

## **Sistema de Inventario**

**John Alejandro Pastor Sandoval**

**Brallan Ferney Mendoza Delgado**

**Marbel Alexandra Garavito Cordero**

**Santiago Bejarano Ariza**

**Marie Virazaels**

**Juan David Loaiza Reyes**

**No. de equipo de trabajo: 6**

### **I. INTRODUCCIÓN**

El objetivo del proyecto es diseñar un software que funcione como un sistema de inventario el cual cuenta con varias funcionalidades dependiendo el tipo de usuario que lo maneje. Para este software se usan conceptos de estructuras de datos secuenciales, conceptos de programación orientada a objetos como clases y herencias las cuales serán importantes para el desarrollo del software, además de que la información necesaria de los productos y usuarios será almacenada en una base de datos la cual en gran medida será accedida por medio de métodos implementados en el programa. También en el proyecto se describe cada función, como se realizó, el tiempo que usa para realizar ciertas funciones con cierta cantidad de datos y hacer un resumen sobre los fallos y acierto que presenta este primer prototipo desarrollado por el grupo de trabajo.

### **II. DESCRIPCIÓN DEL PROBLEMA A RESOLVER**

En el proyecto se busca crear un sistema de inventario que sea capaz de simular una venta de productos, permitiendo al usuario acceder de manera eficaz a cada producto buscando por el nombre del producto, también se busca usar estructuras de datos avanzadas capaces de optimizar este proceso y que permitan ampliar la capacidad de búsqueda en el programa, por ejemplo, un filtro para los productos que permitirá que la búsqueda sea más rápida.

### **III. USUARIOS DEL PRODUCTO DE SOFTWARE**

En el software existen dos tipos de usuario, un vendedor y un comprador, ambos usuarios dependen del acceso a un correo electrónico y una contraseña para poder usar el software. Además, en una base de datos se guardarán cada uno de los usuarios con su información. En esta base de datos los usuarios tendrán dos tipos de identificadores, que permitirán definir qué funcionalidades tendrá acceso cada tipo de usuario, por ejemplo, el vendedor tendrá la funcionalidad de crear, borrar, editar y leer productos, mientras que el comprador tendrá acceso solo a las funcionalidades de: comprar, devolver, o pedir un producto específico (Funciones que un vendedor también puede realizar).

### **IV. REQUERIMIENTOS FUNCIONALES DEL SOFTWARE**

#### **Funcionalidades habilitadas para ambos usuarios.**

##### **Autenticación de Usuarios:**

La autenticación de usuarios da acceso a estos mismos al software dependiendo si se encuentran en la base de datos, además también permite conocer el rol de cada usuario y de esta manera permitirles el acceso a las distintas funcionalidades.

El software le pedirá al usuario que digite su correo electrónico y contraseña, el software realizará una verificación y permitirá el acceso al usuario al programa y a las respectivas funciones. En caso de que el usuario no se haya registrado en la base de datos puede crear un nuevo usuario para acceder al programa.

##### **Compra de Productos:**

La compra de productos permitirá al usuario ver los productos disponibles en el inventario y podrá comprar artículos. El software le preguntará al usuario por el producto que desea comprar y la cantidad y dependiendo del stock que hay determinará si es posible la venta o si toca comprar una cantidad menor.

##### **Pedir producto:**

Si el usuario no encuentra el producto que desea en el inventario, tiene la posibilidad de pedir este producto por encargo, de tal manera que se incluya en el inventario para su respectiva venta. El software le permitirá al usuario digitar el producto que desea pedir y la cantidad que este desea. En este prototipo el pedir producto crea un ítem con precio nulo, de manera que se busca que el vendedor después de pedir el producto le establezca el precio pertinente.

##### **Devolver producto:**

En caso de que el usuario desee devolver un producto, puede hacerlo y el producto se devolverá al inventario con su respectiva cantidad. El programa preguntará al usuario el producto que desea devolver y hará el respectivo proceso para devolver el producto al inventario.

Funcionalidades habilitadas solo para el Usuario con rol de vendedor.

### Crear producto:

Esta funcionalidad permitirá al vendedor agregar un nuevo producto al sistema de inventario con sus respectivas propiedades. El software le pedirá al usuario el nombre, precio y stock inicial del producto para agregarlo al inventario.

### Borrar producto:

Esta funcionalidad permitirá que el vendedor pueda borrar un producto del sistema de inventario. El software sólo pedirá el id del producto y lo ubicará para respectivamente eliminar el producto del inventario.

### Editar producto:

Si el usuario desea cambiar las propiedades de un producto el software le pedirá el id del producto para después actualizar los datos que el usuario desee.

### Leer producto:

En caso de que el usuario solo desee verificar la existencia del producto o quiera conocer la existencia de este, puede acceder esta funcionalidad donde puede dar el nombre del producto y el software le dirá si el producto se encuentra o no en el inventario.

## V. DESCRIPCIÓN DE LA INTERFAZ DE USUARIO PRELIMINAR

En esta versión por motivos de tiempo, se decidió que el desarrollo de los mockups sean avanzados pero que la manera como el usuario interactúa con las funcionalidades sea a través de consola, cosa que en siguientes versiones claramente será actualizado a algo gráfico, es importante decir que como se hace todo a través de consola se limitan varios aspectos como legibilidad y orden, y la forma del menú es más cercana a una pregunta-respuesta que una interfaz de usuario, esto obviamente por las limitaciones de la consola



Sign up

ID:

Name:

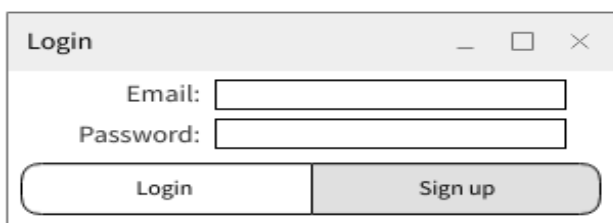
Email:

Cellphone:

Password:

☒ Buyer ☐ Seller

Create Back

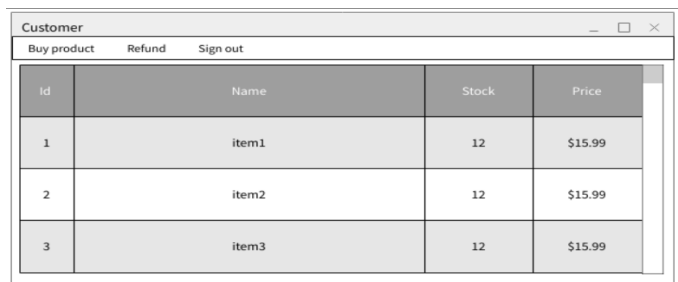


Login

Email:

Password:

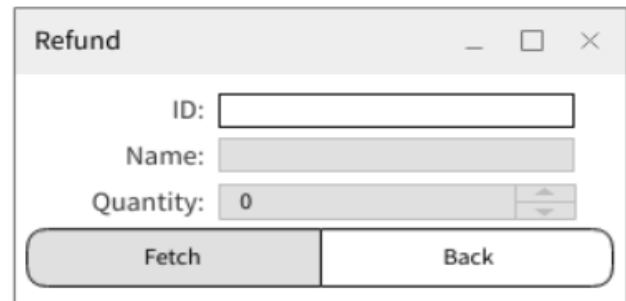
Login Sign up



Customer

Buy product Refund Sign out

Id	Name	Stock	Price
1	item1	12	\$15.99
2	item2	12	\$15.99
3	item3	12	\$15.99



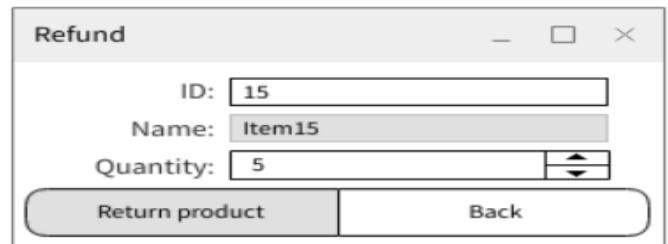
Refund

ID:

Name:

Quantity:

Fetch Back



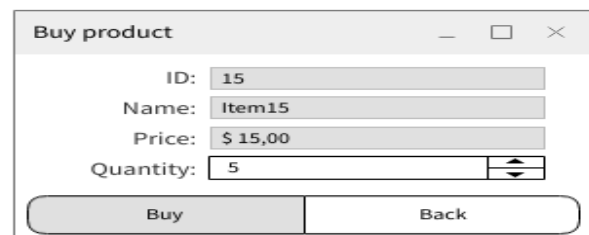
Refund

ID:

Name:

Quantity:

Return product Back



Buy product

ID:

Name:

Price:

Quantity:

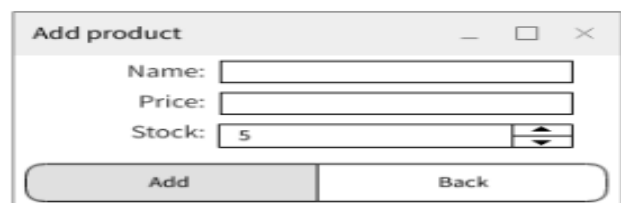
Buy Back



Seller

Add product Update product Delete product Sign out

Id	Name	Stock	Price
1	item1	12	\$15.99
2	item2	12	\$15.99
3	item3	12	\$15.99



Add product

Name:

Price:

Stock:

Add Back



Update product

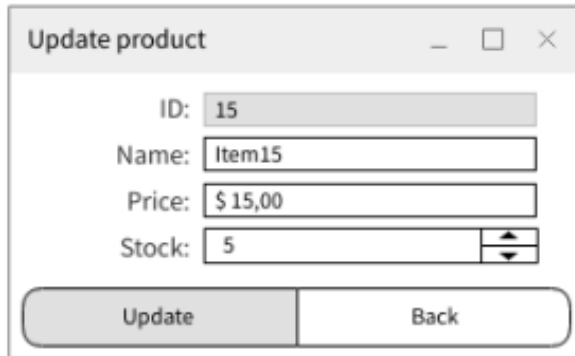
ID:

Name:

Price:

Stock:

Fetch Back



Update product

ID:

Name:

Price:

Stock:

Update Back



Delete product

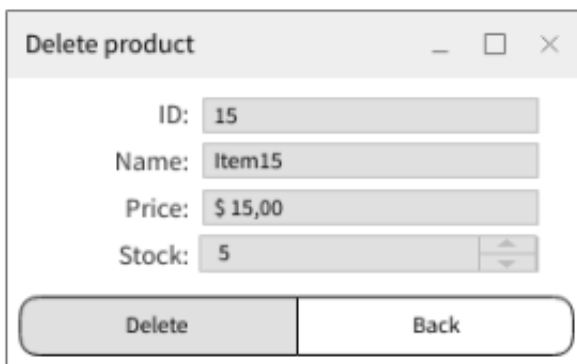
ID:

Name:

Price:

Stock:

Fetch Back



Delete product

ID:

Name:

Price:

Stock:

Delete Back

## VI. ENTORNOS DE DESARROLLO Y DE OPERACIÓN

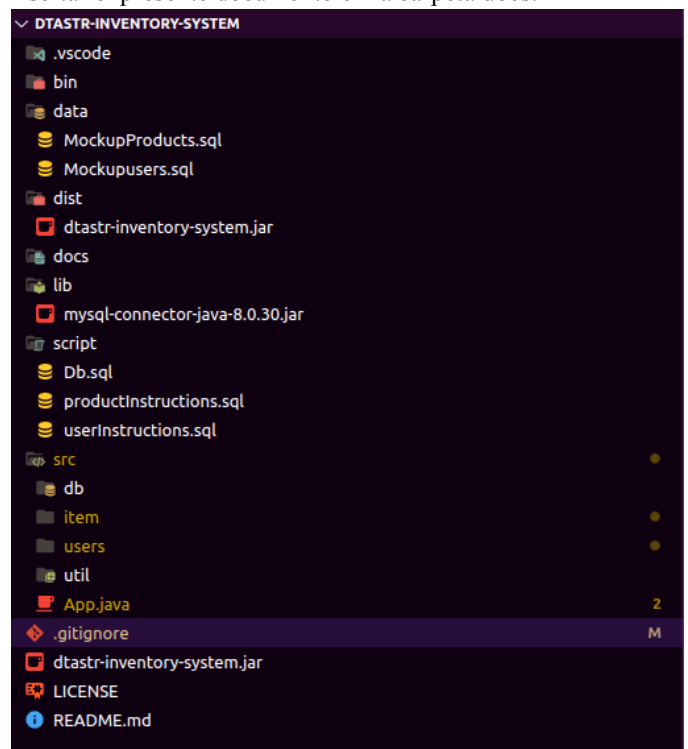
En el entorno de desarrollo se usó el lenguaje de programación Java, en este lenguaje usaron principalmente las librerías: Java Utility Library, Java Database Connectivity API (JDBC) para el manejo de las bases de datos y de estructuras de datos secuenciales. Como herramientas de desarrollo se utilizó el editor de código Visual Studio Code, en este editor trabajamos

los 6 integrantes del proyecto donde cada uno se enfocó en desarrollar funcionalidades específicas para el funcionamiento del software. También usamos herramientas como el cliente MySQL para el manejo de la base de datos y el repositorio online GitHub para facilitar el manejo compartido del código con el grupo de trabajo.

En cuanto a especificaciones que necesitará un dispositivo para usar el software. El software está hecho para ser usado en los sistemas operativos: Windows 8 x64, Ubuntu Linux 18.04 LTS x64 y OS X 10.11+ x64, como requisitos mínimos para que el software funcione correctamente se recomienda el uso de un dispositivo que tenga un procesador de mínimo 1GHZ puede ser Ryzen o Intel de quinta generación por ejemplo, en cuanto a memoria Ram lo recomendado es que sea de 1 GB mínimo y 4 GB de espacio libre en el disco duro, también es necesario tener instalado tanto el JDK como MySQL.

## VII. PROTOTIPO DE SOFTWARE INICIAL

En el prototipo, esta es la estructura que se usa en el repositorio (Visualizado desde VS code). En la imagen falta insertar el presente documento en la carpeta docs.



Cabe aclarar que el archivo dentro de dist apunta a una base de datos con el usuario y contraseña de Juan David Loaiza Reyes esto porque fue quien compilo el proyecto. El código fuente es encontrado.

<https://github.com/juanloaiza21/dtastr-inventory-system>

## VIII. DISEÑO, IMPLEMENTACIÓN Y APLICACIÓN DE LAS ESTRUCTURAS DE DATOS

Antes de la descripción de clases entra la aclaración de que los paquetes `item` y `user` contienen un archivo clase `Main`, que tiene la función de ser un controlador que hace implementaciones a través de consola para facilitar la codificación del archivo principal del proyecto, el cual es `App.java`. La funcionalidad de estos “main” es conectarse con `App` y hacer implementaciones de métodos por consola ya creados en cada uno de estos.

**Class Item:** la clase `Item` permite determinar las características que tendrá cada producto, es decir el id, el nombre, el precio y la cantidad existente, en esta clase se implementan métodos para obtener todos los productos, donde usando un método para obtener todos los productos de la base de datos los agrega a una lista enlazada para después implementar otro método que al recibir la lista donde fueron almacenados los productos y un nombre, donde devolverá el producto si se encuentra en la lista, si no devolverá un valor nulo.

**Clase Users:** Esta clase funciona como una interfaz para métodos que se implementaran en la clase `User`, además de usar listas enlazadas para almacenar los datos de los usuarios recibiendo como parámetro una lista con uno o varios datos, también implementa métodos para validar el rol y el id de cada usuario para verificar que si existen en la base de datos.

**Clase User:** La clase `User` implementa los métodos previamente creados en la clase `Users` para desarrollarlos y crear las funcionalidades específicas para el funcionamiento del prototipo.

Como primera medida implementa el método `login` que devuelve en tipo de dato booleano, donde al recibir un email y una contraseña verifique si existe el usuario. Los métodos de `update` son funciones que usando listas llenan cada espacio determinado por un índice que define el tipo de dato que debe existir, entonces así se actualizan nombres, correos y otros tipos de datos usando listas. También se implementa un método que permite encriptar las contraseñas haciendo cambios entre sistemas hexadecimales y binarios. Finalmente existe un método para crear usuarios donde usando métodos previamente hechos en la clase `Users` permite agregar el usuario y los datos necesarios a la base de datos.

**Interfaz Query:** La interfaz `Query` crea los métodos sin implementar las funcionalidades que se usarán en la clase `Connector`.

**Class Conector:** La clase `Conector` se encarga principalmente de crear las conexiones del software con la base de datos para obtener los datos necesarios de los usuarios y los productos, además de implementar los métodos necesarios para la manipulación de productos y usuarios, por ejemplo los métodos de inserción de datos reciben los datos y los campos que se buscan añadir, entonces para esto genera secuencias

SQL con los datos que se quieren insertar, de esta manera luego usa métodos creados específicamente para ejecutar las secuencias SQL para alterar la base de datos y generar los cambios.

Los métodos para eliminar uno o todos los datos funcionan de una manera similar al método para insertar, debido a que también genera secuencias SQL específicas que permiten eliminar los datos de la base de datos.

También existen métodos que permiten mostrar en consola los ítems o usuarios de manera ordenada con sus respectivos atributos y valores, donde solo usando operaciones de cadenas de texto y obteniendo los valores actuales de la base de datos los muestra al usuario.

Existen varios métodos “get” que son específicamente para mostrar o devolver los datos de las bases de datos, primero se conectan a la base de datos y preparan instrucciones específicas para que el usuario añada los datos que quiere obtener y así generar la secuencia SQL que muestra los datos que pida.

Finalmente los métodos que permiten usar de manera óptima las secuencias SQL y crearlas de una manera simple se conocen como los métodos `query`, estos métodos generan nuevas cadenas de texto y dependiendo los datos que el usuario proporcione genera la secuencia que será usada luego para ejecutarla y realizar el proceso que necesite.

Las estructuras de datos que implementa este módulo son las `linked list` y los `arrays`, estos proviniendo de la clase de `java.util`.

**Class Sell Items:** Esta clase permite la interacción entre el usuario y el software para realizar la respectiva compra así como los métodos para realizar la actualización de stock de los datos ya fueron implementados en otras clases, se procede a importar estas clases y sus métodos de manera que solo se necesitan entradas por consola para su funcionamiento. Por lo tanto solo se usa un método para vender que depende de una lista enlazada para mostrar todos los productos permitiendo su búsqueda rápida y su actualización dependiendo de la operación que realice el usuario.

**Class AskProducts:** En esta clase se implementa principalmente la función que permite al usuario pedir un producto y la cantidad que desee comprar, en caso de que el producto no se encuentre en el inventario, usando una cola para priorizar el primer pedido que se haya realizado cada pedido se va añadiendo a la base de datos con un precio de 0 para que luego el vendedor le establezca el precio a este producto.

**Class Refund:** En esta clase se implementa una función que permite al usuario devolver un producto que haya comprado previamente, tiene un funcionamiento similar al de compra debido a que en vez de descontar stock de la base de datos, se aumenta.

## IX. PRUEBAS DEL PROTOTIPO Y ANÁLISIS COMPARATIVO

Para las pruebas de entorno y análisis comparativo se escogieron 3 funcionalidades, el login (User.login), donde para hacer la prueba haremos la cantidad de login como datos de pruebas, insert item e insert users, esto por como la data pasa de ser string a un array y luego a una linked list. La manera de realizar la prueba será un tanto rudimentaria, lo haremos mediante for en un archivo nuevo a la altura de App.java, estos for tendrán la cantidad de iteraciones iguales a la cantidad de datos. Para el login se usará la cantidad de logins por la imposibilidad computacional local (El equipo de pruebas cuenta con un i5 de 10ma, 8 de ram y una 1050, pero al abrir tantas instancias de java no resiste la pc) de hacer un millón de login a la vez, se harán un millón de login consecutivos y se promedió el tiempo mediante la fórmula:

$$\frac{(Tf_0 - Ti_0) + (Tf_1 - Ti_1) + \dots + (Tf_{n-1} - Ti_{n-1})}{n}$$

En esta fórmula Tf representa el tiempo final, Ti el tiempo inicial y el subíndice es la iteración, todo esto dividido por el número de datos totales que representan”.

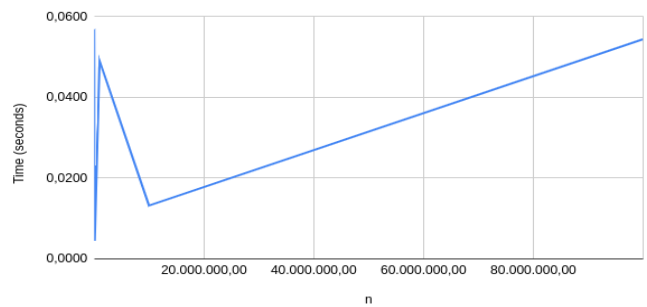
Para los métodos insert Item (askProducts.adding) e insert Users (User.createUser) generamos un n número de datos mediante java y la iteración de for añadiendo solamente el número de iteraciones, después de eso se empieza a calcular el tiempo añadiendo los datos, es en este momento donde empezamos a hacer cálculo de tiempo del tiempo total para la inserción de estos datos.

Todos los tiempos serán medidos con la función de java System.nanoTime(). La elección de nano responde a que la ejecución puede ser menor a mili segundos, esto quiere decir que todos los cálculos dentro de las tablas serán entregados en nano segundos. Todas las pruebas se anidaran en el archivo test, que tendrá una rama propia en github.

Finalmente y a modo de aclaración antes de empezar con las tablas, se hacen ciertas modificaciones a los métodos, en especial lo que devuelven estos sobre consola, ya que no es necesario ver n veces un mensaje tipo “Agregado exitosamente”

Login			
n	Time (nano)	Time (seconds)	Tiempo ejecución total (minutos)
10.000,00	12569382	0,0569	9,483333333
100.000,00	451887	0,0045	7,53145
1.000.000,00	490430	0,0490	816,6666667
10.000.000,00	132432	0,0132	2200
100.000.000,00	544032	0,0544	90666,66667

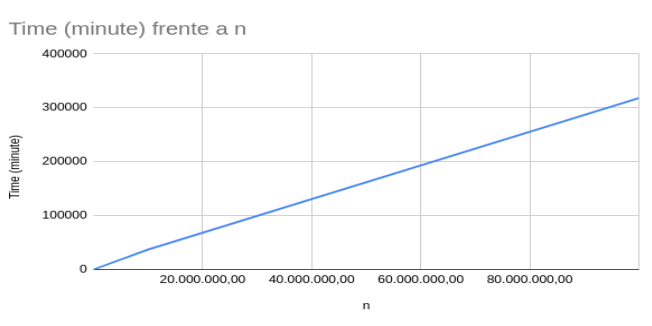
Time (seconds) frente a n



En la tabla del login, vemos como todos los tiempos rondan entre los 0.0569 y 0.544, esto se lo atribuimos a múltiples factores, entre ellos que está escribiendo y obteniendo data desde un HDD y demás, a partir de los 10 millones de datos se decidió generar aleatoriamente los números en rangos que ya teníamos, ya que el hecho de hacer la prueba con 10 millones de datos nos pudo haber tomado mediodía, y con 100 millones el tiempo estimado es de alrededor de 24 días. Esto se debe a la forma de hacer el testeó con los datos, que se encuentra en el [github](#) rama test, se hizo de manera iterativa, es decir en O(n) donde se aunque la función de login es de tipo O(1), los tiempos se juntan y es algo irrisorio el tiempo que hay que esperar.

El tiempo de ejecución de la función login tiene una dificultad de O(n) dado que, al brindar el usuario y la contraseña, comparar si tiene el rol “seller” es una operación de tiempo constante O(1), así que la ejecución total se reduce a O(n) dada la cantidad de veces que se ejecuta el login.

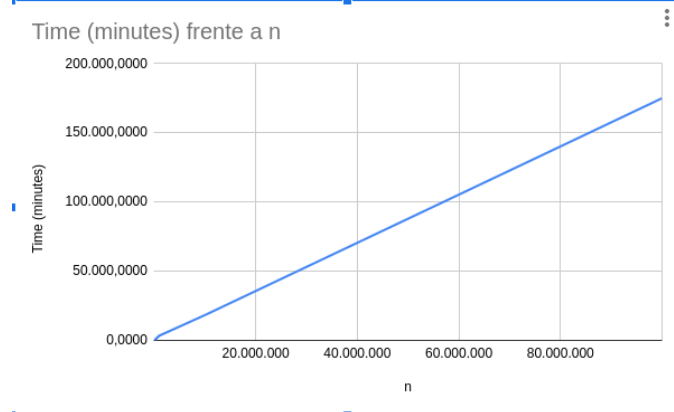
Insert item			
n	Time (nano)	Time (seconds)	Time (minutes)
10.000,00	1.992.760.870.400	1.992,7609	33,2127
100.000,00	17.993.578.496.500	17.993,5785	299,8930
1.000.000,00	230.397.996.817.500	230.397,9968	3.839,9666
10.000.000,00	2.223.497.162.440.000	2.223.497,1624	37.058,2860
100.000.000,00	19.068.123.282.200.000	19.068.123,2822	317.802,0547



En esta función apreciamos más claramente una linealidad, cabe aclarar y resaltar que es virtualmente imposible esperar para el millón de datos, al notar esto se hicieron pruebas con los datos de 100 a 100.000 donde íbamos doblando el número para así llegar hasta los 100.000 y mediante la media ponderada de cada una de las operaciones llegamos a la estimación de los datos del millón en adelante.

El tiempo de ejecución de la función insertItem tiene una dificultad de  $O(n)$ , dado que, agregar un elemento al final de una linkedList tiene una dificultad constante  $O(1)$  y al tener una cantidad de elementos de  $n$ , la dificultad aumentará a  $O(n)$  por la cantidad de elementos.

Insert User			
n	Time (nano)	Time (seconds)	Time (minutes)
10.000	1.701.978.136.717	1.701,9781	28,3663
100.000	10.377.920.065.250	10.377,9201	172,9653
1.000.000	203.709.707.512.023	203.709,7075	3.395,1618
10.000.000	1.094.736.980.510.880	1.094.736,9805	18.245,6163
100.000.000	10.499.052.465.050.800	10.499.052,4651	174.984,2078



Con este método hicimos algo similar al anterior, hicimos una media ponderada con los datos de 100 a 10.000 y le añadimos

el margen de error, ya que los tiempos son asumibles hasta los 100.000 datos. Se denota que es de tipo  $O(n)$ .

<https://vdo.ninja/?view=YM8TEe2E> El tiempo de ejecución de la función insertUser tiene una dificultad de  $O(n)$ , dado que, la función createUser tiene una dificultad constante  $O(1)$  y al tener una cantidad de elementos de  $n$ , la dificultad aumentaría a  $O(n)$  por la cantidad de elementos.

#### X. INFORMACIÓN DE ACCESO AL VIDEO DEMOSTRATIVO DEL PROTOTIPO DE SOFTWARE

El video demostrativo del primer prototipo del software se encuentra en el siguiente link:

[https://drive.google.com/file/d/1gJAntd6CdL\\_ByKjqGbE\\_MtX4wH3WQ2E/view?usp=sharing](https://drive.google.com/file/d/1gJAntd6CdL_ByKjqGbE_MtX4wH3WQ2E/view?usp=sharing)

#### XI. ROLES Y ACTIVIDADES

INTEGRANTE	ROL(ES)	ACTIVIDADES REALIZADAS (LISTADO)
JUAN LOAIZA	LÍDER, COORDINADOR	CRUD USUARIOS CONECTOR DB MERGE
JOHN PASTOR	ANIMADOR, TÉCNICO	VENTA PRODUCTOS DEVOLVER PRODUCTOS
SANTIAGO BEJARANO	TECNICO	CRUD PRODUCTOS
BRALLAN MENDOZA	TÉCNICO, QA	INTERFAZ GRÁFICA
MARIE VIRAZELS	TECNICO	PEDIR PRODUCTOS

#### XII. DIFICULTADES Y LECCIONES APRENDIDAS

Durante el desarrollo del software se aprendieron lecciones importantes al momento de implementar las funciones para que funcionara puesto que se debía implementar una estructura de datos a cada función entonces se debía estudiar qué estructura cumplía con los requisitos para que la función opere de manera óptima y eficaz. En este caso se debería realizar un bosquejo primero de cómo debería funcionar cada método y en qué estructuras de datos se deben manejar los datos que se usen.

También el grupo tuvo dificultades al momento de unir los métodos y clases que creo cada uno, aunque se usó el repositorio GitHub que da la opción de unir de manera rápida los archivos del software, debido a que cada integrante maneja

sus archivos y carpetas de una manera diferente se complica este proceso, por lo tanto para evitar este tipo de problemas se debe primero establecer unos parámetros de entrega para los archivos grupales

Finalmente se dificulto el proceso para debuggear el software y tratar los bugs o errores que presentaba el software durante su implementación, esto se debe más a que el grupo está aprendiendo a manejar este tipo de errores y a manejar los problemas que presente el proyecto.

### XIII. REFERENCIAS BIBLIOGRÁFICAS

- [1] *MySQL :: MySQL Workbench Manual :: 8.1.1 SQL Query Tab.* (s. f.).  
<https://dev.mysql.com/doc/workbench/en/wb-sql-editor-query-panel.html>
- [2] *Java Iterator.* (s. f.).  
[https://www.w3schools.com/java/java\\_iterator.asp](https://www.w3schools.com/java/java_iterator.asp)
- [3] *Java Exceptions (Try. . Catch).* (s. f.).  
[https://www.w3schools.com/java/java\\_try\\_catch.asp](https://www.w3schools.com/java/java_try_catch.asp)
- [4]