COMISIÓN DE REGULACIÓN DE COMUNICACIONES
REPÚBLICA DE COLOMBIA

Arquitectura de referencia Sistema de Información

Tecnologías y Sistemas de Información
Coordinador: Ingrid Picón.
Líder: Helder Cerón.

www.crcom.gov.co
@CRCCol /CRCCol /CRCCol CRCCOL

CONTENIDO

1. Lineamientos 3
2. ¿A quién va dirigido? 3
3. VISTA LOGICA 4
4. MODELO DE ARQUITECTURA CONCEPTUAL BASE 6
4.1 MODELO DE ARQUITECTURA MICROSERVICIOS BASE 11
4.2 ARQUITECTURA LOGICA BASE 12
4.3 TECNOLOGIAS 14
4.4 METODOLOGIAS Y RECOMENDACIONES DEL DESARROLLO 15
4.5 INTEROPERABILIDAD MEDIANTE EL BUS DE SERVICIOS 15
4.6 SEGURIDAD DEL SISTEMA DE INFORMACION 15
4.6.1 SEGURIDAD DE LA PLATAFORMA 15
4.6.2 SEGURIDAD DE LA APLICACION 16
5. Firmas y Aprobación 21

INFORMACIÓN DEL DOCUMENTO

Nombre del Proyecto:	Sistema de Información de Tramites		
Preparado por:	Helder Cerón	**Versión del Documento No:**	4.0
Título:	Avance N° 1 Arquitectura de referencia Sistema de Información Trámites CRC	**Fecha de Versión del Documento:**	30 abril 2020

Lista de Distribución

Descripción	Rol	Fecha	Correo electrónico
Helder Cerón	Arquitecto de TI	11/11/2020	Helder.ceron@crcom.gov.co

Historial de Cambios del Documento

Versión	Fecha de Versión	Actualizado por	Descripción
1.0	17/05/2020	Helder Cerón	Creación de modelo base
2.0	08/06/2020	Helder Cerón	Actualización de la estructura y tecnología
3.0	08/12/2020	Helder Cerón	Actualización de la estructura y tecnología
3.5	05/02/2021	Helder Cerón	Actualización de la estructura y tecnología
4.0	15/05/2021	Helder Cerón	Actualización de la estructura en interoperabilidad

1. Lineamientos

La base de este documento es la definición del marco de tecnologías soportadas por la oficina de ti para la diseño, desarrollo y entrega de soluciones o plataformas tecnológicas de sistemas de información e informacion.

2. ¿A quién va dirigido?

A el área de TI y desarrollo de la Comisión de Regulación de Comunicaciones (CRC) , en las tareas de documentación, recepción, desarrollo, adquisición o mejoramiento de Sistemas de Información; A quienes tiene a su cargo la gestión y administración de las plataformas de los sistemas de informacion; A quienes realizan procesos de auditoría de las plataformas , de gestión de la calidad y de gestión de la seguridad de la información; A los responsables De los Sistemas De Información; En general a todos los que requieran de este instrumento para alguna de sus tareas.

3. VISTA LOGICA

La vista lógica de la arquitectura muestra los componentes internos y los aspectos que determinan la construcción de la aplicación.

Los siguientes esquemas arquitectónicos cumplen con el patrón "N Layers + I" el cual va a ser el lineamiento de construcción de la solución, y que permite sea implementada por bloques de

construcción pertenecientes a capas específicas de funcionalidad que van desde los datos hasta el cliente final.

Este tipo de arquitecturas ofrecen un manejo más claro de alta cohesión, bajo acoplamiento al permitir definir responsabilidades propias a cada capa, de igual manera hacen que sea más sencillo que la solución esté abierta a la extensión y cerrada a la modificación.

La naturaleza del proyecto junto con los requerimientos, reflejan los principios y objetivos estratégicos de la entidad, que opta por la aplicación de arquitecturas robustas que maximicen la entrega de requerimientos no funcionales, la mantenibilidad, la viabilidad y adaptabilidad de la solución a futuro, de tal manera que se fomente la integralidad de datos y aplicaciones entre las diferentes intensiones tecnológicas que actualmente se están desarrollando.

Estas son las principales características de la arquitectura clásica:

La plantilla arquitectónica seleccionada para todos los sistemas de información corresponde a una arquitectura tipo "N-Layers + I", utilizando arquitectura J2EE o Microsoft .NET.

- **Flexibilidad y reutilización de código**: Existe una tendencia mundial hacia la construcción de grandes aplicaciones a partir de pequeños componentes. Se pueden construir nuevas aplicaciones o cambiar la interfaz de usuario sin tener que realizar un esfuerzo adicional en la implementación de la lógica de negocio o viceversa.
- **Separación de intereses**: Cada capa tiene su responsabilidad propia e interés, así como los componentes en su interior, esto permite que la lógica sea reutilizable y que los bloques de construcción de la solución puedan ser cambiados según nuevas modificaciones sin afectar la lógica cohesiva de otras capas.

Cumplimiento de Requerimientos no Funcionales:

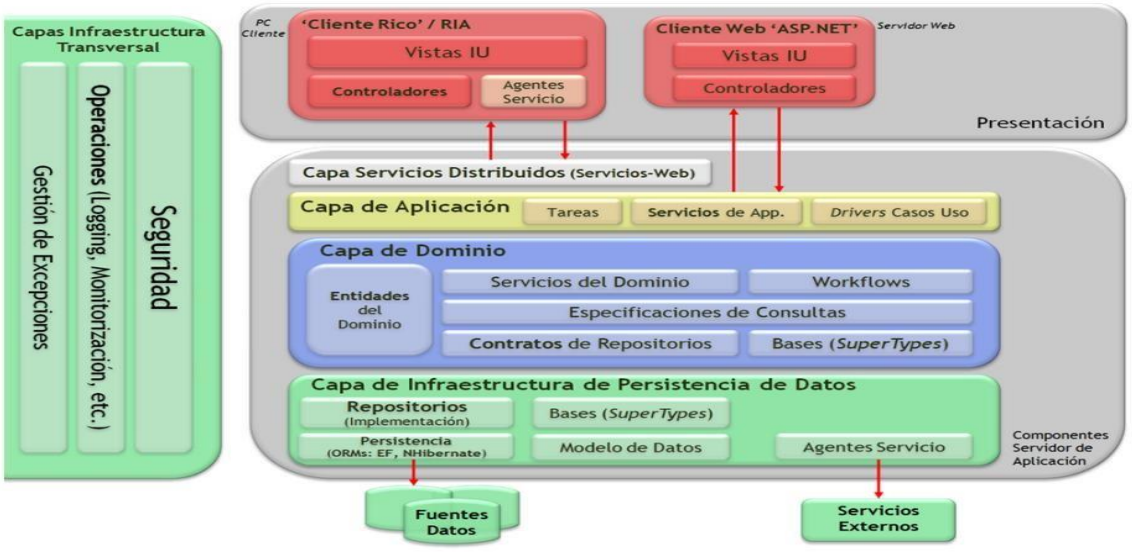
- **Extensión y Adaptabilidad**: Al tener una capa de servicios independientes se puede exponer la misma lógica para diferentes conjuntos de funcionalidades requeridas en la aplicación.

- **Mantenibilidad y Flexibilidad**: La aplicación se basa en un estándar de arquitectura con tecnología nueva que permite que la aplicación pueda mantenerse adecuadamente y con la flexibilidad que permite JAVA, C# y/o PHP.
- **Escalabilidad**: Por el principio de separación de intereses y de acuerdo con un plan de capacidad se podrá separar en nodos diferentes de tal manera que el procesamiento se independice.

A continuación, se presentan los esquemas correspondientes a la arquitectura base

4. MODELO DE ARQUITECTURA CONCEPTUAL BASE

Presenta una agrupación funcional de los artefactos, basándose en el estilo arquitectónico multicapas (N-Tier) y orientado a servicios, que va en pro de mejorar la escalabilidad, disponibilidad, mantenibilidad y utilización de los recursos de la solución que se construya.



A continuación, se realizará una descripción de cada una de las capas y los componentes indicados en la gráfica previa:

- **Capa de Presentación**: Esta capa se encarga de soportar toda la interacción con el usuario. El lineamiento de construcción de la capa de presentación corresponde al modelo "razor" el cual coloca el código en un embebido en el html dando un compacto de la página en caso de que sea una solución .NET o Java, este último debe contemplar características similares como JSF o Primefaces.

Como principal característica base los nombres de los archivos reportan en modelos vista controlador en donde están basados en el nombre del archivo. cshtml para la vista y dos .cs para el controlador y modelo, la vista posee los siguientes elementos:

- **JavaScript**: Componente de lógica encargado de validaciones del lado del cliente, y algunas funciones gráficas que no deben fluir hasta el servidor.

- **Bootstrap:** Conjunto de controles enriquecidos cuyo comportamiento genérico es el de recoger y enviar información desde el cliente al servidor sin ningún tipo de recarga de la página. Lo anterior repercute en una interfaz atractiva y al mismo tiempo efectiva.

- **CSS3**: Corresponde a las hojas de estilos CSS (Cascade Style Sheet) que marcan la experiencia de usuario con la aplicación.

- **HTML5:** Versión 5 del lenguaje básico de la World Wide Web.

- **Administrador de contenidos**: Componente que brinda la administración del contenido, estilos e imágenes.

- **Capa de Aplicación**: esta capa encapsula y ejecuta toda la lógica de negocio y de transformación necesaria para completar la funcionalidad requerida.

- **Controladores**: contiene todos los procesos lógicos del negocio definidos en el análisis y a su vez necesarios para cumplir con los requerimientos funcionales a partir de los eventos que ocurran en la capa de presentación hasta ser atendidos de forma completa.

- **Adaptadores DTO**: Contiene toda la lógica de transformación y mapeo entre objetos de transporte y objetos de datos. Sirve para marcar la separación entre la capa de datos y la capa de presentación.

- **Capa de Modelo de Dominio**: encapsula toda la lógica relacionada con el acceso a los datos y la gestión de persistencia.
  - **Entidades de Dominio**: Conjunto de clases (poco's), donde cada clase es una representación tangible de una tabla de la base de datos.

  - **Servicios de Datos**: Conjunto de clases que contienen los métodos orientados para gestionar la lógica de acceso a datos agrupados por especialidades y responsabilidades específicas.

  - **LINQ**: Language Integrated Query (LINQ) es un componente de la plataforma Microsoft .NET que agrega capacidades de consulta a datos de manera nativa a los lenguajes .NET.

- **Capa de Persistencia de Datos**: Está conformada por un conjunto de componentes que establecen la infraestructura necesaria para gestionar la persistencia de datos. Particularmente la solución está construida sobre Entity Framework en modo Model First.
  - **Contexto de Datos**: representa a los espacios de trabajo que son utilizados para acceder a la base de datos.

- **Modelo / Mapping**: Contiene la representación del modelo de datos en un marco orientado a objetos, donde se establecen y respetan de manera clara y precisa la estructura de las entidades de negocio y su cardinalidad

- **JPA**: El marco de desarrollo propuesto por la plataforma Java se denomina Java Persistence API (JPA), cuya implementación más extendida es Hibernate. Sobre ambas se han definido pautas seguir para mejorar su uso.

- **JPA**: El marco de desarrollo propuesto por la plataforma Java se denomina Java Persistence API (JPA), cuya implementación más extendida es Hibernate. Sobre ambas se han definido pautas seguir para mejorar su uso.

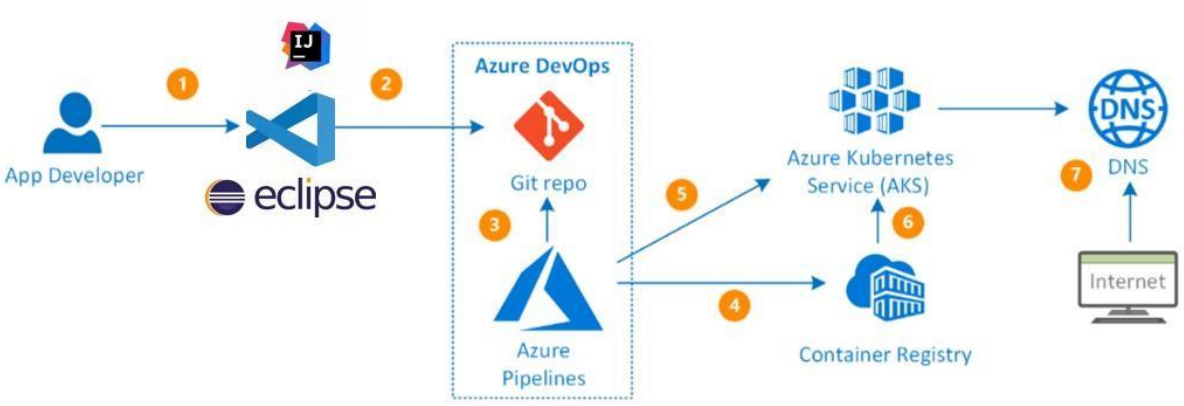
- **Componentes transversales:** Este conjunto de funcionabilidades que no son de uso de una única capa o sección, si no que han de ser usados por cualquier capa bajo una directrices definidas y finitas.

  - **Seguridad**: Encapsula toda la lógica y componentes relacionados con los procesos de autenticación y autorización. Esta capa está implementada mediante autenticación con:
    - **Usuarios internos**: Autenticación por ACTIVE DIRECTORY mediante el servicio web de interoperabilidad provisionado por la oficina de TI para dicha integración.
    - **Usuarios internos**: Estarán asociados al modelo de datos de su solución particular y los datos base estarán sobre el repositorio de datos maestros de la entidad.
  - **Auditoria**: Encapsula toda la lógica tendiente a generar seguimiento sobre las acciones de tipo funcional sobre la solución, en cuanto a:
    - Informacion nueva ○ Informacion antigua ○ Tipo crude (UPDATE, INSERT, DELETE) ○ Fecha del evento ○ Usuario de la plataforma.
    - Sección/modulo/formulario
  - **Cache**: Contiene un conjunto de variables del tipo aplicación y sesión orientadas a mitigar el impacto a la base de datos, permitiendo aplicar buenas prácticas para no generar consumo de recursos innecesarios sobre datos que son paramétricos.
  - **Microservicios**: Representan un estilo de arquitectura y un modo de programar software. Con los microservicios, las aplicaciones se dividen en sus componentes

más pequeños, y son independientes entre sí. A diferencia del enfoque tradicional y monolítico de las aplicaciones, en el que todo se compila en una sola pieza, los microservicios son independientes y funcionan en conjunto para llevar a cabo las mismas tareas. Cada uno de estos elementos o procesos es un microservicio. Este tipo de arquitectura se debe implementar cuando los **componentes de la aplicación son aplicaciones independientes.**

4.1 MODELO DE ARQUITECTURA MICROSERVICIOS BASE

Como parte de las funcionalidades de la arquitectura orientada a microservicios se podrá disponer de acceso al Docker Registry de la plataforma y GIT, donde se almacenen las diferentes imágenes de contenedores debidamente versionadas, funcionalidades para realizar Pruebas Unitarias con Junit, por ejemplo, y funcionalidades de compilación y construcción de pods usando características Azure Kubernetes Service (Source to Image) para construir contenedores, desplegar, orquestar y actualizar contenedores mediante el código tomado del repositorio GIT más la imagen Docker base que el desarrollador defina partir de los insumos indicados en este documento., como se muestra a continuación:



Para lograr esta estrategia de implementación se requerirán los siguientes componentes desplegados como contenedores:

- **Automatización de pruebas (Pipelines) :**es un entorno de pruebas de software para aplicaciones. **Pipelines** provee una herramienta de grabar/reproducir para crear pruebas sin la necesidad de usar un lenguaje de scripting para pruebas**.**

- **Calidad de Código (Test-plan):** es una plataforma para evaluar código fuente. Para obtener métricas que pueden ayudar a mejorar la calidad del código de un programa mediante una trazabilidad completa.

- **GIT:** Es un repositorio de software que se convierte en el lugar de almacenamiento del cual pueden recuperados e instalados los paquetes de software de la plataforma

4.2 ARQUITECTURA LOGICA BASE

La arquitectura lógica de la aplicación se debe caracterizar por:

- **Calidad de Código (Test-plan):** La solución deberá ser construida usando el Framework 4.5 de .Net, java 1.8 o superior, bajo la arquitectura tanto lógica como física implementada por la comisión de Regulación de Comunicaciones**.**

- **El lenguaje** de programación deberá ser JAVA, C#, PHP o PYTHON.

- **La implementación de tecnologías**, ensamblados y/o API's de terceros deberá ser aprobada por la comisión de Regulación de Comunicaciones.

Deberán usarse los siguientes estilos arquitecturales: ○
N capas.

- ○ Cliente / servidor ○ Basado en componentes ○ Diseño dirigido por dominios ○ Orientado a objetos
- ○ La solución deberá ser particionada en N Capas ○ **Cada capa deberá poseer un diseño cohesivo** ○ Integración con el directorio activo mediante los servicios expuesto por la entidad.
- ○ Integrada al repositorio de datos maestros de empresas. mediante los servicios expuesto por la entidad.
- ○ Se deberán aplicar patrones estándar de arquitectura para que la dependencia en las capas está basada en abstracciones y no dependiendo una capa directamente de la otra.
- ○ **La solución deberá** superar las pruebas de vulnerabilidad e intrusión TOP 10 de OWASP
- ○ Se debe implementar seguridad en los Servicios Web, tokens de seguridad y firmado de tramas.

- **Toda la estructura** de la arquitectura de la solución debe estar desacoplada con base en el principio de inversión de dependencias (IoC) y contenedores DI(inyección de dependencia).
- La Comisión de Regulación de Comunicaciones proveerá el marco de referencia que soporta la implementación de la arquitectura definida (Estructura de directorios para la solución y distribución de proyectos).
- La Comisión de Regulación de Comunicaciones realizará auditoria y validación técnica, de seguridad y de las mejores prácticas en el desarrollo e implementación de software.
- Dependiendo del tipo de sistema pueden hacerse opcionales algunos de los requerimientos establecidos en este documento, pero esto sólo si es aprobado por la comision reguladora de comunicaciones.
- La solución deberá ser construida usando el Framework 4.5 de .Net, java 1.8 o superior, bajo la arquitectura tanto lógica como física implementada por la comisión de Regulación de Comunicaciones.
- El paso a producción lo hace la entidad, mediante el comité de paso a producion. Se debe entregar manual de instalación, técnico y de operación. Deben generarse las capacitaciones correspondientes.

4.3 TECNOLOGIAS

La solución debe ser desarrollada usando java en su verision1.7 o superior, C# en .Net Framework 4.5, PHP o PYTHON. A continuación, se detallan las tecnologías a usar de acuerdo con cada capa:

- Microsoft Enterprise Library 5.0 ○ Entity framework 5.0 o superior ○ SQL Server 2016 o superior ○ Windows Server AppFabric

- Jquery/Utilidades (última versión) – Boostrap (Controles RIA - Responsive) ○ Ajax ○ HTML5 ○ CSS3
- WWF (Windows Workflow Foundation) , es una tecnología de Microsoft que proporciona una API, un motor de flujo de trabajo en proceso y un diseñador rehospedable para implementar procesos
- WCF (Windows Communication Foundation), es la plataforma de mensajería que forma parte de la API de la Plataforma
- RESTFUL

4.4 METODOLOGIAS Y RECOMENDACIONES DEL DESARROLLO

Dentro del documento "**GUIA METODOLOGICA PARA EL DESARROLLO DE PLATAFORMAS**" encontramos las recomendaciones dadas por la coordinación de TI para la implementación de plataformas.

4.5 INTEROPERABILIDAD MEDIANTE EL BUS DE SERVICIOS

Dentro del documento "**LINEAMIENTOS DE INTEROPERABILIDAD**" encontramos las recomendaciones dadas por la coordinación de TI para las integraciones entre sistemas de informacion.

4.6 SEGURIDAD DEL SISTEMA DE INFORMACION

4.6.1 SEGURIDAD DE LA PLATAFORMA

- **Optimización y hardening de componentes**: Para el desarrollo de los sistemas de información cualquiera que este sea, se requiere que el proveedor y/o contratista presente las guías que considera necesarias en el proceso de hardening o aseguramiento de servidores, y/o servicios a instalar, lo cual permitirá:

    - Aseguramiento de bases de datos
    - Aseguramiento de servicios Web, si son requeridos
    - Aseguramiento de servidores de aplicaciones
    - Aseguramiento de Sistemas Operacionales

En caso de que no se requiera ninguna guía o no la posee, debe garantizar que ninguna de las reglas de hardening de la organización deban ser cambiadas, refiriéndose exactamente a Cambio en los privilegios de acceso, Cambio en los privilegios de usuario, Cambio en las configuraciones existentes de servicios o servidores y cambio en las configuraciones de seguridad de los elementos de control de la organización.

- **Pruebas de Intrusión**: Es necesario que se realicen una vez entregada la aplicación pruebas de intrusión del mismo software por parte de quien recibe el sistema de información de tal manera que se garantice el cumplimiento de los requerimientos de seguridad, en caso de encontrar fallas de seguridad el proveedor debe garantizar que estas sean corregidas y realizar una reprueba para la misma durante el proceso de entrega por parte del proveedor

4.6.2 SEGURIDAD DE LA APLICACION

- **Validación de los datos de entrada y salida**: Se debe verificar que los datos son los apropiados y con los formatos que se esperan en pro de evitar posibles ataques de inyección o

código malicioso. Para todas las situaciones de entrada y salida de datos se debe realizar los procesos de validación.

- **Diseño simple**: Los mecanismos de seguridad de la aplicación debe ser lo más simple y sencillo posible, de tal forma que la facilidad para el usuario final no se pierda. La probabilidad de que los pasos para proteger una aplicación compleja no se cumplan es muy alta.

- **Utilización de mecanismos de cifrado de datos**: Es muy importante la utilización de SSL o TLS al interior de la aplicación teniendo en cuenta que esta solo garantiza la protección a través del cifrado de los datos entre el servidor y el cliente, pero se debe tener presente que no garantiza la seguridad de la aplicación sin embargo brinda comienza a los usuarios que llegan a las pagina s públicas.

- **Verificación de privilegios**: Es importante que el sistema sea diseñado con la menor cantidad de privilegios posibles.

- **Ofrecer la mínima información**: Es indispensable y de carácter obligatorio que, en caso de un error o una validación negativa, la aplicación responda ofreciendo la menor cantidad posible de información.

- **Documentación de Información**: La aplicación debe tener de manera clara la documentación del software todos sus componentes, así mismo la documentación de las configuraciones necesarias en todo el proceso requerido para su funcionamiento.

- **Usuarios**: Es importante que se realice de manera adecuada la autenticación del usuario teniendo en cuenta, un esquema de autenticación basado en la definición del modelo de roles y perfiles, centralizado y escalable.

- **Rol**: El Rol es el nombre que se le confiere al conjunto de perfiles que le son asignados al usuario para el ejercicio de sus funciones.
- **Perfil**: Es la descripción detallada de las posibles transacciones que puede realizar un usuario en el sistema.
- **Usuario**: Persona que interactúa con el sistema. Por lo tanto, un usuario definido para el sistema debe tener definido un rol dentro del mismo que tiene un perfil asociado con el cual puede realizar las operaciones del sistema. Es importante en este modelo garantizar.
- **El cifrado de la contraseña**, para las siguientes situaciones:
    - Almacenamiento en Base de Datos ○ Viaje a través de la red, para integrarse con otros sistemas de autenticación.
    - Soporte de esquemas fuertes de contraseña, que contemple caracteres especiales ascii.

- **Cuidado** al momento del manejo de sesiones para evitar ataques de hombre en el medio, en el caso del manejo de sesiones se deben autenticar las sesiones a ser utilizadas para las operaciones, y consideraciones en el manejo de los tiempos de vida de las sesiones. El desarrollo de aplicaciones con autenticación de los usuarios debe evitar los procesos de recordar contraseñas en los browsers, en caso de utilizarse.
- **Auditoria y registro**, Es importante que se registren las actividades de la aplicación a desarrollar, diferente de los eventos registrados a través de la infraestructura implementada para la operación del sistema de información o plataforma. Sin ser los únicos (logs de sistema operacional, bases de datos, servidores de aplicaciones, servidores de bases de datos, y demás componentes de la infraestructura tecnológica) para ello es indispensable que se registre:
    - Tiempo en el que un evento se presenta. ○ Quien realiza la acción.
    - Descripción de la actividad realizada.

- **Interfaces de Administración**: Deben estar totalmente segregadas de las funciones de usuario final, los usuarios convencionales no deben utilizar de ninguna forma dicha interfaz, es

indispensable que se provee del suficiente nivel de auditoria, de igual manera si un usuario está asociado a un perfil, este no debe poder ejecutar actividades de otro perfil, a menos que sea autorizado.

- **Cuentas por defecto:** Para ningún caso, sea en las interfaces de administración y las bases de datos se pueden dejar habilitados y/o configuradas cuentas por defecto, y por ninguna circunstancia debe realizarse conexiones a la base de datos con usuario administrador.

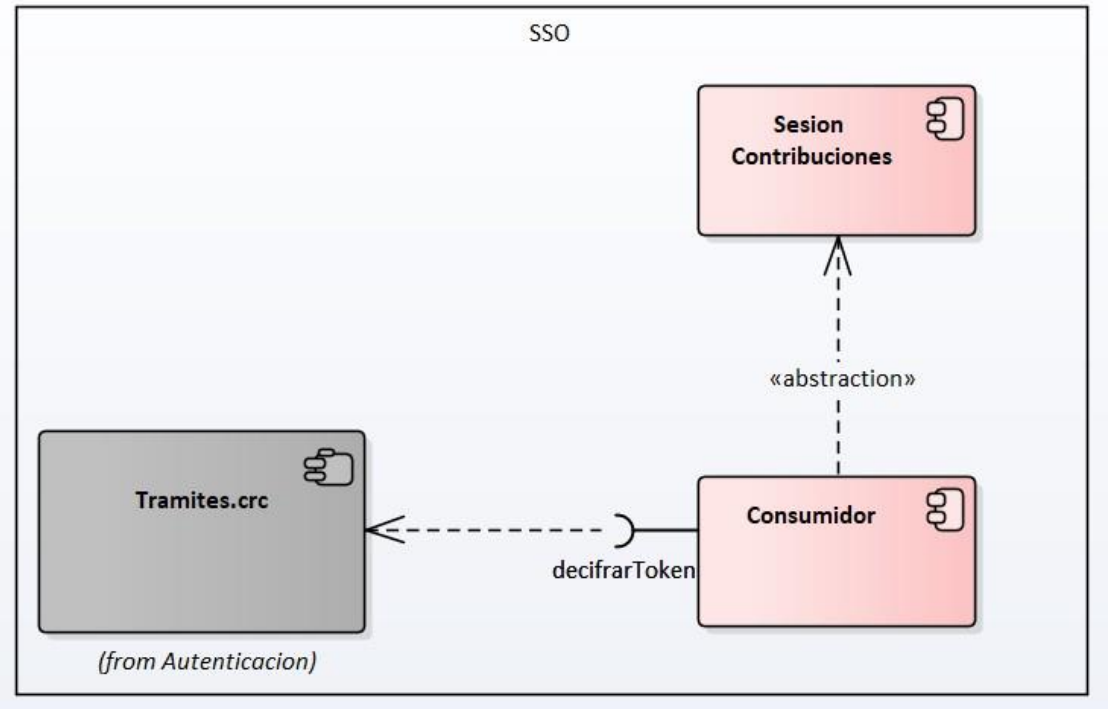
- **Longitud de contraseñas**: En caso de ser necesarias, estas deberán tener una longitud mínima de seis caracteres y deben validar la utilización y combinación de letras y números para la creación de contraseñas. No se deben permitir por ninguna razón password en blanco o con menos de la cantidad especificada, de igual manera la contraseña no debe ser igual a los nombres o apellidos de la persona que realiza el ingreso, de acuerdo los estándares de la industria y recomendaciones.

- **Autenticación**: Es importante que el proceso de ingreso a la aplicación a través de un sistema de credenciales valide el número de intentos ejecutados para tal fin de tal manera que al momento de más de tres ingresos debe bloquear al usuario y registrar en los sistemas de logs de la aplicación dicha operación, igualmente realizar la destrucción total de la sesión utilizada para tal fin.

- **Múltiple factor**: método de control de acceso a sistemas o plataformas en el que a un usuario se le concede acceso al sistema solo después de que presente dos o más pruebas diferentes de que es quien dice ser.

- **Single Sign On**: Permite a los usuarios tener acceso a múltiples aplicaciones ingresando solo con una cuenta a los diferentes sistemas y recursos. En proyectos de referencia como tramites

y contribuciones se aplica este concepto para saltar de tramites a contribuciones, como se muestra a continuación:



- **Tiempos de Vida**: Es importante validar y determinar los tiempos de inactividad dentro del sistema, prudente un tiempo máximo de 15 minutos dentro del mismo, después de este tiempo de inactividad debe destruirse la sesión y realizar el proceso de logoff del sistema

5. Firmas y Aprobación

Fecha de Firma _/_/_

_____

Helder Cerón

Arquitecto de TI _____

Ingrid Picón

Coordinadora Tecnologías y Sistemas de Información