PROYECTO SEGUNDO PARCIAL “SÚPER TIENDAS”

Juan Donoso

Cristina Gatia

Mishelle Menéndez

Mateo Montenegro

Brenda Simbaña

Universidad Politécnica Salesiana

Programación Aplicada, Grupo 1

Ing. Daniel Díaz

01 de agosto de 2023

“Memoria selectiva para recordar lo bueno,
prudencia lógica para no arruinar el presente y
optimismo desafiante para encarar el futuro”
Isabel Allende

“La programación no se trata solo de escribir
código. Se trata de crear soluciones inteligentes
a problemas reales”
Linus Torvalds

1. Objetivos

El objetivo primordial de este proyecto es aplicar nuestros conocimientos sobre clases genéricas, manejo de archivos, expresiones regulares, programación concurrente, conexión a bases de datos e interfaz de comunicación, para desarrollar una aplicación de gestión de inventario destinada a una tienda. A través de este proceso, también buscamos aprender de nuestros errores y adquirir una mayor fluidez en la programación, lo que nos permitirá desarrollar nuevas habilidades de manera efectiva.

Adicionalmente, nos planteamos el desafío de llevar a cabo una estrategia de comercialización exitosa con el fin de vender esta aplicación. Para ello, estudiaremos y aplicaremos las mejores prácticas en marketing, identificaremos el mercado objetivo y crearemos una interfaz de usuario intuitiva y atractiva que satisfaga las necesidades de los potenciales clientes.

2. Descripción del proyecto

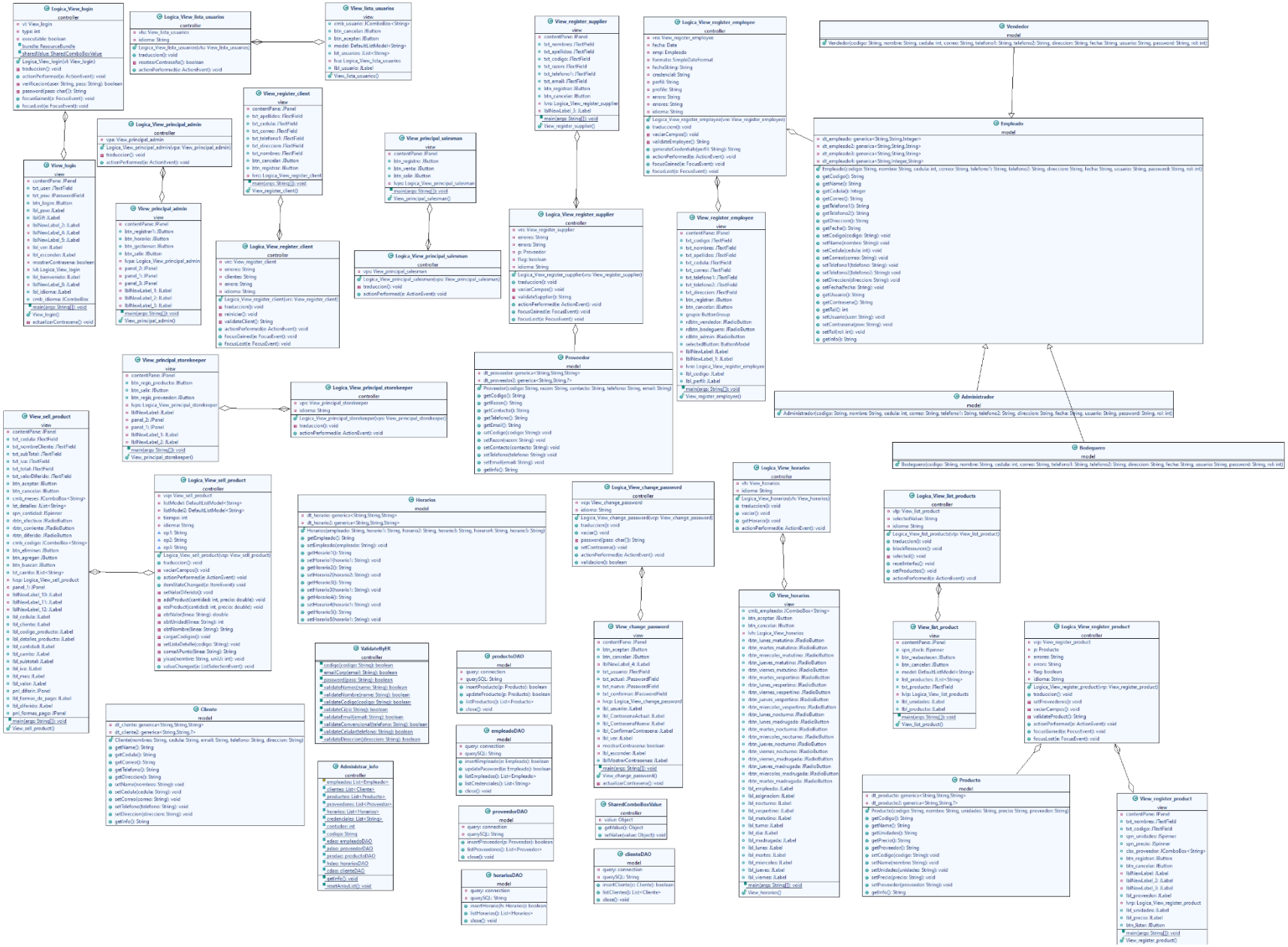
La cadena Super Tiendas busca implementar un sistema informático integral para mejorar su eficiencia y gestión. El sistema deberá encargarse de mantener un registro actualizado del inventario de productos, así como de la información de empleados, clientes y proveedores. Además, la aplicación debe contar con funciones como; realizar la facturación de ventas y obtener el peso de los productos.

El sistema informático será fundamental para asegurar un adecuado control de stock y evitar desabastecimientos o excedentes. Asimismo, permitirá optimizar los procesos de atención al cliente.

3. Arquitectura y diseño

En el desarrollo del proyecto, se adoptó la arquitectura Modelo-Vista-Controlador (MVC) para lograr una estructura organizada y modular.

3.1. Diagrama UML



3.2. Código (MVC)

MODEL

Administrador: es una subclase de Empleado que representa a empleados con el rol de administrador en el sistema. Hereda atributos y métodos de la clase Empleado.
Su constructor inicializa los datos del empleado y establece su rol como administrador.

Bodeguero: es una subclase de Empleado que representa a empleados con el rol de bodeguero en el sistema. Al heredar de la clase Empleado, esta clase adquiere todos los atributos y métodos de la clase padre. Su constructor inicializa los datos del empleado y establece su rol como bodeguero.

Cliente: representa a un cliente y almacena su información básica, como nombres, cédula, correo electrónico, teléfono y dirección. La clase proporciona métodos para acceder y actualizar cada atributo del cliente.

ClienteDAO: facilita la interacción con la base de datos "supertiendas" para realizar operaciones relacionadas con clientes. Utiliza una instancia de la clase connection para establecer la conexión con la base de datos y cerrarla cuando sea necesario.

Empleado: es una representación de un empleado en el sistema, que almacena y gestiona su información básica como código, nombre, cédula, etc. Es una estructura para manejar y administrar la información de un empleado en el sistema.

EmpleadoDAO: se encarga de gestionar el acceso a la base de datos para operaciones relacionadas con empleados, como la inserción de nuevos empleados, actualización de contraseñas y obtención de listas de empleados y nombres de usuario.

Horarios: se encarga de manejar y organizar la información de los horarios de un empleado, utilizando instancias de la clase genérica generica para almacenar los datos de forma estructurada.

HorariosDAO: se encarga de gestionar el acceso a la base de datos para operaciones relacionadas con los horarios de los empleados, como la inserción de nuevos horarios y la obtención de una lista de todos los horarios registrados.

Producto: se encarga de manejar y organizar la información básica de un producto, utilizando instancias de la clase genérica para almacenar los datos de forma estructurada.

ProductoDAO: se encarga de gestionar el acceso a la base de datos para operaciones relacionadas con productos, como la inserción de nuevos productos, actualización de unidades disponibles y obtención de una lista de todos los productos registrados.

Proveedor: se encarga de manejar y organizar la información básica de un proveedor, utilizando instancias de la clase genérica generica para almacenar los datos de forma estructurada.

ProveedorDAO: se encarga de gestionar el acceso a la base de datos para operaciones relacionadas con proveedores, como la inserción de nuevos proveedores y la obtención de una lista de todos los proveedores registrados.

Vendedor: representa a un empleado con el rol específico de vendedor y hereda los atributos y métodos de la clase Empleado, lo que le permite ser tratado como un tipo especializado de empleado dentro del sistema de la aplicación.

VIEW

View_change_password: representa la interfaz gráfica para que un usuario cambie su contraseña en la aplicación y proporciona funcionalidades para mostrar/ocultar las contraseñas ingresadas.

View_horarios: crea una interfaz gráfica para la asignación de horarios de empleados.

View_list_product: esta clase representa una interfaz gráfica que permite listar productos y realizar operaciones de reabastecimiento de stock. Los productos se muestran en una lista, y el usuario puede seleccionar uno de ellos para reabastecerlo especificando la cantidad en el Jspinner.

View_lista_usuarios: esta clase representa una interfaz gráfica para listar usuarios y permitir la interacción con ellos. Los usuarios se muestran en una lista desplegable.

View_login: representa una interfaz gráfica para un formulario de inicio de sesión. Los usuarios pueden ingresar su nombre de usuario y contraseña, y luego hacer clic en el botón "Iniciar sesión" para intentar iniciar sesión en la aplicación.

View_principal_admin: representa una interfaz gráfica para el panel de control principal del administrador. El administrador puede realizar acciones como registrar empleados, gestionar empleados, asignar horarios y cerrar sesión.

View_principal_salesman: representa una interfaz gráfica para el panel de control principal del vendedor. El vendedor puede realizar acciones como registrar clientes, vender productos y cerrar sesión.

View_principal_storekeeper: representa una interfaz gráfica para el panel de control principal del almacenista. El almacenista puede realizar acciones como registrar productos y proveedores.

View_register_client: el usuario puede ingresar información del cliente en los campos de texto proporcionados y luego elegir entre "Cancelar" o "Registrar".

View_register_employee: representa una interfaz gráfica para registrar un empleado. El usuario puede ingresar información del empleado en los campos de texto proporcionados.

View_register_product: El usuario puede ingresar información del producto en los campos de texto y seleccionar el número de unidades y el precio utilizando los spinners.

View_register_supplier: esta clase representa una interfaz gráfica para registrar un proveedor. El usuario puede ingresar información del proveedor en los campos de texto, como nombres, apellidos, código, razón social, teléfono y correo electrónico.

View_sell_product: puede seleccionar productos, agregarlos al carrito de compras, ingresar la cantidad de productos que desea vender, seleccionar la forma de pago.

CONTROLLER:

Administrador_info: es responsable de administrar y almacenar la información relacionada con los empleados, clientes, productos, proveedores y horarios en un sistema.

Logica_View_change_password: maneja la lógica relacionada con la vista de cambio de contraseña.

Logica_View_horarios: se encarga de cargar los productos en la lista, permitir el reabastecimiento de productos seleccionados, mostrar mensajes de éxito o error.

Logica_View_list_product: se encarga de cargar los productos en la lista, permitir el reabastecimiento de productos seleccionados.

Logica_View_lista_usuarios: carga los usuarios en el combo box y en la lista de usuarios, permitir el restablecimiento de contraseñas.

Logica_View_login: verifica las credenciales de inicio de sesión, permitir el cambio de idioma y mostrar mensajes de error en caso de credenciales incorrectas.

Logica_View_principal_admin: Configuración de los botones para que se abran las ventanas correspondientes.

Logica_View_principal_salesman: permite al vendedor acceder a la funcionalidad de registro de clientes y ventas, y proporciona la capacidad de cambiar de vista según las acciones del vendedor.

Logica_View_principal_storekeeper: permite al bodeguero acceder a la funcionalidad de registro de productos y proveedores, y proporciona la capacidad de cambiar de vista según las acciones del bodeguero.

Logica_View_register_client: permite al vendedor registrar nuevos clientes en el sistema y realiza validaciones en los datos ingresados.

Logica_View_register_employee: permite al administrador registrar nuevos empleados en el sistema y realiza validaciones en los datos ingresados.

Logica_View_register_product: permite al encargado de bodega registrar nuevos productos en el sistema.

Logica_View_register_supplier: permite al encargado de bodega registrar nuevos proveedores en el sistema.

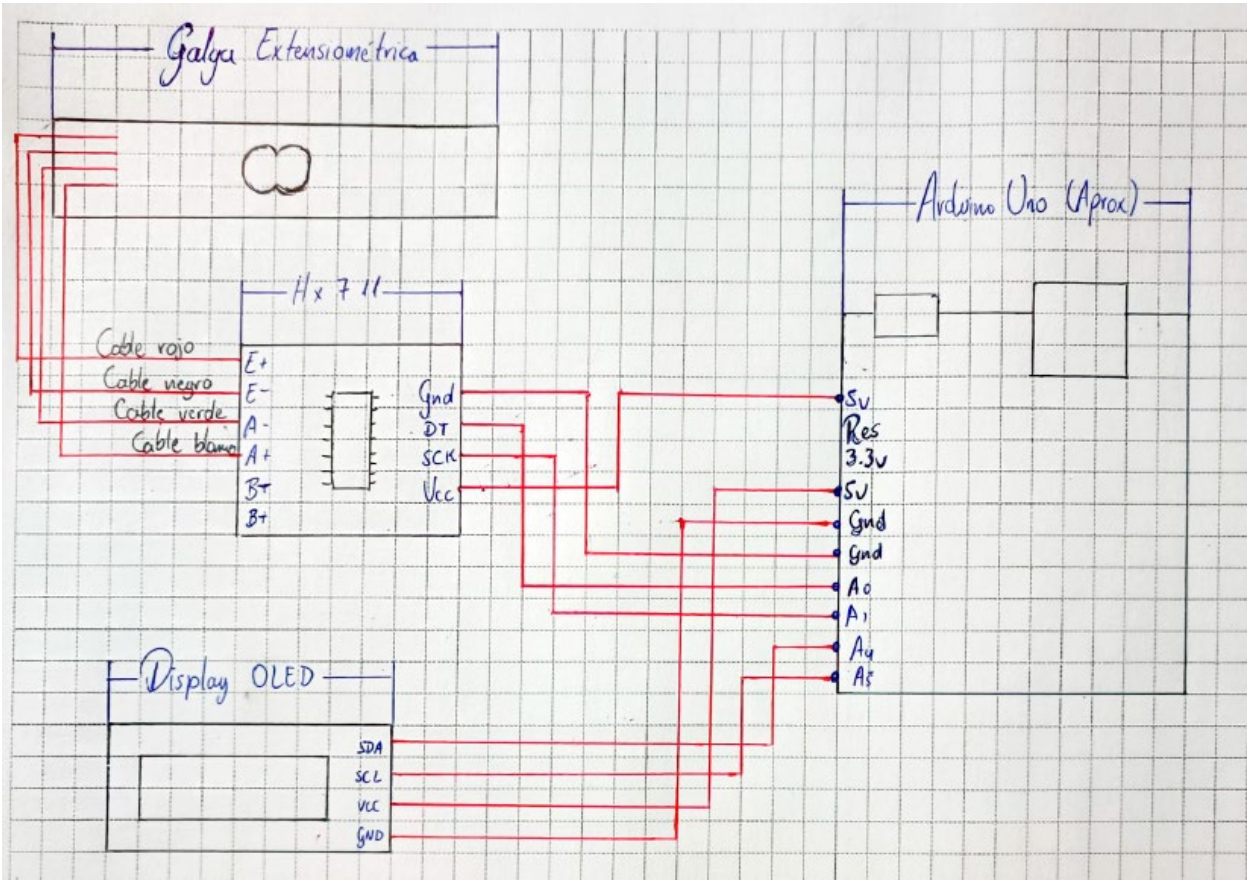
Logica_View_sell_product: permite al vendedor buscar clientes, agregar y quitar productos al carrito de compra, calcular el subtotal, el IVA y el total a pagar, y realizar la venta.

SharedComboBoxValue: actúa como un contenedor para compartir un valor entre diferentes componentes o clases en una aplicación**.**

ValidateByER: se utiliza para verificar si los datos ingresados por el usuario cumplen con ciertos patrones específicos. Proporcionan una forma eficiente y precisa de realizar estas validaciones en la aplicación.

3.3. Arduino

3.3.1 Esquemático

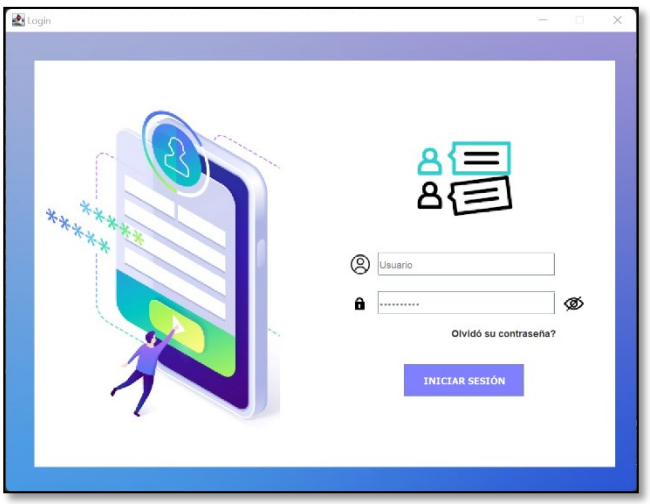


3.3.2 Código Fuente

Se ha implementado el código en Arduino IDE tanto para recibir como enviar datos a Java, así como también se ha incluido imágenes transformadas a mapas de bits como factor diferenciante ya que hemos conseguido tener una pantalla OLED que interprete este tipo de códigos, de esta manera pudimos incluir el logo de la marca.

4. Resultados

Se obtuvo como resultado una aplicación que está destinada a manejar productos, proveedores, clientes, y permite registrar ventas de productos a clientes, utiliza una interfaz gráfica de agradable para el usuario basada en para interactuar con los usuarios, en la que se ha implementado soporte para internacionalización mediante el uso de archivos de propiedades para traducir la interfaz a diferentes idiomas. Los mensajes se muestran en español e inglés según el idioma seleccionado. Se han implementado funciones de validación para asegurar que los datos ingresados por el usuario cumplan con ciertos criterios, como formato de código, email, nombres, etc. Como también permite agregar productos al carrito de compras, calcular el subtotal, impuestos y total, y aceptar el pago con distintas opciones.



5. Conclusiones

1. El código muestra una organización modular, lo que facilita el mantenimiento y la extensibilidad de la aplicación. Los diferentes controladores y clases se encargan de manejar diferentes aspectos de la aplicación, como el registro de productos, proveedores, clientes y ventas.
2. Validación de datos: Se ha implementado una serie de funciones para validar los datos ingresados por el usuario, asegurando que cumplan con ciertos criterios, esto ayuda a garantizar la integridad de los datos almacenados y procesados por la aplicación.
3. Registro de ventas: La aplicación permite realizar el registro de ventas y calcular automáticamente el subtotal, impuestos y total a pagar. También permite opciones de pago diferido en diferentes meses, lo que brinda flexibilidad al cliente.
4. Manejo de archivos: El hecho de manejar archivos en el proyecto demuestra la capacidad de interactuar con el sistema de archivos, lo cual es útil para leer y escribir datos persistentes.
5. Interfaz gráfica: La aplicación utiliza Java Swing para la interfaz gráfica, lo que proporciona una experiencia de usuario intuitiva y fácil de usar.
6. Internacionalización: La aplicación implementa el soporte para internacionalización mediante el uso de archivos de propiedades para traducir la interfaz a diferentes idiomas.
7. Arduino y Java: implementar una balanza utilizando Arduino es un proyecto e que combina el aprendizaje teórico con la experiencia práctica.

8. El proyecto proporciona una excelente oportunidad para aprender sobre los conceptos básicos de electrónica, cómo funcionan los sensores de carga, la programación del Arduino y la interacción con periféricos como pantallas LCD.

6. Recomendaciones

1. Se debe asegurar implementar correctamente la gestión de archivos para evitar errores y garantizar la integridad de los datos.
2. Implementar una adecuada validación de las entradas de datos para evitar errores y garantizar que los usuarios ingresen información válida y coherente.
3. Se recomienda asegurarse de seguir buenas prácticas de codificación, como el uso de nombres de variables y métodos descriptivos, comentarios adecuados, estructuras de control claras y organización lógica del código. Esto facilitará el mantenimiento del proyecto a largo plazo.
4. Se recomienda documentar adecuadamente el proyecto, incluyendo descripciones de las funcionalidades, la arquitectura, las decisiones de diseño y cualquier otro aspecto relevante. Esto será útil para futuros desarrolladores y para nosotros mismos en caso de que necesitemos realizar modificaciones o mejoras en el proyecto en el futuro.
5. Optimizar el código y las consultas de base de datos para garantizar una ejecución rápida y eficiente de la aplicación.
6. Antes de comenzar con el diseño y la construcción, es necesario investigar a fondo los conceptos de electrónica y Arduino necesarios para el proyecto. Planifica y esboza el diseño de tu balanza, considerando los componentes necesarios y cómo se conectarán entre sí.

8. Referencias

- Campus MVP. (2018, 29 de mayo). *Java: cómo listar, filtrar y obtener información de carpetas y archivos.* Recuperado de https://www.campusmvp.es/recursos/post/java-como-listar-filtrar-y-obtener-informacion-de-carpetas-y-archivos.aspx
- Mindware SRL. (2015, 5 de julio). *Programación genérica en Java.* Mindware SRL. Recuperado el 30 de mayo de 2023, de https://mindwaresrl.com/2015/07/05/programacion-generica-en-java/#:~:text=La%20programaci%C3%B3n%20gen%C3%A9rica%20tiene%20como,momento%20de%20desarrollar%20el%20algoritmo.
- Macaray, J.-F., & Nicolas, C. (1996). *Programacion java*. Gestion 2000.
- GoCoding. (s.f.). *Introducción a la programación genérica*. GoCoding. Recuperado el 30 de mayo de 2023, de https://gocoding.org/es/introducci%C3%B3n-a-la-programaci%C3%B3n-gen%C3%A9rica/
- Álvarez, C. (2014, January 13). *Uso de Java Generics (I). Arquitectura Java*. Retrieved 30 de mayo 2023, from https://www.arquitecturajava.com/uso-de-java-generics/
- Jansen, J. (2019). *Java File Input Output (I/O) Handling: A Tutorial.* Retrieved from https://www.baeldung.com/java-file-io
- *¿Cómo hacer una balanza digital con Arduino?* (s/f). La Electrónica. Recuperado el 31 de julio de 2023, de https://laelectronica.com.gt/extras/como_hacer_una_balanza_digital_con_arduino
- Sergio, C. (2022, marzo 7). *Balanza Electronica con HX711 y Arduino*. Control Automático Educación. https://controlautomaticoeducacion.com/arduino/balanza-electronica-hx711-arduino/