GUÍA TÉCNICA

Lineamientos para Desarrollo de Software

Diciembre 2018

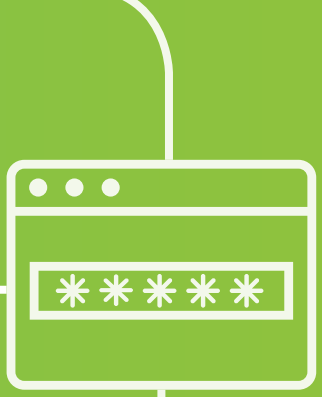
gob digital CHILE

Ministerio Secretaría General de la Presidencia

Gobierno de Chile

índice

I.
INTRODUCCIÓN



Conforme al numeral cuarto "Guías técnicas y de implementación" del articulo primero de las disposiciones transitorias del Decreto N° 14, de 2014, del Ministerio de Economía, Fomento y Turismo, respecto de la publicación de guías necesarias para la adecuada comprensión, implementación y especificación de las normas técnicas, en relación a: el desarrollo de plataformas web abiertas seguras; la accesibilidad y despliegue de contenidos digitales web; el desarrollo e implementación de la interoperabilidad en el Estado, la presente Guía Técnica de Lineamientos para Desarrollo de Software se dicta en cumplimiento de lo anterior.

Esta guía técnica entrega los lineamientos generales y recomendaciones específicas que debe seguir todo equipo que desarrolle software al interior de la Administración del Estado y los grupos de desarrollo de los proveedores, contribuyendo a la construcción de sistemas de alta calidad en todas las instituciones.

La adopción de buenas prácticas es fundamental para todas las etapas de desarrollo de un sistema o aplicación informática. La mitigación de problemas de software permite un uso correcto e íntegro de estos sistemas, ayudando al usuario final a tener un recurso eficiente, confiable, seguro y privado. Asimismo, se minimizan los riesgos y costos de mantención del sistema en horas hombre.

Al desarrollar software, se deben tener en cuenta diversos factores que impactan directa e indirectamente su correcto uso. Estos factores abarcan desde la comunicación en el equipo de desarrollo y pruebas, pasando por consideraciones dentro de la calidad del código fuente, específicamente factores como seguridad, QA y finalmente el deployment, así como su futuro uso en producción.

Para ello, es fundamental adoptar una metodología de desarrollo de software rápida, flexible e incremental, para minimizar el riesgo de desviaciones en su proceso de creación e implementación, y permitir las adecuaciones tempranas, basadas en la retroalimentación de los usuarios o en las modificaciones de los requisitos durante el proyecto.

Los aspectos que serán abordados en esta guía son:

- Calidad del software, tanto en servicio implementado, como en código fuente resultante.
- Entregas tempranas en base a productos mínimos viables, que permitan comenzar a aprovechar las ventajas que ofrece la solución requerida a la brevedad.
- Comunicación efectiva dentro del equipo de desarrollo y también con los stakeholders.
- Uso de estándares, en el que todos los actores e instituciones utilicen el mismo lenguaje.

La adopción de buenas prácticas es fundamental para todas las etapas de desarrollo de un sistema o aplicación informática. La mitigación de problemas de software permite un uso correcto e íntegro de estos sistemas, ayudando al usuario final a tener un recurso eficiente, confiable, seguro y privado. Asimismo, se minimizan los riesgos y costos de mantención del sistema en horas hombre.

II. CONSIDERACIONES PARA EL DESARROLLO DE LAS APLICACIONES

El análisis, desarrollo e implementación deberá ser realizado en base a metodologías ágiles, incorporando al menos las recomendaciones de la metodología denominada The Twelve-Factor App, documentado en https://12factor.net/es/, con el objeto de satisfacer, al menos, los siguientes aspectos requeridos:

- Usar formatos declarativos para la automatización de la configuración. Así es posible minimizar el tiempo y coste que supone que nuevos desarrolladores se unan al proyecto.
- Tener un contrato claro con el sistema operativo sobre el que se trabaja, ofreciendo la máxima portabilidad entre los diferentes entornos de ejecución.
- Disponer, por defecto, despliegues en plataformas modernas en la nube, obviando la necesidad de servidores y administración de sistemas.
- Minimizar las diferencias entre los entornos de desarrollo y producción, posibilitando un despliegue continuo para conseguir la máxima agilidad.
- Poder escalar la arquitectura o las prácticas de desarrollo sin cambios significativos para las herramientas.

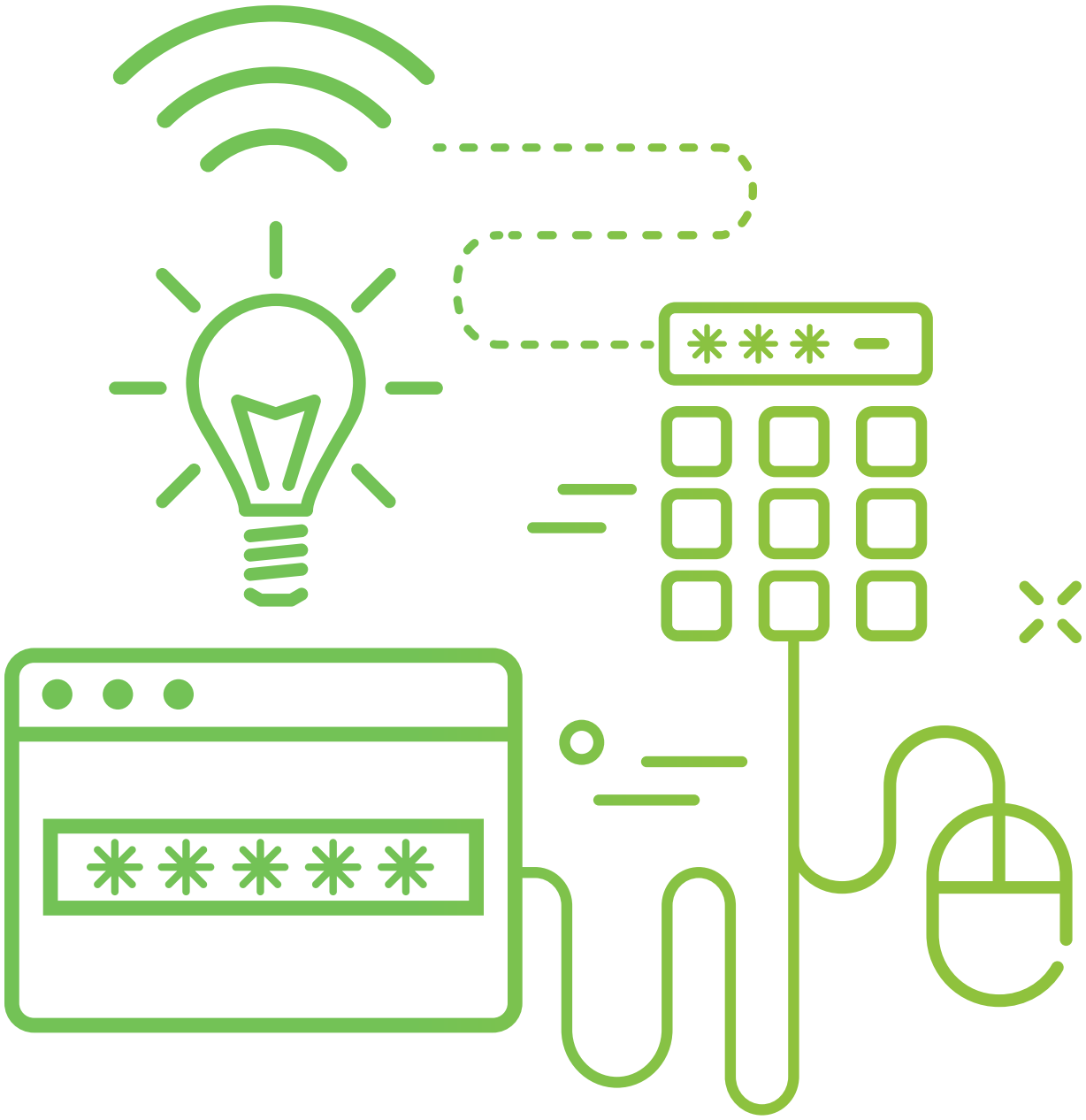
El equipo de desarrollo deberá detallar en su propuesta el uso de metodologías ágiles, tales como Scrum o Kanban, Programación Extrema, Modelamiento Ágil, Feature Driven Development (FDD) o cualquier otra que considere relevante.

Dado que actualmente las instituciones pueden no contar con un equipo con los conocimientos de metodologías de desarrollo ágil, se recomienda iniciar esta capacitación para trabajos futuros y, de esta forma, iniciar los nuevos proyectos con estas metodologías y, con el tiempo, adquirir la experiencia para aplicarla a proyectos ya existentes y mejorarlos en pro de obtener productos ágiles de forma rápida.

Ambientes de desarrollo

Considerando que al estar en una constante mejora el desarrollo podría no terminar nunca, se sugiere la siguiente cantidad de ambientes para cada proyecto en su desarrollo : producción, *staging*, *test* y *development*, cada uno de ellos representado en una rama del proyecto en https://git.gob.cl para, de esta forma, poder detectar rápidamente cuál código es el que está en producción.

Al contar con los resguardos descritos en esta guía, cada paso de un ambiente a otro contará con procedimientos de calidad de código, por lo que se apunta a lograr, en el proceso completo de desarrollo, un código revisado y que esté apto para su entrega en el ambiente en el que es requerido.



III. CONSIDERACIONES PARA SEGURIDAD EN EL DESARROLLO

Las librerías y frameworks de terceros confiables que incorporan mecanismos de seguridad son fundamentales para evitar errores de implementación en áreas en las que el desarrollador no se encuentre tan familiarizado.

1. HACER USO DE LIBRERÍAS Y FRAMEWORKS DE SEGURIDAD

Se deben seleccionar librerías y *framework* de autores confiables, con mantención y desarrollo activo, y que sean ampliamente utilizadas (y por ende, validadas).

Las librerías y *frameworks* de terceros confiables que incorporan mecanismos de seguridad son fundamentales para evitar errores de implementación en áreas en las que el desarrollador no se encuentre tan familiarizado.

Es importante inventariar y catalogar dichas librerías y *frameworks* para mantenerlas actualizadas y prevenir vulnerabilidades en ellas.

Otra alternativa recomendada es encapsular la librería y exponer sólo la funcionalidad requerida en el sistema que se está desarrollando.

Algunas librerías y *frameworks* recomendados se encuentran en la sección Tecnologías de Preferencia.

2. CONSULTAS SEGURAS A LAS BASES DE DATOS

Las vulnerabilidades de tipo SQL injection ocurren cuando datos no confiables son incorporados de forma dinámica a una consulta SQL (muchas veces a través de la concatenación directa de dos strings). Este tipo de ataques permite a un adversario extraer, modificar o eliminar datos desde la base de datos e, incluso en ocasiones, tomar total control del sistema comprometido.

Para mitigar o prevenir este tipo de ataques, se deben implementar medidas de protección acordes a la tecnología y plataforma utilizada:

1. En el caso de estar utilizando un lenguaje orientado a objetos, se debe utilizar el mapeo a través de ORM, asegurarse que la herramienta utilizada no tenga vulnerabilidades de inyección, e implementar otros mecanismos de defensa como codificar y escapar los datos, como se explica en la próxima sección de esta guía.

2. En el caso de no poder utilizar ORM de forma justificada, se deben utilizar prepared *statements* para prevenir posibles inyecciones de código SQL.

3. En el caso de contar con tecnología de base de datos específica o legacy (Oracle, SQL Server, etc), en la cual se utilicen procedimientos almacenados directamente en el motor, éstos deben ser securizados de forma correcta, por ejemplo, utilizando bind variables.

3. CODIFICAR Y ESCAPAR LOS DATOS

Éstas son técnicas defensivas, cuyo objetivo es detener ataques de inyección, ya sean SQL, XSS u otros. Codificar consiste en transformar o traducir ciertos caracteres especiales en otros caracteres equivalentes pero inofensivos para el intérprete. Por ejemplo, el carácter "<" se traduce en "<".

Escapar caracteres es una técnica similar, con la diferencia que en lugar de reemplazar un carácter por otro, se añade un carácter especial antes del dato a escapar, para prevenir interpretaciones erróneas, por ejemplo, añadiendo un "\" antes de caracteres como ["] (comillas dobles), para que sea interpretado como texto y no como el cierre de un string.

Es fundamental realizar este escapado y/o codificación de datos en un entorno confiable (en el servidor y no en el navegador del usuario), de forma tal, de evitar el "doble escapado" que genera errores de despliegue de los datos (por ejemplo, si escapamos antes de almacenar en la base de datos y nuevamente cuando se despliega al usuario).

Para evitar ataques XSS se deben utilizar los mecanismos conocidos como *Contextual Output Encoding*, que nos permiten prevenir el despliegue de contenido malicioso en el navegador del usuario.

Existen otros tipos de codificación y escape, tales como shell escaping, XML escaping, LDAP escaping y otros, que podrán ser utilizados en la medida que sea necesario.

Un sistema debe considerar como potencialmente inseguros todos los datos que provengan desde fuera de la aplicación, así como también los datos que provienen desde la base de datos (en caso que se haya modificado maliciosamente la misma). Por ello, deben ser correctamente codificados y/o escapados siempre, pero sin olvidar el contexto de los datos, por ejemplo, si es HTML, datos alfanuméricos u otros.

Se deben considerar también las validaciones de datos, descritas a continuación:

4. VALIDACIÓN DE DATOS

Ésta es una técnica para asegurar que sólo los datos con el formato correcto podrán ingresar al sistema a desarrollar. Esta validación debe ser correcta, tanto sintáctica como semánticamente. Por ejemplo, si esperamos que en un campo se puedan ingresar sólo dígitos, no debemos aceptar otro tipo de caracteres (validación sintáctica). La validación semántica es un

poco más compleja y consiste en validar que los datos tengan sentido en el contexto de la aplicación, por ejemplo, una fecha de inicio no podría estar después de una fecha de fin.

La validación de datos debe ser, como mínimo, a través de mecanismos de lista negra (datos conocidos como maliciosos). Como una mejor práctica también se validará a través de lista blanca (datos conocidos como correctos, por ejemplo, una lista de regiones).

La validación de datos, al igual que la codificación y escape, deben ser siempre realizados en el servidor y nunca del lado del cliente (por ejemplo, no se debe realizar en el navegador del usuario). No se debe confundir un aviso al usuario (por ejemplo, para alertar de un campo mal ingresado) con la validación de datos.

La validación de datos no convierte automáticamente a los datos en seguros, por lo que se debe utilizar en conjunto con otras defensas, tales como la parametrización de consultas y escapado de datos, mencionados anteriormente.

Para la validación de datos, incluyendo datos complejos, tales datos serializados, HTML u otros, se deberían utilizar librerías apropiadas para ello *(HTML Purifier, Bleach*, entre otras).

5. IMPLEMENTAR MECANISMOS DE AUTENTICACIÓN SEGUROS

La autenticación hace referencia a un proceso para verificar que un individuo o entidad es quien dice ser. Para estos efectos, podemos utilizar varios mecanismos, también llamados factores de autenticación:

5.1 NIVEL 1: CONTRASEÑAS

Las aplicaciones deben exigir al usuario el uso de contraseñas de características y calidad adecuadas, y que no sean contraseñas previamente filtradas[1] Asimismo, se deben implementar mecanismos seguros de recuperación de contraseñas, ya sea utilizando un segundo factor de autenticación o a través de canales diferentes (como teléfono móvil o correo electrónico).

Una contraseña fuerte es aquélla que es difícil de detectar, tanto por humanos como por *software*, protegiendo efectivamente los datos de un acceso no autorizado.

Una contraseña segura consta de al menos ocho (8) caracteres (y mientras más caracteres, más fuerte es la contraseña), que son una combinación de letras, números, símbolos y el uso de mayúsculas y minúsculas. Ésta no debe contener palabras que se pueden encontrar en un diccionario o partes del nombre del usuario.

1 https://github.com/danielmiessler/SecLists/tree/master/Passwords

Para almacenar las contraseñas, se deben utilizar algoritmos criptográficos especialmente diseñados para este fin, tales como bcrypt, PBKDF2 y Argon2. Nunca se deben utilizar algoritmos de hash "a secas", tales como SHA-1, SHA-2 y MD5 para el almacenamiento de contraseñas.

También se deben implementar mecanismos de limitación de intentos de inicio de sesión, ya sea ralentizando los intentos de inicio de sesión, bloqueando la IP de origen de las pruebas fallidas o bloqueando las cuentas después de un número predeterminado de intentos fallidos. Esta estrategia no debe bloquear permanentemente las cuentas, puesto que esto podría causar una denegación de servicio a los usuarios legítimos.

En el caso de que los ataques de fuerza bruta sean un problema, es recomendado implementar múltiples factores de autenticación.

5.2 NIVEL 2: MULTIFACTOR

Es el uso de múltiples factores de autenticación para verificar al usuario. Habitualmente se usa una combinación de dos o más de los siguientes factores:

4. Lo que el usuario sabe (por ejemplo, una contraseña),
5. lo que el usuario tiene (por ejemplo, un dispositivo o llaves de seguridad) y,
6. algo que el usuario es (por ejemplo, la biometría).

La biometría no se debe considerar un dato secreto, puesto que algunas características biométricas pueden ser fácilmente reproducibles u obtenidas sin conocimiento del usuario, por ejemplo, a través de una foto del rostro o "levantando" huellas digitales de objetos, por lo que siempre debe usarse en conjunto con otro factor de autenticación.

5.3 NIVEL 3: AUTENTICACIÓN CRIPTOGRÁFICA

Se logra a través del uso de módulos de hardware criptográfico seguro, en conjunto con algún mecanismo de autenticación adicional (*password*, biometría, otros).

6. IMPLEMENTAR MECANISMOS DE MANEJO DE SESIÓN

Para el manejo de las sesiones, se debe considerar al menos lo siguiente:

- El identificador de sesión debe ser único, suficientemente largo y aleatorio.
- Se deben generar (o rotar) los identificadores de sesión durante la autenticación y re-autenticación.
- Se debe implementar un timeout por inactividad que fuerce la re-autenticación al usuario. La duración de este timeout debe ser inversamente proporcional a la sensibilidad de los datos a proteger, vale decir, mientras más sensible, menor duración.

7. USO DE COOKIES PARA MANEJO DE SESIÓN

Para el uso de *cookies* se debe considerar lo siguiente:

- Deben ser accesibles por el mínimo de

dominios requeridos para el correcto funcionamiento del sistema.

- Deben caducar al momento en que expira la sesión, o luego de un corto período.
- Deben tener el flag *"secure"*. Esto fuerza su transferencia a través de TLS.
- Deben tener el flag *"HttpOnly"*. Esto previene su acceso a través de JavaScript.

8. USO DE TOKENS DE SESIÓN

Cuando sea necesario manejar sesiones stateless, por razones de performance u otras, se recomienda el uso de JWT (JSON Web Tokens), puesto que es un mecanismo seguro e interoperable de manejo de sesión.

Un JWT[2] tiene garantías criptográficas si es generado de forma segura, para lo cual se recomienda utilizar los siguientes parámetros:

- Nunca definir, en la configuración, el algoritmo para firmar o cifrar como *"none"*. Esto deshabilita todas las consideraciones criptográficas. El parámetro alg es el que permite definir el algoritmo que será utilizado para estas tareas.
- En relación al punto anterior, se debe seleccionar un algoritmo lo suficientemente fuerte. La División de Gobierno Digital lo recomendará de forma oportuna, en caso de que alguna de las opciones disponibles deje de ser segura.
- Se deben generar los secretos (o secrets) de forma criptográficamente segura, con una correcta entropía por parte del servidor que los genere.

9. IDENTIFICACIÓN DE DATOS

Los datos sensibles o críticos deben estar identificados para poder implementar las protecciones que requieran de forma correcta. Por ejemplo, si se manejan datos sensibles de ciudadanos, se requieren algunas protecciones específicas que deben estar presentes.

10. DATOS EN TRÁNSITO

Las comunicaciones de los componentes que transporten información de los usuarios entre sistemas, deberán siempre estar protegidas mediante TLS, y los sistemas correctamente configurados para seleccionar el cifrado más fuerte disponible.

11. DATOS EN REPOSO

Se debe evitar, cuando sea posible, el almacenamiento de datos sensibles por parte de la aplicación.

En caso que no se pueda evitar almacenar datos sensibles, éstos deben ser protegidos mediante encriptación, para prevenir la modificación o acceso no autorizado.

Considerando que la encriptación es un tema altamente complejo, en particular las buenas prácticas, mecanismos de cifrado y gestión de llaves, la División de Gobierno Digital prestará el apoyo técnico apropiado para asegurar una correcta implementación y manejo de estas medidas.

2 https://tools.ietf.org/html/rfc7519

En caso de manejar datos que deban considerarse como abiertos, éstos deberán ser manejados y almacenados utilizando formatos estándar que permitan su fácil exportación a las plataformas correspondientes.

12. MANEJO DE SECRETOS

Las aplicaciones habitualmente contienen múltiples secretos que son necesarios para su operación. Éstos pueden incluir certificados digitales, contraseñas para la base de datos, credenciales a otros servicios y llaves criptográficas, entre otros. Toda la información relacionada a certificados digitales, contraseñas, llaves, credenciales, incluidas las rutas de almacenamiento u otras referencias a estos objetos, deben quedar debidamente parametrizadas en un archivo de variables de entorno y éste debe ser excluido de la herramienta de control de versiones Git.

13. DESARROLLO ORIENTADO A PRUEBAS

Para efectos de trabajo con datos o ejercicios, el equipo de desarrollo deberá tener un set de datos no válidos, escogido para trabajar y cargar el sitio para efectos de pruebas de integración continua o pruebas unitarias. Este set no debe contener información que sea real y la cantidad de datos debe ser reducida, pero suficiente para generar pruebas sobre el código.

14. CALIDAD DE CÓDIGO

El código deberá ser revisado de forma continua durante su construcción con herramientas de inspección.

Las herramientas deseadas y por orden de prioridad son:

- Codeclimate
- Sonarqube

El objetivo principal es encontrar posibles bugs de seguridad, tanto dentro del código creado por el equipo de desarrollo, como en sus dependencias. La configuración de ambas, o al menos de una de las aplicaciones, debe ser guardada como archivo de configuración dentro del código, para luego ser usado en la integración continua.

15. DETECCIÓN PREVENTIVA

Para todo lo relacionado con las aplicaciones, se debe configurar y usar algún proyecto OWASP (ejemplo: OWASP Zap) para la revisión en búsqueda de vulnerabilidades en la aplicación. Esto, para corregir lo que sea necesario antes de entregar la aplicación. Sin embargo, como se considera un desarrollo continuo después de una entrega final, la configuración de la aplicación usada debe quedar documentada dentro del código, tanto para uso futuro como para ser integrada dentro del desarrollo/ integración continua.

16. MODELO INICIAL DE DATOS

Las estructuras de base de datos no deberán contener datos, tan sólo los esquemas y, de requerir la carga de datos externos

producto de un proceso de inicialización, estos datos no deben ir en el código y serán procesados por una vía interna y privada, en el caso que se requiera.

17. SEEDING DE LA BASE DE DATOS Y CREACIÓN DE DATOS INICIALES EN EL DESPLIEGUE

La creación de usuarios administradores iniciales o de cualquier tipo de información sensible, debe quedar fuera del sistema de versionamiento del código.

El proceso de creación de un usuario inicial de administración en este proceso, así como cualquier otro dato sensible para el funcionamiento de la aplicación cuya filtración involucre una merma de seguridad, debe ser documentando, entregado confidencialmente y nunca ser registrado en un repositorio de código.

18. IMPLEMENTAR MECANISMOS DE REGISTRO

Se deben registrar distintos eventos del sistema para permitir que éstos sean monitoreados de forma automatizada para efectos de seguridad:

- Utilizar un formato común de registros al interior de la institución. Esto permite facilitar la configuración y parametrización de los mecanismos de monitoreo.
- Registrar la cantidad de información correcta.
- Siempre registrar un timestamp e información de identificación (IP, usuario, etc).
- Nunca registrar información sensible en los logs.

Dentro del *stack* de herramientas detallado en Tecnologías de Preferencia, se encuentra el software Open Source elegido para estos efectos.

19. MANEJO SEGURO DE ERRORES Y EXCEPCIONES

Se debe desarrollar el sistema, de forma tal, que aplique el principio de "fallar seguro", es decir:

- No exponer información sensible o privada en los mensajes de error. En particular, nunca olvidar desactivar el modo *debug* al pasar a producción.
- Asegurarse de que una excepción o fallo no comprometa la seguridad por un error de programación en el sistema. Por ejemplo, causar una denegación de servicio o ejecución de código con privilegios incorrectos.
- Registrar las excepciones adecuadamente en los sistemas que correspondan.

20. COMPILACIÓN LIMPIA

De ser un lenguaje compilado y no interpretado, no es necesario contar con ningún tipo de alerta en el momento de compilación. Para conseguir este objetivo, lo mejor es dar las opciones al compilador para tratar las alertas *(warnings)* como si fueran errores y, de esta forma, fallar en caso de existir una alerta *(warning)* al momento de la compilación.

Un sistema debe considerar como potencialmente inseguros todos los datos que provengan desde fuera de la aplicación, así como también los datos que provienen desde la base de datos (en caso que se haya modificado maliciosamente la misma).

IV. ASEGURAMIENTO Y CERTIFICACIÓN DE CALIDAD

Se requerirá que los equipos de desarrollo de las instituciones cumplan con una estrategia de aseguramiento de la calidad, la cual incluirá:

- Plan de pruebas.
- Diseño de las pruebas.
- Registro de su ejecución y resultados.

Para cada caso, el equipo de desarrollo deberá dejar registro del aseguramiento de la calidad en un documento que certifique la misma, el cual servirá para ser contrastado con cualquier control de calidad que pueda realizar la contraparte técnica.

La División de Gobierno Digital utiliza el servicio web de control de versiones Gitlab, y por esto se requiere utilizar gitlab-ci.yml para configurar integración/entrega continua (CI/CD). Esto, además de la utilización de herramientas para revisión del código (ejemplo: Codeclimate).

Algunos ejemplos para la utilización serán provistos por la División de Gobierno Digital al final de este documento. De existir un caso no cubierto en esta versión de la guía, éstos serán creados por la División de Gobierno Digital y agregados al documento en una nueva versión.

V. TECNOLOGÍAS DE PREFERENCIA

Para cada proyecto se recomienda el uso de ciertas tecnologías de desarrollo y arquitectura del *software,* basado en las recomendaciones del equipo de infraestructura y el equipo de desarrollo de la División de Gobierno Digital. En caso de querer usar otra tecnología no listada se debe justificar su uso.

1. PARA LENGUAJES DE PROGRAMACIÓN

- Sistemas de información
  - Python 2.7+
  - PHP 7.1
  - Java
  - C#
- Microservicios
  - Python 2.7+
  - NodeJS 8+
  - Go
- Los *framework* y bibliotecas para desarrollo recomendados son:
  - PHP: Laravel, Symfony, CodeIgniter
  - Python: Django, Flask, Tornado (para desarrollo de microservicios en Python)
  - Go: Revel, Gin, Martini
  - Java: Spring Boot, Splunk
  - C#: .NET Framework
  - ReactNative
  - ExpressJS
- Bibliotecas gráficas y herramientas:
  - Bootstrap
  - D3JS
  - JQuery
  - Sass, Less
  - Grunt
  - NPM
- Frameworks Javascript
  - AngularJS
  - VUE
  - ReactJS

2. PARA ALMACENAMIENTO DE DATOS RELACIONALES

- PostgreSQL 10 o superior.
- MySQL 5.7 o superior.
- MSSQL 2016 o superior.
- Oracle 12c Release 2 o superior.

3. PARA ALMACENAMIENTO DE DATOS NO RELACIONALES

- MongoDB 3.6 o superior
- Elasticsearch 5.5 o superior
- Redis 5.0 o superior

4. PARA TAREAS DE ENCOLAMIENTO

- RabbitMQ
- ZeroMQ

5. PARA TAREAS DE REGISTRO DE EVENTOS (LOGGING)

- Elasticsearch 5.5 o superior
- Logstash
- Kibana
- Sentry

6. DESPLIEGUE DE APLICACIONES

- Kubernetes
- Docker
- CloudFoundry

VI. USO DE LAS TECNOLOGÍAS

1. LENGUAJE

Para escoger el lenguaje de programación, se debe hacer uso de un *framework* para desarrollar bajo el paradigma de la programación orientada a objetos y utilizando un patrón de arquitectura modelo-vista-controlador (MVC).

Algunos ejemplos de *frameworks* que soportan este paradigma en diversos lenguajes son: Laravel, Symfony, CodeIgniter, Flask, Django, Beego, Revel, Spring Boot, etc.

2. ALMACENAMIENTO

Para la gestión de datos, tanto relacionales como no relacionales, se debe considerar que las bases de datos deben estar instaladas en modo cluster y, por ello, se deben tomar las precauciones necesarias al momento de crear las consultas y hacer uso de las mismas.

En el caso de las bases de datos relacionales, se debe considerar, entre otros:

- Tiempo de respuesta; puede variar si un cluster es muy grande.
- No tener referencias circulares.

Con respecto al uso de bases de datos no relacionales, se debe tener cuidado en la elección del driver, debiendo ser capaz de conectarse a más de un nodo del cluster a la vez para, de esta forma, mantener el paradigma de un sistema en cluster y capaz de crecer de forma horizontal.

3. TAREAS FUERA DE LÍNEA O PROGRAMADAS

Para las tareas asíncronas o que tengan la característica de una tarea programada diaria o de tiempo definido, éstas deben usar tecnologías de encolamiento, como las que se mencionan en Tecnologías de Preferencia. El consumo de las colas debe ser con procesos que no tengan relación con el sitio y debe ser posible desarrollarlos e implementarlos como un servicio separado.

4. FORMATEO DE CÓDIGO

Para una mejor lectura del código por personas en la organización y para el futuro, se recomienda de sobremanera el uso de los formateadores de texto.

Éstas son herramientas que permiten limitar el largo de las líneas, lugares de llaves y paréntesis, así como también ayudan a ordenar el código , lo que debe ser una tarea constante.

Estas herramientas pueden ser configuradas en el editor de texto preferido de los programadores. Para lenguajes como Python, se sugiere PEP8, sin embargo, el estilo y formateo de una organización debe ser elegido por la misma y se sugieren los estándares propuestos para cada lenguaje por sus creadores.

VII. USO OBLIGATORIO REPOSITORIO GIT git.gob.cl

El servicio es administrado por la División y las credenciales para el acceso serán otorgadas oportunamente para que el equipo de desarrollo realice las tareas necesarias sobre esta plataforma.

La plataforma de gestión de versiones de código fuente a utilizar debe ser obligatoriamente el repositorio que provee la División de Gobierno Digital: https://git.gob.cl.

El servicio es administrado por la División y las credenciales para el acceso serán otorgadas oportunamente para que el equipo de desarrollo realice las tareas necesarias sobre esta plataforma.

En el caso de las instituciones, se debe usar el dominio *.gob.cl respectivo de correo en el registro de usuarios, para poder acceder al sistema que es ampliamente usado por la administración del Estado. De no existir un dominio *.gob.cl para la institucion, ésta debe contactar a la División de Gobierno Digital para agregar el dominio a la lista.

Con respecto a empresas proveedoras de servicios del Estado, éstas deben contactar a la División de Gobierno Digital para agregar el dominio de correo del proveedor a la lista de dominios permitidos de registro.

Para la administración de la cuenta y gestión de los grupos, recomendamos la documentación esencial del proyecto GitLab (en inglés) que está en la siguiente URL https://docs.gitlab.com/ce/README.html

En el caso de instituciones que ya estén usando un sistema de versionamiento distinto, se incentiva que el mismo sistema suba una versión de forma automática a https://git.gob.cl para centralizar los esfuerzos y compartir el código y segmentos del mismo con otras instituciones, de forma fácil y rápida.

VIII. DESARROLLO Y CONSUMO DE APIs

Las aplicaciones construidas bajo los lineamientos expuestos hasta acá, deberán cumplir con dos características fundamentales:

- Independencia de la plataforma.
- Evolución del servicio.

Para cumplir esto, se deben desacoplar las implementaciones de clientes y servicios, y poner a disposición sus métodos y operaciones utilizando exclusivamente arquitectura REST. Esta característica debe ser implementada tanto para el consumo interno de la aplicación, como para interoperar con otras aplicaciones.

Todas las operaciones de las interfaces programables deben utilizar el protocolo HTTP [3] , ser procesadas en tiempo real y entregar una respuesta según la codificación establecida en el protocolo HTTP:

- GET: Solicitar un recurso.
- POST: Crear un nuevo recurso subordinado dentro de una URL existente.
- DELETE: Eliminar un recurso.
- PUT: Crear un nuevo recurso a una nueva URL o modificar un recurso existente en una URL.
- HEAD: Idéntica a GET, excepto que no se retorna un cuerpo del mensaje en la respuesta.
- CONNECT: Establece un túnel hacia el servidor identificado por el recurso.
- OPTIONS: Utilizado para describir las opciones de comunicación para el recurso de destino.
- TRACE: Realiza una prueba de bucle de retorno de mensaje a lo largo de la ruta al recurso de destino.
- PATCH: Es utilizado para aplicar modificaciones parciales a un recurso.

Es importante identificar que, de los métodos precedentes, los cuatro primeros son los más utilizados, en particular para atender las operaciones CRUD (create, read, update y delete).

1. INTERACCIÓN CON SERVICIOS LEGADOS

La interacción de un sistema nuevo con otros servicios antiguos o legados, debe ser hecha mediante una integración. Ésta debe considerar la traducción del servicio legado que se desea consumir, hacia un servicio REST. Es decir, la arquitectura REST será la única forma en la que se interopere por parte de las nuevas aplicaciones. La razón de esta restricción es tener una interoperabilidad resiliente entre todos los puntos.

3 https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html

2. EXPOSICIÓN Y AUTENTICACIÓN DE SERVICIOS

La exposición de los servicios API REST debe utilizar mecanismos de autenticación para su consumo privado, usando ClaveÚnica[4] como medio de autenticación entre puntos. Se debe mantener la autenticación y el *token* que compartirán las aplicaciones, permitiendo una interoperabilidad entre todas las futuras y actuales aplicaciones del Gobierno de Chile.

El diseño de la aplicación debe considerar que consumirá y proveerá recursos por medio de arquitectura REST, por lo que el uso apropiado de funciones y/o *frameworks* para este propósito es un requisito de su diseño. Esto ya fue expuesto en la sección de Tecnologías de Preferencia.

3. DESARROLLAR ORIENTADO A INTEROPERABILIDAD

La definición de URI de recursos debe seguir las prácticas expresadas con ejemplos, como se ve a continuación:

- Usar sustantivos para describir los recursos:
  - GET /usuarios/ lista todos los usuarios
  - GET /usuarios/12 muestra detalle del usuario con id 12
  - POST /usuarios crear usuario
  - PUT /usuarios/12 actualiza usuario con id 12
  - PATCH /usuarios/12 actualiza parcialmente usuario con id 12
  - DELETE /usuarios/12 borra usuario con id 12
- Usar "/" para indicar la relación de jerarquía:
  - GET /usuarios/12/mensajes muestra mensajes del usuario con id 12
  - GET /usuarios/12/mensajes/93 obtiene mensaje con id 93 del usuario con id 12
  - POST /usuarios/12/mensajes crear mensaje para usuario con id 12
  - DELETE /usuarios/12/mensajes/3 elimina mensaje 3 de usuario con id 12
- Usar parámetros de consulta *(query strings)* para realizar operaciones de filtro, ordenamiento, paginación y/o búsqueda:
  - GET /instituciones/ buscar?nombre=ministerio* busca organizaciones cuyo nombre comienza con "ministerio"
  - GET /instituciones?ordenar=fecha_creacion,desc lista y ordena

4 https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html

organizaciones por fecha de creación, descendente

» GET /instituciones/2/usuarios?estado=inactivo lista todos los usuarios de la institución 2 con estado inactivo

- Formato de las URIs:
  - » No usar "/" al final de la URI.
  - » No usar mayúsculas.

    /instituciones/2/usuarios

    /Instituciones/2/Usuarios/

- Usar guión "-" (y no guión bajo "_") para facilitar la comprensión de la URI en los casos en que sea más de una palabra para el servicio.

  /oficinas/134/como-llegar

  /oficinas/134/como_llegar

- Usar sólo letras en minúscula dentro de la URI. No utilizar caracteres especiales ni acentos.
- Versionamiento de los servicios de interoperabilidad. Cada servicio de interoperabilidad contará con un número de versión en la URI, el cual deberá ser incrementado en la medida en que el servicio se vea modificado en su naturaleza o los datos que entrega (ver Artículo 11 de la Norma Técnica).

  URI Versión 1 antigua:

  GET /v1/oficinas/134/como-llegar

  URI Version 2 actual:

  GET /v2/oficinas/134/como-llegar

- No usar extensiones de archivos.

  /instituciones/2/descripcion

  /instituciones/2/descripcion.txt

  /instituciones/2/descripcion.json

La descripción de servicios será realizada utilizando el estándar OpenAPI Specification (OAS) v2.0, siendo este punto un requisito fundamental en la documentación entregable de un proyecto.

4. USO DEL IDIOMA EN EL CÓDIGO

El uso de idioma castellano e inglés a nivel de programación y definición de esquemas para la construcción de servicios será de la siguiente forma:

- Castellano
  - » Variables y contenido
  - » Construcción de las URIs
  - » Documentación
  - » Metadatos
- Inglés
  - » Operaciones
  - » Encabezados

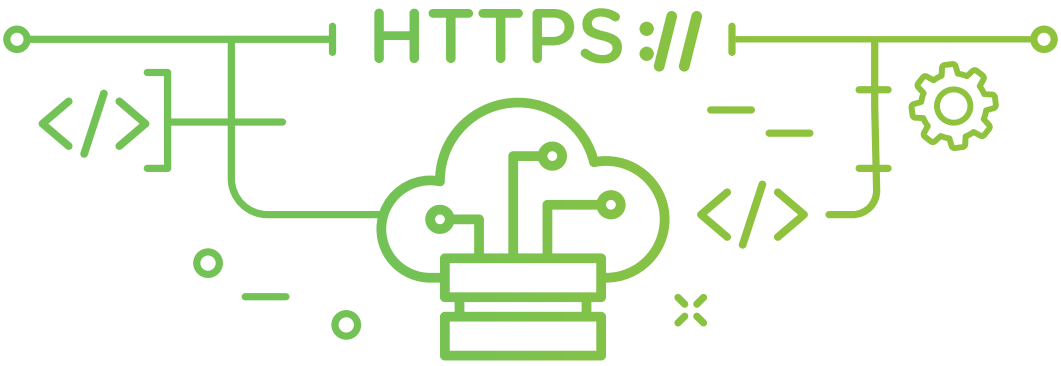
IX. USO DE CONTENEDORES EN EL DESPLIEGUE

Para el despliegue de las aplicaciones es altamente recomendado el uso de contenedores, debido a la facilidad de movilidad de los mismos y replicación de los distintos ambientes. Dada su característica de idempotente, nos permiten replicar en una plataforma para crecer de forma horizontal.

En el caso de plataformas de servidores y con contenedores Docker, el uso de herramientas para la gestión y automatización es muy recomendable. En este caso, se sugiere usar Ansible, Chef o Puppet para la administración de los servidores y levantar los contenedores en los mismos, utilizando la herramienta docker-compose provista por Docker. Este set de herramientas permite su fácil y rápida replicación.

Para situaciones en las cuales se puede contar con cluster Kubernetes o CloudFoundry, las herramientas provistas por cada uno es suficiente para el despliegue, ya que estas herramientas están preparadas para la gestión, replicación y respaldo de los ambientes, y es posible generar más de un ambiente usando las herramientas de cada uno para el efecto.

Se entiende también que muchas veces las aplicaciones no pueden ingresar a un container, o por su carácter legacy no es posible introducirlas en estos containers idempotentes. Sin embargo, se sugiere considerar su uso en los futuros desarrollos y, de esta forma, ir avanzando hacia infraestructuras más flexibles y que pueden crecer de forma horizontal.



X. LICENCIAS

Todo desarrollo realizado al interior del Estado debe estar licenciado, acorde a las necesidades de uso con que fue creado. Para el Estado es fundamental la colaboración entre instituciones, por lo que es mandatorio, salvo fundadas excepciones, la construcción de *software* cuyo código fuente sea accesible y modificable por otras instituciones, así como también para estar frente al escrutinio de los ciudadanos y que éstos puedan realizar los aportes que consideren necesarios para mejorar el código de aplicaciones que ellos mismos pueden usar.

Para estos efectos, están a disposición las siguientes licencias:

1. GPLv2 y GPLv3

Este conjunto de licencias es conocido como *copyleft*, es decir, requiere que las modificaciones realizadas al *software*, incluyendo cambios efectuados por terceros, sean puestos a disposición de los receptores del *software* utilizando la misma licencia. Esto impulsa la colaboración entre instituciones y da la posibilidad de que todas ellas puedan aportar a las mejoras del *software* y, aquellas que no tengan la capacidad de desarrollo, puedan usarlo sin problemas futuros.

Para todos los desarrollos, se recomienda el uso de la licencia GPLv3, pero se permite el uso de la licencia GPLv2 en caso de que alguna institución requiera modificar *software* que utilice dicha licencia y no pueda apelar a re-licenciar la misma a partir de una fecha específica.

2. LGPLv3

Esta licencia sirve cuando el *software* debe ser enlazado con módulos que son privativos o no compatibles con alguna de las versiones de GPL.

La licencia LGPL puede ser usada para mantener la viralidad del licenciamiento y de esta forma poder enlazar con módulos privativos.

3. APACHE 2.0

Esta licencia, compatible con GPLv3 (no inferiores), supone muchas ventajas para mantener el código libre y también permite trabajos secundarios y uso del código en otros proyectos. Se recomienda este uso en todo proyecto que requiera de una cierta interacción con el mundo privado y cuando esta interacción sea a largo tiempo. Un ejemplo de esto último es cuando se genera un producto complementario que podría llegar a ser usado por otros gobiernos e, incluso, por el mundo privado.

4. DOMINIO PÚBLICO (Creative Commons 0 y equivalentes)

Estas licencias actúan como el equivalente a poner en dominio público el trabajo realizado. Son eficaces para compartir datos que puedan ser utilizados por la comunidad científica, medios audiovisuales y otros datos que requieran una distribución lo más amplia posible.

Habitualmente, esta licencia se aplica a los datos generados por un *software* y no al *software* en sí, salvo algunas excepciones.

CASOS DE USO DE LICENCIA

Al momento de aplicar las licencias se debe tener especial consideración en cómo se desea perpetuar el código en el tiempo, es decir, cuando el código sea un proyecto relevante, éste debe considerar que al estar público da lugar a que la comunidad tome el código y pueda hacer cambios en el mismo, para uso propio o para vender de vuelta estos cambios al Estado.

Sin embargo, al usar una licencia GPLv3 esto se puede evitar, ya que fuerza a que los cambios mayores realizados al código sean contribuidos al proyecto principal y, de esta forma, mantener todo cambio en el código libre en el tiempo.

Al dejar el código en dominio público, sin licencia o usando una licencia clásica Creative Commons, no se está buscando que los cambios al código vuelvan a la base sino, más bien, mantener la autoría del mismo. De esta forma, toda persona dentro de la comunidad puede tomar el código y realizarle cambios, cerrarlo y utilizarlo para prestar servicios al Estado, usando el código del Estado. Es por ello que estas licencias no son recomendadas para *software* de carácter crítico o que su uso pueda masificarse de forma importante en el tiempo.

Por otro lado, es necesario considerar que muchas veces el *software* podría estar usando módulos con licencias no compatibles con una GPLv3 o GPLv2. Por esto, el uso de LGPLv3 es recomendado, ya que permite linkear con módulos propietarios. Este caso se da para aplicaciones que son compiladas y enlazadas con otros módulos. No aplica para lenguajes interpretados.

En el caso de que la institución no sea capaz de decidir el medio y/o la forma de licenciamiento, la División de Gobierno Digital prestará la ayuda y orientación necesaria a la institución que lo solicite.

BUENAS PRÁCTICAS DE LICENCIAMIENTO

Al momento de licenciar el software es necesario agregar el texto exacto de la licencia usada en una archivo llamado LICENSE, COPYING, LICENSE.TXT o COPYING.TXT. Este texto no debe ser modificado ni alterado de forma alguna, ya que vivirá con el código y será transportado por todo medio al momento de desplegar el mismo.

Es deseable, pero no mandatorio, incluir al principio de cada archivo del código un aviso de licenciamiento *(copyright notice)* con un extracto sugerido en la licencia. En el caso de la licencia GPLv3, debe ser de la siguiente forma:

Nombre de aplicación

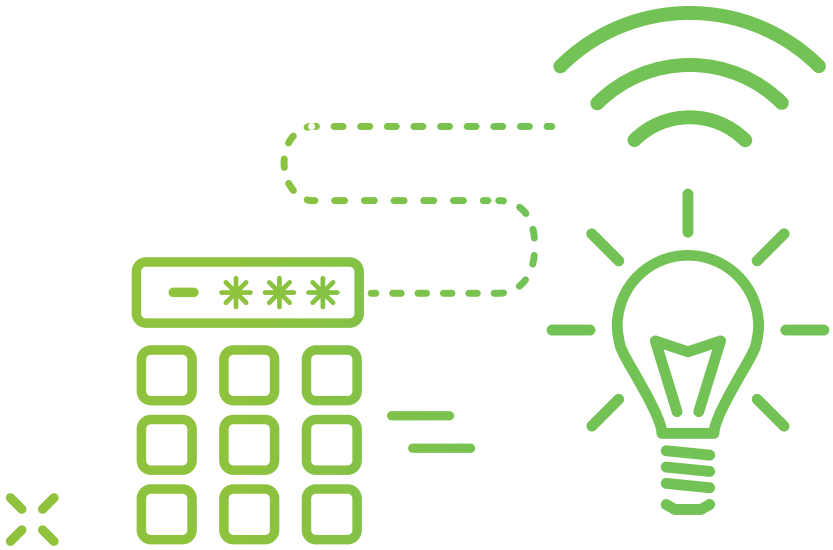
XI. REVISIONES Y ACTUALIZACIONES A ESTA GUÍA

Ésta será la instancia oficial en la cual la sociedad civil podrá realizar una revisión y evaluación para integrar nuevas mejoras a la guía, según su propia opinión y experiencia en el mundo de las tecnologías.

Dado que el mundo de la tecnología avanza cada vez a pasos más y más agigantados, esta guía será sometida a revisión cada 3 meses, en busca de las actualizaciones que puedan ocurrir, de forma de mantenerse actualizada. Esta revisión será realizada por la División de Gobierno Digital en colaboración con todas las instituciones que deseen participar. Con esto se espera incluir todas las mejoras y aprendizajes de las instituciones conforme han seguido esta guia y pudieran encontrar mejoras a la misma.

Con respecto a la revisión por parte de la sociedad civil, ésta será programada una vez al año, previa convocatoria pública. Ésta será la instancia oficial en la cual la sociedad civil podrá realizar una revisión y evaluación para integrar nuevas mejoras a la guía, según su propia opinión y experiencia en el mundo de las tecnologías.

Sin desmedro de lo anterior, la guía podrá contener un anexo con fe de erratas apropiadas, que serán incorporadas en la medida que éstas sean detectadas, sin la necesidad de esperar a la revisión anual pública antes mencionada..



Sin desmedro de lo anterior, la guía podrá contener un anexo con fe de erratas apropiadas, que serán incorporadas en la medida que éstas sean detectadas, sin la necesidad de esperar a la revisión anual pública antes mencionada.

XII. EJEMPLOS GITLAB-CI.YML

Contexto

Las aplicaciones se pueden dividir en dos grandes grupos: Procesamiento y *Assets*.

Se entiende por procesamiento, toda parte de la aplicación que se enfoca en el lenguaje de programación elegido y es el encargado de procesar la información por el lado del servidor. Es decir, todo código que recibe una entrada, procesa la misma y tiene una salida esperada y/o predecible.

Los assets son toda parte de la aplicación que no cambia y se mantiene en el tiempo desde que es creado en algún proceso previo. También son las imágenes que se mantienen como parte del diseño.

Por parte de assets procesado, se puede entender todo archivo minificado o comprimido (js, css, etc). En el caso de imágenes, éstas pueden tener un origen en el diseño o ser creadas a partir de preprocesamiento para optimización en web de las mismas, como compresión en el formato, mejoras en su calidad y/o agrupación para ser usadas en el diseño.

Los artefactos se entenderán como todo conjunto finito que puede ser descargado para ser usado en la aplicación. Dentro de éstos, se encuentran los assets, paquetes Java compilados y cualquier elemento de la aplicación que puede ser creado una vez y que se mantendrá en el tiempo sin cambios.

Los ejemplos ofrecidos en este capítulo pueden cambiar en el tiempo, por lo que se adjunta como adicional la URL de referencia al mismo, en caso que se lea esta guía antes de su próxima actualización.

Aplicaciones PHP

Se comienza con un archivo Dockerfile, el cual es el que da el inicio a la construcción de la imagen que será cargada en el servicio *registry* de https://git.gob.cl para su posterior uso.

Usaremos como ejemplo el caso de ChileAtiende:

Ejemplo: ChileAtiende

https://git.gob.cl/chileatiende/chileatiende/blob/master/Dockerfile

```
FROM php:7.1-fpm
ARG CREDENTIALS_GIT
ARG REPO=git.gob.cl/chileatiende/chileatiende
ARG DIRECTORY_PROJECT=/var/www/chileatiende

WORKDIR $DIRECTORY_PROJECT

%23 Install Packages
RUN apt-get update && apt-get install -y \
     libxml2-dev \
     git \
     zip \
     unzip \
     zliblg-dev \
     libpng-dev \
     --no-install-recommends \
  %23 Docker extension install
  && docker-php-ext-install \
         opcache \
         pdo_mysql \
         pdo \
         mbstring \
         tokenizer \
         xml \
         ctype \
         json \
         zip \
         gd \
         bcmath \
  %23 error to stderr php-fpm
  && { \
      echo "log_errors = On" ; \
      echo "error_log = /dev/stderr" ; \
      echo "error_reporting = E_ALL" ; \
      echo "memory_limit = 256M" ; \
  } > /usr/local/etc/php/php.ini \
  %23 Install composer
  && curl -sS https://getcomposer.org/installer | php -- --install-dir=/usr/local/bin --filename=composer

%23 Clone repository
COPY . $DIRECTORY_PROJECT
  %23 Install dependencies from project
RUN composer install && composer clearcache && rm -rf /root/.composer /usr/local/bin/composer

%23 Permissions
RUN find $DIRECTORY_PROJECT -type f -exec chmod 644 {} \; \
  && find $DIRECTORY_PROJECT -type d -exec chmod 755 {} \; \
  && chown -R www-data:www-data $DIRECTORY_PROJECT \
  %23 Reduce image size
  && apt-get remove --purge -y git curl \
  && apt-get autoremove -y \
  && apt-get clean \
  && apt-get autoclean \
  && apt-get autoremove -y \
  && rm -rf /usr/share/locale/* \
  && rm -rf /var/cache/debconf/*-old \
  && rm -rf /var/lib/apt/lists/* \
  && rm -rf /usr/share/doc/* \
  && rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/* /var/cache/apt/archives/*.deb /var/cache/apt/archives/partial/*.deb /var/cache/apt/*.bin

ENV LANG es_CL.UTF-8
ENV LANGUAGE es_CL:es
ENV LC_ALL es_CL.UTF-8
ENV TZ America/Santiago

RUN echo "APP_KEY=$(php artisan key:generate --show)"    > .env

EXPOSE 9000
CMD ["php-fpm"]
```

https://git.gob.cl/chileatiende/chileatiende/blob/master/.gitlab-ci.yml#L14

```
xxxxxxxxxx
image: docker:stable

stages:
- build
- code_quality

variables:
MYSQL_DATABASE: homestead
MYSQL_ROOT_PASSWORD: secret
DB_HOST: mysql
DB_USERNAME: root


build:
stage: build
variables:
  DOCKER_DRIVER: overlay2
services:
  - docker:stable-dind
script:
  - docker login git.gob.cl:4567 -u gitlab-ci-token -p $CI_BUILD_TOKEN
  - docker build -t git.gob.cl:4567/chileatiende/chileatiende:$CI_COMMIT_REF_NAME .
  - docker push git.gob.cl:4567/chileatiende/chileatiende:$CI_COMMIT_REF_NAME
only:
  - master
  - staging

code_quality:
stage: code_quality
variables:
  DOCKER_DRIVER: overlay2
allow_failure: true
services:
  - docker:stable-dind
script:
   - export SP_VERSION=$(echo "$CI_SERVER_VERSION" | sed 's/^\([0-9]*\)\.\([0-9]*\).*/\1-\2-
 stable/')
  - docker run
      --env SOURCE_CODE="$PWD"
      --volume "$PWD":/code
      --volume /var/run/docker.sock:/var/run/docker.sock
      "registry.gitlab.com/gitlab-org/security-products/codequality:$SP_VERSION" /code
artifacts:
  paths: [gl-code-quality-report.json]

build site:
stage: build
image: node:10-stretch
script:
  - apt-get update
  - apt-get -y install libpng16-16 libpng-tools libpng-dev
  - npm i npm@latest -g
  - npm install
  - npm audit fix --force
  - npm run prod
artifacts:
  expire_in: 30 days
  paths:
    - public/*
```

Luego procedemos a crear los artefactos para este sitio, que básicamente son los js, css, svg, png y jpg servidos en el sitio. Esto lo podemos ver de la siguiente forma:

https://git.gob.cl/chileatiende/chileatiende/blob/master/.gitlab-ci.yml#L45

```
xxxxxxxxxx
image: docker:stable

stages:
- build
- code_quality

variables:
MYSQL_DATABASE: homestead
MYSQL_ROOT_PASSWORD: secret
DB_HOST: mysql
DB_USERNAME: root


build:
stage: build
variables:
  DOCKER_DRIVER: overlay2
services:
  - docker:stable-dind
script:
  - docker login git.gob.cl:4567 -u gitlab-ci-token -p $CI_BUILD_TOKEN
  - docker build -t git.gob.cl:4567/chileatiende/chileatiende:$CI_COMMIT_REF_NAME .
  - docker push git.gob.cl:4567/chileatiende/chileatiende:$CI_COMMIT_REF_NAME
only:
  - master
  - staging

code_quality:
stage: code_quality
variables:
  DOCKER_DRIVER: overlay2
allow_failure: true
services:
  - docker:stable-dind
script:
   - export SP_VERSION=$(echo "$CI_SERVER_VERSION" | sed 's/^\([0-9]*\)\.\([0-9]*\).*/\1-\2-
 stable/')
  - docker run
      --env SOURCE_CODE="$PWD"
      --volume "$PWD":/code
      --volume /var/run/docker.sock:/var/run/docker.sock
      "registry.gitlab.com/gitlab-org/security-products/codequality:$SP_VERSION" /code
artifacts:
  paths: [gl-code-quality-report.json]

build site:
stage: build
image: node:10-stretch
script:
  - apt-get update
  - apt-get -y install libpng16-16 libpng-tools libpng-dev
  - npm i npm@latest -g
  - npm install
  - npm audit fix --force
  - npm run prod
artifacts:
  expire_in: 30 days
  paths:
    - public/*
```

Como se puede ver, sólo se ejecutan los comandos relacionados para crear los *assets*. Éstos están almacenados en el directorio public/ por lo que es el mismo el que es empaquetado.

Un último trabajo es el control de calidad de código, el cual también genera un artefacto, pero en este caso, el mismo contiene el resultado del análisis realizado en este proceso, para luego revisarlo.

https://git.gob.cl/chileatiende/chileatiende/blob/master/.gitlab-ci.yml#L28

```
xxxxxxxxxx
image: docker:stable

stages:
- build
- code_quality

variables:
MYSQL_DATABASE: homestead
MYSQL_ROOT_PASSWORD: secret
DB_HOST: mysql
DB_USERNAME: root


build:
stage: build
variables:
  DOCKER_DRIVER: overlay2
services:
  - docker:stable-dind
script:
  - docker login git.gob.cl:4567 -u gitlab-ci-token -p $CI_BUILD_TOKEN
  - docker build -t git.gob.cl:4567/chileatiende/chileatiende:$CI_COMMIT_REF_NAME .
  - docker push git.gob.cl:4567/chileatiende/chileatiende:$CI_COMMIT_REF_NAME
only:
  - master
  - staging

code_quality:
stage: code_quality
variables:
  DOCKER_DRIVER: overlay2
allow_failure: true
services:
  - docker:stable-dind
script:
   - export SP_VERSION=$(echo "$CI_SERVER_VERSION" | sed 's/^\([0-9]*\)\.\([0-9]*\).*/\1-\2-
 stable/')
  - docker run
      --env SOURCE_CODE="$PWD"
      --volume "$PWD":/code
      --volume /var/run/docker.sock:/var/run/docker.sock
      "registry.gitlab.com/gitlab-org/security-products/codequality:$SP_VERSION" /code
artifacts:
  paths: [gl-code-quality-report.json]

build site:
stage: build
image: node:10-stretch
script:
  - apt-get update
  - apt-get -y install libpng16-16 libpng-tools libpng-dev
  - npm i npm@latest -g
  - npm install
  - npm audit fix --force
  - npm run prod
artifacts:
  expire_in: 30 days
  paths:
    - public/*
```

Adjuntamos el contenido del archivo .gitlab-ci.yml para el proyecto ChileAtiende completo.

https://git.gob.cl/chileatiende/chileatiende/blob/master/.gitlab-ci.yml#L28

```
xxxxxxxxxx
image: docker:stable

stages:
- build
- code_quality

variables:
MYSQL_DATABASE: homestead
MYSQL_ROOT_PASSWORD: secret
DB_HOST: mysql
DB_USERNAME: root


build:
stage: build
variables:
  DOCKER_DRIVER: overlay2
services:
  - docker:stable-dind
script:
  - docker login git.gob.cl:4567 -u gitlab-ci-token -p $CI_BUILD_TOKEN
  - docker build -t git.gob.cl:4567/chileatiende/chileatiende:$CI_COMMIT_REF_NAME .
  - docker push git.gob.cl:4567/chileatiende/chileatiende:$CI_COMMIT_REF_NAME
only:
  - master
  - staging

code_quality:
stage: code_quality
variables:
  DOCKER_DRIVER: overlay2
allow_failure: true
services:
  - docker:stable-dind
script:
   - export SP_VERSION=$(echo "$CI_SERVER_VERSION" | sed 's/^\([0-9]*\)\.\([0-9]*\).*/\1-\2-
 stable/')
  - docker run
      --env SOURCE_CODE="$PWD"
      --volume "$PWD":/code
      --volume /var/run/docker.sock:/var/run/docker.sock
      "registry.gitlab.com/gitlab-org/security-products/codequality:$SP_VERSION" /code
artifacts:
  paths: [gl-code-quality-report.json]

build site:
stage: build
image: node:10-stretch
script:
  - apt-get update
  - apt-get -y install libpng16-16 libpng-tools libpng-dev
  - npm i npm@latest -g
  - npm install
  - npm audit fix --force
  - npm run prod
artifacts:
  expire_in: 30 days
  paths:
    - public/*
```

Aplicaciones Ruby

En el proceso general existe una gran similitud con la aplicación PHP, ya que los procesos son los mismos, sin embargo, se agrega un nuevo paso de test, que permite el aseguramiento de calidad sobre la aplicación.

https://git.gob.cl/chileatiende/chileatiende/blob/master/.gitlab-ci.yml#L28

```
xxxxxxxxxx
image: docker:stable

stages:
- build
- code_quality

variables:
MYSQL_DATABASE: homestead
MYSQL_ROOT_PASSWORD: secret
DB_HOST: mysql
DB_USERNAME: root


build:
stage: build
variables:
  DOCKER_DRIVER: overlay2
services:
  - docker:stable-dind
script:
  - docker login git.gob.cl:4567 -u gitlab-ci-token -p $CI_BUILD_TOKEN
  - docker build -t git.gob.cl:4567/chileatiende/chileatiende:$CI_COMMIT_REF_NAME .
  - docker push git.gob.cl:4567/chileatiende/chileatiende:$CI_COMMIT_REF_NAME
only:
  - master
  - staging

code_quality:
stage: code_quality
variables:
  DOCKER_DRIVER: overlay2
allow_failure: true
services:
  - docker:stable-dind
script:
   - export SP_VERSION=$(echo "$CI_SERVER_VERSION" | sed 's/^\([0-9]*\)\.\([0-9]*\).*/\1-\2-
 stable/')
  - docker run
      --env SOURCE_CODE="$PWD"
      --volume "$PWD":/code
      --volume /var/run/docker.sock:/var/run/docker.sock
      "registry.gitlab.com/gitlab-org/security-products/codequality:$SP_VERSION" /code
artifacts:
  paths: [gl-code-quality-report.json]

build site:
stage: build
image: node:10-stretch
script:
  - apt-get update
  - apt-get -y install libpng16-16 libpng-tools libpng-dev
  - npm i npm@latest -g
  - npm install
  - npm audit fix --force
  - npm run prod
artifacts:
  expire_in: 30 days
  paths:
    - public/*
```

Adjuntamos el archivo completo de ejemplo para su registro, que corresponde al proyecto de kioskos de ChileAtiende.

https://git.gob.cl/chileatiende/kioskos/blob/dev/.gitlab-ci.yml

```
xxxxxxxxxx
image: docker:stable

stages:
- test
- build

test:
image: ruby:2.3.3
stage: test
services:
  - postgres:9.6
variables:
  POSTGRES_USER: test
  POSTGRES_PASSSWORD: test-password
  POSTGRES_DB: test_db
  DATABASE_URL: postgres://${POSTGRES_USER}:${POSTGRES_PASSSWORD}@postgres/${I
  RAILS_ENV: test
  CHILEATIENDE_TOKEN: KAZXSW3Hpyc9PGqa
  SIMPLE_URL: http://simple-cer.digital.gob.cl
  GET_CLAVEUNICA_URL: https://apis.digital.gob.cl/claveunica/users
  GET_CLAVEUNICA_TOKEN: 123
  MOC_SECRET: 1234567890
  SMTP_ADDRESS: smtp.sendgrid.net
  SMTP_DOMAIN: totems.cl
  SMTP_PORT: 587
  SMTP_SECRET: SG.s99lfHKTR_imP8_mCX-Svg.myTEdasdaeefeqTeW1yOuQj9VTqxLAg_WTGE2
  SMTP_USER: apikey
  SMTP_FROM_EMAIL: info@totems.cl
  PHANTOMJS_VERSION: 2.1.1

script:
  - apt-get update -qy
  - apt-get install -y nodejs
  - apt-get install -qq -y --no-install-recommends build-essential nodejs libp
   - curl -L -O https://bitbucket.org/ariya/phantomjs/downloads/phantomjs-$PHA
 tar.bz2
  - tar xvfj phantomjs-$PHANTOMJS_VERSION-linux-x86_64.tar.bz2
  - cp phantomjs-$PHANTOMJS_VERSION-linux-x86_64/bin/phantomjs /usr/local/bin
  - rm -rf phantomjs-$PHANTOMJS_VERSION-linux-x86_64
  - bundle install
  - bundle exec rake db:create
  - bundle exec rake db:migrate
  - bundle exec rails test

code_quality:
stage: build
variables:
  DOCKER_DRIVER: overlay2
allow_failure: true
services:
  - docker:stable-dind
script:
  - export SP_VERSION=$(echo "$CI_SERVER_VERSION" | sed 's/^\([0-9]*\)\.\([0-9
  - docker run
      --env SOURCE_CODE="$PWD"
      --volume "$PWD":/code
      --volume /var/run/docker.sock:/var/run/docker.sock
      "registry.gitlab.com/gitlab-org/security-products/codequality:$SP_VERSIO
artifacts:
  paths: [gl-code-quality-report.json]

build:
stage: build
variables:
  DOCKER_DRIVER: overlay2
services:
  - docker:stable-dind
script:
  - docker login git.gob.cl:4567 -u gitlab-ci-token -p $CI BUILD TOKEN
```

GUÍA TÉCNICA

Lineamientos para Desarrollo de Software

Diciembre 2018

gob digital CHILE

Ministerio Secretaría General de la Presidencia

Gobierno de Chile