

## RESUMEN

El presente documento fue elaborado siguiendo la estructura de reporte final ITH-AC-PO-007-07 y que describe las actividades realizadas durante el semestre 2019-2.

Este reporte abarca todo el proceso que involucro el proyecto ***“Desarrollo de módulo de inventario”*** para la plataforma Industore Global. El documento provee una vista general de alto nivel, define metas de la arquitectura, los casos de uso soportados por el sistema, estilos de arquitectura y los componentes que fueron seleccionados, así como definir las especificaciones funcionales, no funcionales y del sistema. El presente también provee una justificación de cada una de las decisiones tomadas sobre la arquitectura y diseño desde la idea conceptual hasta la implementación.

El proyecto consiste en una API que funcione como inventario (altas, bajas y modificaciones de productos, sucursales, usuarios y matrices) para que sea independiente de tecnologías y pueda ser consumido por cualquier tipo de cliente (móvil, web, escritorio).

## HISTORIAL DE CAMBIOS

Versión	Descripción	Participantes	Fecha
1.0	Comienza la elaboración del documento para modelar el sistema “módulo de inventario” utilizando el patrón 4+1 vistas	Erick Jara	12/09/19
2.0	Se agregaron y modifíco la sección de drivers arquitectónicos.	Erick Jara	12/10/19
3.0	Se añadió la sección de estándares.	Erick Jara	12/11/19
4.0	Documento finalizado.	Erick Jara	19/12/19
5.0	Documento actualizado y revisado.	Erick Jara	20/12/19

<b>INDICE DE FIGURAS .....</b>	<b>i</b>
<b>INDICE DE TABLAS .....</b>	<b>ii</b>

## **INDICE DE CONTENIDO**

<b>GENERALIDADES DEL PROYECTO .....</b>	<b>1</b>
Introducción .....	1
Descripción de la empresa u organización y del puesto o área del trabajo del estudiante .....	1
Problemas a resolver .....	2
Objetivos .....	2
General: .....	2
Específicos: .....	2
Justificación .....	3
<b>MARCO TEÓRICO .....</b>	<b>3</b>
<b>DESARROLLO.....</b>	<b>6</b>
Alcance .....	6
Arquitectura de software .....	7
Técnica para el levantamiento de requerimientos .....	7
Modelo Utilizado (Iterativo Incremental) .....	7
Metodología Ágil (SCRUM) .....	7
Drivers Arquitectónicos.....	8
Stakeholders .....	11
Diagrama de contexto.....	12
Restricciones .....	12
Riesgos.....	13
Bosquejo del sistema.....	14
Entorno de operación .....	14
Vistas arquitectónicas .....	15
Arquitectura lógica .....	15
Arquitectura de desarrollo (despliegue) .....	17
Arquitectura física .....	17

<b>RESULTADOS</b>	18
Sprint 1: Usuarios	18
Sprint 2: Matrices	21
Sprint 3: Sucursales	24
Sprint 4: Productos	26
Normatividades, regulaciones y restricciones	29
De documento	29
De diseño	30
De código	33
Tamaño y desempeño	36
<b>CONCLUSIONES</b>	37
<b>COMPETENCIAS DESARROLLADAS</b>	37
<b>FUENTES DE INFORMACIÓN</b>	38
<b>DEFINICIONES</b>	38
<b>ANEXOS</b>	41

## INDICE DE FIGURAS

Ilustración 1: Diagrama de contexto .....	12
Ilustración 2: Bosquejo del sistema .....	14
Ilustración 3: Diagrama de clases .....	15
Ilustración 4: Diagrama de comunicación.....	16
Ilustración 5: Diagrama E/R .....	16
Ilustración 6: Diagrama de componentes .....	17
Ilustración 7: Diagrama de despliegue .....	17
Ilustración 8: Código Ejemplo - Creación usuario .....	19
Ilustración 9: Código ejemplo - Protección autenticación .....	21
Ilustración 10: Prueba sprint 1 - Activación usuario .....	21
Ilustración 11: Código ejemplo - Listar matriz .....	22
Ilustración 12: Código ejemplo - Crear matriz .....	24
Ilustración 13: Prueba sprint 2 - Solo una matriz por administrador .....	24
Ilustración 14: Código ejemplo - Eliminar sucursal.....	25
Ilustración 15: Prueba sprint 3 - Asignación de sucursal a matriz .....	26
Ilustración 16: Código ejemplo - Editar producto.....	27
Ilustración 17: Prueba sprint 4 - Creación de producto .....	28
Ilustración 18: Control de versiones, proyecto subido a repositorio remoto .....	29
Ilustración 19: Modelo 4+1 Vistas .....	30
Ilustración 20: Recursos y colecciones .....	31
Ilustración 21: Respuesta JSON .....	32
Ilustración 22: Respuesta enviada por el servidor.....	35
Ilustración 23: Limitar peticiones .....	35
Ilustración 24: Grafica de Gantt.....	36
Ilustración 25: Seguimiento Entregas Funcionales .....	41

## INDICE DE TABLAS

Tabla 1: Alcance .....	6
Tabla 2: Driver Arquitectónico (Del negocio) .....	8
Tabla 3: Driver arquitectónico (De usuario) .....	9
Tabla 4: Driver arquitectónico (Funcional) .....	10
Tabla 5: Driver arquitectónico (De calidad) .....	10
Tabla 6: Riesgos .....	13
Tabla 7: Prueba sprint 1 .....	21
Tabla 8: Prueba sprint 2 .....	24
Tabla 9: Prueba sprint 3 .....	25
Tabla 10: Prueba sprint 4 .....	28
Tabla 11: Definiciones .....	38

# GENERALIDADES DEL PROYECTO

## Introducción

En el presente documento se describe el análisis, desarrollo, implementación y funcionamiento de una API RESTful que realiza las funciones de inventario que será conectada con la plataforma Industore Global en un futuro.

El objetivo es mejorar el manejo de las existencias de productos que los proveedores reportan en la plataforma y mejorar el proceso que se tiene actualmente, ya que se utiliza un archivo Excel, lo cual hace el proceso más tedioso ya que debes llenar un documento para poder cargar las existencias de producto y esto genera que las existencias reales no siempre coincidan con las reportadas en la plataforma.

El alcance del proyecto es realizar las operaciones de consultas, altas, modificaciones, bajas y asignaciones en los diferentes recursos del software desarrollado. El proyecto pretende servir como base para continuar el desarrollo del mismo y añadir más funcionalidades y características que le agreguen valor a la empresa y a la plataforma de Industore Global, así como ayudar a los proveedores a mejorar el manejo de existencias de productos y gestión de sus sucursales y matriz como de los usuarios que pueden tener acceso.

Para representar el software con la mayor precisión posible, ciertas partes del documento están basadas en el modelo “4+1” de Phillippe Kruchten<sup>1</sup>.

## Descripción de la empresa u organización y del puesto o área del trabajo del estudiante

La empresa es de tipo privada y lleva por nombre Encor Development, están orientados a la prestación de servicios en desarrollo de software y todo lo que ello conlleva (análisis, desarrollo, implementación, mantenimiento, etc.), son una empresa de tamaño mediana (según la clasificación del INEGI por número de trabajadores de 11 a 50).

La empresa está ubicada en la calle Enrique González Martínez 19A esquina con 14 de abril en la colonia Valle Hermoso y código postal 83209.

Actualmente la empresa tiene un proyecto de desarrollo propio llamado Industore Global (industoreglobal.com), en el cual conectan proveedores de productos industriales con clientes finales, es un sitio de comercio electrónico.

---

<sup>1</sup> <https://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf>

Mis labores fueron elaboradas en el área de tecnologías de información con el rol de desarrollador backend, en donde estuvo a mi cargo el Ing. Iván Arvizu Bejarano que tiene el puesto de coordinador del área de TI.

## **Problemas a resolver**

Se me dio la tarea de proponer una solución al problema que tienen con el manejo de los inventarios que los proveedores reportan en la plataforma, esto debido a que se me comento que ya ha ocurrido la situación en la que usuarios han querido realizar una compra y el stock que el proveedor reportaba no era acorde a la realidad, por lo que al momento de solicitarse el pedido la empresa proveedora no contaba con existencias suficientes para satisfacer la orden.

Actualmente la forma en la que se suben productos a la plataforma es utilizando un archivo en formato Excel, lo cual genera esa falta de precisión en cuanto a la cantidad de existencias reportadas ya que no se actualiza constantemente, es por esta razón que era necesario hacer el proceso más rápido y que los datos se actualicen con más frecuencia para que sean más acordes a la realidad, por lo cual se me solicito el desarrollo de un software que aborde esta problemática.

## **Objetivos**

### **✓ General:**

- Dar solución mediante el desarrollo de un software a la problemática generada por el reporte incorrecto de existencias de producto en el inventario por parte de las empresas proveedoras de productos industriales en la plataforma Industore Global.

### **✓ Específicos:**

- Análisis, desarrollo, validación e implementación de un software que actúe a modo de inventario.
- Utilización de estándares y principios de diseño orientados a APIs REST.
- Mantener tiempos de respuesta del servidor baja.
- Utilización de lenguaje JavaScript de principio a fin junto con base de datos NoSQL.
- Utilización de metodologías ágiles (SCRUM, Kanban).
- Minimizar incoherencias y errores en registros en la base de datos.
- Maximizar esfuerzos en materia de seguridad (roles, inicio de sesión, cookies, prevención de ataques HTTP, etc.).



## **Justificación**

Los motivos que me llevaron a seleccionar la tecnología requerida para desarrollar una API REST es que siendo una API tienes la flexibilidad de exponer rutas HTTP que puedan ser utilizadas por cualquier tipo de cliente y realice peticiones a estas, lo anterior es el justificante principal ya que el cliente (en este caso interno, Encor) me solicito que el módulo desarrollado pueda ser conectado a la plataforma principal que es Industore Global, siendo una de las restricciones o impedimentos para hacer esto el no tener acceso al código fuente de la plataforma (al no ser empleado no podía ver el código, por motivos de seguridad).

Otra de las consideraciones o justificaciones más importantes que tome en cuenta al momento de decidir desarrollar una API fue el que esta seria independiente de tecnologías y lenguajes del cliente que la consume, en este caso Industore Global que está desarrollada con tecnologías PHP. Esto me daría la libertad de usar un lenguaje, framework y/o base de datos que me fuera más familiar y que considerara más acorde al problema que se le pretende dar solución y así permitirme ser más eficiente con los tiempos y necesidades del proyecto.

Además, esto permite al personal encargado de desarrollar el frontend para la aplicación enfocarse en la presentación de la información y la manera de realizar peticiones de una manera intuitiva mediante el uso de HTML, CSS, JavaScript, sin tener que preocuparse por el código detrás y de esta manera hacer peticiones a las rutas de la API y obtener una respuesta en formato JSON que posteriormente el frontend mostrara y de esta manera optimizar el flujo de trabajo del equipo asignado al proyecto.

## **MARCO TEÓRICO**

El proyecto descrito en este documento se basa en una API que hace uso del estilo de arquitectura de software REST.

Para comenzar, ¿qué es una API?, son las siglas en ingles de Application Programming Interface, o en español, Interfaz de Programación de Aplicaciones. Lo anterior puede no hacer mucho sentido, es por eso que debemos enfocarnos en la palabra ‘interfaz’ ya que una API es la interfaz que un programa presenta a otros programas, humanos, y hacia el mundo (en el caso de las APIs Web).

“Las APIs son los bloques de construcción que permiten a interoperabilidad hacia las principales plataformas de negocios en la web. Son la manera en que una identidad es creada y mantenida a través de cuentas de software en la nube, desde tu correo electrónico hasta las aplicaciones web que te ayudan a pedir comida a domicilio. Las APIs son la manera en cómo los datos de pronostico del tiempo de una fuente confiable

como The Weather Channel pueden ser compartidos a múltiples aplicaciones de software que se especializan en su presentación (frontend). Las APIs procesan tus transacciones con tarjeta de crédito en compras en línea para que las empresas puedan fácilmente obtener el dinero sin tener que preocuparse por crear tecnología financiera y sus leyes y regulaciones correspondientes. “(1)

Las API pueden ser privadas para el uso de una empresa (como es el caso), abiertas sólo para socios, o públicas (como la de GitHub) para que cualquier desarrollador interactuar con ellas o crear sus propias API para que lo hagan. A los servicios que no son de código abierto también les permite dejar que otros utilicen funciones concretas de sus aplicaciones o servicios sin tener que proporcionarles todo el código. Esta es una de las grandes ventajas de las API, exponer rutas a las cuales los desarrolladores pueden mandar y recibir información sin que el creador de la API comparta el código fuente, solo con saber qué es lo que se debe de mandar a dicha ruta y con qué parámetros.

Una de las principales funciones de las API es poder facilitarles el trabajo a los desarrolladores y ahorrarles tiempo y dinero. Hablando específicamente de este proyecto, ahorra tiempo (y dinero en costos de desarrollo) a los desarrolladores ya que el inventario ya está hecho y ellos solo harán uso de los métodos y generarán un cliente que presente de manera agradable de cara al usuario la información generada.

Entonces, podemos resumir que las API surgen de la necesidad de compartir información a otras aplicaciones o sistemas, sin que las personas que consumirán o harán uso de tu API deban preocuparse de solucionar problemas que tú ya resolviste, además, estas son muy versátiles, ya que pueden ser utilizadas por aplicaciones web, móvil, de escritorio, IoT (Internet de las Cosas) y así mantener consistencia a través de tus productos tecnológicos. Este proyecto fue desarrollado teniendo lo antes mencionado en cuenta, ya que tener una solución tecnológica de este tipo te permite escalar rápidamente e implementar también y así, alinearte con la estrategia de negocio de la empresa y ayudando a conseguir más clientes.

Dentro del mundo de las API hay distintas arquitecturas, guías de diseño, patrones, etc., pero la elegida (y mas popular hoy en día) es REST, ¿qué es REST?, es un paradigma o estilo arquitectónico de software utilizado para construir APIs que permiten comunicar el servidor con los clientes utilizando el protocolo HTTP mediante URIs inteligentes que puedan satisfacer la necesidad del cliente. En este estilo arquitectónico, la implementación del cliente y del servidor pueden ser independientes sin conocer uno del otro, esto significa que el código en el lado del cliente puede ser cambiado en cualquier momento sin afectar la operación del servidor y el código en el servidor puede ser cambiado sin afectar la operación del cliente. Solo es necesario conocer el formato de los mensajes que se enviaran entre ambos para poder mantenerlos de forma separada

y modular, de esta manera mejoramos la flexibilidad de la interfaz a través de plataformas y mejoramos la escalabilidad.

REST tiene ciertas reglas que hay que tomar en cuenta al momento de desarrollar, por ejemplo, es STATELESS, esto significa que si nosotros ofrecemos la manera de autenticarse a la API no podemos guardar la sesión del usuario ya que la memoria RAM de nuestro servidor se acabaría muy rápido, es por esto que hice uso de TOKENS y COOKIES para guardar la información de sesión. Es por esto que los tokens y cookies posteriormente se gestionan a través del cliente utilizando algún framework para frontend como React, Vue o Angular.

Otra regla es que cada acción en nuestra API es un verbo (en inglés) y dicha acción corresponde al verbo, por ejemplo, GET (obtener) sirve para recibir información desde la API que se conecta a una base de datos, POST (publicar) sirve para mandar datos y que la API haga algo con ellos y te mande una respuesta, etc.

Ahora bien, ¿qué es RESTful? Es simplemente la forma en que se les denomina a las API que hacen uso de la arquitectura REST, es decir, toda API que haga uso del estilo arquitectónico REST es una API RESTful.

Para el desarrollo de la API se hizo uso de Node.js con un framework llamado Express.js que es básicamente una colección de herramientas para desarrollar aplicaciones web de manera más rápida que si lo hiciéramos de cero. Node.js es un desarrollo de código abierto para ejecutar código de JavaScript de lado del servidor y suele ser utilizado para aplicaciones en tiempo real. Una de las características más atractivas es que Node es asíncrono, es decir, si tienes a múltiples usuarios realizando peticiones al mismo tiempo a tu servidor, cada una de estas peticiones serán gestionadas de manera independiente y pueden ser resueltas en poco tiempo, contrario a lo que pasa cuando no tienes un entorno asíncrono, ya que cada petición tomaría su tiempo de procesamiento y hasta no terminar con un usuario no podrías continuar con el siguiente.

Al separar el servidor del cliente y trabajar de manera independiente, es necesario contar con una forma de interactuar con nuestra API/servidor, es por ello que se utilizó la herramienta Postman. Esta herramienta permite enviar peticiones mediante el protocolo HTTP a nuestra API y visualizar la respuesta del servidor, así como el tiempo que tardó en procesar la petición y el código de respuesta que obtuvimos (404, 500, entre otros).

Para un desarrollo más dinámico y adaptable a las exigencias del cliente (además de ser la metodología ágil utilizada dentro de la empresa) fue que utilice SCRUM y Kanban para la gestión del proyecto con el cliente.

SCRUM es un conjunto de reglas y buenas prácticas que sirven para trabajar en equipo, especialmente en el área de tecnologías de la información. En esta metodología, se realizan entregas parciales del proyecto o producto en desarrollo al cliente (denominados

sprints y generalmente son de 1 a 2 semanas), de esta manera podemos recibir retroalimentación y ajustar en caso de que lo realizado no sea de agrado del cliente sin que esto implique grandes pérdidas de tiempo y, por lo tanto, de dinero. Esta metodología es muy ágil en entornos complejos o cambiantes en donde los resultados deben entregarse rápidamente o que debido a los cambios latentes podamos ser flexibles.

Kanban ayuda a mantener todo mas organizado, ya que mediante un tablero con tarjetas podemos separar lo que se esta haciendo, lo que se hará, lo que esta en revisión, lo que ya esta concluido y otras maneras de gestionar tu trabajo, se utilizo Trello como software para utilizar la metodología Kanban.

Para mantener un buen manejo de los cambios que se fueron desarrollando conforme avanzaron los meses de trabajo, se utilizo un control de versiones, en este caso Git, el cual nos sirve para tener control sobre el código fuente del software en desarrollo y así, tener control sobre que es lo que se ha cambiado, a que hora y en que día, para que de ser necesario poder descartar los cambios realizados en caso de haber ocurrido algún error.

## DESARROLLO

### Alcance

Se me dio la libertad de elegir las tecnologías que consideré apropiadas para el proyecto, es por ello que me di a la tarea de analizar el problema para poder ofrecer la mejor solución posible teniendo en cuenta el tiempo disponible para desarrollarlo.

Este proyecto consiste solo en la parte del servidor (backend), es decir, el código que se ejecutará y devolverá datos. El módulo realiza las siguientes operaciones:

*Tabla 1: Alcance*

Numero de entrega	Tema principal	ID de características a incluir
1.0	Usuarios	HU-USR-01, HU-USR-02, HU-USR-03, HU-USR-06, HU-USR-07, HU-USR-08
2.0	Matrices	HU-USR-04, HU-PRNT-001, HU-PRNT-002
3.0	Sucursales	HU-USR-05, HU-BRNC-001, HU-BRNC-002
4.0	Productos	HU-PROD-001, HU-PROD-002

## Arquitectura de software

### ➤ Técnica para el levantamiento de requerimientos

Se hizo uso de dos técnicas para levantamiento de requerimientos, en este caso fue la observación e historias de usuario.

La observación se aplicó viendo cómo se maneja actualmente el inventario, que es lo que necesitan los interesados en el proyecto (Stakeholders) y los clientes, tomando nota de las dificultades que se iban mencionando en cuanto a la gestión de existencias.

Las historias de usuario tomaron mayor relevancia ya que el mismo cliente es el que indica cuáles son las funciones que el sistema debe realizar, estas historias están descritas en el apartado de **drivers arquitectónicos (de usuario)**.

### ➤ Modelo Utilizado (Iterativo Incremental)

Para llevar un mejor control en cuanto al desarrollo del proyecto fue necesario utilizar un modelo, para este caso el más conveniente (por utilizar SCRUM) es el iterativo incremental.

¿Qué significa iterativo? Básicamente, consiste en dividir el proyecto en fases cíclicas en las que el proyecto avanza progresivamente, esto permite ir detallando el proyecto mediante avanza el mismo y así reducir la incertidumbre del resultado. Los pasos claves en el proceso son comenzar con una implementación simple de los requerimientos del sistema, e iterativamente mejorar la secuencia evolutiva de versiones hasta que el sistema completo este implementado.<sup>2</sup>

¿E incremental? Es la forma en que se llama a cada uno de los ciclos realizados en la parte iterativa, en el caso de un producto de software es cada porción completa de código que se establece en el plan de desarrollo y vamos iterando una a una, parte por parte del código hasta que llegamos al producto terminado.

### ➤ Metodología Ágil (SCRUM)

Para este proyecto se hizo uso de la metodología ágil SCRUM ya que en la empresa Encor hace uso de ella para la gestión de proyectos, es necesario mencionar que no se utiliza la metodología completa, ya que ellos hicieron cambios a la metodología para adaptarla a las necesidades que tienen.

Uno de los cambios es el reemplazar las reuniones diarias por reuniones semanales en las cuales se discuten y evalúan avances de cada uno de los miembros del proyecto y

---

<sup>2</sup> [https://www.ecured.cu/Metodolog%C3%ADa\\_de\\_desarrollo\\_iterativo\\_y\\_creciente](https://www.ecured.cu/Metodolog%C3%ADa_de_desarrollo_iterativo_y_creciente)

saber qué es lo que se hará durante dicha semana, estas reuniones a diferencia del SCRUM tradicional no se hacen con el equipo de pie. Otro cambio es que no se realiza el 'planning poker' como tal para realizar estimaciones, si no que por la dificultad y opinión del equipo de desarrollo se llega a un tiempo estimado para el sprint o desarrollo/mejora a realizar al software.

Las historias de usuario si se utilizan de manera tradicional para saber qué es lo que el cliente requiere que se haga, se hace uso de un SCRUM board en la oficina para que todos puedan visualizar que es lo que se va a hacer, que está en proceso, impedimentos, etc.

Como apoyo adicional a SCRUM se hace uso de una herramienta que tradicionalmente se usa con Kanban mediante tarjetas, llamada Trello. Mediante la utilización de esta podemos dar seguimiento al proyecto de manera más sencilla y rápida ya que tienes la posibilidad de compartir tareas y avances con el resto del equipo de desarrollo o con el coordinador de TI y así tener visualmente una referencia del estado del proyecto y de cualquier imprevista que pueda surgir para evitar desviarse mucho de la fecha de entrega estimada de cualquiera de los sprints o entregables.

#### ➤ Drivers Arquitectónicos

- Del negocio

*Tabla 2: Driver Arquitectónico (Del negocio)*

ID	Descripción del objetivo de negocio
ON-1	Asegurar hasta un 85% que el inventario reportado a la plataforma sea acorde a la realidad.
ON-2	Proporcionar mecanismos que permitan integrar el sistema con otros proyectos que requieran de la información almacenada en la aplicación como propuesta antes del 20 de diciembre de 2019.
ON-3	Prevenir la fuga de clientes por stock inexistentes o no apegados a la realidad en un 100%.
ON-4	Liberar el sistema antes del 20 de diciembre de 2019.

- De usuario (historia de usuario)

*Tabla 3: Driver arquitectónico (De usuario)*

ID	Rol	Característica / Funcionalidad	Razón / Resultado
HU-USR-001	Como administrador	Necesito registrar usuarios nuevos	Con la finalidad de asignarles roles, y darles acceso
HU-USR-002	Como administrador	Necesito listar todos los usuarios asignados a mi matriz y sucursales	Con la finalidad de conocer quien tiene acceso a qué recursos
HU-USR-03	Como administrador	Necesito activar y desactivar usuarios	Con la finalidad de bloquear el inicio de sesión a aquellos usuarios que ya no forman parte de la empresa.
HU-USR-04	Como administrador	Necesito asignar usuarios a la matriz.	Con la finalidad de que puedan realizar tareas dentro de ella.
HU-USR-05	Como administrador / gerente	Necesito asignar usuarios a las sucursales	Con la finalidad de que puedan realizar tareas dentro de ella
HU-USR-06	Como administrador	Necesito eliminar usuarios	Con la finalidad de que no queden en el registro (en caso de despido, por ejemplo)
HU-USR-07	Como publicante	Necesito registrarme	Con la finalidad de que mi administrador pueda asignarme a la matriz/sucursal que corresponde
HU-USR-08	Como publicante	Necesito iniciar sesión	Con la finalidad de tener acceso a los recursos disponibles
HU-PRNT-001	Como administrador	Necesito dar de alta, modificar, eliminar y listar una matriz	Con la finalidad de poder realizar operaciones sobre esta
HU-PRNT-002	Como usuario	Necesito dar de alta productos en matriz	Con la finalidad de tener el inventario reportado

HU-BRNC-001	Como administrador	Necesito asignar gerentes para cada sucursal	Con la finalidad de tener un control y establecer gente de confianza
HU-BRNC-002	Como gerente	Necesito asignar usuarios para mi sucursal	Con la finalidad de que ejecuten las tareas de alta de productos
HU-PROD-001	Como administrador / gerente / usuario	Necesito dar de alta, modificar, eliminar y listar productos.	Con la finalidad de llevar un control
HU-PROD-002	Como administrador	Necesito ver la lista de productos de cada una de las sucursales en adición a la matriz	Con la finalidad de conocer las existencias de producto en todo momento

- Funcional

*Tabla 4: Driver arquitectónico (Funcional)*

El sistema debe funcionar como una API RESTful por lo cual debe funcionar con el protocolo HTTP para realizar peticiones GET, POST, PUT y DELETE a las diferentes rutas que estén disponibles.
Las interfaces graficas deben estar separadas de la lógica de programación para que funcionen cada una de manera independiente.
Debe contener una base de datos que permita exportar o de como salida documentos JSON como MongoDB.
El sistema debe estar desarrollado en JavaScript para funcionar en un stack tecnológico 'MERN'.

- De calidad (atributos de calidad)

1 = Mayor prioridad

5 = Menor prioridad

*Tabla 5: Driver arquitectónico (De calidad)*

ID	Categoría	Escenario	Prioridad
ODC-1	Desempeño	El sistema tiene una baja en velocidad debido al servidor y a peticiones concurrentes en un periodo corto de tiempo. El sistema debe volver	1



		a la normalidad una vez la carga disminuya, además deben implementarse formas de prevenir esta situación mediante programación asíncrona.	
ODC-2	Seguridad	Un atacante intercepta la comunicación entre el dispositivo de un usuario y el sistema mientras se realizaba un registro o inicio de sesión en un momento normal de operación. El atacante no obtiene ninguna información clara ya que esta debe estar cifrada (hash y https).	1
ODC-3	Disponibilidad	El sistema se cae en un momento de funcionamiento normal. El sistema debe estar de vuelta en operación en no más de 5 minutos.	2
ODC-4	Escalabilidad	La plataforma a la cual se conectará el sistema crece. El sistema debe estar preparado para soportar la escalabilidad para adaptarse a las nuevas necesidades del cliente.	3

### ➤ Stakeholders<sup>3</sup>

- Punto de vista interactuadores
  - Representa a las personas u otros sistemas que interactúan directamente con el sistema.
  - En el caso de este proyecto se tratan de los proveedores de Industore Global (personas) y la plataforma misma (sistema).
- Punto de vista dominio
  - Representa a las personas que no utilizan el sistema ellos mismos pero que influyen en los requerimientos de algún modo.
  - En este caso nos referimos al coordinador del área de TI y sus superiores, ya que, si bien no harán uso de la API, si toman decisiones en cuanto a lo que debe o no considerarse para su desarrollo.
- Punto de vista indirectos
  - Representan las características y restricciones del dominio que influyen en los requerimientos del sistema.

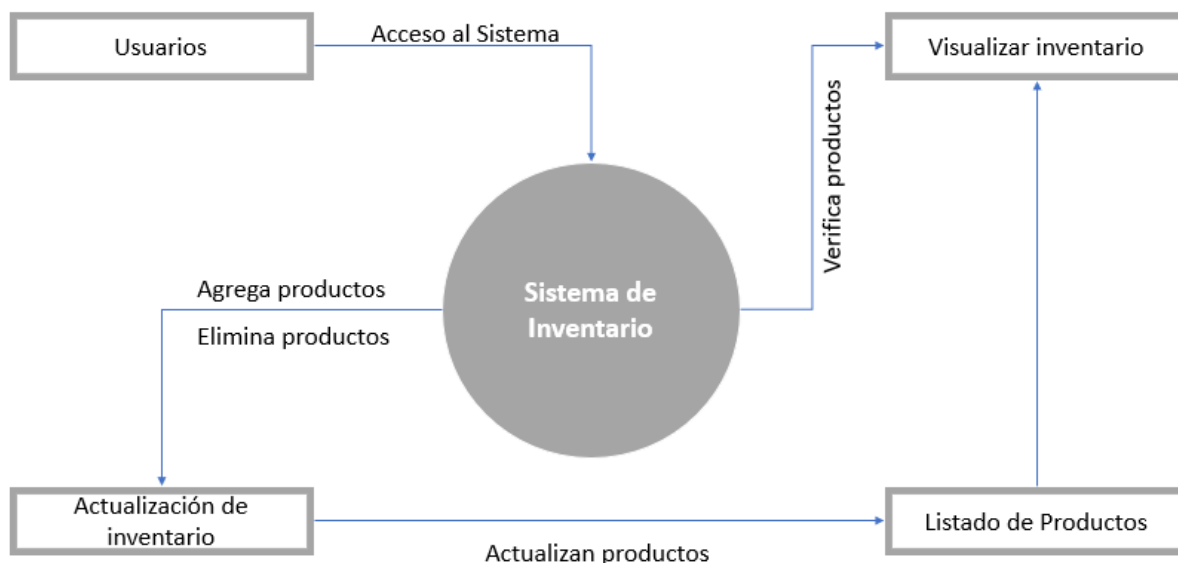
<sup>3</sup> <https://books.google.com.mx/books?id=gQWd49zSut4C&printsec=frontcover#v=onepage&q&f=false>

- Se habla entonces de los estándares de comunicación hacia y desde la API, que en este caso sería HTTP, ya que mediante él se gestionan todas las peticiones al servidor.

### ➤ Diagrama de contexto

Un diagrama de contexto es un tipo de método de modelado de procesos, y es una representación visual de las entradas y salidas de datos en un sistema. Las compañías analizan los diagramas de contexto para averiguar qué circunstancias, o contexto, afecta el flujo de datos.<sup>4</sup>

En este diagrama se muestra como el usuario puede acceder al sistema y puede realizar las siguientes acciones: Ver inventario para ver los registros, Actualizar inventario para registrar transacciones de todos los artículos que entran y salen del almacén y Extraer / eliminar artículo.



*Ilustración 1: Diagrama de contexto*

### ➤ Restricciones

- Técnicas
  - Todo el software, lenguaje, librerías, base de datos y todo lo que esté relacionado con el desarrollo de la API deberá ser gratuito.
  - Debe poder conectarse a la plataforma Industore Global.

<sup>4</sup> <https://techlandia.com/crear-diagrama-contexto-sistema-cliente-como-230087/>

- El proyecto debe ser desarrollado en JavaScript al ser un lenguaje que los empleados del área de TI ya conocen (por cuestión de mantenimiento y mejoras).
- Administrativas
  - La API debe desarrollarse en un periodo no mayor a 4 meses hasta donde indica el alcance del proyecto.

➤ **Riesgos**

*Tabla 6: Riesgos*

<b>Tipo de riesgo</b>	<b>Descripción</b>	<b>Probabilidad</b>	<b>La acción a realizar</b>
Técnico	Las tecnologías elegidas no son las adecuadas para el proyecto.	Baja	Hacer los cambios pertinentes.
Técnico	Los tiempos de respuesta del servidor se elevan al cargar con muchos datos la base de datos.	Media	Buscar y optimizar consultas problemáticas.
Proyecto	Se le asignan tareas distintas del proyecto al residente.	Media	Ver la viabilidad y llegar a un acuerdo en cuanto al tiempo a destinarse.

## ➤ Bosquejo del sistema

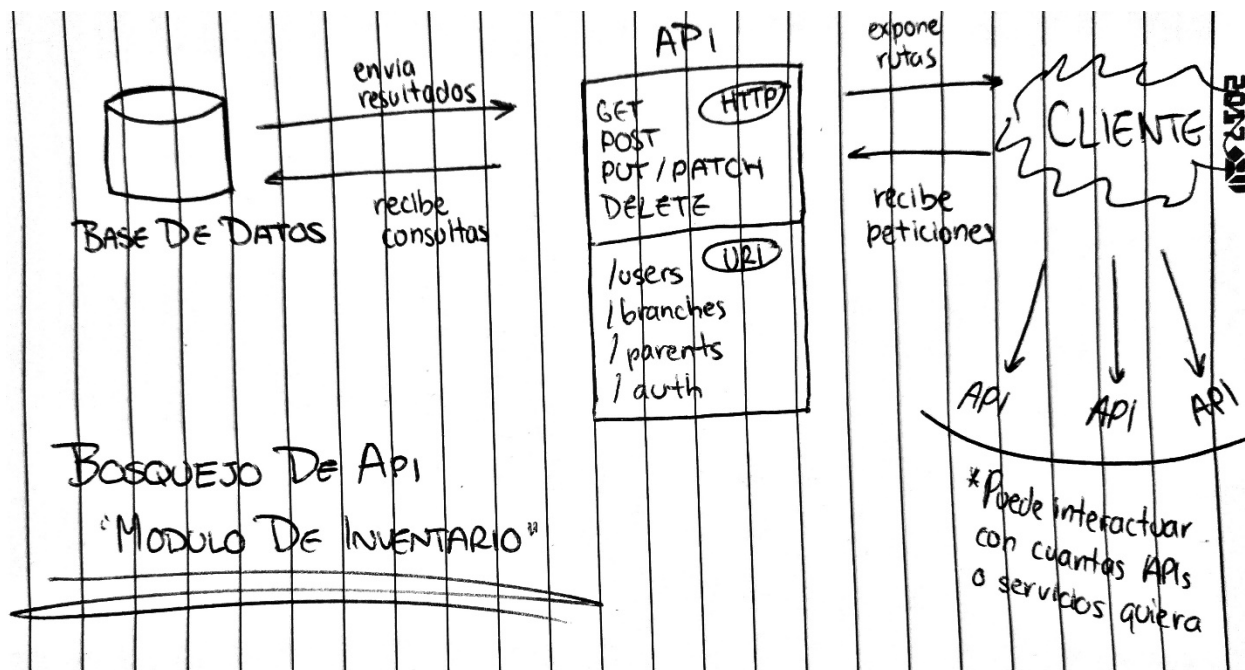


Ilustración 2: Bosquejo del sistema

## ➤ Entorno de operación

La API será utilizada por aplicaciones clientes mediante HTTP, principalmente con un frontend y un framework para la vista de cara al cliente y mediante Postman para los desarrolladores.

La API debe ser completamente independiente, pero al mismo tiempo estar preparada para conectarse a otros desarrollos (principalmente Industore Global) de la empresa, para de esta manera poder acceder a los datos que se encuentran en dicha aplicación. Es por lo anterior que separarse la lógica de las vistas, y manejar la aplicación como un módulo que se pueda adaptar a las necesidades de la empresa.

La base de datos, lenguajes, tecnologías, protocolos, etc., quedaron a criterio de los residentes trabajando en el proyecto, siempre y cuando fueran tecnologías que los empleados actuales de Encor Development supieran manejar para que ellos puedan manejar la aplicación en un futuro.

Debe ser una aplicación ligera y rápida, además de intuitiva y que presente información relevante para el usuario.

Se debe de manejar ciertos criterios de seguridad en cuanto a los usuarios para que no haya problemas con la información ya que pudiera llegar a ser sensible, así como el correcto manejo de las contraseñas mediante encriptación.

## Vistas arquitectónicas

### ➤ Arquitectura lógica

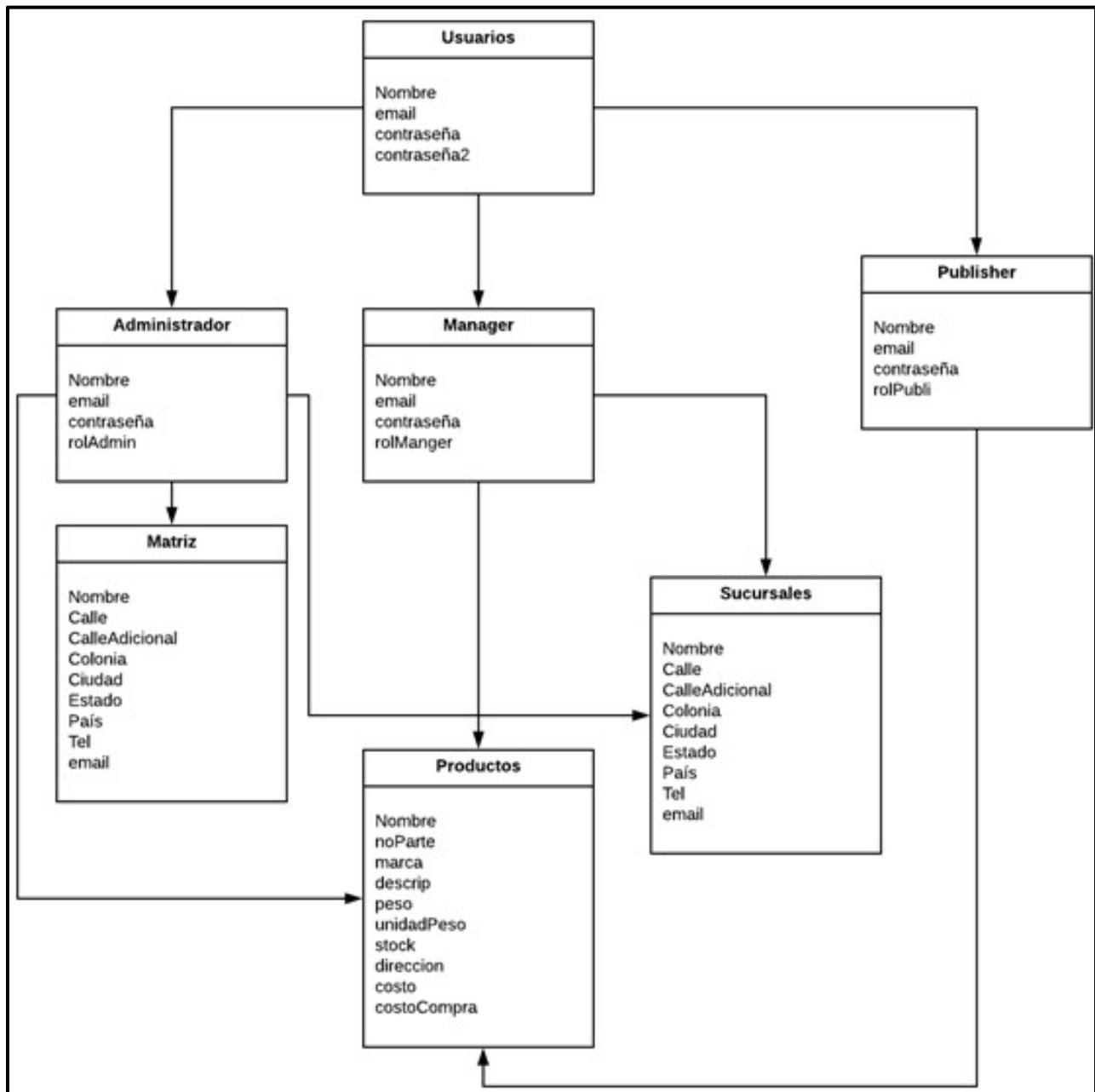


Ilustración 3: Diagrama de clases

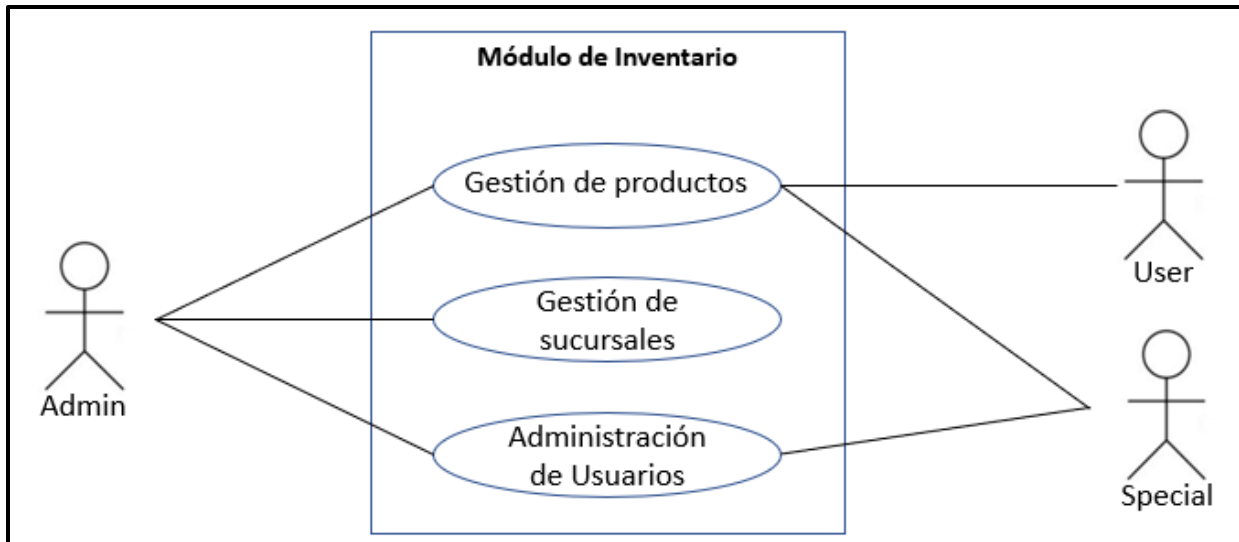


Ilustración 4: Diagrama de comunicación

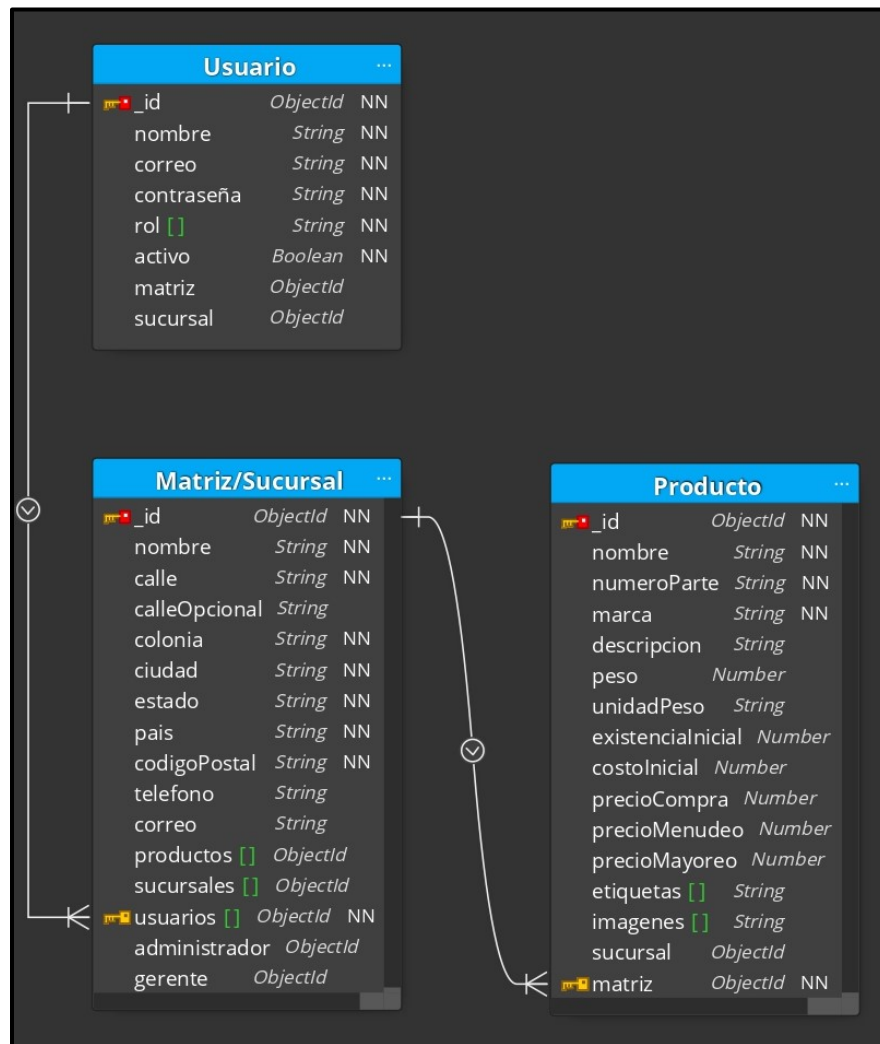


Ilustración 5: Diagrama E/R

## ➤ Arquitectura de desarrollo (despliegue)

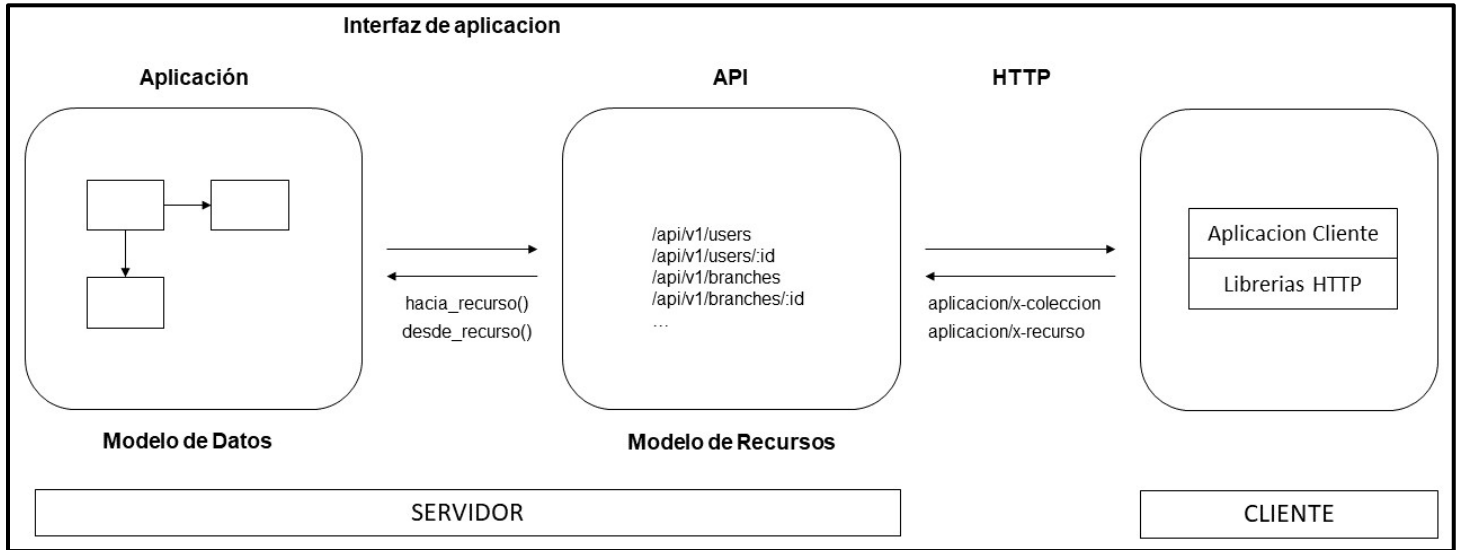


Ilustración 6: Diagrama de componentes

## ➤ Arquitectura física

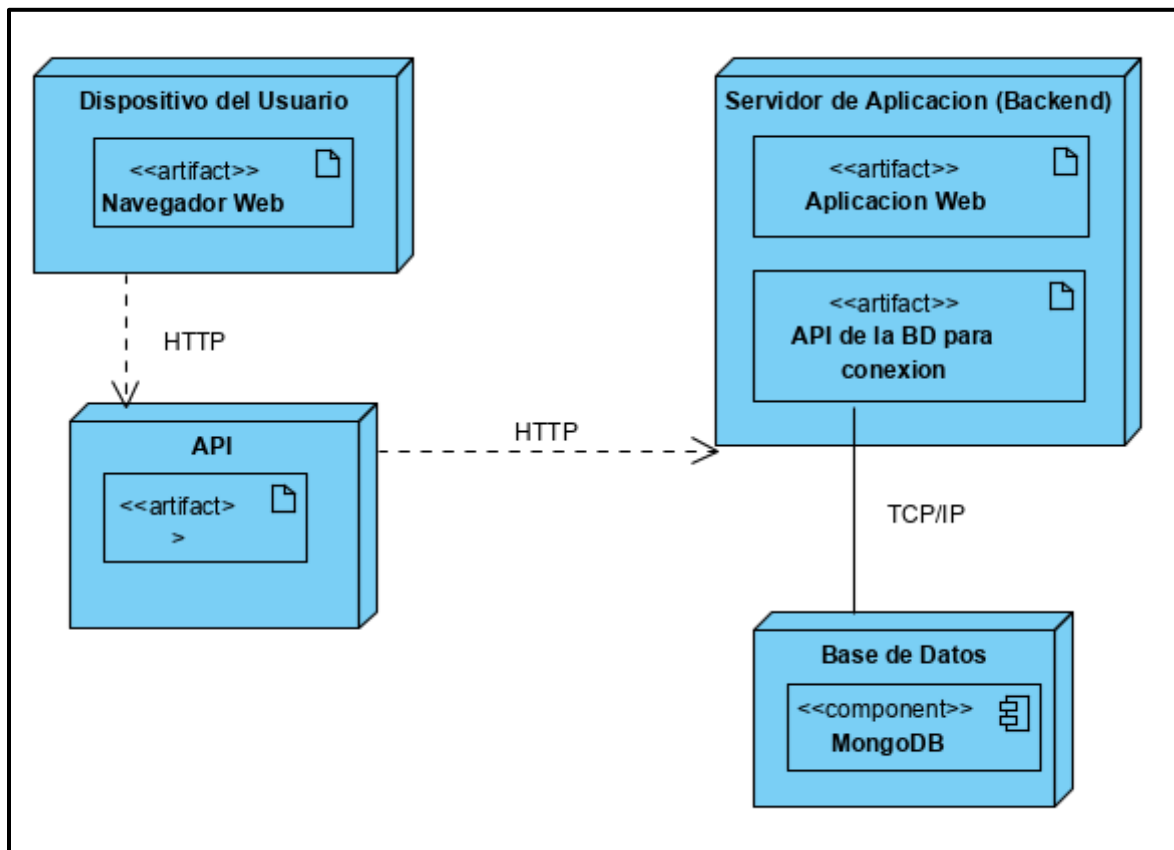


Ilustración 7: Diagrama de despliegue

## RESULTADOS

Para seguimiento de los entregables y firma de aceptación por parte del cliente, revisar la figura 25 en el apartado anexos.

### **Sprint 1: Usuarios**

- Objetivo

En este sprint se desarrollo lo que concierne a los usuarios, es decir, el poder crear una cuenta y acceder al sistema, también se desarrollo la base para que en futuros sprints se añadieran funcionalidades como asignar usuarios a matriz/sucursal, eliminar usuarios (solo administradores) y modificar los mismos.

- Código de Ejemplo



```

// @desc    Crear un usuario
// @route    POST /api/v1/users
// @access   Private [Admin, Manager]
exports.createUser = asyncHandler(async (req, res, next) => {
  // Campos a ingresar
  let { name, email, password, passwordConfirm, role, active } = req.body;

  // Buscar al usuario en la base de datos mediante su correo
  let user = await User.findOne({ email: req.body.email });

  // Validar que el usuario no exista previamente en la BD
  if (!isEmpty(user)) {
    return next(
      new ErrorResponse(
        `El correo electronico que ingresaste ya existe en nuestros registros.`,
        400
      )
    );
  }

  // Validar que la contraseña y su confirmacion coinciden
  if (password !== passwordConfirm) {
    return next(new ErrorResponse(`Las contraseñas no coinciden.`, 400));
  }

  // Si se trata de ingresar el rol de administrador se retorna un error
  if (role === "admin") {
    return next(new ErrorResponse("No puedes crear administradores.", 401));
  }

  // Crear el usuario
  user = await User.create({
    name,
    email,
    password,
    role,
    active
  });

  // Enviar respuesta
  res.status(201).json({
    success: true,
    data: user
  });
});

```

*Ilustración 8: Código Ejemplo - Creación usuario*



### Ilustración 9: Código ejemplo - Protección autenticación

- Pruebas realizadas en el sprint

Tabla 7: Prueba sprint 1

<b>Plan de pruebas</b>	Alta, modificación, eliminación y activación de usuarios
<b>Nombre</b>	CP-01
<b>Autor</b>	Erick Jara
<b>Fecha</b>	12/09/19
<b>Realizador de prueba</b>	Daniel Bordón
<b>Historia de usuario</b>	HU-USR-01, HU-USR-03, HU-USR-06 y HU-USR-07
<b>Descripción</b>	Se valida tanto la entrada como la salida
<b>Precondiciones</b>	Se ingresaron los datos del usuario
<b>Postcondiciones</b>	Se altera un registro en la base de datos
<b>Criterio de aceptación</b>	Usuario registrado o alterado

```
{
  "success": true,
  "data": {
    "role": "publisher",
    "active": false,
    "_id": "5df8d47360e48a37c8967e9a",
    "name": "Erick Jara",
    "email": "xczx@gmail.com",
    "createdAt": "2019-12-17T13:13:23.100Z",
    "__v": 0,
    "parent": "5df41202b10a741a68354e6e"
  }
}
```

Ilustración 10: Prueba sprint 1 - Activación usuario

## Sprint 2: Matrices

- Objetivo

Para que un inventario pueda tener control, es necesario separar matriz de sus sucursales para que los productos y usuarios estén siempre bien localizados. Es por lo anterior, que en este entregable se realizó todo lo correspondiente a las matrices para así poder continuar con los derivados de una matriz dada.

- Código

```
// Middlewares y Utilidades
const ErrorResponse = require("../utils/errorResponse");
const asyncHandler = require("../middlewares/asyncHandler");
const isEmpty = require("../utils/isEmpty");

// Carga de los modelos a utilizar
const Parent = require("../models/Parent");
const User = require("../models/User");

// @desc    Obtener matriz
// @route    GET /api/v1/parents
// @access   Private [Admin]
exports.readParent = asyncHandler(async (req, res, next) => {
  // Buscar matriz donde el campo administrador contenga el ID del usuario
  // haciendo la petición
  // Solo listar el nombre, dirección y contacto
  const parent = await Parent.find(
    { admin: req.user.id },
    "name address contact"
  );

  if (!isEmpty(parent)) {
    res.status(200).json({ success: true, data: parent });
  } else {
    return next(
      new ErrorResponse("No encontramos una matriz registrada.", 404)
    );
  }
});
```

*Ilustración 11: Código ejemplo - Listar matriz*

```

// @desc    Crear matriz
// @route    POST /api/v1/parents
// @access   Private [Admin]
exports.createParent = asyncHandler(async (req, res, next) => {
  // Buscar matriz donde el campo administrador contenga el ID del usuario
  // haciendo la peticion
  let parent = await Parent.findOne().where({ admin: req.user.id });

  // Si la query anterior retorna algun resultado, significa que
  // el usuario ya ha registrado una matriz y no tiene permitido crear otra
  if (!isEmpty(parent)) {
    return next(new ErrorResponse(`Ya tienes una matriz registrada.`, 403));
  }

  // Datos a llenar
  const newParent = new Parent({
    name: req.body.name,
    address: {
      street: req.body.street,
      streetAditonal: req.body.streetAditonal,
      district: req.body.district,
      city: req.body.city,
      state: req.body.state,
      country: req.body.country,
      zipCode: req.body.zipCode
    },
    contact: {
      phone: req.body.phone,
      email: req.body.email
    },
    // El campo admin sera igual al ID del usuario haciendo la peticion
    admin: req.user.id
  });

  // Creamos una nueva matriz con los datos proporcionados
  parent = await Parent.create(newParent);

  // Buscamos al usuario que realiza la operacion mediante su ID
  const user = await User.findById(req.user.id);

  // Insertamos el ID de la matriz y se la asignamos al usuario y guardamos
  user.parent = newParent.id;
  user.save();

  // Enviamos la respuesta
  res.status(201).json({
    success: true,
    data: parent
  });
});

```

*Ilustración 12: Código ejemplo - Crear matriz*

- Pruebas realizadas

*Tabla 8: Prueba sprint 2*

<b>Plan de pruebas</b>	Listar, alta, modificación y eliminación de matriz
<b>Nombre</b>	CP-02
<b>Autor</b>	Erick Jara
<b>Fecha</b>	12/10/19
<b>Realizador de prueba</b>	Daniel Bordón
<b>Historia de usuario</b>	HU-USR-04, HU-PRNT-001
<b>Descripción</b>	Existe la restricción de una sola matriz por administrador
<b>Precondiciones</b>	Se ingresan los datos de la empresa
<b>Postcondiciones</b>	Se altera un registro en la base de datos
<b>Criterio de aceptación</b>	Se registra, modifica o elimina una matriz

```
{  
  "success": false,  
  "error": "Ya tienes una matriz registrada."  
}
```

*Ilustración 13: Prueba sprint 2 - Solo una matriz por administrador*

### **Sprint 3: Sucursales**

- Objetivo

Una vez terminado el código correspondiente a dar funcionalidad a las matrices, fue posible comenzar con las sucursales y que estas fueran creadas como hijas de la matriz y de esta manera el administrador tendrá acceso al contenido tanto de su matriz como de las sucursales. Este entregable es muy similar al anterior ya que contiene (en su mayoría) los mismos datos, sin embargo, me asegure que existiera una relación directa entre la sucursal y su matriz por temas de seguridad.

- Código



```

// @desc    Eliminar sucursal
// @route    DELETE /api/v1/branches/:id
// @access   Private [Admin]
exports.deleteBranch = asyncHandler(async (req, res, next) => {
  // Buscar la sucursal mediante el ID
  const branch = await Branch.findById(req.params.id);

  // Validar que la sucursal exista
  if (isEmpty(branch)) {
    return next(new ErrorResponse("La sucursal proporcionada no existe.", 404));
  }

  // Validar que la sucursal a editar pertenezca al administrador
  if (req.user.parent.toString() !== branch.parent.toString()) {
    return next(
      new ErrorResponse(`Esta sucursal no pertenece a tu empresa.`, 401)
    );
  }

  // Eliminar sucursal
  branch.remove();

  // Enviar respuesta
  res.status(200).json({ success: true, data: {} });
});

```

*Ilustración 14: Código ejemplo - Eliminar sucursal*

- Pruebas realizadas

*Tabla 9: Prueba sprint 3*

<b>Plan de pruebas</b>	Listar, alta, modificación y eliminación de sucursal
<b>Nombre</b>	CP-03
<b>Autor</b>	Erick Jara
<b>Fecha</b>	12/11/19
<b>Realizador de prueba</b>	Daniel Bordón
<b>Historia de usuario</b>	HU-USR-05, HU-BRNC-001, HU-BRNC-002,
<b>Descripción</b>	Cada matriz puede tener “n” número de sucursales
<b>Precondiciones</b>	Se ingresaron los datos de la sucursal
<b>Postcondiciones</b>	Se altera un registro en la base de datos
<b>Criterio de aceptación</b>	Sucursal añadida, alterada o removida de la BD

```

{
  "success": true,
  "data": {
    "products": [],
    "users": [],
    "_id": "5df8e08b60e48a37c8967e9c",
    "parent": "5df41202b10a741a68354e6e",
    "name": "Sucursal Kino",
    "address": {
      "street": "Blvd. Kino #4635",
      "district": "Pitic",
      "city": "Hermosillo",
      "state": "Sonora",
      "country": "Mexico",
      "zipCode": "83254"
    },
    "contact": {
      "phone": 6622109824,
      "email": "sucursalkino@ford.mx"
    },
    "__v": 0
  }
}

```

*Ilustración 15: Prueba sprint 3 - Asignación de sucursal a matriz*

## **Sprint 4: Productos**

- Objetivo

Una vez se culminó la parte más complicada que era el tener una matriz con múltiples sucursales asignadas y usuarios, se pudo comenzar a crear productos, se dejó al final para que estos fueran asignados a la matriz o sucursal a la cual el usuario tiene acceso y de esta manera evitar problemas de seguridad o de integridad de la información en el sistema.

- Código



```

// @desc    Modificar un producto
// @route    PUT /api/v1/products/:id
// @access   Private ["Admin", "Manager", "Publisher"]
exports.updateProduct = asyncHandler(async (req, res, next) => {
  // Buscar producto mediante ID
  let product = await Product.findById(req.params.id);

  // Validar que exista el producto
  if (isEmpty(product)) {
    return next(new ErrorResponse(`No se encontró ese producto.`, 404));
  }

  // Validar que el usuario tenga derechos sobre el producto
  if (
    !isEmpty(product.parent) &&
    !isEmpty(req.user.parent) &&
    product.parent.toString() !== req.user.parent.toString()
  ) {
    return next(
      new ErrorResponse("Este producto no pertenece a tu matriz.", 401)
    );
  }

  // Validar que el usuario tenga derechos sobre el producto
  if (
    !isEmpty(product.branch) &&
    !isEmpty(req.user.branch) &&
    product.branch.toString() !== req.user.branch.toString()
  ) {
    return next(
      new ErrorResponse("Este producto no pertenece a tu sucursal.", 401)
    );
  }

  if (!isEmpty(req.body.tags)) {
    // Si el campo 'tags' no esta vacio hacemos la siguiente validacion
    // Removemos los posibles espacios dentro del campo 'tags' y despues separamos
    // por comas e insertamos en el arreglo
    req.body.tags = req.body.tags.replace(/ +/g, "").split(",");
  }

  // Actualizar el producto
  product = await Product.findByIdAndUpdate(req.params.id, req.body, {
    new: true,
    runValidators: false
  });
});

```

*Ilustración 16: Código ejemplo - Editar producto*

- Pruebas realizadas

Tabla 10: Prueba sprint 4

<b>Plan de pruebas</b>	Listar, alta, modificación y eliminación de productos
<b>Nombre</b>	CP-04
<b>Autor</b>	Erick Jara
<b>Fecha</b>	12/12/19
<b>Realizador de prueba</b>	Daniel Bordón
<b>Historia de usuario</b>	HU-PRNT-002, HU-PROD-001, HU-PROD-002
<b>Descripción</b>	El producto puede existir tanto en una matriz como en una sucursal, cada una es independiente y responsable de su inventario
<b>Precondiciones</b>	Se ingresa la información del producto
<b>Postcondiciones</b>	Se genera un registro en la BD y se puede visualizar, eliminar y modificar
<b>Criterio de aceptación</b>	El registro está bien formateado y se asigna al lugar correcto

```

{
  "success": true,
  "data": {
    "images": [
      "no-image.jpg"
    ],
    "_id": "5df8e14f60e48a37c8967e9d",
    "name": "BALERO 6200-ZZ",
    "partNumber": "6200-ZZ",
    "brand": "Generico",
    "description": "LOS RODAMIENTOS RÍGIDOS SON POPULARES DEBIDO A QUE ESTE TIPO DE RODAMIENTO SOPORTA CARGAS ALTAS. LOS RODAMIENTOS RÍGIDOS DE BOLAS SON POPULARES DEBIDO A QUE SON FÁCILMENTE MANTENIBLES.",
    "weight": 30,
    "weightUnit": "g",
    "initialStock": 5000,
    "initialCost": 76250,
    "buyPrice": 15.25,
    "retailPrice": 30.78,
    "wholesalePrice": 25.65,
    "tags": [
      "Bolas",
      "balero",
      "timken",
      "ejemplo"
    ],
  },
}

```

Ilustración 17: Prueba sprint 4 - Creación de producto

API RESTful que permite gestionar usuarios, matrices, sucursales y productos.

Edit

Manage topics

2 commits

1 branch

0 packages

0 releases

1 contributor

Branch: masterNew pull requestCreate new fileUpload filesFind fileClone or download

erickjara7 Update README.mdLatest commit 2dc4ef2 19 days ago

_data	Proyecto finalizado	19 days ago
config	Proyecto finalizado	19 days ago
controllers	Proyecto finalizado	19 days ago
middlewares	Proyecto finalizado	19 days ago
models	Proyecto finalizado	19 days ago
public	Proyecto finalizado	19 days ago
routes	Proyecto finalizado	19 days ago
utils	Proyecto finalizado	19 days ago
.gitignore	Proyecto finalizado	19 days ago
README.md	Update README.md	19 days ago
package-lock.json	Proyecto finalizado	19 days ago
package.json	Proyecto finalizado	19 days ago

Ilustración 18: Control de versiones, proyecto subido a repositorio remoto

## Normatividades, regulaciones y restricciones

### ➤ De documento

Para el documento se utilizó el patrón 4+1 vistas. El modelo 4+1 describe la arquitectura del software usando cinco vistas concurrentes. Tal como se muestra en la Figura 14, cada vista se refiere a un conjunto de intereses de diferentes Stakeholders del sistema.

- La vista lógica describe el modelo de objetos del diseño cuando se usa un método de diseño orientado a objetos. Para diseñar una aplicación muy orientada a los datos, se puede usar un enfoque alternativo para desarrollar algún otro tipo de vista lógica, tal como diagramas de entidad-relación.
- La vista de procesos describe los aspectos de concurrencia y sincronización del diseño.
- La vista física describe el mapeo del software en el hardware y refleja los aspectos de distribución.
- La vista de desarrollo describe la organización estática del software en su ambiente de desarrollo.

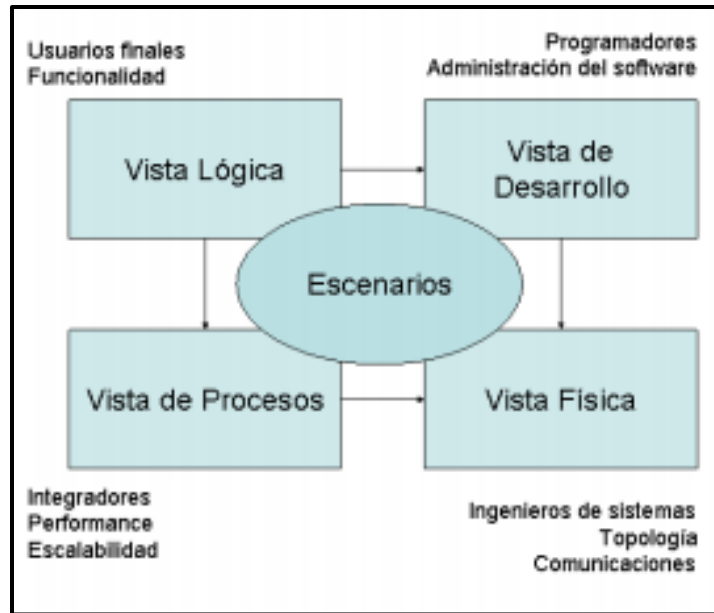


Ilustración 19: Modelo 4+1 Vistas

## ➤ De diseño

Se utilizaron los principios, técnicas y patrones de diseño descritos en el marco teórico y lo largo del documento, siendo esencialmente REST para la creación de APIs RESTful.

- **Recursos:**

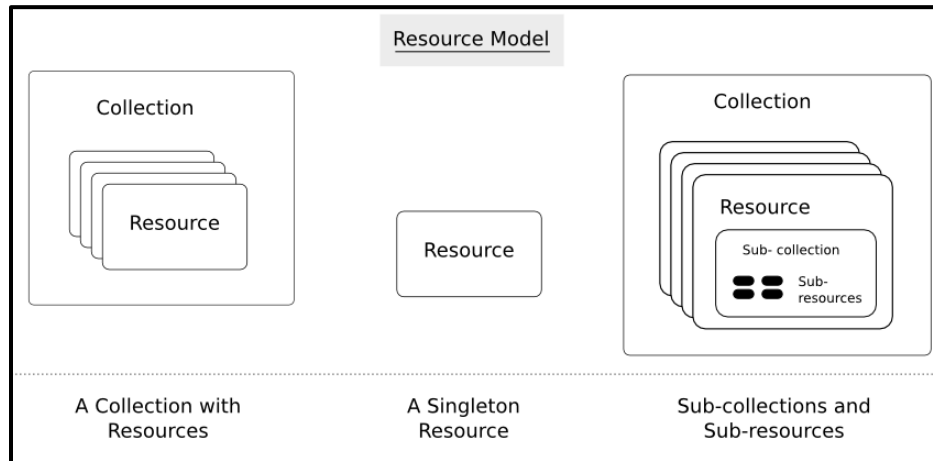
- Una API suele ser creada para resolver un problema de negocio de algún tipo, por lo cual, antes de hacer cualquier cosa, debemos decidir qué recursos queremos usar. Un recurso es un objeto con un tipo, datos asociados, relaciones a otros recursos, y una serie de métodos que operan con él.

- **Identificadores de Recursos:**

- Cada recurso creado debe tener un identificador, a esto se le llama URI (Uniform Resource Identifier) en la industria. Lo más importante a tener en cuenta con estos identificadores es que deben ser únicos, solamente deben identificar cierta entidad (ejemplo, el campo `_id` con un valor ObjectId creado por MongoDB en este proyecto).
- Para acceder a los recursos podemos hacerlo mediante las URIs, generalmente tienen esta forma:

- <http://industoreglobal.com/api/products>

- Si ejecutamos una petición HTTP GET en esta URI debemos obtener todos los productos. También podemos pensarlo como colecciones, la colección de productos contiene todas las entidades individuales de productos.



*Ilustración 20: Recursos y colecciones*

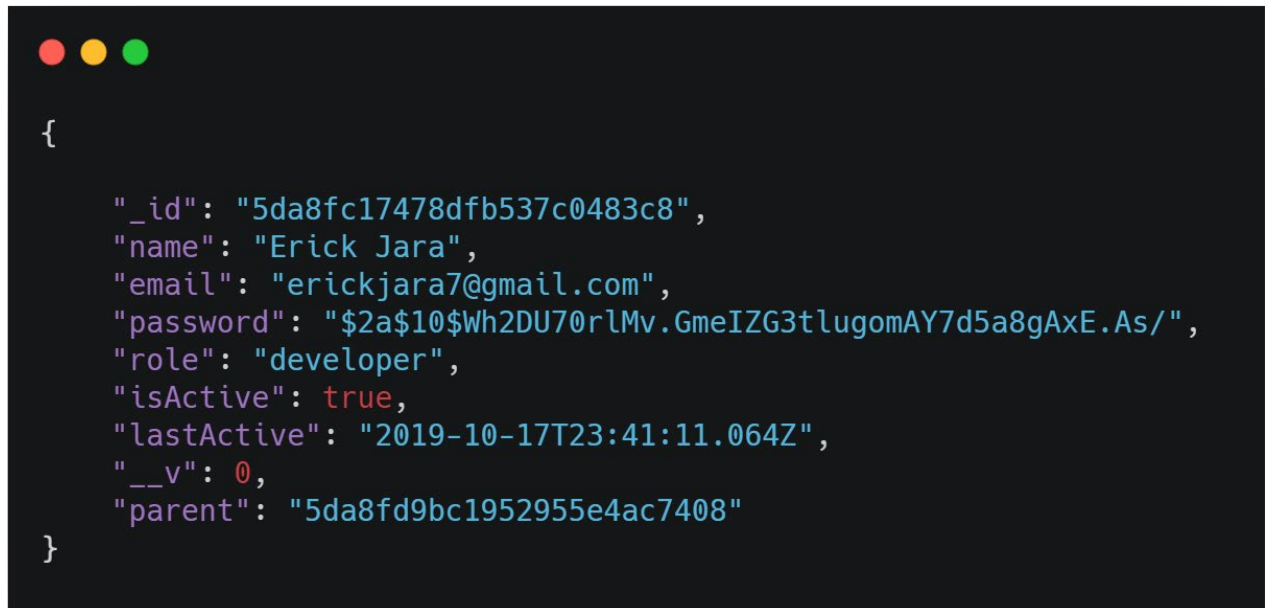
- **Jerarquía de los recursos:**

- Una vez los recursos/entidades y sus URIs correspondientes se pensaron de manera adecuada, se puede definir cómo organizar y representar la relación entre las entidades.
- En este proyecto desarrolle una API para un sistema de inventario. Las entidades resultantes fueron: Usuarios, Productos, Matrices y Sucursales. En este punto podemos pensar en la jerarquía. Los usuarios pertenecen a una matriz o sucursal y estos a su vez almacenan productos. Por lo tanto, podemos establecer las colecciones o URIs que nos permitan obtener, por ejemplo, una lista de todas las sucursales con sus productos y usuarios.
- Si quisiéramos obtener información sobre una sucursal en particular debemos ser capaces de dar como parámetro un ID de sucursal y de ahí hacer la siguiente petición HTTP GET:
  - <http://industoreglobal.com/api/sucursales/43396>
  - Si quisiéramos adentrarnos más, podríamos ver los usuarios asignados a esa sucursal, <http://industoreglobal.com/api/sucursales/43396/usuarios>

- **Representación de datos:**

- Se debe definir la estructura de datos que retornaran las peticiones, que datos y en que formato retornarlos cuando alguien solicite información.

- El formato más utilizado actualmente y el implementado en este proyecto es JSON (JavaScript Object Notation), aunque también se utiliza XML en otros softwares.
- Lo anterior sirve para que el cliente obtenga información de manera organizada y/o que el framework del frontend solamente se encargue de presentar dicha información de manera atractiva.
- Ejemplo de un JSON del módulo de inventario:



```
{
  "_id": "5da8fc17478dfb537c0483c8",
  "name": "Erick Jara",
  "email": "erickjara7@gmail.com",
  "password": "$2a$10$Wh2DU70rlMv.GmeIZG3tlugomAY7d5a8gAxE.As/",
  "role": "developer",
  "isActive": true,
  "lastActive": "2019-10-17T23:41:11.064Z",
  "__v": 0,
  "parent": "5da8fd9bc1952955e4ac7408"
}
```

*Ilustración 21: Respuesta JSON*

- **Utilizar métodos HTTP para peticiones:**

- **GET:** Retorna un recurso. Puede ser una colección entera o un recurso individual. Ambos basados en una URI:
  - <http://industoreglobal.com/api/productos>
  - <http://industoreglobal.com/api/productos/24356>
- **POST:** Crea un nuevo recurso en la URI especificada como parte de la petición. Esta petición requiere de un payload, que contiene los detalles del recurso a ser añadido, en el caso de esta API es el contenido que se envíe mediante **req.body**.
- **PUT:** Crea o reemplaza un recurso de una URI especificada.
- **PATCH:** Aplica una actualización parcial a un recurso especificado por una URI.

- **NOTA:** La diferencia entre PUT y PATCH es que en el método PUT, por ejemplo, si tenemos un formulario y solo llenamos el nombre y no los demás campos, estos quedarían vacíos, en cambio, si utilizamos PATCH y solo llenamos el campo nombre, solo este campo se actualizaría para dicho recurso.
- DELETE: Elimina un recurso en una URI específica.
- **Códigos HTTP (ejemplos)**
  - GET: 200 (OK) o 404 (Not Found).
  - POST: 201 (Created) o 400 (Bad Request).
  - PUT: 201 (Created) o 204 (No Content).
  - PATCH: 400 (Bad Request) o 204 (No Content).

### ➤ De código

Para el desarrollo de la API se hicieron uso de las buenas prácticas de programación, así como ciertos estándares y políticas para el desarrollo cuando el lenguaje JavaScript y NodeJS son utilizados.

#### 1. Utilizar métodos HTTP y rutas API.

Las rutas de la API siempre deben utilizar sustantivos como identificadores de los recursos. En el caso de este proyecto se ven de esta manera:

- **POST /products** para crear un nuevo producto,
- **GET /products** para obtener una lista de productos,
- **GET /products/:id** para obtener un producto,
- **PATCH /products/:id** para modificar un producto existente,
- **DELETE /products/:id** para eliminar un producto.

#### 2. Utilizar códigos de estado HTTP correctamente.

Si algo sale mal durante una petición, debemos enviar el código de estado correcto en la respuesta:

- **1xx**, informacional, la solicitud se recibió y se continua el proceso.
- **2xx**, exitoso, si todo salió bien.
- **3xx**, redirección, si el recurso se cambió de lugar y/o se requiere otra acción para completar la petición.

- **4xx**, error del cliente, si la petición no puede ser completada por un error en el cliente (como solicitar un recurso que no existe).
- **5xx**, error del servidor, si algo salió mal de lado de la API (como una excepción) o el servidor fallo en completar una petición aparentemente valida.

En el proyecto se utilizaron los siguientes códigos de estado en la respuesta:

- **200 OK**: Respuesta estándar para peticiones HTTP exitosas.
- **201 CREATED**: La petición ha sido realizada, resultando en la creación de un nuevo recurso.
- **204 NO CONTENT**: El servidor proceso exitosamente la petición y no retorna ningún contenido.
- **400 BAD REQUEST**: El servidor no puede o no podrá procesar la petición debido a un aparente error del cliente (como que un campo requerido no fue llenado).
- **401 NOT AUTHORIZED**: Similar al error 403 FORBIDDEN, pero este es específicamente para utilizarse cuando se requiera autenticación y esta haya fallado o no se ha provisto aún.
- **403 FORBIDDEN**: La petición contiene datos válidos y fue entendida por el servidor, pero este se rehúsa a realizar alguna acción. Puede deberse a que el usuario no tiene los permisos necesarios para un recurso o requiere una cuenta de algún tipo, o está intentando una acción que está prohibida (crear un registro duplicado cuando solo uno es permitido, por ejemplo).
- **404 NOT FOUND**: El recurso solicitado no se encontró, pero puede estar disponible en un futuro.
- **405 METHOD NOT ALLOWED**: El método HTTP para la petición no está soportado por el recurso solicitado, por ejemplo, una petición GET en un formulario que requiera ser presentado mediante POST.
- **500 INTERNAL SERVER ERROR**: Un error genérico, se proporciona cuando una condición inesperada fue encontrada y ningún mensaje es adecuado.
- **503 SERVICE NOT AVAILABLE**: El servidor no puede procesar la petición (porque se ha sobrecargado o está en mantenimiento). Generalmente es un estado temporal.

Al haber sido utilizado ExpressJS se enviaron los mensajes de error o éxito en un JSON junto con un código de estado en un middleware que se desarrolló.





```
res.status(200).json({
  success: true,
  data: user
});
```

*Ilustración 22: Respuesta enviada por el servidor*

### 3. Utilizar encabezados 'headers' HTTP para enviar metadatos.

Para adjuntar metadatos sobre la carga útil (payload) que se enviara se hace uso de los encabezados HTTP. Estos encabezados pueden servir para enviar información como autenticación o paginación.

En este proyecto se utilizaron varios encabezados como los siguientes (algunos generados automáticamente por paquetes utilizados):

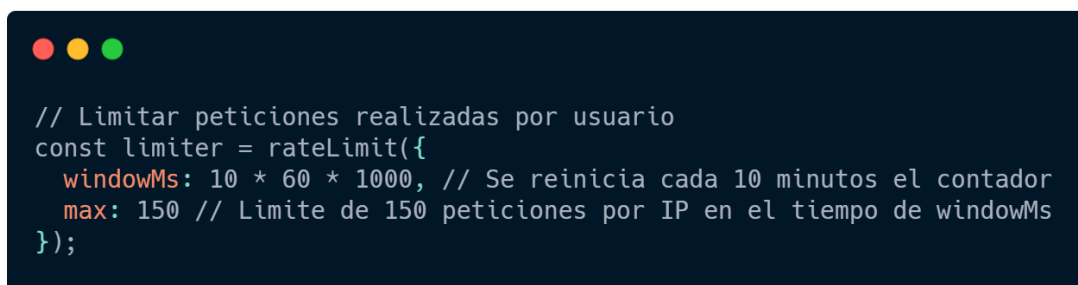
- AUTHORIZATION. Sirve para implementar autenticación mediante JWT (JSON Web Tokens) en un formato similar a este: **Bearer eyJhbGciOiJI..**
- X-Frame-Options, X-XSS-Protection: Prevenir ataques XSS.

### 4. Tener autenticación sin estado basada en JWT.

El tener autenticación sin estado te permite gestionarlo en el frontend lo cual es muy útil para equipos de desarrollo en los cuales hay varios involucrados. El servidor autentica mediante JSON Web Tokens al usuario donde va la información encriptada.

### 5. Soportar un límite en la tasa de peticiones.

Sirve para evitar ataques de negación de servicios (DDoS) ya que limita la cantidad de peticiones que puede hacer una IP al servidor en un tiempo determinado, estos parámetros son personalizables de manera sencilla para que el administrador del sistema pueda hacer los cambios que considere pertinentes.



```
// Limitar peticiones realizadas por usuario
const limiter = rateLimit({
  windowMs: 10 * 60 * 1000, // Se reinicia cada 10 minutos el contador
  max: 150 // Limite de 150 peticiones por IP en el tiempo de windowMs
});
```

*Ilustración 23: Limitar peticiones*

## 6. Tener en consideración la seguridad y los ataques posibles.

### ➤ De comunicación

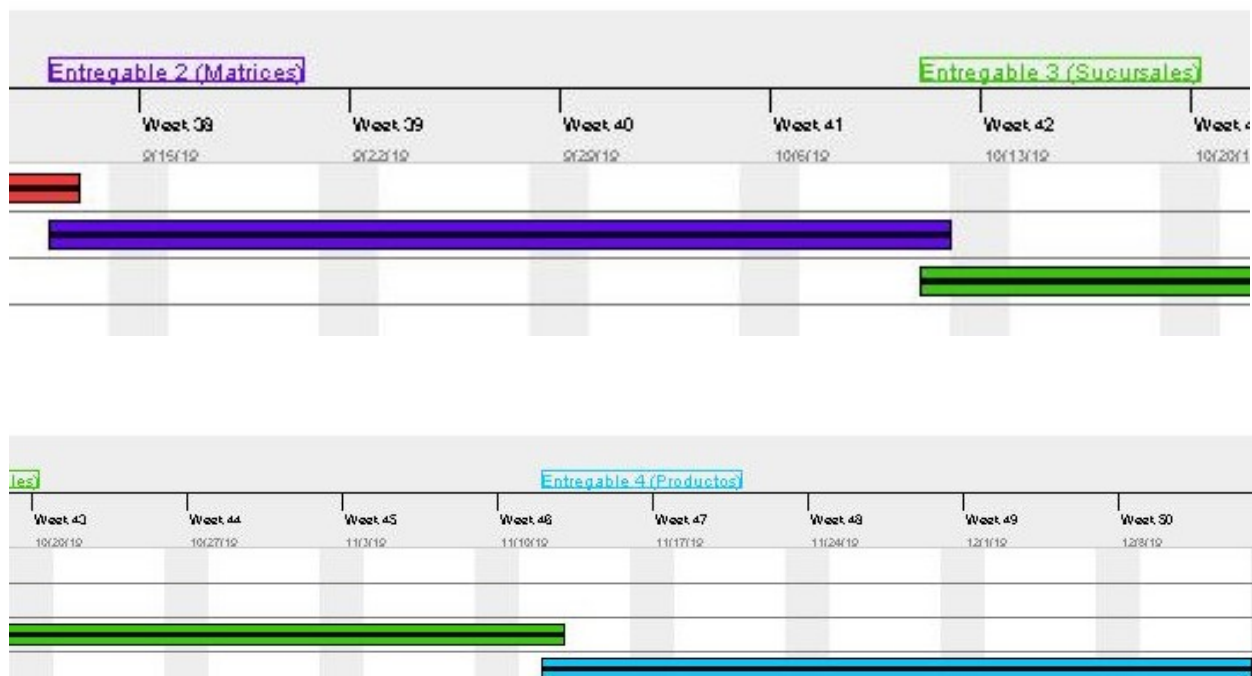
La comunicación con el equipo siempre fue en persona y solo pláticas breves para solucionar algún problema puntual o resolver dudas con respecto al proyecto, por lo tanto, no fue necesario un protocolo o estándar de comunicación, para este desarrollo se me dio mucha libertad por parte del cliente interno ya que ellos trabajan de esta manera.

### ➤ Tamaño y desempeño

- Planeación del proyecto (grafica de Gantt)



Ilustración 24: Grafica de Gantt



## **CONCLUSIONES**

Durante el tiempo que estuve en la empresa Encor Development realizando el proyecto aquí descrito considero que pude lograr bastante en poco tiempo, tomando en cuenta que al mismo tiempo llevaba dos materias que me tocaban de noveno semestre. Creo que el tiempo invertido se aprovechó al máximo dentro de las posibilidades y que fue un proceso de aprendizaje en el cual pude potencializar conocimientos que ya tenía pero que nunca había aplicado y de que mejor manera que en un proyecto de software real para un cliente (interno) que tiene necesidades reales.

Como es normal en los proyectos de software, me encontré con adversidades como lo son el tiempo ya que al mismo tiempo que desarrollaba el proyecto, me encontraba estudiando los conceptos, normas, lenguajes y tecnologías que el proyecto involucraba. Además, durante el transcurso de los meses me fueron involucrando en parte del proyecto Industore Global por lo cual mi atención se desviaba por cierta cantidad de días en la cual debía dar prioridad a lo que se me pedía.

Puedo decir que me encuentro satisfecho con lo logrado y que me queda un largo camino por recorrer para seguir creciendo como profesionista y que este proyecto me amplió la visión y aclaro el panorama un poco sobre qué es lo que la industria requiere hoy en día de los ingenieros en informática o sistemas y como debemos aprender a adaptarnos a la tecnología tan cambiante para poder establecernos como profesionales de calidad.

## **COMPETENCIAS DESARROLLADAS**

Se aplicaron muchos de los conocimientos y competencias adquiridas en la carrera, ya que muchos de los fundamentos y conceptos que no había tenido la oportunidad de poner en práctica ahora si pude hacerlo.

Las competencias que a mi parecer desarrolle durante este proyecto y extraído del perfil de egreso de ingeniería en informática son:

1. Analizar, modelar, desarrollar, implementar y administrar sistemas de información para aumentar la productividad y competitividad de las organizaciones.
2. Aplicar normas, marcos de referencia, estándares de calidad y seguridad vigentes en el ámbito del desarrollo y gestión de tecnologías y sistemas de información.
3. Identificar y aplicar modelos pertinentes en el diseño e implementación de base de datos para la gestión de la información en las organizaciones.

## FUENTES DE INFORMACIÓN

### Libros consultados:

- (1): Jin, B., Sahni, S. & Shevat, A. (2018). Designing web APIs: building APIs that developers love. Sebastopol, CA: O'Reilly Media.
- Jacobson, D., Brail, G. & Woods, D. (2011). APIs: a strategy guide. Beijing Sebastopol Calif: O'Reilly.
- Massé, M. (2012). REST API design rulebook. Sebastopol, CA: O'Reilly.

### Sitios web y documentos consultados:

- Architectural Blueprints—The “4+1” View Model of Software Architecture. Por IEEE, noviembre 1995, pp. 42-50  
<https://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf>
- Metodología de desarrollo iterativo y creciente  
Sitio web:  
[https://www.ecured.cu/Metodolog%C3%ADa\\_de\\_desarrollo\\_iterativo\\_y\\_creciente](https://www.ecured.cu/Metodolog%C3%ADa_de_desarrollo_iterativo_y_creciente)

## DEFINICIONES

Tabla 11: Definiciones



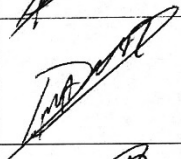

Definición, Acrónimos y Abreviaciones	Concepto
NodeJS	Entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor. Servidor Web.
ExpressJS	Es el framework web más popular de NodeJS. Está diseñado para construir aplicaciones web y APIs.
JavaScript	Es un lenguaje de programación interpretado, dialecto del estándar ECMAScript.
JWT	JSON Web Token. Estándar abierto basado en JSON propuesto por IETF (RFC 7519) para la creación de tokens de acceso que permiten la propagación de identidad y privilegios.
HTTP	Hypertext Transfer Protocol. Protocolo de transferencia donde se utiliza un sistema mediante el cual se permite la

	transferencia de información entre diferentes servicios y los clientes que utilizan páginas web.
API	Application Programming Interface. Conjunto de reglas (código) y especificaciones que las aplicaciones pueden seguir para comunicarse entre ellas.
REST	Representational State Transfer. Es un estilo de arquitectura para proveer estándares entre sistemas de computación en la web, haciendo más sencilla la comunicación entre sistemas.
RESTful	Son los sistemas que adoptan el estilo de arquitectura REST, se caracterizan por separar el cliente y el servidor, mejorando la escalabilidad y flexibilidad.
WWW	Sistema de distribución de documentos de hipertexto o hipermedia interconectados y accesibles a través de Internet.
URL (endpoint)	Es un sistema de distribución de documentos de hipertexto o hipermedia interconectados y accesibles a través de Internet. Secuencia estándar de caracteres que permite localizar y recuperar una información determinada en Internet.
URI	Uniform Resource Identifier. Sirve para identificar recursos en Internet, precisamente lo que el nombre indica.
Recurso	Es un objeto con un tipo, datos asociados, relaciones con otros recursos y un conjunto de métodos que operan con él. Es similar a una instancia de un objeto, con la diferencia de que este solo tiene unos cuantos métodos correspondientes a HTTP: GET, POST, PUT y DELETE.

UML	Unified Modeling Language. Lenguaje gráfico para visualizar, especificar, construir y documentar un sistema.
Usuario	Cualquier usuario que este registrado en la aplicación web.
MERN	MongoDB. ExpressJS. ReactJS. NodeJS. Stack tecnológico para crear aplicaciones full-stack.
SCRUM	Metodología ágil que sirve para que equipos multidisciplinares trabajen en entornos complejos, donde los requisitos son muy cambiantes, y los resultados se tienen que obtener en un plazo corto de tiempo.
Sprint	Se trata de un miniproyecto de no más de un mes (ciclos de ejecución muy cortos - entre una y cuatro semanas), cuyo objetivo es conseguir un incremento de valor en el producto que estamos construyendo.
Planning Poker	Técnica para calcular una estimación basada en el consenso, en su mayoría utilizada para estimar el esfuerzo o el tamaño relativo de las tareas de desarrollo de software.
Backend	Parte del programa que se ejecuta en un servidor (en vez de en el navegador o dispositivo del cliente).
Frontend	Trabaja principalmente del lado que se ve, del lado que podemos ver todos los usuarios de un sitio web o una aplicación web. Es la parte visual de un programa.

## ANEXOS

- Seguimiento Entregas Funcionales

Entregable No.	Tema principal	Fecha Entregable	Observaciones del cliente	Aceptación del cliente. Firma.
1	USUARIOS	12/09/19	A falta de la funcionalidad de matrices y sucursales, todo OK	
2	MATRICES	12/10/19	Validación de solo una matriz por administrador	
3	SUCURSALES	12/11/19	Todo OK.	
4	PRODUCTOS	12/12/19	Todo OK.	



Jesus Ivan Aruco Bejarano

Ilustración 25: Seguimiento Entregas Funcionales