connect.
Toda aplicación o autenticación que no sea de directa al ciudadano debe pasar por el Active Directory del GCABA. (Ver Estándar de Seguridad)
Los permisos y roles en el marco de la aplicación deberán asignarse desde un BackOffice de la misma.

8.2 Georeferenciación

Las aplicaciones deben contar con direcciones de calles normalizadas y validadas a través del servicio de PDI disponible en el catálogo.

Adicionalmente, para el caso de visualización de direcciones en mapa, debe utilizarse el mapa del GCABA.

8.3 File System

Todo documento adjunto que necesite guardar una aplicación debe resguardarse en el repositorio estándar para tal fin del GCABA, respetando las condiciones mínimas y necesarias. La tecnología actual está basada en el protocolo S3.

Debido a la forma en que HCP almacena objetos, las carpetas que crea y la forma en que almacena los mismos en ellas pueden generar un impacto degradante en su rendimiento, se brinda a continuación algunas pautas para crear estructuras de carpetas efectivas:

- Planificar la estructura de carpetas antes de almacenar objetos.
- Evitar estructuras que provoquen que una sola carpeta obtenga una gran cantidad de tráfico en poco tiempo
- Si se almacena objetos por fecha y hora, tener en cuenta la cantidad de objetos durante un período de tiempo determinado al planificar la estructura de carpetas. Por ejemplo, usar una estructura de carpetas como año / mes / día / hora / minuto / segundo.
- Seguir las siguientes pautas sobre el tamaño de la estructura de carpetas:
    - Intentar equilibrar el ancho y la profundidad de la estructura de la carpeta.
    - No crear estructuras de carpetas que tengan más de 20 niveles de profundidad.
    - Evitar colocar una gran cantidad de objetos (más de 100,000) en una sola carpeta. En su lugar, crear varias carpetas y distribuir uniformemente los objetos entre ellas.

8.4 Reporting

Las aplicaciones deben contar con la mínima capacidad de generar reportes de la operativa diaria exportables a PDF y Excel, teniendo en cuenta criterios mínimos de filtros que acoten el volumen y sean performante.

La capa analítica no se encuentra dentro del alcance del desarrollo de los sistemas, a no ser que se especifique lo contrario. La capa analítica la GCABA la resuelve a través de SAC (Sap Analitic Cloud)

El diseño del modelo de datos debe contemplar, facilitar la exportación y explotación a través de SAC o con las mismas características.

8.5 Base de Datos Documental

Se recomienda utilizar una base de datos documental, en proyectos que requieran alta escalabilidad, los cuales tienen características fundamentales como: alta velocidad en los tiempos de respuesta, la gestión de gran volumen de contenidos y variabilidad donde cada documento puede almacenar campos distintos, pudiendo ser flexibles en cuanto al esquema de la información. En ningún caso puede utilizarse como BD relacional y en sistemas transaccionales. La BD documental homologada es MongoDB.

8.6 BPM

Las aplicaciones que necesiten flujos de procesos de negocio dinámicos deben gestionarse con una herramienta de WorkFlow del mercado. La herramienta de BPM aceptada por el GCABA es JBPM2.

8.7 Motor de Reglas (ME)

Para la automatización de la gestión de procesos muy variables es recomendable utilizar un motor de Reglas donde se definan las condiciones en forma dinámica. El motor de reglas aceptada es Drools.

8.8 ChatBot

Ante la necesidad de contar con un chatbot que genere un servicio al ciudadano, es necesario utilizar el chatBot del GCABA (BOTI).

9 Metodología de Desarrollo e Integración continua

Todos los proyectos que contemplen un desarrollo a medida o proyectos de software deben guiarse con la metodología SCRUM, respetando los principios del proceso de desarrollo Agile en proporcionar software de trabajo en incrementos más pequeños y más frecuentes, en oposición al enfoque de "big bang" del método de cascada.
Esto se complementa con un flujo de trabajo al estilo de “DevOps”, cuyo objetivo es aumentar la tasa de cambio, como así también desplegar funciones exitosamente en producción sin causar caos e interrumpir otros servicios, a la vez que detecta y corrige incidentes rápidamente cuando ocurren.

En resumen con equipos de desarrollo ágiles, sumado a la integración continua y la entrega continua, se obtienen resultados más eficientes.

10 Despliegue Continuo

Todas las aplicaciones a implementarse en GCABA deben contemplar un proceso automático de despliegue que permita agilizar la llegada a producción con la menor interacción manual posible. Alineado al concepto de Integración Continua implementado en GCABA.

Para esto debe cumplir con los estándares para la implantación en el datacenter local y Cloud.

Ver Anexo de Despliegue continuo.

11 Especificaciones No Funcionales

Los requerimientos no funcionales (RNF) son restricciones de los servicios o funciones ofrecidas por el software. Incluyen restricciones de tiempo y recursos, sobre el proceso de desarrollo, estándares, etc.
Son aquellos requerimientos que no se refieren directamente a las funciones específicas del sistema, sino a las propiedades emergentes de éste como la fiabilidad, la respuesta en el tiempo y la capacidad de almacenamiento.
Muchos RNF se refieren al sistema completo y no tanto a rasgos particulares del mismo. Esto significa que muchas veces pueden resultar más críticos que los requerimientos funcionales (RF) particulares. Mientras que el incumplimiento de este último degradará el producto final, una falla en un RNF puede inutilizar el mismo.
Surgen de la necesidad del usuario, debido a las restricciones en el presupuesto, a las políticas de la organización, a la necesidad de interoperabilidad con otros sistemas de software o hardware o a factores externos como los reglamentos de seguridad, las políticas de privacidad, etcétera.
En general, los RF definen lo que un software se supone que debe hacer, mientras que los RNF definen cómo un software se supone que es.

A continuación se detallan algunos requisitos a cumplir:

Clase	Requisito
Performance	Cuando se necesite acceso rápido a información con baja probabilidad de cambios y empleada con mucha frecuencia se debe implementar sistemas de caché de contenido y de datos. Esto evita realizar invocaciones innecesarias que degradan la performance del aplicativo. Para servir contenido cacheado (html, css, imágenes o cualquier contenido estático) deberá emplearse el acelerador http Varnish. Para el caché de datos deben utilizarse los servicios de Memcached o Redis provistos por la agencia. Es responsabilidad del proveedor el mantenimiento de los archivos de configuración de Varnish (definición y actualización).
Disponibilidad	La aplicación debe ser 7x24 sin ventana de mantenimiento, sin interrumpir la prestación de servicios interactivos o Web Services. En caso de contar con la necesidad de bajar la aplicación por un tiempo determinado, debe contemplar desde la administración del usuario que un administrador pueda realizar una baja controlada del servicio que permita visualizar a los usuarios un aviso de la situación.
Escalabilidad	Se debe proveer escalabilidad horizontal en el sistema a desarrollar teniendo en cuenta las especificaciones realizadas en el Documento de Arquitectura, como ser la integración con balanceador de capa 7, Application servers granjeables y stateless.
Seguridad	Ver Estándar de Seguridad (ES0902) que complementa este documento
Robustez	Los distintos servicios, Application servers, base de datos, etc. deben estar desacoplados, como se encuentra especificado en el Documento de Arquitectura
Usabilidad	• Estética: Los sitios web de la Ciudad de Buenos Aires deben tener una concordancia estética homogénea que se corresponde con la marca integral del Gobierno de la Ciudad. Por este motivo, los diseños web ya sean de aplicaciones accedidas desde la intranet o desde internet deben tener la conformidad de comunicación. • Responsive: Todo Producto que fuera accedido por un ciudadano debe cumplir con la condición de ser Responsive Web Design (RWD) de tal forma que la apariencia se adapte a diferentes canales tecnológicos como tablets, smartphones, portátiles, etc.
Hardware y Software	Los requerimientos de Hardware y Software de las aplicaciones deben encontrarse dentro de las especificaciones de cuadro de tecnología especificado en este pliego. El dimensionamiento del Hardware es responsabilidad del proveedor y debe ser especificado durante la etapa del proyecto. Es necesario comunicar al GCABA con anticipación para disponibilizar el mismo y verificar el correcto uso.
Mantenibilidad	• Administración de Errores: La aplicación debe evitar enmascarar los mensajes de error http de forma tal que los balanceadores / proxy puedan entender el estado actual del aplicativo. Ante un error inesperado no debe visualizarse por front/backend/api características del servidor, ip, path, o cualquier información que permita conocer propiedades de infraestructura. Por ejemplo, si ocurre un error en el aplicativo la respuesta http de estado debe ser código 500 y el mismo no debe reemplazarse por ningún otro código.

Clase	Requisito
	• Servicio de Monitoreo: Las aplicaciones deben implementar el patrón "healthcheck" para chequear conexiones con la infraestructura y salud de los servicios core. • Despliegues: Las aplicaciones deben contemplar un proceso de despliegue automático para sus actualizaciones. Ver Anexo "Despliegue continuo."
Interoperabilidad	La ASI aspira a maximizar la interoperabilidad entre los sistemas del GCABA buscando que cada sistema se atenga a la resolución de los problemas propios del dominio en que actúa. Al mismo tiempo, es habitual que las áreas de gobierno interactúen y, por lo tanto, haya interdependencia de datos. Se espera las aplicaciones expongan sus servicios del Gateway del GCABA
Depuración	Las aplicaciones deben contemplar mecanismos de archivado de datos internamente, con el fin que durante un uso prolongado en el tiempo los acceso a Base de Datos no se vean afectados. Los datos archivados deben tener mecanismos de acceso opcionales y controlados desde las aplicaciones y no deben generar degradación en el servicio.
Auditoria	Se deben aplicar los conceptos vertidos en la sección "Uso de Logs y monitoreo" del espacio en Confluence de la Agencia cuya url es https://asijira-confluence.buenosaires.gob.ar/display/ASI Las aplicaciones deben capturar las excepciones y dejar el registro siguiendo el estándar de la tecnología utilizada y permitir que el nivel de logueo sea configurable. El GCABA utiliza el stack tecnológico ELK (Elastic Search, Filebeat/LogStash, Kibana) para centralizar y generar indicadores de la salud de la aplicación.

12 Catálogo de Servicios estándar

La Agencia posee un catálogo de Servicio disponible para evaluar servicios generales y comunes a necesidades generales de las aplicaciones. Todos los servicios que se detallan se encuentran implementados en el módulo integrador, garantizando la seguridad, control, robustez y documentación. En el caso que una aplicación posea la necesidad de integrarse con otra aplicación, sea para consultar, o para disponibilizar un servicio debe atravesar el integrador, no es aceptable integraciones directas.

Adicionalmente, al contar con el servicio disponible por parte de la ASI, durante la fase del proyecto debe existir un acuerdo y permisos de cómo se va a utilizar entre las partes referentes la información que se integran. En algunos casos implica costos que hay que contemplar o acuerdos con terceros.

Este catálogo detalla la lista de los servicios existentes, como usarlos y el alcance de cada caso:

Integración con Renaper
Integración con AD
Integración con Catastro
Integración con Gestor Documental (SADE)
Integración con Plataforma de Pago
Integración con Plataforma de Correo
Integración con SMS

Integración con Servicios de Almacenamiento de Datos Seguros

Haga click aquí para acceder al catálogo disponible. En el caso que alguno no se encuentre visible aquí debe realizar las gestiones durante la etapa del proyecto.

13 Condiciones de aceptación

Cada entregable pasará por un proceso de certificación, dependiendo el mismo, el responsable del GCABA podrá ser diferente y deberá será acordado en la etapa de planificación del proyecto.

Todos los entregables con excepción de "**Paquete de Software"** tienen un límite de tiempos estipulado de 10 días hábiles. Al respecto, previa solicitud, el GCABA podrá extender dicho plazo, justificando la demora.
En particular el entregable "**Paquete de Software** será sometido a un proceso exhaustivo de verificación que deberá incluir como parte esencial del mismo el mecanismo para demostrar que la aplicación se comporta:

1) correctamente (tests unitarios y de integración),
2) según lo esperado (tests de aceptación)
3) con un tiempo de respuesta aceptable (tests de performance).

Adicionalmente, se tomarán los recaudos de comprobación en la Infraestructura, Red y condiciones del GCABA teniendo en cuenta los criterios de severidad y prioridad. Dependiendo de los issues reportados con los criterios de mencionados se establece que el software entregado se aprobará cuando exista:

- **Según su severidad:**

  0% incidentes críticos y 0% incidentes mayores. El porcentaje de incidentes menores e incidentes cosméticos permitidos abiertos para dar la aceptación de calidad por parte del GCABA se acordarán durante el transcurso y planificación del proyecto.

- **Según su prioridad:**

  0% incidentes de prioridad urgente y 0% incidentes de prioridad Alta. El porcentaje de incidentes con prioridad normal y baja permitidos abiertos para dar la aceptación de calidad por parte del GCABA se acordarán durante el transcurso y planificación del proyecto.

La severidad puede ser:

Crítica	Mayor	Menor	Cosmética
a) Cuando los usuarios no pueden utilizar las funcionalidades principales del sistema. b) Cuando no es posible realizar algún trabajo productivo. c) Cuando no se puede prestar el servicio a los usuarios.	a) Cuando el sistema está operando pero con restricciones. b) Existe impacto en la prestación del servicio a los usuarios. c) Existe impacto para los usuarios.	a) Cuando los usuarios no pueden utilizar las funcionalidades secundarias del sistema. b) Cuando no se encuentran disponibles algunas funciones o componentes del Sistema, que generan un impacto mínimo para los clientes y para los usuarios. c) Cuando las limitaciones no son críticas para la operación. d) El impacto no genera un riesgo considerable, pero es necesario resolverlo.	El error se refiere a un mal funcionamiento o diseño de la interfaz de usuario, que no impide la correcta ejecución del sistema.

La prioridad indica la precedencia o el orden en el tiempo en que deben solucionarse los errores reportados.

La prioridad puede ser:

Urgente	Alta	Normal	Baja
El Error debe solucionarse inmediatamente dado que no permite continuar, puede requerir una entrega especial.	El error debe solucionarse tan pronto como sea posible dado que se requiere para la próxima salida a producción.	Prioridad media, es deseable la corrección del error para la salida a producción.	Corregir el error sólo si no afecta calendario ni la corrección de otros errores de mayor prioridad o severidad, puede salirse a producción sin corregir.

El paquete de software además del funcionamiento, será sometido a una revisión que compruebe los requisitos exigidos en este estándar.
En el caso del código tendrá foco en:

- Controles estáticos de código con las buenas prácticas del mercado : Libre de código duplicado, Libre de issues y bugs, Cobertura de código superior a 80%, Complejidad ciclomática
- Controles estáticos de OWASP
- Controles dinámicos de calidad
- Selección de muestras de código de servicios o funcionalidades críticas.
- Calidad de los test Unitarios y su cobertura
- Profiling. Se realizará profiling de las aplicaciones construidas para visualizar óptimo consumo de recursos (CPU y RAM).

***IMPORTANTE**: El incumplimiento de la regla de sólo paquetes homologados puede ocasionar el rechazo del entregable y la necesidad de retrabajo a último momento.*

ANEXO: REPOSITORIO DE CONTROL DE VERSIONES

1. OBJETIVO

El objetivo del presente documento es unificar las entregas de paquetes de software en una única herramienta de control de versiones o SCM (del inglés Software Control Management), a través de un proceso y estructura definidos que permitan a las diferentes áreas de la ASI realizar las actividades e intercambios con los proveedores. Es de carácter obligatorio realizar este proceso para llevar a cabo las actividades de control de cambios tanto para aplicaciones nuevas como entregas de nuevas versiones de aplicaciones existentes.

2. AUDIENCIA

La audiencia del presente documento involucra a todas las áreas técnicas de las reparticiones y proveedores de software del GCABA que realizan entregas de software a la ASI.

3. PROCEDIMIENTO

El sistema de control de versiones seleccionado por la **ASI** es **GIT,** el cual pertenece al tipo de arquitectura de repositorios de información distribuidos.

El repositorio para un proyecto dentro de GIT y sus usuarios son creados en la etapa denominada **Setup del Proyecto** en la cual el referente técnico de un proyecto declara a la ASI el inicio de un proyecto de Software.

Todo el código fuente necesario para un proyecto así como también para el esquema, sus migraciones, scripts y todo el material entregable para el proyecto, deberá estar presente en el repositorio GIT. Se exceptúa la documentación del proyecto como los documentos funcionales, arquitectura que debe residir en el Confluence de ASI.

4. ESTRUCTURA DE UN PROYECTO

La ASI generará el repositorio del proyecto y otorgará los acceso en GIT al momento del iniciar el proyecto.

En la carpeta source/ debe estar contenido todo el código fuente de la aplicación y el archivo de configuración de dependencias, el cuál debe listar las mismas especificando el número de versión exacta para cada una de ellas. No se aceptarán paquetes compilados como parte de la entrega.

En el directorio raíz del repositorio deberá encontrarse un archivo README.md el cual debe contener la información necesaria para realizar la primera implementación del aplicativo.

Luego, para cada nueva versión que se entrega es obligatorio sumar al archivo UPGRADE.md, con las instrucciones para llevar a cabo la instalación, indicando la fecha en la que se produjo dicha actualización y los cambios que se realizaron en el archivo CHANGELOG.md a nivel funcional, incluyendo en el mismo lo siguiente:

- Nro. de tickets de bugs/requerimientos/incidentes resueltos (nro. asociado en la Herramienta de seguimiento y control de cambios definidas en el proyecto)

- Funcionalidades incluidas (si corresponde),

- Sprint correspondiente (versión preliminar).

Es obligatorio contar en la carpeta scripts para el caso de utilizar despliegues en máquinas virtuales. Debe contar con los comandos necesarios para efectuar un rollback en caso de que el deploy falle. Dichoscript deberá efectuar tanto rollback de código como de base de datos y configuración. Es responsabilidad del desarrollador mantener la lógica necesaria para evitar que se pierda información valiosa en este proceso. Todos los script deben contar con una forma adicional que verifique y valide que la ejecución es exitosa o no, a modo de validación, como por ejemplo que contenga las instrucciones que logueen en un log que pueda ser rescatado y enviado a los interlocutores.

Adicionalmente, debe ser utilizado el comando de mayor nivel de información (verbouse). En caso de existir integraciones/acoplamiento con servicios externos al paquete a instalar, se debe verificar que la instalación relacionada se comunica correcta o incorrectamente y es obligatorio detallar la forma de integración en el documento de arquitectura.

Si se considera que el despliegue a realizar puede poner en riesgo la información almacenada en la base de datos del proyecto, se deberá realizar un backup de los datos en el README.md y en la solicitud del pedido del pasaje de ambiente.

Los proyectos deberán contar como mínimo con una rama DEV y una MASTER. En DEV se realizarán las entregas parciales y verificaciones implementadas en la herramienta con el fin de contar en forma proactiva un auto examen de calidad. Dicha rama es de uso exclusivo del equipo que construye. Una vez que se cuenta con la versión candidata, en condiciones de avanzar a otro ambiente, se actualiza a la rama MASTER con el TAG correspondiente.

A partir de esta acción se considera la entrega formal y se inicia el proceso de despliegue y seguimiento.

En la rama MASTER se ejecutarán los mismos controles que en DEV y se avanzará con el despliegue si los mismos son satisfactorios.

En el archivo .gitignore dentro del repositorio deberán incluirse los archivos de configuración y todos aquellos archivos que no formen parte del proyecto y que por alguna razón existan en el directorio del mismo (Ej.: archivos de sistema, configuración, archivos subidos por usuarios, etc.)

5. CONTROLES DE CALIDAD

Cada proyecto de Git cuenta con la asociación de los controles Code Quality, SAST, DAST y Dependency Scanning. Dichas revisiones se activan a partir de los commits y tags que se van generando y dependiendo de la tecnología utilizada.

Code Quality proporciona una revisión automatizada del código, está basada en la herramienta gratuita Code Climate Engines.

Static Application Security Testing (SAST) soporta:

Language / framework	Scan tool
.NET	Security Code Scan
C/C++	Flawfinder
Go	Gosec
Groovy (Ant, Gradle, Maven and SBT)	find-sec-bugs
Java (Ant, Gradle, Maven and SBT)	find-sec-bugs
JavaScript	ESLint security plugin
Node.js	NodeJsScan
PHP	phpcs-security-audit

Python	bandit
Ruby on Rails	brakeman
Scala (Ant, Gradle, Maven and SBT)	find-sec-bugs
Typescript	TSLint Config Security

Dynamic Application Security Testing (DAST) está basada en OWASP Zed Attack Proxy (ZAP)

Dependency Scanning soporta:

Language (package managers)	Scan tool
JavaScript (npm, yarn)	gemnasium, Retire.js
Python (pip) (only requirements.txt supported)	gemnasium
Ruby (gem)	gemnasium, bundler-audit
Java (Maven)	gemnasium
PHP (Composer)	gemnasium

El usuario cuenta con los informes de resultados correspondientes accediendo al Dashboard de la aplicación en Git.

6. CONFIGURACION

A continuación, se detalla una guía de los pasos necesarios para vincular un puesto al GIT GCBA:

1. Acceder a la URL que al momento del setup el GCABA entregará. Dicho acceso se encontrará habilitado temporalmente durante el proyecto. En caso de inactividad del usuario en la herramienta (60 días), se procederá a la desactivación por cuestiones seguridad y ante una nueva necesidad, deberá solicitar la reactivación con la Mesa de ayuda de la Agencia.

2. Para vincular el puesto con el proyecto en GIT, se cuentan con dos opciones:

   Generar una Key SSH y vincularla en el profile de GIT.

   Utilizar un cliente GIT tipo UI y autenticarse con el usuario y contraseña de GIT.

3. Clonar el repositorio para obtener la estructura de carpetas Estándar.

4. Realizar el commit de los archivos generados respetando la estructura de carpetas.

7. VERSIONADO DE CODIGO - GITLAB

En la etapa de desarrollo continuo se debe utilizar la rama DEV, y versionar cada commit indicando el ISSUE asociado o un comentario asociado al cambio.

Ejemplo:

```
git commit –m “1.0.1 APP-DEMO-04”
git commit –m “1.0.2 Fix funcionalidad …”
```

Donde “APP-DEMO-04” refleja el ISSUE asociado a la herramienta de Gestión del Cambio. Asimismo, el commit tiene que estar identificado a través de un tag que posea la siguiente nomenclatura “1.0.2.DEV-1”

Ejemplo:

```
git tag “1.0.2.DEV-1”
```

Consiguiente nomenclatura del tag se debe aumentar en el término DEV-x, siendo la x el valor numérico incremental en cada subida a la rama DEV.

Al momento de realizar una entrega en la rama Master se deberá identificar el commit como un Release Candidate de la siguiente forma:

git commit -m <identificación del release candidate>” refleja la intención del Proveedor de entregar un nuevo release -incorporando N cambios sobre la versión anterior- y por lo tanto el mensaje que describe el commit representa dicha intención.
Ejemplo para una aplicación que tiene la versión 3.3 en Producción se incorporan 3 nuevas funciones y 2 cambios (todos detallados en el documento CHANGELOG.MD).
Se realiza el comando:

```
git commit –m “3.4.RC-1”
```

Además, de incrementar el número de versión (en este caso el segundo dígito por tratarse de una modificación menor) se le agrega la identificación “RC” para señalar que se trata de un candidato a ser implementado (release candidate) y con el número #1 porque es la primera entrega luego de una implementación exitosa.

Si durante el proceso de deploy se detecta algún problema que impide llegar a la implementación exitosa y que requiere subir nuevos cambios al código, simplemente se repite la operación pero esta vez el comando será:

```
git commit –m “3.4.RC-2”
```

De esta forma queda documentado un segundo candidato, y así sucesivamente hasta que se obtenga una versión exitosa.

En ese momento, el equipo de despliegues procederá a generar una etiqueta (git tag) con el número de versión reemplazando el “RC-x” por un “.0”, por lo que la versión implementada exitosamente estará identificada así:

```
git commit "3.4.0"
```

Versionado de FIX

Supongamos que mientras el equipo de desarrollo sigue avanzando en el proyecto y entrega la siguiente versión (en nuestro ejemplo, 3.5.RC-1), se produce un problema que requiere la inmediata corrección del código productivo (3.4.0).
En ese caso, se crea una nueva rama (git branch) identificada en la rama FIX con la última versión productiva (3.4.0) y se corrige el problema, subiendo al repositorio el arreglo como:

```
git commit –m "3.4.FIX-1"
```

Una vez que el fix ha sido autorizado e implementado, el equipo de la Agencia genera la etiqueta:

```
git tag "3.4.1"
```

El tercer dígito indica que la versión productiva tiene uno o más arreglos, independientemente de los release candidates que se estén implementando. Estos arreglos podrían ser agregados o no al código en desarrollo, como en el caso de workarounds para problemas que se solucionan justamente con una próxima versión.

8. Consideraciones a tener en cuenta

- El equipo de desarrollo solamente indica las versiones candidatas a despliegue (RC) o arreglos (FIX) en el comentario del "git commit", seguido de un número secuencial correlativo.

- El equipo de Despliegue es quien genera las etiquetas de TAGS para las versiones productivas y arreglos de emergencia.

- La lista de los cambios realizados con respecto a la versión anterior deben estar documentados (ej. CHANGELOG.MD) y pasos a seguir para la implementación del Release Candidate entregado en el archivo UPGRADE.md (ej, ejecución de script .sh, pasos para ejecutar archivo Ansible).

ANEXO: DESPLIEGUE CONTINUO

Los sistemas a implementar OnPremise en GCABA deben adaptarse a la plataforma de delivery continuo Openshift. La misma provee los ambientes de Desarrollo, Calidad, Homologación, Producción Interna y Producción DMZ.

Para poder implementar los desarrollos en la plataforma, se deberá cumplir con los siguientes requerimientos:

- Poseer un archivo de configuración YAML para definir todas las variables de entorno e informar la ruta del mismo en el archivo README.md.
- La base de datos estará fuera de la plataforma Openshift, esto no debe afectar la performance. El host, puerto y esquema deben ser parametrizables.
- No está permitido utilizar almacenamiento local, ya que se utiliza el concepto de containers efímeros.
- En el caso de los framework PHP (Symphony o Laravel) se debe contar con el archivo composer.json en la raiz del proyecto en donde se cuenta con la descripción de librerías asociadas al desarrollo.
- En el caso de los desarrollos NodeJs, se debe contar con el archivo package.json.
- Para aplicaciones con tecnología Java, se debe contar con el archivo pom.xml donde se declare las dependencias y versiones a utilizar.
- Se debe documentar tanto las rutas a exponer como los servicios externos a utilizar.
- Se debe utilizar las herramientas disponibles dentro del framework elegido para versionar la estructura de base de datos evitando utilizar Scripts.

  EJ:

  Framework Symphony (PHP) se utiliza Doctrine.
  Framework Laravel (PHP) se utiliza Artisan.
  Framework Spring (JAVA) se utiiza JPA, Hibernate.

El proceso de setup inicial en openshift comienza con las siguientes tareas realizadas por el equipo de GCABA con colaboración del equipo desarrollador:

- Creación del proyecto en la plataforma
- Creación de un repositorio para configuraciones (ConfigMaps y DeployConfig)
- Elección de la imagen S2I (Source to Image) apropiada a la tecnología utilizada
- Configuración del Pipeline de construcción de despliegue.
- Creación de usuarios y asignación de roles dentro de la plataforma (Entorno Desarrollo).

Posteriormente, una vez iniciado el desarrollo y primer deploy exitoso, se generan los proyectos y configuran los pipelines en los demás ambientes.

Ambientes Virtualizados

En el caso de no ser factible implementar en la plataforma Openshift, pero se cumple con el estándar tecnológico, se contempla la opción de despliegue en máquinas virtuales (Previa acuerdo de trabajo y justificación).
Para implementar el despliegue se deberá contar con los siguientes requisitos:

- Poseer un archivo de configuración YAML para definir todas las variables de entorno e informar la ruta del mismo en el archivo README.md
- Proveer un script de instalación parametrizable generado con la herramienta ANSIBLE y probado en el ambiente de desarrollo.

Ambientes Cloud

- **Plataforma como Servicio (PaaS)**

Las implementaciones en ambientes Cloud deben contemplar el proceso de integración continua de GCABA, para esto tanto la infraestructura como el código a implementar debe estar automatizado.
Los requerimientos para realizar las implementaciones Cloud son los siguientes:

- Se debe entregar en GIT el código fuente y todo los scripts necesarios para la implementación.
- Deben definir la infraestructura / servicios a utilizar como código (Infraestructure as code), para esto se pueden utilizar las herramientas Terraform o Cloudformation.
- La implementación del código debe automatizarse con la herramienta ANSIBLE.
- Se debe contar con un archivo de configuración de servicios y base de datos, externo al código de la aplicación, parametrizable por ambiente.
- Script para carga de datos iniciales.

- **Software como servicio (SaaS)**

En el caso de la customización de productos, se debe contar con un proceso para realizar la actualización de todos los ambientes.
Para esto se debe contar con los siguientes requerimientos:

- Documentación necesaria para generar el ambiente inicial.
- Entrega en GIT de scripts o archivos de configuración para la implementación.