2025

MARCO NORMATIVO DE IT

ES0901 - Estándar de Desarrollo
Agencia de Sistemas de Información

Diciembre 2025

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 https://buenosaires.gob.ar/agencia-de-sistemas-de-informacion/estandares-de-la-agencia

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 formalizada y acordada.

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.

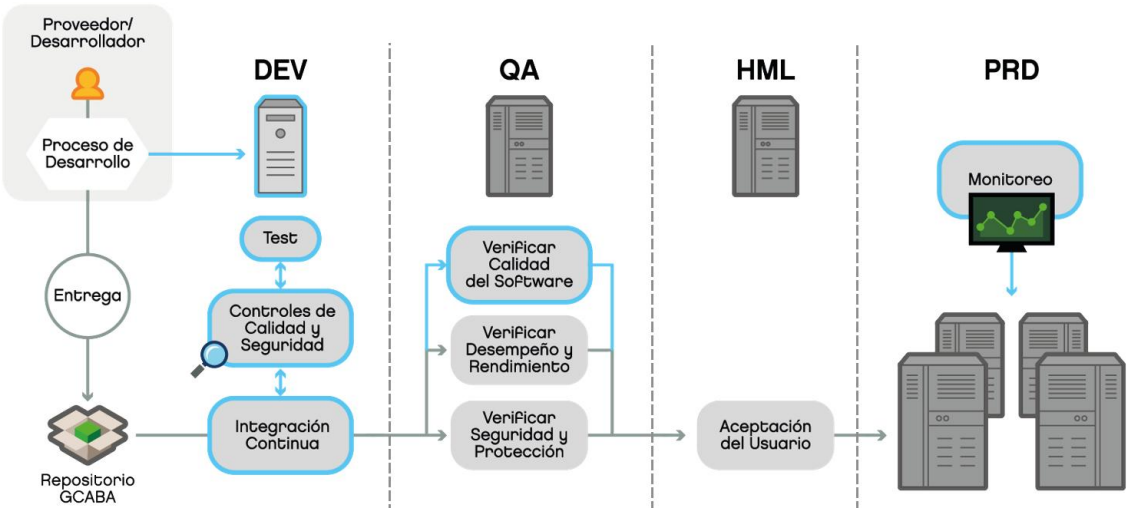


fig. 1 - Proceso de gestión de cambios de software

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 I: Repositorio de control de versiones*).

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 *ES902: Estándar de Seguridad y Anexo V: Assessment 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. Además, probar escalamiento horizontal en contenedores de la plataforma OpenShift.
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%. • Archivos de configuración de Varnish (definición y actualización). • Plan de pruebas de funcionalidades core (.jmx). • 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	• Todas 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.

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 *Round Robin*.

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 creadas 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 encontrara 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 alertará 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 Herramientas y Versiones aceptadas por la ASI

Con el objetivo de garantizar la homogeneidad, la compatibilidad y la mantenibilidad de los desarrollos dentro del GCABA, se ha definido un conjunto de herramientas y versiones homologadas que deben utilizarse en esos proyectos.

La información detallada sobre estas herramientas, incluyendo lenguajes de programación, frameworks, bibliotecas, entornos de ejecución, bases de datos, herramientas de análisis, integración y despliegue, se encuentra disponible en el *Anexo II: Herramientas de Desarrollo y Versiones homologadas*.

Este anexo será actualizado periódicamente por la ASI y constituye la referencia oficial sobre el stack tecnológico aprobado al momento del desarrollo.

Criterios de homologación de herramientas y versiones

Para que una herramienta -exceptuando sistemas operativos, plataformas de búsqueda, bases de datos y servidores web- pueda ser incorporada a este estándar, deberá cumplir con los siguientes criterios mínimos:

Respecto a la versión:

- No presentar vulnerabilidades conocidas.
- Estar homologada en OpenShift para la generación de imágenes.
- Corresponder a una versión estable con al menos 3 meses de maduración.
- La última versión publicada debe tener una antigüedad menor a 2 años.
- No estar en estado de deprecación (EOL). Si el fin de soporte ocurrirá en menos de 6 meses, no podrá agregarse a este estándar.
- Licenciamiento gratuito y con licencia open source compatible (ej*: MIT, Apache 2.0, BSD, EPL, MPL)*.

Respecto a su comunidad y mantenimiento:

- Actividad reciente: issues, pull requests y discusiones resueltas en menos de 1 año.
- Frecuencia de commits y releases regulares.
- Se dará mayor relevancia a la combinación de métricas de comunidad (forks + stars + issues/pull requests recientes) junto con la frecuencia de actualizaciones.

Estos criterios aplican también para bibliotecas que no estén incluidas en el estándar. Si se necesita utilizar una biblioteca no homologada, la ASI evaluará tanto su uso como su posible incorporación al estándar.

En caso de dudas respecto a los criterios de homologación, o para iniciar el proceso de evaluación de una herramienta, framework o biblioteca no contemplada en el Estándar de Desarrollo, los equipos podrán canalizar sus consultas a través de la casilla de correo oficial:

estandares.DGISIS@buenosaires.gob.ar

7 Plataformas de Desarrollo

El GCABA contempla diferentes tecnologías 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 analizar debidamente 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 a fin de implementar 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 *Anexo II: Herramientas de Desarrollo y Versiones homologadas*).

G2. Aplicar todas las buenas prácticas del mercado, aplicables 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 deberá delegar el ingreso de credenciales de usuarios según lo descripto en el apartado “Autenticación”.

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 en forma inmediata.

D8. Los servicios deben estar protegidos con sistema de token.

A Nivel Programación

P1. Todo desarrollo debe realizarse utilizando frameworks homologados en este documento. No se permite el uso del lenguaje puro (“vanilla”) sin el soporte de su respectivo framework. Esto garantiza la adopción de buenas prácticas en seguridad, modularización, manejo de dependencias y favorece la mantenibilidad del código. Las bibliotecas o componentes de bajo nivel podrán utilizarse únicamente cuando estén integrados de forma indirecta a través del framework correspondiente (por ejemplo, conectores de base de datos usados por un ORM).

Nota: En desarrollos que utilicen tecnologías basadas en Node.js (como Angular, React o NestJS), se establece que el administrador de paquetes permitido es NPM (Node Package Manager). No está permitido el uso de YARN u otras alternativas, a fin de garantizar la compatibilidad y trazabilidad de dependencias en los entornos de GCABA.

Por otro lado, en el caso de plataformas de negocio (ERP, CRM, FSM, etc.), se considera que estas ya proveen un entorno estructurado y normado para el desarrollo, por lo que su uso está habilitado bajo los lineamientos propios de cada plataforma, siempre que no contradigan los principios contenidos en este documento.

P2. Utilizar patrones de diseño estándares, por ejemplo: Singleton, Visitor, Strategy, Adapter, Lazy Loading.

P3. 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.

P4. Disponer de una capa de servicios core API REST que permita integrarse con otros sistemas del Ecosistema del GCABA.

P5. Las validaciones se realizan con control duplicado, por frontend y backend.

P6. Todo componente adicional utilizado en código debe ser validado y acordado.

P7. No se debe incluir lógica de negocio en la capa de base de datos. Toda funcionalidad relacionada con reglas de negocio debe implementarse en la capa correspondiente de la aplicación, evitando el uso de store procedures, triggers, funciones u otros artefactos de lógica en el motor de base de datos. Se deberá preservar la mantenibilidad, trazabilidad y portabilidad del código, así como asegurar el cumplimiento de los principios de arquitectura n-capas y desacoplamiento.

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 acceso 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 con 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 la transferencia de conocimientos (*skill transfer*) y mantenimiento.

7.2 Arquitecturas de Sistemas

Atento a lo detallado hasta aquí, a continuación, se describe cada opción aceptada, agrupada por enfoque y con su justificación marco de implementación y requisitos.

Arquitecturas basadas en capas

a. Monolítica estructurada (N-Capas o MVC)

Ante un proyecto de baja complejidad, con baja demanda de volumen y criticidad, es posible optar por una arquitectura monolítica estructurada (monolito tradicional), basada en el modelo cliente-servidor

y organizada según patrones consolidados como MVC o arquitectura en N-capas.

Para hacer esta selección debe estar específicamente definido que no existe una necesidad de escalar en ninguno de los siguientes criterios básicos: volumen de transacciones, concurrencia, disponibilidad, integración con otros sistemas, mantenibilidad en módulos desacoplados o despliegue independiente.

En esta opción las herramientas recomendadas son:

- Backend / Frontend: framework Laravel (PHP), JavaScript, HTML5, CSS3
- Como sistema de diseño debe utilizarse Obelisco (https://gcba.github.io/estandares/)
- Motor de base de datos: MariaDB + ORM (Eloquent)

b. Monolítica modular

Esta es una alternativa válida para proyectos de complejidad baja, en los que se busca mantener una única unidad de despliegue, pero con una organización interna clara y desacoplada por dominios o responsabilidades. Se considera especialmente útil cuando el sistema puede requerir cierto nivel de desacoplamiento organizativo, sin los costos adicionales de operación, monitoreo y seguridad propios de una arquitectura distribuida.

Herramientas recomendadas:

- Backend: Spring Boot (Java), Django REST Framework (Python), NestJS (Node.js)
- Frontend: Angular
- Como sistema de diseño debe utilizarse Obelisco (https://gcba.github.io/estandares/)
- Base de datos: Oracle, MariaDB

Arquitecturas distribuidas

Dentro de este grupo se incluyen distintos enfoques para construir sistemas con múltiples componentes que colaboran entre sí de forma desacoplada, habitualmente mediante servicios, eventos o interfaces bien definidas. A continuación, se detallan los estilos más relevantes para el GCABA:

a. Arquitectura orientada a servicios (SOA)

Para los casos de complejidad media o alta, con lógica compartida, que requieran integración con otros sistemas del universo GCABA, que demanden escalabilidad, robustez, con alto volumen de transacciones y alta disponibilidad, se propone el uso de esta arquitectura.

Se podrá seleccionar esta opción con los requisitos que se definen a continuación:

- Cumplir con los principios de diseño de servicios
- Utilizar el estilo arquitectónico REST siguiendo el estándar REST API URI, con mensajes de datos en formato JSON y con documentación generada mediante SWAGGER o RAML.
- Como sistema de diseño debe utilizarse Obelisco (https://gcba.github.io/estandares/)

Herramientas recomendadas:

- Backend:
    - **Java Spring Boot**: *Lenguaje y plataforma base:* Java JDK, Apache Maven. *Spring Suite:* Boot, MVC, Core (IoC), AOP. *Persistencia y acceso a datos:* Hibernate ORM, JPA (Java Persistence API), Flyway. *Integración y mensajería:* Apache Camel con Spring Boot, JMS (Java Message Service), AMQ. *Servidor embebido:* Tomcat WebServer. *Logueo:* Logback.
    - **Node.JS**: frameworks NestJS, Express o Fastify, con TypeORM (o Sequelize Migrations)
    - **Python:** con Django REST Framework o FastAPI
- Frontend: frameworks NextJS, Angular, React (opcional), HTML5 y CSS3
- Motor de base de datos: ORACLE.

b. Arquitectura de microservicios

Para los casos de complejidad media y alta, que requieren integración con otros sistemas o servicios del GCABA, con alto volumen de transacciones y alta disponibilidad. Con demanda de robustez, resiliencia y donde se pueda desarrollar, implementar, operar y escalar sin afectar el funcionamiento de otros servicios del sistema. Es decir, donde aplique un enfoque de desarrollo de software que divida aplicaciones en servicios independientes.

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 el estilo arquitectónico REST siguiendo el estándar REST API URI, con mensajes de datos en formato JSON con documentación generada mediante SWAGGER o RAML.
- Como sistema de diseño debe utilizarse Obelisco (https://gcba.github.io/estandares/)

Herramientas recomendadas:

- Backend:
    - **Java Spring Boot**: *Lenguaje y plataforma base:* Java JDK, Apache Maven. *Spring Suite:* Boot, Cloud, MVC, Core (IoC), AOP. *Persistencia y acceso a datos:* Hibernate ORM, JPA (Java Persistence API), Flyway. *Integración y mensajería:* Apache Camel con Spring Boot, JMS (Java Message Service), AMQ. *Servidor embebido:* Tomcat WebServer. *Logueo:* Logback.
    - **Python**: con framework Django o FastAPI
- Frontend: frameworks NextJS, Angular, React (opcional), HTML5 y CSS3
- Base de datos: Oracle o base por microservicio (persistencia políglota)

c. Arquitectura orientada a eventos (EDA)

Es recomendable utilizar este enfoque para sistemas que requieren procesamiento asincrónico, bajo acoplamiento entre componentes, y la capacidad de responder a eventos en tiempo real. Aplicada especialmente en contextos donde se necesite lograr una mayor escalabilidad, resiliencia o auditabilidad de flujos complejos.

Esta arquitectura permite que los distintos componentes o servicios del sistema se comuniquen a través de eventos publicados y consumidos, sin conocer directamente a sus emisores o receptores. Es ideal para escenarios donde la actividad del sistema se desencadena en función de eventos y no por llamados directos entre componentes.

Se podrá seleccionar esta opción con los siguientes requisitos:

- Cumplir con los principios de desacoplamiento, reactividad y procesamiento orientado a mensajes.
- Definir claramente los eventos de negocio, su estructura, y los canales o tópicos involucrados.
- Implementar mecanismos de garantía de entrega y gestión de errores, según la criticidad de los procesos.
- Mantener trazabilidad de los flujos mediante logs, correlación de eventos o herramientas de trazabilidad.

Herramientas recomendadas:

- Backend:
    - Java con Spring Boot + Spring Cloud Stream
    - Python con FastAPI o Django REST Framework
- Mensajería: Apache Kafka
- Frontend (si aplica): frameworks NextJS, Angular, React (opcional), HTML5 y CSS3
- Persistencia: bases de datos desacopladas por servicio, event sourcing opcional

7.3 Plataformas de Desarrollo

Plataforma CMS

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

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 características creadas a tales efectos.

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

Plataforma Cloud

El uso de plataformas Cloud podrá considerarse en aquellos casos en los que se justifique técnica y funcionalmente su necesidad, en comparación con una implementación basada en infraestructura autogestionada del GCABA (on-premise).

Será admisible recurrir a soluciones Cloud cuando se trate de productos o servicios Cloud preexistentes (SaaS, PaaS) con claras ventajas funcionales, de seguridad, mantenimiento o integración.

En todos los casos, el uso de plataformas Cloud deberá ser evaluado por la ASI y estar debidamente documentado en el análisis de factibilidad del proyecto.

Deberán considerarse aspectos como:

- Cumplimiento de normas de seguridad de la información.
- Ubicación geográfica y jurisdicción de los datos.
- Costos totales de propiedad (TCO).
- Estrategia de portabilidad y salida (vendor lock-in).

Plataforma Mobile

Los desarrollos orientados a plataformas móviles deberán contemplar criterios de calidad, rendimiento, seguridad y experiencia de usuario acordes con los lineamientos del GCABA.

Cuando se justifique la necesidad de contar con una aplicación móvil dedicada —por ejemplo, en casos donde se requiera funcionalidad offline, uso intensivo de sensores del dispositivo, acceso a recursos del sistema operativo, o una experiencia de usuario optimizada— se podrá optar por una implementación nativa.

Las plataformas recomendadas son:

- **Android**: desarrollo nativo utilizando lenguaje **Kotlin**, a través del uso de frameworks según su funcionalidad y siguiendo las recomendaciones de arquitectura moderna.
    - **Frameworks:** *UI:* Jetpack Compose. *Arquitectura:* Android Jetpack. *Inyección de dependencias:* Hilt. *Persistencia local:* Room. *Multiplataforma:* Kotlin Multiplatform (KMP)
- **iOS**: desarrollo nativo utilizando lenguaje **Swift**, a través del uso de frameworks según su funcionalidad y las prácticas actuales recomendadas por Apple.
    - **Frameworks:** *UI:* SwiftUI. *Concurrencia/estado:* Combine o Swift Concurrency. *Persistencia local:* Core Data.

En todos los casos, el diseño de las aplicaciones deberá:

- Integrarse con los mecanismos de autenticación aprobados por el GCABA.
- Considerar requisitos de accesibilidad, rendimiento, uso offline, consumo de batería y almacenamiento.
- Cumplir con las normas de publicación, incluyendo certificados, políticas de privacidad, permisos requeridos y revisión por áreas técnicas correspondientes.
- Utilizar componentes visuales alineados al Sistema de Diseño Obelisco.

En casos donde se justifique la necesidad de una solución híbrida, y siempre que no se vean comprometidos los criterios de rendimiento, experiencia de usuario o integración con funcionalidades críticas del dispositivo, se permitirá el uso de tecnologías como Ionic en conjunto con Capacitor.

Esta alternativa será admisible para proyectos de baja o media complejidad, y deberá ser validada previamente por la ASI. Se deberá garantizar que las aplicaciones generadas para Android y iOS cumplan con los mismos criterios de seguridad, autenticación, rendimiento y calidad definidos para las aplicaciones nativas.

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 u otorgarle un tratamiento individual.

8.1 Integración mediante el ESB del GCABA

El Enterprise Service Bus (ESB) es un componente estratégico provisto por la ASI para facilitar la integración y la comunicación entre los distintos sistemas y aplicaciones del GCABA, independientemente del enfoque arquitectónico adoptado.

El uso del ESB permite incorporar mecanismos de seguridad, control de concurrencia y acceso formal a fuentes de información, siguiendo un enfoque basado en servicios.

Conforme a lo establecido en los apartados "Principios de Aplicaciones" y "Entregables", la comunicación entre aplicaciones debe estar formalizada y acordada. En este marco, si es necesario desarrollar servicios para integrar sistemas, dichos servicios deberán utilizar el ESB provisto por el GCABA.

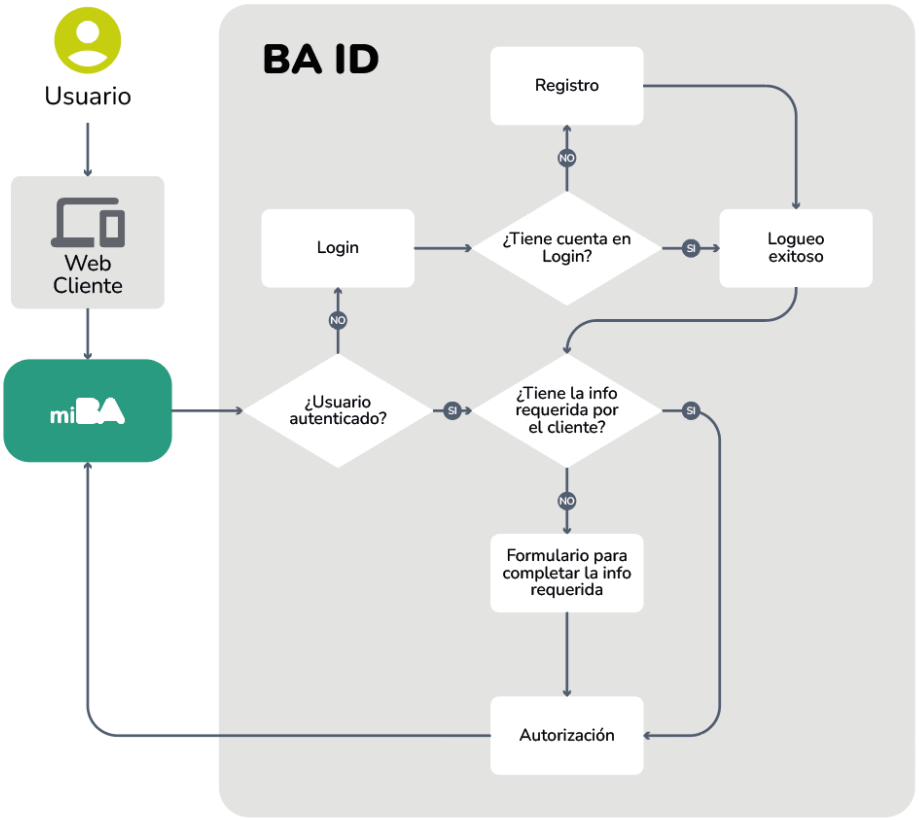
Esta integración deberá estar reflejada en el Documento de Arquitectura que forma parte de los entregables del proyecto.

8.2 Autenticación

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

miBA 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 uso directo del ciudadano debe pasar por el Active Directory del GCABA. Es decir, debe autenticar contra el directorio de GCBA delegando el ingreso de las credenciales en el portal de autenticación del OpenID administrado por la DGSEI (Ver *ES0902 - Estándar de Seguridad*)

Los permisos y roles en el marco de la aplicación deberán asignarse desde un backoffice de la misma.

Para más información sobre miBA, solicitar acceso a la wiki de miBA Login:
https://gcaba.sharepoint.com/sites/grupo_ManualdeIntegracinmiBALogin

Autenticación institucional con Keycloak – Aplicaciones no orientadas a ciudadanía

El mecanismo de autenticación a implementar en las aplicaciones que no poseen interacción con el ciudadano debe basarse en el protocolo **OpenID Connect**, cumpliendo con los lineamientos de seguridad informática definidos por la ASI.
En todos los casos, el proveedor autorizado y obligatorio de identidad para la implementación de OpenID en los entornos del GCABA es **Keycloak**, gestionado por la DGSEI.

Las aplicaciones deberán:

- Integrarse con Keycloak utilizando los flujos adecuados de OpenID.
- Registrarse en el servidor correspondiente.
- Respetar las políticas de autenticación, autorización y protección de recursos definidas por la ASI.

Además, se deberá tener presente que no se brindarán credenciales sobre el anterior OpenID (https://oauth2-server.apps.buenosaires.gob.ar/). Por el contrario, las nuevas versiones de aplicaciones deberán actualizar la autenticación a la nueva solución, redirigiendo al usuario para el ingreso de sus credenciales a este portal de autenticación: https://identidad-gcaba.apps.buenosaires.gob.ar/.

La asignación de roles a los usuarios es una función que queda delegada a la propia aplicación, pudiendo utilizar grupos de AD de ser necesario. Se recomienda acotar la autenticación del usuario en base a su ubicación en el árbol de AD o por membresías de grupos.

8.3 Georreferenciación

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.

8.4 Sistema de Archivos

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 (Simple Storage Service).

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 almacenan 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.

8.5 Generación de Reportes

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 performantes.

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 se resuelve a través de SAC (SAP Analytics Cloud).

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

8.6 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.

8.7 Gestión de Procesos de Negocio

Las aplicaciones que necesiten flujos de procesos de negocio dinámicos deben gestionarse con una herramienta de WorkFlow del mercado. La plataforma de orquestación de procesos BPM (*Business Process Management*) que se utiliza para controlar procesos complejos aceptada por el GCABA es Camunda, basada en estándares como BPMN**,** DMN **y** CMMN.

8.8 Motor de Reglas de Negocio

Para la automatización de la gestión de procesos muy variables es recomendable utilizar un motor de reglas (BRMS – *Business Rules Management System*) donde se definan las condiciones en forma dinámica. El motor de reglas aceptado es Drools, integrable con Java, Spring Boot y Camunda entre otros.

8.9 Gestor de Servicios de Campo

En el caso de necesitar implementar una herramienta de gestión de servicios de campo (FSM - *Field Service Management*) para la gestión de rutas se debe utilizar la herramienta del GCABA o, en el caso de proponer otra, tiene que ser evaluada y validada por la ASI.

8.10 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 data center local y Cloud.

Ver *Anexo III: 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:

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 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 24x7 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, servidores de aplicación granjeables y stateless.

Seguridad

Consultar *Estándar de Seguridad (ES0902)* que complementa este documento.

Robustez

Los distintos servicios, servidores de aplicación, 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.
- Adaptable (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 el 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 poner a disposición 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 / proxys puedan entender el estado actual del aplicativo. Ante un error inesperado no debe visualizarse por frontend/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.

- Servicio de Monitoreo: Toda aplicación deberá implementar mecanismos de *health check, liveness y readiness* como condición excluyente para su homologación y pasaje a producción. Esto permitirá verificar conexiones con la infraestructura y salud de los servicios core. Para más detalles sobre la implementación y buenas prácticas, consultar el *Anexo IV: Requisitos de Health Check para aplicaciones.*
- Despliegues: Las aplicaciones deben contemplar un proceso de despliegue automático para sus actualizaciones. Ver *Anexo III: 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 que 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 accesos 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

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 son aceptables integraciones directas.

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

Para consultar el catálogo disponible, puede contactarse con la DGISIS.

13 Condiciones de aceptación

Cada entregable pasará por un proceso de certificación, dependiendo el mismo, el responsable del GCABA podrá ser diferente y deberá ser 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.

Grados de Severidad

Crítica

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.

Mayor

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.

Menor

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.

Cosmética

El error se refiere a un mal funcionamiento o diseño de la interfaz de usuario, que no impide la correcta ejecución del sistema.

Prioridad

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

Urgente

El error debe solucionarse inmediatamente dado que no permite continuar, puede requerir una entrega especial.

Alta

El error debe solucionarse tan pronto como sea posible dado que se requiere para la próxima salida a producción.

Normal

Prioridad media, es deseable la corrección del error para la salida a producción.

Baja

Corregir el error sólo si no afecta al 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 (lógica del sowtware).
- 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
- Se realizará profiling de las aplicaciones construidas para visualizar el 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 I: 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

El presente documento está dirigido 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 ser los documentos funcionales o arquitectura, que debe residir en el SharePoint de ASI.

4. Estructura de un proyecto

La ASI generará el repositorio del proyecto y otorgará los accesos 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. Dicho

script 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 scripts 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 (verbose). 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 DEVELOP y una MASTER. En DEVELOP 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 DEVELOP 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)	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. Configuración

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 código - GITLAB

El desarrollo continuo debe realizarse en la rama 'develop'. Esta rama representa el estado activo del entorno de desarrollo (DEV) y debe ser utilizada para integrar nuevas funcionalidades, cambios o correcciones que aún no estén destinados a producción.

Al finalizar una versión del software, se debe generar un tag desde la rama ‘master’ en Git siguiendo el esquema de versionado semántico: MAJOR.MINOR.PATCH. Además, se aplicará una nomenclatura adicional según el tipo de versión:

Tipo de versión	Sufijo en el tag	Uso
Beta	**-BETA**	Para pruebas funcionales, no funcionales o preparación de ambientes.
Release Candidate	**-RC**	Candidata a producción.
Hotfix	**-FIX**	Correcciones urgentes sobre una versión ya productiva.

BETA

Las versiones BETA están orientadas a pruebas funcionales y no funcionales, de integración o preparación de ambientes. Estas versiones solo podrán llegar al ambiente de QA, pero nunca a producción.

Release Candidate (RC)

Si se libera la versión candidata '1.0.0-RC' y no logra llegar a producción (por ejemplo, por vulnerabilidades de seguridad), se debe crear una nueva versión incrementando su número, por ejemplo '1.0.1-RC'. Este proceso se repite hasta que una versión RC cumpla con los criterios para ser liberada en producción.

FIX

Las versiones FIX deben ser siempre generadas a partir de una versión ya productiva. Solo corrige errores o vulnerabilidades detectadas en una versión previa que no permite al ambiente productivo funcionar, sin agregar nuevas funcionalidades ni cambios mayores. Dado que los tiempos de despliegue en producción deben ser inmediatos, a este tipo versión se le realizará el assessment a posteriori del despliegue productivo. Dicho assessment se realizará en el ambiente de QA.

Ejemplo general de versionado:

Si la versión **'2.4.0-RC'** fue liberada en producción:
El primer HotFix será **2.4.1-FIX** y el segundo será **2.4.2-FIX**
Para los sucesivos casos, podría darse: **2.5.0-BETA, 2.5.1-BETA, 2.5.2-RC, 2.5.3-FIX**, etc.
Los números de versión nunca deben repetirse.

8. Consideraciones a tener en cuenta

- Asegurarse que todo tag sea único y numerado correlativamente.
- Documentar los cambios que se agregan en cada tag en el changelog.md y pasos a seguir para la implementación en el archivo UPGRADE.md (ej: ejecución de script de base de datos, agregar variables, montar volúmenes).
- Usar Git de forma consistente para facilitar trazabilidad, auditoría y despliegues automatizados.

Anexo II: Herramientas de Desarrollo y Versiones Homologadas

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

Lenguaje	Versiones Homologadas	Web / Móvil
PHP	8.2.29 | 8.3.25	Web
Python	3.11.13 | 3.12.11	Ambos
Java (OpenJDK)	17.0.16 | 21.0.8	Ambos
Node.js	20.19.5 | 22.19.0	Web
Kotlin	2.1.21 | 2.2.10	Mobile
Swift	6.0 | 6.1 | 6.2	Mobile

Importante:

- Según los principios de desarrollo expresados en el apartado 7, debe utilizarse alguno de los frameworks incluidos en este documento, no el propio lenguaje.
- Para todos los entornos basados en Node.js, el administrador de paquetes homologado es NPM. YARN no está autorizado para la instalación, administración o ejecución de dependencias.
- **Node.js:** Para todos los entornos basados en Node.js, el administrador de paquetes homologado es NPM. Su versión viene incluida con Node.js, por lo que queda resuelta automáticamente. No está autorizado YARN para la instalación, administración o ejecución de dependencias.
- **Kotlin:** Siempre verificar compatibilidad de Kotlin con versiones homologadas de Java, ya que para compilación mobile y desarrollos backend, se utiliza ese lenguaje.
- **Swift:** Si bien sirve tanto para desarrollos de apps mobile y backend, este debe utilizarse únicamente para el desarrollo de aplicaciones nativas en plataforma Mobile.

Framework Frontend	Versiones Homologadas	Tipo
Angular	18.2.13 (LTS) | 19.2.15 | 20.3.0	Web
Next.js	14.2.32 | 15.5.7	Web
Ionic / Capacitor	8.7.3	Híbrida o PWA

Framework Backend	Versiones Homologadas
NestJS	10.4.18 | 11.1.6
Django	4.2.24 (LTS) | 5.2.6 (LTS)
FastAPI	0.116.1
Laravel	11.46.0 | 12.28.1
Express.js	4.21.2 | 5.1.0
Fastify.js	4.29.1 | 5.6.0
Spring Boot	3.3.13 | 3.4.9 | 3.5.5
.NET (.NET Core)	8.0.20 | 9.0.9
Kotlin	2.1.21 | 2.2.10

.NET: Se deberá justificar debidamente el motivo de su elección. Microsoft ha unificado .NET Framework y .NET Core en la plataforma .NET.

Kotlin: Siempre verificar compatibilidad de Kotlin con las versiones homologadas de Java, ya que en desarrollos backend, se utiliza ese lenguaje.

Motor de Base de Datos	Versiones Homologadas
Oracle	19c (LTR)
PostgreSQL	15.13
MariaDB	10.11.x | 11.4.x
MongoDB	8.x
Redis	7.2.x

Importante: para nuevas implementaciones, se deberá validar previamente con DGISIS y DGINFRA la versión del motor a utilizar.
PostgreSQL: Elegible solo cuando sea obligatorio para: uso de PostGIS, enlatados (sin otra opción), necesidad de múltiples esquemas en una sola DB, que no pueda utilizarse Oracle o MariaDB (ej: X-Road), etc. Se deberá justificar debidamente el caso y se analizará con DGINFRA cada implementación en particular.

Plataforma de Búsqueda	Versiones Homologadas
Apache Solr	9.9.0
Elastic Search	8.17.10

Bibliotecas

Biblioteca Android	Versiones Homologadas
core	1.13.1 | 1.15.0 | 1.16.0 | 1.17.0
material-components	1.12.0 | 1.13.0
Maps	18.2.x | 19.x
Places	3.5.0 | 4.4.1
Compose	1.3.2 | 1.5.15 | 1.9.1
ViewPager2	1.1.0
Compose Navigation	2.7.7 | 2.8.9 | 2.9.4
WorkManager	2.9.1 | 2.10.4
RXJava3	3.1.11
camerax	1.3.4 | 1.4.2 | 1.5.0
retrofit	2.10.x | 2.11.x | 2.12.x | 3.0.x
Okhttp3	5.0.x | 5.1.x
paging-runtime	3.3.6
livedata	2.7.0 | 2.8.7 | 2.9.3
dagger hilt	2.51.x a 2.57.x
room	2.7.x | 2.8.x
Coil	2.7.0 | 3.1.0 | 3.2.0 | 3.3.0

Biblioteca Java	Versiones Homologadas
Spring Cloud	2023.0.5 | 2024.0.2 | 2025.0.0
Okhttp3	5.0.x | 5.1.x
retrofit	2.10.x | 2.11.x | 2.12.x | 3.0.x
junit-jupiter-api	5.12.x | 5.13.x
modelmapper	3.2.4
Hibernate	6.6.29 | 7.1.1
ojdbc11	21.19.x | 23.9.x
mockito-core	5.17.x | 5.18.x | 5.19.x
apache poi	5.3.x | 5.4.x
aws-java-sdk-core	2.31.x | 2.32.x
unirest-java-core	4.4.x | 4.5.x
MapStruct	1.6.x
Gson	2.11.x | 2.12.x | 2.13.x
lombok	^1.18.32
commons-codec	1.17.x | 1.18.x | 1.19.x
commons-lang3	3.16.0 | 3.17.0 | 3.18.0

Biblioteca Python	Versiones Homologadas
Matplotlib	3.8.4 | 3.9.4 | 3.10.6
Bokeh	3.6.x | 3.7.x | 3.8.x
NumPy	1.26.x | 2.2.x | 2.3.x
SciPy	1.14.1 | 1.15.3 | 1.16.1
SpaCy	3.7.5 | 3.8.7
Pandas	2.2.3 | 2.3.2
PyTorch	2.7.x | 2.8.0
NLTK	3.9.1
Gensim	4.3.x
transformers	4.53.x | 4.54.x | 4.55.x
Pillow	10.4.x | 11.2.x | 11.3.x
Scrapy	2.12.x | 2.13.x
TensorFlow	2.18.x | 2.19.x | 2.20.x
Oracledb [(1)]	3.1.1 | 3.2.x | 3.3.x
mysql-connector-python [(1)]	9.3.0 | 9.4.0

[(1)] Estas bibliotecas están permitidas exclusivamente como **conectores de base de datos subyacentes**, utilizadas a través de un framework aprobado (ej. Django, FastAPI) y en combinación con un ORM estructurado como SQLAlchemy. Su uso directo (sin ORM ni framework) no está autorizado, conforme al lineamiento de desarrollo basado en frameworks y no en lenguaje puro (“vanilla”).

Biblioteca Angular	Versiones Homologadas
ngx-extended-pdf-viewer	19.7.x | 20.5.x | 21.4.x | 22.3.x | 23.3.x | 24.3.x
ngx-permissions	17.1.x | 19.0.x
ngx-spinner	(depende de la versión Angular)
PrimeNg	18.0.2 | 19.1.4 | 20.1.x

Biblioteca JavaScript	Versiones Homologadas
React	18.3.x | 19.0.1 | 19.1.2
React-Redux	9.1.2 | 9.2.x
Redux	5.0.1
Maps	3.60 | 3.61 | 3.62 | quarterly
jwt-decode	4.0.0
Chart.js	4.4.x | 4.5.0
Anime.js	4.1.x
Apache ECharts	5.6.0 | 6.0.0
ApexCharts	4.7.0 | 5.3.x
JOSE	4.15.9 | 5.8.x | 5.10.x | 6.1.x
Modernizr	3.13.1
Openlayers.js	9.2.4 | 10.6.0
Day.js	1.11.18
@fullcalendar/core	6.1.x
helmet	7.2.x | 8.1.x
core-js	3.41.x | 3.45.x
dotenv	17.2.x
express-fileupload	1.5.x
Underscore	1.13.7
Backbone.js	1.6.x
CKEditor 5	45.2.x | 46.1.x
TinyMCE	6.8.x | 7.9.x | 8.1.x

Importante: Para todos los entornos basados en Node.js, el administrador de paquetes homologado es NPM**.** YARN no está autorizado para la instalación, administración o ejecución de dependencias.

Biblioteca .NET	Versiones Homologadas
Entity Framework Core	8.0.x | 9.0.x
Quartz.NET	3.15
NUnit	4.3.2 | 4.4.x
FluentValidation	11.11.0 | 12.0.0
NLog	5.5.x
Log4Net	3.2.0
NServiceBus	9.2.4

Biblioteca .NET	Versiones Homologadas
MimeKit	4.10.0
Polly	8.6.0
Hangfire	1.8.18

Otras Bibliotecas	Versiones Homologadas
lottie-android	6.6.x
Open CV	4.10.x | 4.11.x
Libpng	1.6.48
Zlib-ng	2.2.5

Sistema de Diseño	Versiones Homologadas
Obelisco	2
Bootstrap	5.3.x

Nota: Por resolución RES-94-SECITD-23, para el desarrollo debe utilizarse Obelisco como sistema de diseño, el cual a su vez utiliza Bootstrap como librería.

Sistemas de Gestión

CMS (Content Management System)	Versiones Homologadas
Wordpress	6.6.2 | 6.7.2 | 6.8.x
Drupal	10.4 | 11.1

LMS (Learning Management System)	Versiones Homologadas
Moodle	4.5.3 (LTS) | 5.0

DMS (Document Management System)	Versiones Homologadas
Ckan	2.10.7 | 2.11.2
GeoServer	2.27.x

Administrador de migración de base de datos	Versiones Homologadas
Flyway	11.x
Laravel Migrations	(1)
Django Migrations	(1)
Entity Framework Core Migrations	(1)
Alembic	1.15.2 | 1.16.x
TypeORM	0.3.26
Sequelize Migrations	6.37.7

Nota: complementariamente a lo citado en la RESOL-2013-177-ASINF (PO0001 - Política Catálogo Gral) y teniendo en cuenta el riesgo que conlleva la ejecución manual de scripts sobre las bases de datos en ambientes productivos, se insta a utilizar administradores de migración de bases de datos en los procesos de despliegue. Para más detalles, consultar *Anexo III: Despliegue continuo*.
(1) la versión de estos administradores de migración está sujeta a la del framework utilizado.

BPM (Gestor de Procesos de Negocio)	Versiones Homologadas
Camunda	7.22 Community Edition

Gestor de Colas de Mensajes	Versiones Homologadas
Apache Kafka	3.6.2

Servidor Web	Versiones Homologadas
Apache Web Server	2.4.62
Apache Tomcat	10.1.36
Nginx	1.24
JWS (JBoss Web Server)	6.0 SP1 | 6.1

Nota: Nginx podrá emplearse únicamente en escenarios puntuales donde no exista otra alternativa técnica viable. Por defecto, el servidor web de uso preferente es Apache HTTP Server.

Sistema Operativo	Versiones Homologadas
Red Hat Enterprise Linux - RHEL	8.7 | 9.6
Android	11 | 12 | 13 | 14 | 15
iOS	16 | 17

Nota: Para Android y IOS, dado que cada usuario puede elegir libremente que versión de estos SO utilizar, se enumeran las versiones sobre las que debiera validarse el correcto funcionamiento de los desarrollos sobre plataforma móvil. Las apps modernas suelen requerir como mínimo Android 11, por lo que se lista como versión mínima. En comparación, los usuarios de Android suelen adoptar nuevas versiones más lentamente debido a la fragmentación entre dispositivos y fabricantes, mientras que Apple se beneficia de un ecosistema de hardware y software más controlado.

Seguridad	Versiones Homologadas
OpenSSL	3.5 (LTS)
PrimeKey EJBCA	9.3.x
OpenID Connect (OIDC)	(1)
Keycloak	(1)

(1) Las versiones homologadas de estas herramientas serán provistas por la DGSEI al momento de ser requeridas para un fin específico.

Otras Herramientas	Versiones Homologadas
BusyBox	1.36.x

Anexo III: Despliegue continuo

Los sistemas a implementarse on premise en GCABA deben adaptarse a la plataforma de entrega continua (Continuous Delivery o CD) Openshift. La misma provee los ambientes de Desarrollo, Calidad, Homologación, Producción Interna y Producción DMZ.

La estrategia utilizada consiste en realizar un build único en el ambiente de QA y luego promover la imagen con variables externas en cada ambiente. Esto implica que la imagen del contenedor se construye una única vez en el ambiente de Calidad y luego es promovida sin modificaciones a los ambientes de Homologación y Producción. Cada ambiente aplica su propia configuración mediante variables externas (por ejemplo, config maps, secrets u otros mecanismos de inyección de configuración).

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 contenedores efímeros.
- Para las tecnologías:
  - Framework Laravel PHP: se debe contar con el archivo composer.json en la raíz del proyecto en donde se cuenta con la descripción de bibliotecas asociadas al desarrollo.
  - Node.js: se debe contar con el archivo package.json.
  - 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 deben utilizar las herramientas disponibles dentro del framework elegido para versionar la estructura de base de datos (administradores de migración de bases de datos) evitando utilizar scripts. Para conocer las versiones homologadas, consultar *Versiones y Herramientas aceptadas por la ASI.*
  EJ:
  Framework Laravel (PHP) se utiliza Artisan.
  Framework Spring (JAVA) se utiliza JPA, Hibernate.

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

- 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 (previo acuerdo de trabajo y debida 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 todos los scripts necesarios para la implementación.
- Se debe 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.
- Debe proveerse un mecanismo automatizado (por ejemplo, migradores o seeders) para la carga de datos iniciales, en línea con la pauta establecida para evitar la ejecución manual de scripts.

- **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.

Anexo IV: Requisitos de Health Check para aplicaciones

Introducción

La implementación de *health checks* es un requisito excluyente para que una aplicación sea considerada homologada y pueda pasar a producción. Esto responde a la necesidad de garantizar que los servicios estén operativos, detectando estados no confiables causados por errores de configuración, conectividad u otros fallos de aplicación. En entornos como OpenShift, los health checks permiten gestionar automáticamente el ciclo de vida de los contenedores ante fallos o comportamientos inesperados.

Tipos de Health Checks

A continuación, se describen los tres tipos principales de probes utilizados por la plataforma:

- *Liveness Probe*: Detecta si la aplicación está colgada o en estado incorrecto. Si falla, el contenedor se reinicia.
- *Readiness Probe*: Verifica si la aplicación está lista para recibir tráfico. Si falla, se evita el enrutamiento de solicitudes al pod.
- *Startup Probe*: Su implementación es opcional. Se usa durante el arranque de aplicaciones lentas para evitar falsos negativos de liveness.

Requisitos Generales de Implementación

Toda aplicación deberá exponer los siguientes endpoints específicos:

- */health/liveness*
- */health/readiness*

Estos no deben exponer datos sensibles. Además, se recomienda incluir un endpoint de health general (`/health`) para tareas de monitoreo.

Buenas Prácticas

- Liveness debe ser simple: validar que el hilo principal esté activo o que el proceso responda.
- Readiness puede validar conexiones a bases de datos, servicios externos u otros recursos críticos.
- No mezclar responsabilidades: cada endpoint debe tener un objetivo específico.
- Estimar correctamente el tiempo de inicio para evitar falsos negativos.
- Documentar en el entregable técnico el método, path, delays y thresholds utilizados.

Métodos de Comprobación

Existen varios métodos para definir probes según el tipo de aplicación:

- HTTP: Ideal para API REST. Se verifica el estado mediante un path y puerto.
- Comando (exec): Útil para contenedores que no exponen HTTP. Ejecuta scripts que devuelven 0 (OK) o error.
- Socket TCP: Verifica la apertura de puertos, ideal para daemons o servicios no HTTP.

Parámetros de Configuración

Para los probes se deben informar los valores para parámetros como:

- *initialDelaySeconds*: Tiempo antes de la primera verificación.
- *timeoutSeconds*: Tiempo máximo para una respuesta.
- *periodSeconds*: Frecuencia de verificación.
- *successThreshold* y *failureThreshold*: cantidad mínima de aciertos/fallas para considerar el resultado de la prueba.

Ejemplo de Configuración YAML

```yaml
readinessProbe:
  httpGet:
    path: /health/readiness
    port: 8080
  initialDelaySeconds: 10
  timeoutSeconds: 1
  periodSeconds: 10
  successThreshold: 1
  failureThreshold: 3

livenessProbe:
  httpGet:
    path: /health/liveness
    port: 8080
  initialDelaySeconds: 10
  timeoutSeconds: 1
  periodSeconds: 10
  successThreshold: 1
  failureThreshold: 3
```

Los templates de deployment.yaml en GitLab incluyen secciones comentadas para configurar los probes. Estas deben ser completadas con los valores definidos para cada aplicación según sus características.

Responsabilidades del Equipo de Desarrollo

- Implementar y validar los endpoints y mecanismos de health check.
- Estimar tiempos de inicio realistas para evitar falsos negativos.
- Informar en la documentación del sistema los parámetros utilizados.
- Coordinar con el área de despliegues para la integración de estos mecanismos en los manifiestos de OpenShift.

Notas Finales

Este anexo resume los lineamientos necesarios para garantizar una correcta implementación de health checks. Cualquier cambio en el comportamiento estándar deberá ser consensuado con la Gerencia Operativa de Despliegues e Implementaciones.

Anexo V: Assessment de Seguridad

El assessment de seguridad es un control obligatorio que debe cumplirse antes de que una aplicación pase a ambientes superiores (HML y PRD). Se realiza en el ambiente de QA y su propósito es asegurar que las aplicaciones mantengan un nivel adecuado de seguridad y cumplan con los estándares descriptos tanto en este documento, como en *ES0902 - Estándar de Seguridad*.

Frecuencia

- Todo desarrollo debe contar con un assessment aprobado para poder avanzar hacia homologación y producción.
- El assessment debe repetirse cada vez que se cumpla alguna de las siguientes condiciones:
    - Haya un desarrollo y transcurran más de 20 días desde el assessment anterior.
    - Existan correcciones de vulnerabilidades detectadas previamente (assessment parcial).
    - Ocurra un incidente de seguridad en la aplicación.
    - No ocurra ninguno de los 3 puntos anteriores, pero haya un desarrollo con cambios en funcionalidades y/o endpoints dentro de los 20 días desde el assessment anterior.

Motivos para solicitar un nuevo assessment

En caso que un activo con assessment aprobado sufra alguna modificación se deben tener en cuenta estos criterios para solicitar uno nuevo:

1. **Cambios en funcionalidades y endpoints**
    - Adición o modificación de endpoints.
    - Eliminación de secciones de frontend o backend.
    - Cambios en parámetros de formularios o APIs.
2. **Integraciones y recursos externos**
    - Nuevas integraciones con APIs externas, servicios de terceros o webhooks.
    - Inclusión de iframes o embeds externos.
    - Incorporación de nuevos scripts de terceros (ej. trackers, analytics, chatbots).
3. **Configuraciones y controles de seguridad**
    - Alteración de políticas de seguridad (CORS, CSP, cookies).
    - Creación o modificación de roles y permisos.
4. **Dependencias e infraestructura**
    - Agregado o actualización de bibliotecas, frameworks, SDKs o dependencias.
    - Migración a nueva infraestructura.
    - Cambios en protocolos de comunicación.
5. **Funcionalidades críticas**
    - Inclusión de funciones de carga o descarga de archivos.
    - Cambios en flujos sensibles de la aplicación.
    - Modificación en el flujo de autenticación (login, registro, recuperación de contraseña).