Proyecto:

Sistema de Gestión Operacional de Insumos para Laboratorios Clínicos (LABx)

Por Fabián A. Callejas, Julián Gómez, Julián A. Galeano y José L. Jiménez

Revisión: 1.0, basado en la plantilla 7.0 ES (basada en asciidoc), Enero 2017

# 

# Introducción y Metas

El laboratorio LABx, se encuentra administrando el inventario de una forma manual, en la cual se están dejando de lado la normatividad establecida por el INVIMA; de este modo a se está generando el problema de no contar con una trazabilidad completa entre el proveedor y el cliente.

Puesto que el proceso es manual, buscar la trazabilidad de cierto insumo es una tarea ardua debido a que todos los registros que tiene ese insumo pueden estar dispersos entre varios documentos físicos, haciendo que un operario de laboratorio se demore más realizando dicha búsqueda.

Al momento de ingresar los insumos que llegan al laboratorio se torna en una tarea larga y esclavizante por la cantidad de registros que se deben llevar en hojas, ocasionando errores y demoras en el proceso.

Puesto que el manejo es en hojas la actualización del inventario puede tener errores o puede no darse en tiempo real desconociendo la cantidad actual del insumo.

Se tienen problemas para identificar que productos están próximos a vencer debido a que la información está dispersa.

No se tiene información exacta para generar reportes y gráficas precisas acerca de los insumos.

No se está generando un reporte de reactivo de vigilancia debido a que el proceso es manual, incumpliendo la normatividad INVIMA.

Se está generando un alto costo operativo debido al tiempo que es necesario para la categorización manual al ingresar los insumos al inventario.

Objetivos:

* Disminuir los eventos adversos en laboratorio por el uso de reactivos en mal estado en un 20% antes del 31 de diciembre.
* Reducir el tiempo necesario para el ingreso de reactivos al Kardex en un 25% cada semestre.
* Reducir el tiempo de generación de informes al 40% en los primeros 6 meses.
* Tener un inventario sistematizado y en línea en los próximos 4 meses.
* Tener informes automáticos de regulación por ley (INVIMA y otros) en el primer año.

Metas:

* En 1 año se puede obtener la trazabilidad sobre el 40% de los insumos almacenados en el Laboratorio.
* Aumentar un 50% la productividad de los operarios en 10 meses.
* En 1 año reducir los costos operativos en un 30%.
* En 10 meses reducir los errores al registrar insumos en un 50%

Interesados:

* Directivas, esperan que el software mejore la eficiencia aumentando la productividad del Laboratorio, así como que evite las pérdidas de reactivos por vencimiento y sanciones de los entes reguladores por el inadecuado uso del inventario.
* Personal de laboratorio, garantizaría una mayor eficiencia en el trabajo y una capa más de seguridad para los procedimientos y pacientes.
* Proveedores, el software garantizaría una mayor exactitud en los pedidos de productos, por lo que se evitaría tener o despachar reactivos que no necesitan rompiendo cadenas de frio y evitando logística.

## 1.1. Vista de Requerimientos

Se pretende desarrollar un software web de inventarios que permita llevar una trazabilidad completa entre el proveedor y el cliente, que al mismo tiempo permita al cliente final administrar su inventario de acuerdo a la normatividad establecida por el INVIMA.

Los despachos realizados por el proveedor a cada cliente llevaran un código el cual el cliente en el momento de recibir el pedido cargará automáticamente en su propio inventario. Además, el Sistema de Inventario debe contar con las siguientes características:

* Software amigable con búsquedas intuitivas y rápidas.
* Manejo inteligente de productos e inventario: búsqueda de inventario por familias, lotes, marcas y fechas de vencimiento.
* Alertas de stock (mínimos y máximos), posibilidad de configuración de alertas por e-mail.
* Generación de informe de productos próximos a vencer y posibilidad de exportar a Excel.
* Generación de informe de productos vencidos y posibilidad de exportar a Excel.
* Generación de gráficas y promedio de consumo de productos.
* Clasificación y seguimiento de productos por área o familia.
* Manejo de información detallada de cada producto: proveedor, código, cantidad de reactivos, presentación, clasificación de riesgo, certificado INVIMA y temperatura de almacenamiento.
* Manejo de cantidades de cada producto por fracción o por unidades.
* Manejo y seguimiento de lotes.
* Manejo y seguimiento de fechas de vencimiento.
* Manejo de código de colores para fecha de vencimiento (verde, amarillo y rojo).
* Reactivo vigilancia, con todos los parámetros exigidos por ley.
* Informe de reactivo vigilancia generado automáticamente y con la posibilidad de exportarlo fácilmente a Excel.
* Exportación de inventario e informes a Excel.
* Generación de seguimiento de entrega de productos y generación de comprobante de entrega automático.

El sistema de gestión del Sistema de Inventario debe contar con las siguientes características de administración:

* Seguridad de acceso al software y encriptación de la base de datos.
* Manejo de perfiles de usuarios (Control por roles).
* Auditoria completa y continua de acceso y cambios realizados por los usuarios.
* Número de usuarios ilimitado (restringido a la negociación).
* Cantidad de usuarios recurrentes y conexiones ilimitadas.
* Generación de reportes.
* Posibilidad de generar nuevos reportes de acuerdo a la necesidad.
* Tablero de control gerencial con control de gastos vs producción.

La motivación principal es la de eliminar el proceso de manejo de inventario manual que se lleva a cabo en el Laboratorio Clínico, contando con un control online del mismo que permita la reducción de costos por desperdicios, que también garantice la seguridad de los usuarios y la reducción del margen de error en los procedimientos del Laboratorio.

## 1.2. Metas de Calidad

|  |  |  |
| --- | --- | --- |
| **Prioridad** | **Objetivo de Calidad** | **Escenario** |
| 1 | Disponibilidad | El requerimiento del negocio, el cual demanda que los laboratorios trabajen 24x7x365 exige alta disponibilidad para el uso de todos los usuarios. |
| 2 | Eficiencia de desempeño | La respuesta del sistema debe ser en el menor tiempo posible para garantizar que inventario en tiempo real y se pueda consultar el estado de vencimiento de los reactivos. |
| 2 | Corrección funcional | El software debe estar en la capacidad de demostrar los resultados correctos con el nivel de precisión requerido principalmente en las fechas de vencimiento. |
| 3 | Confidencialidad | Se debe restringir el acceso a la aplicación. |
| 3 | Integridad | Manejar roles que tienen diferentes responsabilidades y capacidades dentro del mismo. |
| 4 | Operabilidad | El software debe ser intuitivo y fácil de manejar por los usuarios. |
| 5 | Tolerancia a fallos | El software debe estar en la capacidad de recuperarse ante cualquier falla. |
| 5 | Portabilidad | El software debe ejecutarse en distintos dispositivos. |

## 1.3. Partes interesadas (Stakeholders)

|  |  |  |
| --- | --- | --- |
| **Rol/Nombre** | **Contacto** | **Expectativas** |
| *Directivas* | *Iván Uribe* | *Mejora eficiencia*  *Evitar pérdidas de reactivos*  *Evitar sanciones del INVIMA* |
| *Coordinación de Laboratorio* | *Claudia Clavijo* | *Mejora eficiencia*  *Evitar pérdidas de reactivos*  *Evitar sanciones del INVIMA*  *Asegurar resultados confiables* |
| *Bacteriólogo(a)* | *Diana Sánchez* | *Tener control de los reactivos*  *Garantizar la seguridad del paciente*  *Garantizar la efectividad de las pruebas*  *Uso adecuado de reactivos* |
| *Auxiliar de Laboratorio* | *Katherine Herrera* | *Tener control de los reactivos*  *Uso adecuado de reactivos* |
| *Proveedor* | *Alexander Parra* | *Mejorar el proceso de pedidos*  *Mejorar el proceso de entregas* |

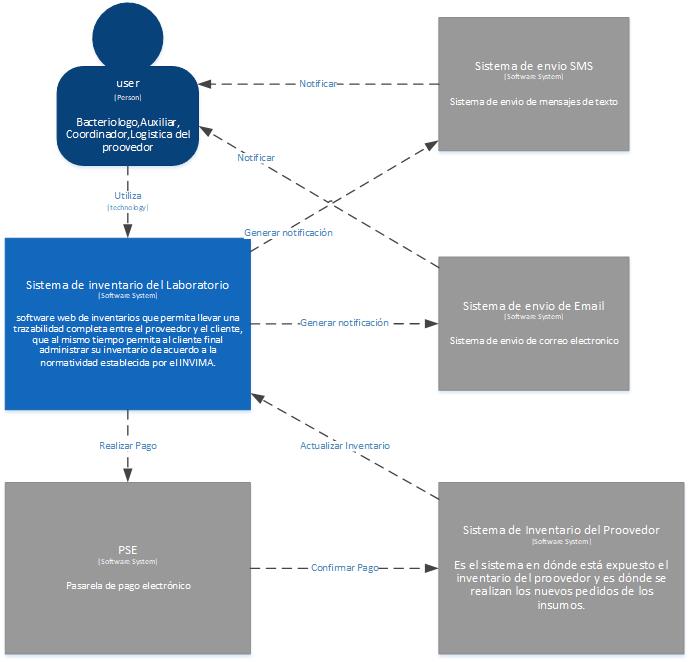
# Restricciones

## 2.1. Restricciones de la Arquitectura

LABx debe:

* Debe ser multiplataforma y trabajar con los sistemas operativos más comunes (Windows, Linux, macOS, Android y iOS).
* La aplicación debe ser web responsive.
* Debe ser realizando con una arquitectura de microservicios.
* Debe seguir los lineamientos de la norma de reactivo vigilancia vigente expedida por el INVIMA.
* Se debe utilizar una base de datos OpenSource preferiblemente MySQL ó MariaDB, debido principalmente a la reducción de costos.
* Se requiere una base de datos centralizada por que se requiere la entrega de la misma por parte de los clientes a las autoridades para ser auditada.
* Se deben enviar alertas de mensajes por SMS y por email para informar a los usuarios y proveedores cuando algún producto se está acabando y cuando ya se terminó.

# Alcance y Contexto del Sistema

**3.1. Contexto de Negocio**

|  |  |
| --- | --- |
| **Sistemas Vecinos** | **Descripción** |
| Usuarios | Los usuarios interactúan con LabX a través de un usuario y contraseña el cual se valida mediante un token, cada usuario tiene un rol asignado que permite su interacción y acceso a las capacidades del sistema. |
| Sistema de envío de mensajes de texto SMS | LabX envía información de los productos próximos a terminarse y a vencerse al sistema de envío de mensajes de texto SMS para que este pueda enviar las notificaciones a los usuarios. |
| Sistema de envío de email | LabX envía información de los productos próximos a terminarse y a vencerse al sistema de envío de email para que este pueda enviar las notificaciones a los usuarios. |
| Sistema inventario del proveedor | LabX permite el acceso a los proveedores de manera que estos puedan actualizar el inventario en sus despachos, así mismo avisarles de la poca existencia de alguno de los productos del inventario. |
| Pasarela de pago PSE | LabX permite el acceso a la Pasarela de pago PSE a los usuarios con permisos de manera que puedan hacer el pago de los productos a los proveedores por este canal. |

**3.2. Contexto Técnico**



|  |  |
| --- | --- |
| **Nodo/Artefacto** | **Descripción** |
| Github | Plataforma de versionamiento de código basada en git |
| Aplicación Web | Servidor VPS con Ubuntu 18.04 alojado en la nube, que contiene el Docker de la aplicación web |
| Docker Ngx | Docker con un servidor de tipo Ngx |
| APIs | Servidor VPS con Ubuntu 18.04 alojado en la nube, que contiene múltiples Dockers con los binarios de las API |
| Binarios (security, inventario, insumo) | Implementaciones realizadas en Java con Maven con Springboot, y JPA |

# Estrategia de Solución

* LABx tendrá un base de datos distribuida en MySQL, la cual estará alojada en un servidor Linux montado sobre una máquina virtual en la nube de Digital Ocean.
* Se realizarán distintos microservicios los cuales estarán programados en Java.
* Los microservicios escritos en Java usarán conexión a la base de datos por medio de JPA (Java Persistence API).
* La capa de presentación estará implementada en Angular con Bootstrap para que esta sea responsive.

# Vista de Bloque

## Nivel1



**Racional:** Utilizamos la descomposición funcional para separar las responsabilidades:

* Las API encapsulan la lógica del negocio y el manejo de los datos.
* La Aplicación Web realiza la presentación de los datos y comunicación con las apis.
* La se conecta a diferentes APIs de terceros para realizar el envío de notificaciones y el pago de productos al proveedor.

Bloques de construcción contenidos

|  |  |
| --- | --- |
| **Bloques** | **Descripción** |
| Sistema de envío de mensajes de texto SMS | Envío de mensajes de texto SMS con la información de los productos próximos a terminarse y a vencerse para notificar a los usuarios del estado de los insumos. |
| Sistema de envío de email | Envío de correo electrónico con información de los productos próximos a terminarse y a vencerse para notificar a los usuarios del estado de los insumos. |
| Sistema inventario del proveedor | Actualizar el inventario en sus despachos, así mismo avisarles de la poca existencia de alguno de los productos del inventario. |
| Pasarela de pago PSE | Acceso a la pasarela de pago PSE para realizar el pago de los productos al proveedor. |

## Nivel 2

### Caja Blanca Aplicación Web



**Racional:** Utilizamos Angular 8 para el desarrollo del frontend, ya que este propone un desarrollo orientado a componentes y favorece las conexiones con las APIs.

|  |  |
| --- | --- |
| **Bloques** | **Descripción** |
| Login | Es un componente transversal a la aplicación el cual, se encarga de la autenticación del usuario y la administración de la cookie de sesión. |
| Buscar | Componente que se encarga de realizar la búsqueda y los filtros para los distintos insumos. |
| Sacar Insumo | Componente encargado de retirar X cantidad de insumo solicitado por el usuario. |
| Tabla de Insumo | Lista todos los Insumos disponibles en una tabla con opciones de ordenar y filtrar. |
| Detalle del Insumo | Muestra una vista detalle del insumo con todos los campos. |
| Agregar Insumo | Añade un insumo nuevo mediante un formulario. |
| Filtrar Reporte | Componente que se encarga de realizar los filtros de búsqueda y orden a un reporte |
| Tabla de Reporte | Crear una tabla de reporte dependiendo del tipo de reporte seleccionado |
| Tipo de reporte | Componente que tiene filtros predefinidos para la generación de distintos reportes. |
| Detalle de Reporte | Detalle de una celda del reporte con todos sus campos. |
| Provider | Componente que se encarga de conectar los demás componentes a las APIs para su consumo y realizar distintas transacciones. |

## Nivel 2

### Caja Blanca API



**Racional:** El desarrollo en el Backend está realizado en JAVA 8, implementando el framework spring data y JPA, aprovechando los beneficios para obtener un código limpio.

|  |  |
| --- | --- |
| **Bloques** | **Descripción** |
| Administrador de Seguridad | Es un componente encargado sobre la administración de los usuarios para la aplicación, genera el token cuando un usuario ingresa al sistema de información, y validación de la sesión. |
| Inventario | Componente encargado de la gestión del inventario de los productos, lleva el registro de la entrada y salida de las diferentes cantidades del inventario. Controla la fecha de vencimiento del inventario, para bloquearlo dentro del sistema. |
| Reportes | Componente que tiene como objetivo permitir generar el reporte con el detalle de movimiento para los productos registrados en el sistema. |
| Usuario | Componente que tiene la gestión de usuarios del sistema, y maneja el control de acceso a los servicios expuestos por las diferentes API’s. |
| Insumo | Componente encargado sobre la gestión de los productos para el sistema. |

## Nivel 3

Clase Inventario  


|  |  |
| --- | --- |
| **Bloques** | **Descripción** |
| **InventoryController** | Clase que expone los endpoint para la gestión de inventario, en el cual se implementa InventoryServiceImpl |
| **InventoryServiceImpl** | Clase que tiene la lógica sobre el manejo de inventario, y tiene la lógica para el manejo del inventario vencido. Implementa InventoryRepositoryImpl |
| **InventoryRepositoryImpl** | Clase que tiene la implementación de las entidades que se encargan de la persistencia de datos, entre la aplicación y la base de datos. |
| **InventorySystemException** | Clase para el tratamiento de la excepción que se puede dar durante la ejecución del aplicativo. |

# Vista de Ejecución

## Login



1. El sistema en el cliente inicialmente revisa si existe una cookie de sesión activa, en este caso no existe la cookie de sesión
2. El usuario ingresa un Email y un Password , en el cliente y este válida que no esté vacío, y que el correo tenga un formato válido.
3. Se envían las credenciales del Usuario y el Password al provider que se encarga del consumo de la API indicada.
4. Las credenciales se validan en el Backend dentro de Security frente a la Base de datos, las credenciales son correctas
5. Se genera un Token de sesión y se envía el Id de Usuario y el Token
6. Se genera una Cookie de sesión con el Id y el Token recibidos en el Provider
7. El Provider responde con un login exitoso al componente de Login y este deja continuar al usuario.

## Consultar Lista de Insumo



1. El usuario desea ver la tabla de insumos, el usuario se encuentra en sesión
2. Se valida la cookie de sesión con el Login, existe la cookie de sesión y el componente del Login la descifra en su Id y Token.
3. El componente de Insumo realiza la consulta al Provider enviando las credenciales del id y del token en la cabecera de la cookie de sesión.
4. El componente de Security en el backend recibe las credenciales y determina su validez.
5. Las credenciales son validas y el componente de security procede a actualizar el tiempo de vida del token.
6. El componente de Security autoriza al componente de Insumo en entregar los datos de la lista de insumos, el componente de Insumo genera la Lista.
7. El componente de Insumo entrega la lista a Provider; esta la serializa y se la entrega a el componente de Insumo.
8. El componente de Insumo renderiza la tabla con la lista de Insumos serializada.

## Sacar una x cantidad de Insumo del Inventario



1. El usuario desea sacar una cantidad X de insumo del inventario, el usuario se encuentra en sesión.
2. Se valida la cookie de sesión con el Login, existe la cookie de sesión y el componente del Login la descifra en su Id y Token.
3. El componente de Inventario realiza la consulta al Provider enviando las credenciales del id y del token en la cabecera de la cookie de sesión.
4. El componente de Security en el backend recibe las credenciales y determina su validez.
5. Las credenciales son válidas y el componente de security procede a actualizar el tiempo de vida del token.
6. El componente de Security autoriza al componente de Inventario en realizar el cambio de cantidad.
7. El componente de Inventario API, valida que exista el insumo en inventario y que la cantidad actual al restarse no sea un valor negativo, en este caso no lo es.
8. El componente de Inventario API realiza la resta y refleja el cambio en la base de datos actualizando el inventario.
9. El componente de Inventario API retorna una lista de inventario.
10. El componente de Provider Serializa la respuesta, se la envía a Inventario.
11. Inventario renderiza la tabla con la respuesta.

# Vista de Despliegue



## Despliegue de la API

|  |  |
| --- | --- |
| **Nodo/Artefacto** | **Descripción** |
| Github | Plataforma de repositorios en donde se encuentra el código fuente del proyecto |
| API | Servidor VPS cloud con 2Gb de RAM, dónde se encuentra alojado los contenedores de las APIs. |
| Git | Software para la gestión de repositorios. |
| Maven | Gestor de las dependencias requeridas para JAVA, y herramienta por medio de la cual se compila el desplegable de la API |
| JRE | Herramienta que es el entorno de ejecución para las aplicaciones Java. |
| Docker | Herramienta para la creación y ejecución de contenedores de tipo Docker |

**Prerequisitos:**

* Los desarrolladores BackEnd, deben tener instalado el JDK 8 o superior, con el IDE de desarrollo Netbeans o Eclipse, con el plugin de spring.
* El servidor debe estar habilitado para la creación de imágenes de Docker, JDK para la compilación del código JAVA y la obtención del código fuente vía git.

|  |  |
| --- | --- |
| **Pom.xml** |  |

## Despliegue de la Aplicación WEB

|  |  |
| --- | --- |
| **Nodo/Artefacto** | **Descripción** |
| Github | Plataforma de repositorios en donde se encuentra el código fuente del proyecto |
| Aplicación Web | Servidor VPS cloud con 2Gb de RAM, dónde se encuentra alojado el contenedor de la aplicación web y las herramientas de compilación. |
| Git | Motor de gestión de repositorios basados en git. |
| Angular CLI | Framework de desarrollo Front End basado en componentes |
| NodeJS | Motor de backend en Javascript, se utiliza el npm y ng para la compilación del proyecto. |
| Docker | Herramienta para la creación y ejecución de contenedores de tipo Docker |
| Front End Angular | Contenedor en dónde está el frontend compilado en Angular. |

Todos los nodos deben estar conectados a internet.

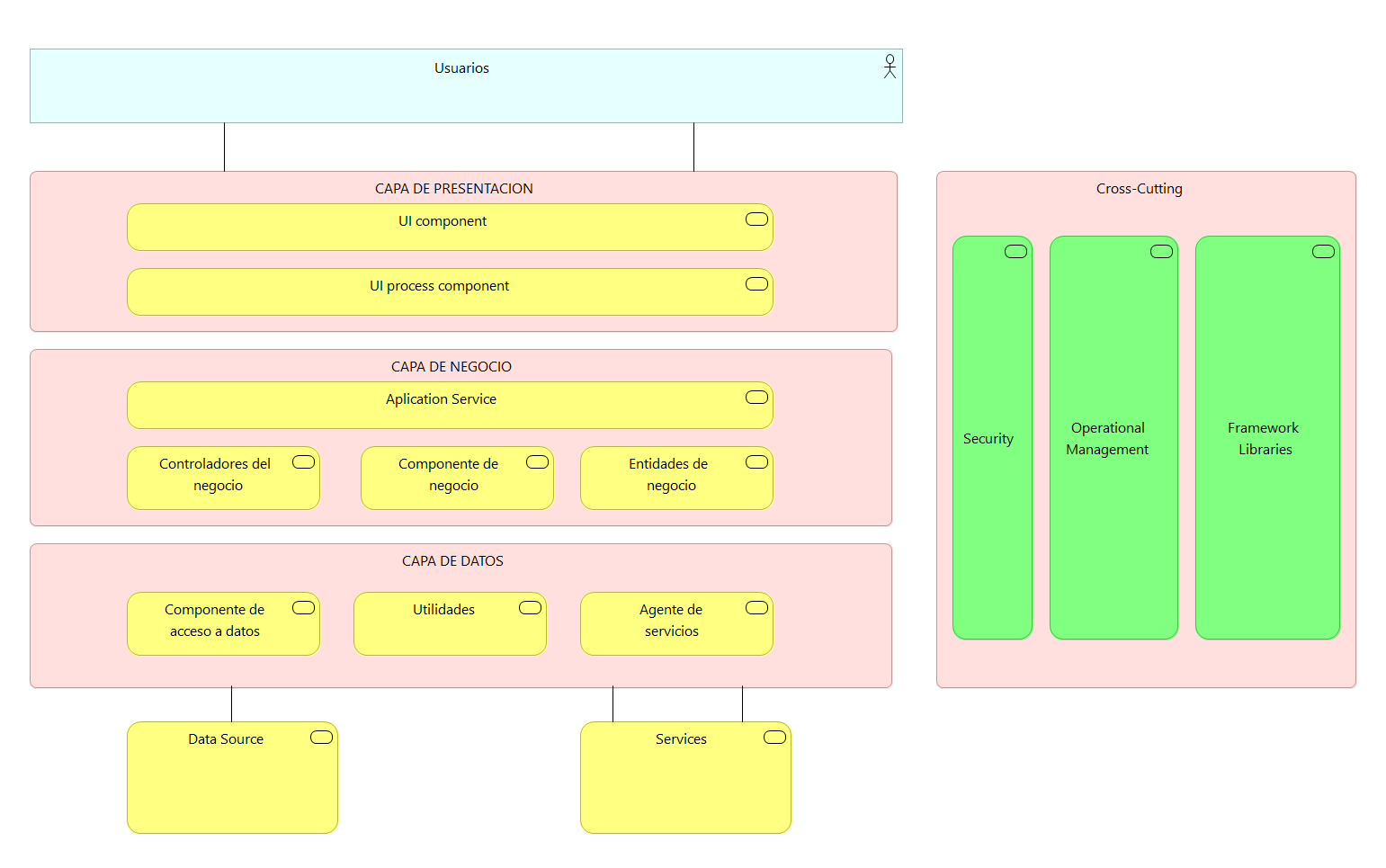
**Prerequisitos:**

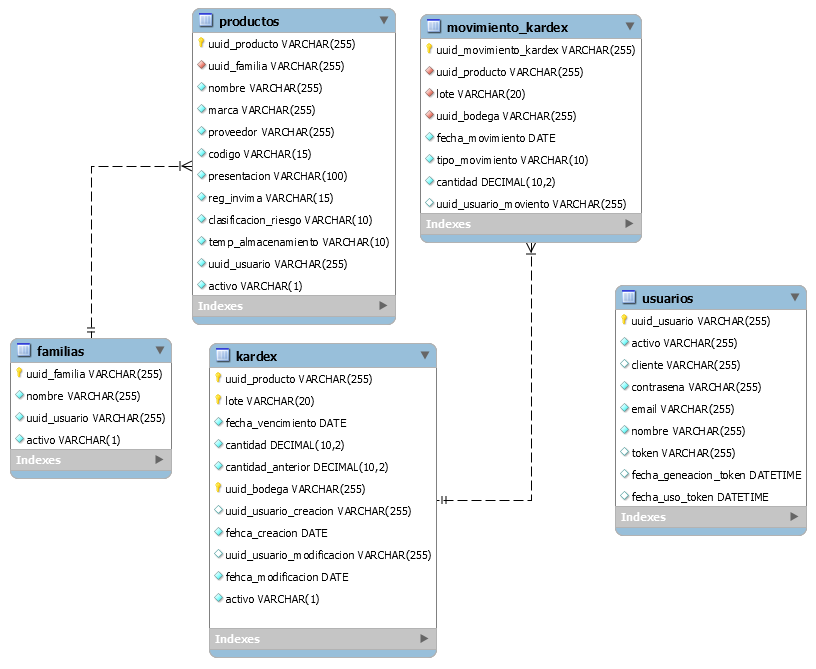
* Los desarrolladores Frontend deben contar con Angular versión 8 instalado y NodeJS.
* Los usuarios deben usar un Navegador moderno que soporte html5, Css3 y Javascript.
* El servidor debe estar habilitado para la creación de imágenes de Docker, la compilación de los scripts y la obtención del código fuente vía git.

|  |  |
| --- | --- |
| **Dockerfile** |  |

FROM nginx:1.17.1-alpine  
EXPOSE 80  
COPY /dist/angular8 /usr/share/nginx/html

# Conceptos Transversales (Cross-cutting)





|  |  |
| --- | --- |
| **Nodo/Artefacto** | **Descripción** |
|  |  |

**8.2 Persistencia:** El sistema de laboratorios usa una serie de bases de datos relacionales pero distribuidas debido a que la aplicación está contenerizada en microservicios.

**8.3 Interfaz de usuario:** Angular con Bootstrap para que esta sea responsiva.

**8.4 Manejo de sesión:** se realiza mediante una cookie, la cual guarda cifrado el id del usuario y el token de sesión, este token expira en 30 minutos si la sesión se encuentra inactiva.

**8.5 Seguridad:** Las APIs tienen un sistema de autorización mediante el UID y el token del usuario, si el usuario no se identifica la Api no podrá ser consumida.

**8.6 Validaciones:**

Validaciones de negocio:

* Al realizar una salida de inventario se valida que la cantidad de la salida no supere a la cantidad disponible del insumo en el inventario.

**8.7 Excepciones y manejo de errores:**   
  
La API muestra una excepción 401 si no está autorizado el usuario, si es otro tipo de error o error de negocio, retorna un mensaje con el error especifico el cual se visualiza en el frontend.

# Decisiones de Diseño

* 1. **Despliegue en la Nube**

**Problema:**

Se debe realizar un despliegue de la aplicación web de bajo costo de manera que sea publicada para el acceso desde cualquier lugar y dispositivo que esté conectado a internet.

**Alternativas consideradas:**

* Usando algún tipo de almacenamiento en la nube.
* Usando un servidor y el sistema de archivos local.

**Decisión:**

Se opta por el sistema de almacenamiento en la nube porque no se tenía presupuesto para la compra de un servidor físico, ni tampoco el pago de una conexión permanente a internet empresarial con una IP pública, adicionalmente se evalúan las siguientes ventajas de la implementación en la nube:

* Menos tiempo de implementación: este tipo de plataformas reducen mucho los tiempos de instalación. Menos tiempo, menos costos.
* Programación desde cualquier sitio: no es necesario estar delante de tu máquina para continuar programando tu producto. Al estar alojada en la nube, es posible hacerlo desde cualquier sitio y dispositivo (ordenador o tableta). Único requisito: estar conectado a Internet.
* Trabajo colaborativo en tiempo real: una de sus grandes ventajas es que varios desarrolladores pueden estar trabajando en el mismo proyecto a la vez y utilizar servicios de chat online para comunicarse.
* Personalización del entorno de desarrollo: este tipo de herramientas permiten instalar dependencias para los proyectos de forma independiente. Cuando se programa en local, en muchas ocasiones se pueden tener complicaciones porque las dependencias para unos proyectos afectan a otros o perjudican a otras aplicaciones web.
  1. **Diseño basado en microservicios**

**Problema:**

Se requiere un desarrollo de software basado en componentes, que permita una división del trabajo dentro de los integrantes del equipo y una mayor agilidad en el proceso de desarrollo utilizando los principios de S.O.L.I.D.

**Alternativas consideradas:**

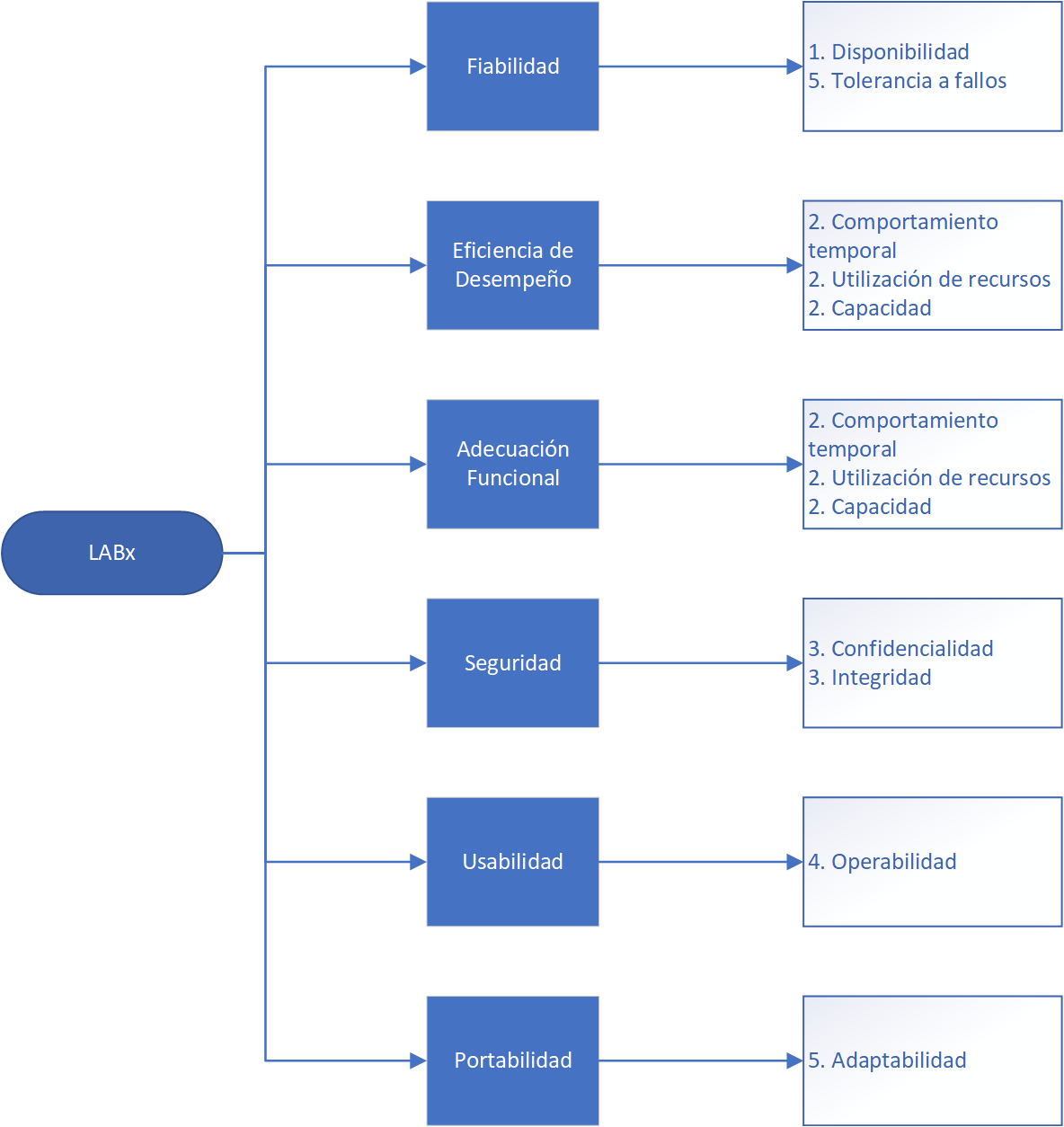
* Desarrollo de la aplicación centrada en el monolito.
* Aplicación diseñada con microservicios.

**Decisión:**

Se opta por utilizar una arquitectura de diseño de la aplicación basada en microservicios y con el uso de contenedores en Linux, los cuales proporcionan a las aplicaciones basadas en microservicios una unidad de implementación de aplicaciones ideal y un entorno de ejecución autónomo. Esto le permite aprovechar mejor el hardware y coordinar los servicios con facilidad, lo que incluye el almacenamiento, las redes y la seguridad.

# Requerimientos de Calidad

* 1. **Árbol de Calidad**



* 1. **Escenarios de calidad**

**Testabilidad / Cobertura:**

Al usar LABx durante el desarrollo y el proceso de construcción, se garantiza una cobertura de código de al menos el 95%. Este cálculo se basa en la mínima disponibilidad de los contenedores de servicios alojado en la nube.

**Testabilidad / Independiente de servicios externos:**

La arquitectura debe diseñarse de tal manera que los algoritmos que dependen de los servicios externos se puedan probar sin tener el servicio externo disponible. Es decir: todas las dependencias externas deben ser simulables.

# Riesgos y deuda técnica

Usar versiones de Java que impidan usar nuevo software o aplicar una actualización de seguridad.

El sistema de inventarios LABx puede llegar a ser tan viejos y personalizados que no se pueden actualizar, ya que sería un esfuerzo de “destripar y reemplazar”.

Sistemas similares que tienen funciones superpuestas en diferentes partes de tu organización.

Existe la posibilidad de que la base de datos de LABx se dañe debido a un apagado inesperado de la VM (es decir, falla del sistema operativo o del hardware en la nube, la cual no es administrada por nosotros). El riesgo se mitiga mediante copias de seguridad periódicas del archivo de base de datos serializado.

# Glosario

|  |  |
| --- | --- |
| **Término** | **Definición** |
| Angular | Angular es un marco de aplicaciones web de código abierto mantenido principalmente por Google y por una comunidad de desarrolladores y corporaciones individuales para abordar muchos de los desafíos encontrados en el desarrollo de aplicaciones de una sola página. |
| Evento Adverso | Es el resultado de una atención en salud que de manera no intencional produjo daño. Los eventos adversos pueden ser prevenibles y no prevenibles. |
| Evento Adverso Prevenible | Resultado no deseado, no intencional, que se habría evitado mediante el cumplimiento de los estándares del cuidado asistencial disponibles en un momento determinado. |
| Evento Adverso No Prevenible | Resultado no deseado, no intencional, que se presenta a pesar del cumplimiento de los estándares del cuidado asistencial. |
| Insumo | Bien de cualquier clase empleado en la producción de otros bienes, en el contexto de la arquitectura se habla indistintamente entre insumos y reactivos de Laboratorio Clínico. |
| Inventario | Registro ó balance de productos usados en el Laboratorio Clínico. |
| INVIMA | Es el Instituto Nacional de Vigilancia de Medicamentos y Alimentos, es una entidad de vigilancia y control de carácter técnico científico, que trabaja para la protección de la salud individual y colectiva de los colombianos, mediante la aplicación de las normas sanitarias asociada al consumo y uso de alimentos, medicamentos, dispositivos médicos y otros productos objeto de vigilancia sanitaria. |
| Kardex | Es un registro utilizado para mantener el control de la mercadería cuando se utiliza el método de permanencia en inventarios, con este registro podemos controlar las entradas y salidas de las mercaderías y conocer las existencias de todos los artículos que posee la empresa para la venta. |
| Registro INVIMA | Conocido también como registro sanitario, es el documento expedido por la autoridad sanitaria correspondiente (INVIMA), mediante el cual se autoriza a una persona natural o jurídica para fabricar, envasar e importar un alimento con destino al consumo humano. |
| Seguridad del Paciente | Es el conjunto de elementos estructurales, procesos, instrumentos y metodologías basadas en evidencias científicamente probadas que propenden por minimizar el riesgo de sufrir un evento adverso en el proceso de atención de salud o de mitigar sus consecuencias. |
| Trazabilidad | Serie de procedimientos que permiten seguir el proceso de evolución de un producto en cada una de sus etapas. |