

2022

Integrantes:

- Erick Alexander Gamboa Trujillo

- Anderson Adrian Pisfil Maza

- Oscar Alexander Caycho Zapata

- Cristofer Eduardo Chumpitaz Pacheco

- De Paz Cotrina David Moises

- Christian Junior Blas Vargas

CURSO: TÉCNICAS DE PROGRAM.ORIE. OBJ.

DOCENTE: ESTHER TARMEÑO JUSCAMAITA

**ESTRUCTURA DEL PROYECTO FINAL**

**INTRODUCCIÓN**

En la actualidad, el sistema de transporte es uno de los servicios públicos más usados y a su vez saturado debido a la variedad de compañías con todo tipo de características como precios, ruta de destino, líneas de buses, variedad de terminales, etc. Por ello, el presente trabajo tiene como propósito implementar una aplicación para mejorar el registro y venta de boletos para tener una mejor organización para la empresa Turismo Cavassa.

**PRESENTACIÓN**

En el presente apartado se puntualiza la organización del proyecto: En la primera sección del informe se muestra el título del trabajo, el alcance que se desea obtener de las ventas y registros de la problemática dada, marco teórico, así mismo, los objetivos generales y específicos de la empresa. En la segunda sección se visualizará el desarrollo ágil del proyecto a la problemática que tiene la empresa Turismo Cavassa, debido a ello se utilizará una metodología de desarrollo de software del proyecto, requerimientos del proyecto / historias de Usuario, Diagrama de clases del proyecto, se creará una aplicación en java que nos da un resumen de colecciones en Java y forma de aplicación, de igual forma, resumen sobre archivos en Java y forma de aplicación, manual de clases, diseño de la interfaz y la prueba del software. Por último, se darán las conclusiones y recomendaciones para el buen uso de la aplicación.

**CAPÍTULO I:**

1. **PROYECTO DE INVESTIGACIÓN**
   1. Título del Proyecto

Sistema de registro y venta de boletos en la empresa de transportes Cavassa.

* 1. Alcance

Nuestro programa se enfocará en examinar y organizar el área de registro y ventas donde se permitirá a la empresa tener un mejor manejo sobre los datos cuando se registren de sus ventas.

* 1. Marco Teórico
  2. Sistema de venta

Un sistema de venta es un proceso de gestión que facilita la administración del negocio, mediante un servidor con base de datos Relacional, automatizando el servicio de atención al cliente en forma equitativa y, por ende, este puede controlar: cotizaciones, control de caja, stock, compras a proveedores y movimientos (Castillo, 2016).

* 1. Área de venta (como funciona un área de venta en una empresa de transporte)

Este sector es muy importante en la estructura de una empresa ya que tiene la función de conseguir los objetivos de venta a través de los clientes. Algunos de los objetivos del área de ventas son:

* Tener metas claras y realistas
* Planear estrategias viables
* Controlar en el aumento o pérdida de clientes
* Mejorar la atención del cliente
* Hacer promociones de la empresa
  1. Java

Java es un lenguaje compilado e interpretado. Todo programa o proyecto en Java ha de compilarse y el código que se genera es interpretado por una máquina virtual. De esta manera se consigue la independencia de la máquina, el código compilado se ejecuta en dichas máquinas virtuales (Belmonte, 2004).

Java es un lenguaje de programación creado por James Gosling en 1995, este lenguaje está orientado a objetos, es decir, un paradigma de la programación donde el uso de las clases genera objetos que interactúan entre sí. Java ganó mucha popularidad debido a su flexibilidad para crear cualquier aplicación como páginas web, formularios, etc., además es posible enlazarlo con base de datos.

* 1. Scrum

Trigas (2012) manifiesta que Scrum es una metodología de desarrollo ágil compuesta por un equipo multifuncional, que tiene como base hacer reuniones, adaptarse a los cambios y crear ciclos de mínimo dos semanas llamadas “*Sprints*” con el objetivo de optimizar un proyecto.

* 1. Términos en Scrum
     1. Product Owner

Es la persona encargada de tener contacto directo con el cliente y los stakeholders para decidir si el producto se acepta o se rechaza. Así mismo, maximiza el valor del producto realizado por el equipo de desarrollo en corto tiempo.

* + 1. Scrum Master

Es el responsable de guiar al equipo de desarrollo y hacerles entender las reglas del scrum, mediante reuniones diarias con el fin de saber que se hizo ayer, hoy y que se hará mañana.

* + 1. Development Team

También llamado equipo de desarrollo, tienen la capacidad de habituarse a los cambios, son muy colaborativos y altamente sincronizados. Estos profesionales son los responsables de desarrollar el producto para el cliente bajo un tiempo limitado por el Scrum Master.

* + 1. Backlog

Es una lista de funcionalidades extensa, el Product Owner tiene la tarea de actualizar el Backlog, de tal manera que las tareas estén con un nivel de prioridad y así, entregar el producto en el menor tiempo posible.

* + 1. Historia de usuario

El Backlog se divide en historias de usuario, estas son funcionalidades precisas que poseen unas ligeras tareas.

* + 1. Sprints

El sprint es la lista de tareas priorizadas para ser desarrolladas por el equipo de trabajo en un lapso de tiempo determinado por el Scrum Master. El tiempo varía entre dos semanas a un mes.

* 1. Objetivos
     1. Objetivo General

Nos hemos propuesto, mejorar la calidad de atención al cliente, además de mejorar el tratamiento de los datos que son necesarios para la venta de pasajes, siendo registrados en listas.

* + 1. Objetivo Específico

Como objetivos específicos tenemos los siguientes:

1. Mejorar la toma de datos del cliente a la empresa.

2. Almacenar los datos en digital.

3. Generar rutas para los buses.

4. Registrar clientes independientes.

5. Facilitar el almacenaje de boletos de pasajes, formato digital para la empresa y físico para el cliente.

**CAPÍTULO II**

1. **DESARROLLO ÁGIL DEL PROYECTO**
   1. Metodología de desarrollo de software del proyecto.

Para el desarrollo del proyecto se utilizará el scrum que es unas metodologías ágiles más usadas para los proyectos de software.

Es un marco de gestión para el desarrollo incremental de productos, valiéndose de uno o más equipos multifuncionales, auto organizados, de aproximadamente siete personas cada uno.

Proporciona una estructura de roles, reuniones, reglas y artefactos. Los equipos son los responsables de crear y adaptar sus procesos dentro de este marco.

Se compone de 3 roles principales:

1. Product owner

Conformado por una persona dentro del equipo de trabajo es el encargado de expresar la necesidad del cliente dentro del product Backlog

1. Scrum máster

Es el líder del equipo y encargado de asegurar que el equipo trabaje ajustándose a las reglas del scrum solicitado.

1. Development Team

Es el equipo de desarrollo se compone de desarrolladores que son las personas responsables de dar cumplimiento a los sprint.

Los Scrum máster deben idear la manera de que estos 3 roles se lleven a cabo, esto se logra a través de las reuniones.

Entre estas reuniones tenemos:

1. Planificación de Sprint

El sprint que es el núcleo central donde se trabajarán los proyectos a tomar. Suelen durar de 2 a 4 semanas.

Es esta reunión en donde el Scrum Máster, Product Owner y el equipo de trabajo se dividen el proyecto en etapas y tareas.

Esta reunión suele durar entre 4 a 8 horas.

1. Daily Scrum

Esta reunión tiene lugar cada día, a la misma hora y en el mismo lugar, los miembros del equipo de desarrollo pasan 15 minutos reportándose entre sí. Cada miembro del equipo resume lo que hizo el día anterior, lo que hará hoy, y qué impedimentos está enfrentando.

1. Ejecución del Sprint

En esta reunión el equipo de trabajo enseña el resultado final del proyecto donde se llevará a cabo la inspección y el análisis para su adaptación a la pila.

1. Retrospectiva del Sprint

Cada Sprint finaliza con una retrospectiva. En esta reunión, el equipo de trabajo y Scrum Máster analizan y reflexionan sobre su propio proceso. Inspeccionan su comportamiento y adoptan las medidas para adaptarlo en los futuros Sprint.

Se estima que puede durar de 2 a 4 horas.

Por último, tenemos a los artefactos que son los elementos que garantizan la transparencia y el registro de información del proceso de scrum.

Entre ellos tenemos:

1. Producto backlog

Lista ordenada que contiene los requerimientos y funcionalidades que serán utilizadas sobre el producto en las futuras entregas.

Los elementos de la pila estarán ordenados de mayor a menor según el rango de importancia y puede estar sometida a cambios si se lo requiere.

1. Sprint backlog

Es aquella lista de las tareas necesarias para la construcción de las historias de usuarios que se tomarán en cada uno de los sprint.

1. Incremento

Es el resultado del sprint, ósea es una versión mejorada del producto.

Una versión con posibilidad de liberación o no en función del product owner.

En conclusión, la metodología **Scrum** es una metodología ágil que hace énfasis en el trabajo en equipo donde la claridad de los objetivos es crucial para avanzar hacia una versión cada vez mejor.

**NIVEL DE PRIORIDAD**

|  |  |  |
| --- | --- | --- |
| **PRIORIDAD** | **NIVEL** | **DESCRIPCIÓN** |
| **1** | Muy alto | El requerimiento es un fundamental para el funcionamiento de la aplicación |
|  |
| **2** | Alto | El requerimiento es secundario, es importante para la aplicación |  |
|  |
| **3** | Medio | El requerimiento es complementario, sin este el formulario sigue funcionando sin este |  |
|  |
| **4** | Bajo | El requerimiento no es tan necesario, ayudan a la comprensión de la aplicación |  |
|  |
| **5** | Muy bajo | El requerimiento no es relevante, aun así, ayuda a una mayor comprensión de la aplicación. |  |
|  |

**PRODUCT BACKLOG**

|  |  |  |
| --- | --- | --- |
| **Requerimientos funcionales** | **Nro. Historia** | **Prioridad** |
| R1: El sistema requiere ingresar nombres, apellidos, dirección, correo, fecha de nacimiento del nuevo cliente, fecha de viaje, destino, todo esto dentro de un formulario. | HU1 | 1 |
| R2: El sistema necesita un menú de inicio donde se podrá acceder al registro de ventas. | HU2 | 1 |
| R3: El sistema necesita unos precios definidos según las condiciones establecidas por el gerente financiero. | HU3 | 1 |
| R4: El sistema requiere que se incluya un formulario bus con las siguientes características: placa, marca, modelo, origen, destino, n# de asientos. | HU4 | 1 |
| R5: El sistema debe validar los datos ingresados del nuevo cliente de acuerdo con las condiciones especificadas. | HU5 | 3 |
| R6: El sistema necesita una mejora en el formulario bus y validación para su buen funcionamiento. | HU6 | 3 |
| R7: El sistema requiere un formulario de registro de viajes. | HU7 | 1 |
| R8: La aplicación necesita un formulario de compras de pasaje y requerirá las siguientes características: fecha, origen, destino, viaje de bus, cliente y numero de asiento. | HU8 | 1 |
| R9: El sistema necesita una mejora en el formulario “Comprar pasaje”. | HU9 | 3 |
| R10: El sistema necesita botones especiales que manipulen los datos del registro de viajes y compra de pasajes. | HU10 | 4 |
| R11: El sistema necesita una mejora en el formulario “Registrar viaje”. | HU11 | 3 |
| R12: El sistema requiere de un inicio de sesión de trabajadores para posteriormente, mostrar el menú de inicio. | HU12 | 2 |
| R13: El sistema requiere de un registro de ventas, donde se podrá visualizar todas las ventas realizadas. | HU13 | 2 |
| R14: El sistema necesita un registro de compras donde se podrá visualizar tosas las compras realizadas (gasolina, buses). | HU14 | 2 |
| R15: El sistema necesita botones especiales que manipulen los datos de los reportes de la aplicación. | HU15 | 4 |
| R16: La aplicación requiere de un registro de empleado, cada vez que es contratado un nuevo trabajador, la lista crecerá. | HU16 | 2 |
| R17: El sistema necesita validación en el formulario de trabajadores. | HU17 | 3 |
| R18: La aplicación necesita una generación de pasajes virtual. | HU18 | 5 |
| R19: El sistema necesita validación en el registro de compras. | HU19 | 3 |
| R20: El sistema requiere de un listado de programación de buses. | HU20 | 5 |

**PRODUCT BACKLOG ORGANIZADO**

|  |  |  |
| --- | --- | --- |
| **Requerimientos funcionales** | **Nro. Historia** | **Prioridad** |
| R1: El sistema requiere ingresar nombres, apellidos, dirección, correo, fecha de nacimiento del nuevo cliente, fecha de viaje, destino, todo esto dentro de un formulario. | HU1 | 1 |
| R2: El sistema necesita un menú de inicio donde se podrá acceder a los formularios. | HU2 | 1 |
| R4: El sistema requiere que se incluya un formulario bus con las siguientes características: placa, marca, modelo, origen, destino, n# de asientos. | HU4 | 1 |
| R7: El sistema requiere un formulario de registro de viajes. | HU7 | 1 |
| R8: La aplicación necesita un formulario de compras de pasaje y requerirá las siguientes características: fecha, origen, destino, viaje de bus, cliente y numero de asiento. | HU8 | 1 |
| R3: El sistema necesita unos precios definidos según las condiciones establecidas por el gerente financiero. | HU3 | 1 |
| R12: El sistema requiere de un inicio de sesión de trabajadores para posteriormente, mostrar el menú de inicio. | HU12 | 2 |
| R16: La aplicación requiere de un registro de empleado, cada vez que es contratado un nuevo trabajador, la lista crecerá. | HU16 | 2 |
| R13: El sistema requiere de un registro de ventas, donde se podrá visualizar todas las ventas realizadas. | HU13 | 2 |
| R14: El sistema necesita un registro de compras donde se podrá visualizar tosas las compras realizadas (gasolina, buses). | HU14 | 2 |
| R5: El sistema debe validar los datos ingresados del nuevo cliente de acuerdo con las condiciones especificadas. | HU5 | 3 |
| R6: El sistema necesita una mejora en el formulario bus y validación para su buen funcionamiento. | HU6 | 3 |
| R9: El sistema necesita una mejora en el formulario “Comprar pasaje”. | HU9 | 3 |
| R11: El sistema necesita una mejora en el formulario “Registrar viaje”. | HU11 | 3 |
| R17: El sistema necesita validación en el formulario de trabajadores. | HU17 | 3 |
| R19: El sistema necesita validación en el registro de compras. | HU19 | 3 |
| R10: El sistema necesita botones especiales que manipulen los datos del registro de viajes y compra de pasajes. | HU10 | 4 |
| R15: El sistema necesita botones especiales que manipulen los datos de los reportes de la aplicación. | HU15 | 4 |
| R18: La aplicación necesita una generación de pasajes virtual. | HU18 | 5 |
| R20: El sistema requiere de un listado de programación de buses. | HU20 | 5 |

**PRODUT BACKLOG PRIORIZADO**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Requerimientos funcionales** | **Nro. Historia** | **Prioridad** |
| **SPRINT 1** | R1: El sistema requiere ingresar nombres, apellidos, dirección, correo, fecha de nacimiento del nuevo cliente, fecha de viaje, destino, todo esto dentro de un formulario. | HU1 | 1 |
| R2: El sistema necesita un menú de inicio donde se podrá acceder al registro de ventas. | HU2 | 1 |
| R4: El sistema requiere que se incluya un formulario bus con las siguientes características: placa, marca, modelo, origen, destino, n# de asientos. | HU4 | 1 |
| R7: El sistema requiere un formulario de registro de viajes. | HU7 | 1 |
| R8: La aplicación necesita un formulario de compras de pasaje y requerirá las siguientes características: fecha, origen, destino, viaje de bus, cliente y numero de asiento. | HU8 | 1 |
| **SPRINT 2** | R3: El sistema necesita unos precios definidos según las condiciones establecidas por el gerente financiero. | HU3 | 1 |
| R12: El sistema requiere de un inicio de sesión de trabajadores para posteriormente, mostrar el menú de inicio. | HU12 | 2 |
| R16: La aplicación requiere de un registro de empleado, cada vez que es contratado un nuevo trabajador, la lista crecerá. | HU16 | 2 |
| R13: El sistema requiere de un registro de ventas, donde se podrá visualizar todas las ventas realizadas. | HU13 | 2 |
| R14: El sistema necesita un registro de compras donde se podrá visualizar tosas las compras realizadas (gasolina, buses). | HU14 | 2 |
| **SPRINT 3** | R5: El sistema debe validar los datos ingresados del nuevo cliente de acuerdo con las condiciones especificadas. | HU5 | 3 |
| R6: El sistema necesita una mejora en el formulario bus y validación para su buen funcionamiento. | HU6 | 3 |
| R9: El sistema necesita una mejora en el formulario “Comprar pasaje”. | HU9 | 3 |
| R11: El sistema necesita una mejora en el formulario “Registrar viaje”. | HU11 | 3 |
| R17: El sistema necesita validación en el formulario de trabajadores. | HU17 | 3 |
| **SPRINT 4** | R19: El sistema necesita validación en el registro de compras. | HU19 | 3 |
| R10: El sistema necesita botones especiales que manipulen los datos del registro de viajes y compra de pasajes. | HU10 | 4 |
| R15: El sistema necesita botones especiales que manipulen los datos de los reportes de la aplicación. | HU15 | 4 |
| R18: La aplicación necesita una generación de pasajes virtual. | HU18 | 5 |
| R20: El sistema requiere de un listado de programación de buses. | HU20 | 5 |

|  |  |  |
| --- | --- | --- |
| **Requerimientos funcionales** | **Nro. Historia** | **Prioridad** |
| R5: El sistema necesita una clase de clientes y creación de archivos para almacenarlo en la carpeta del proyecto | HU5 | 1 |
| R6: El sistema necesita una clase Bus y creación de archivos con serializadores para almacenarlo en la carpeta del proyecto | HU6 | 1 |
| R9: El sistema necesita una clase Registro de viajes y la creación de archivos para almacenar los datos de la lista. | HU9 | 1 |
| R10: El sistema necesita una clase Comprar pasaje y la creación de archivos para almacenar los datos de la lista. | HU10 | 1 |
| R19: El sistema requiere de una clase de Registro de anulados de pasaje y la creación de archivos para almacenar los datos del registro. | HU19 | 1 |
| R20: El sistema necesita un formulario para actualizar la disponibilidad de los asientos del bus, de esa manera nuevos clientes podrán reservar asientos. | HU20 | 1 |
| R1: El sistema requiere ingresar datos del nuevo cliente, usando una lista temporal. | HU1 | 2 |
| R3: El sistema requiere de un registro de clientes donde se mostrará en un formulario todos los datos de este, que se hayan registrado. | HU3 | 2 |
| R4: El sistema requiere que se incluya un formulario bus. | HU4 | 2 |
| R7: El sistema requiere un formulario de registro de viajes donde se reservará el recorrido. | HU7 | 2 |
| R8: La aplicación necesita un formulario de compras de pasaje. | HU8 | 2 |
| R18: La aplicación necesita una generación de pasajes virtual. | HU18 | 2 |
| R2: El sistema necesita un menú de inicio donde se podrá acceder a los formularios. | HU2 | 3 |
| R12: El sistema requiere de un inicio de sesión de trabajadores para posteriormente, mostrar el menú de inicio. | HU12 | 3 |
| R13: El sistema requiere de una clase de Registro de ventas y la creación de archivos para almacenar los datos de la lista. | HU13 | 3 |
| R15: El sistema requiere de un registro de todos los pasajes, ya sea cancelados y anulados. | HU15 | 3 |
| R11: El sistema requiere de un formulario de registro de ventas, donde se podrá visualizar todas las ventas realizadas. | HU11 | 4 |
| R17: El sistema requiere de una clase empleados y la creación de archivos para almacenar los datos del registro. | HU17 | 5 |
| R16: La aplicación requiere de un registro de empleados. | HU16 | 5 |
| R14:  El sistema necesita una mejora en el diseño de los formularios. | HU14 | 5 |

|  |  |  |
| --- | --- | --- |
| **PRIORIDAD** | **NIVEL** | **DESCRIPCIÓN** |
| **1** | Muy alto | El requerimiento es un fundamental para el funcionamiento de la aplicación |
|
| **2** | Alto | El requerimiento es secundario, es importante para la aplicación |
|
| **3** | Medio | El requerimiento es complementario, sin este el formulario sigue funcionando sin este |
|
| **4** | Bajo | El requerimiento no es tan necesario, ayudan a la comprensión de la aplicación |
|
| **5** | Muy bajo | El requerimiento no es relevante, aun así, ayuda a una mayor comprensión de la aplicación. |
|

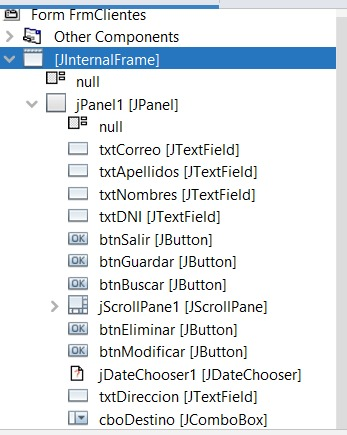
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **HISTORIA DE USUARIO NRO. 1** | | | | | | |
| **DESCRIPCIÓN** | | | | | | |
| Como gerente de operaciones quiero un formulario de ingreso de datos del cliente para tener un mayor control en la obtención de datos. | | | | | | |
|
| **TAREAS** | | | | | | **PRIORIDAD** |
| En el formulario deben existir los campos DNI, nombres, apellidos, dirección, correo electrónico y fecha de nacimiento. Se podrá buscar, eliminar y modificar la lista de clientes. | | | | | | **2** |
|
| **T. ESTIMADO** |
| **4 días** |
|
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| **HISTORIA DE USUARIO NRO. 2** | | | | | | |
| **DESCRIPCIÓN** | | | | | | |
| Como gerente de operaciones quiero un menú de inicio para acceder a los diferentes formularios. | | | | | | |
|
| **TAREAS** | | | | | | **PRIORIDAD** |
| En el formulario deben existir los botones cliente, viaje, pasaje, bus, empleado, registro de ventas, registro de pasajes.  Cada botón se subdivide. | | | | | | **3** |
|
| **T. ESTIMADO** |
| **1 días** |
|
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| **HISTORIA DE USUARIO NRO. 3** | | | | | | |
| **DESCRIPCIÓN** | | | | | | |
| Como gerente de operaciones quiero un formulario de registro de clientes, para almacenar todos los clientes del formulario de clientes. | | | | | | |
|
| **TAREAS** | | | | | | **PRIORIDAD** |
| En el formulario deben existir los campos DNI, nombres, apellidos, dirección, correo electrónico y fecha de nacimiento. En este formulario no se podrá guardar. | | | | | | **2** |
|
| **T. ESTIMADO** |
| **2 días** |
|
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| **HISTORIA DE USUARIO NRO. 4** | | | | | | |
| **DESCRIPCIÓN** | | | | | | |
| Como jefe de transporte quiero un registro de buses para tener una lista de buses para mi empresa. | | | | | | |
|
| **TAREAS** | | | | | | **PRIORIDAD** |
| En el formulario deben existir los botones ingreso, búsqueda, modificación y eliminar . El formulario tendrá campos: placa, marca, modelo, n# de asientos. | | | | | | **2** |
|
| **T. ESTIMADO** |
| **3 días** |
|
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| **HISTORIA DE USUARIO NRO. 5** | | | | | | |
| **DESCRIPCIÓN** | | | | | | |
| Como gerente de operaciones quiero almacenar la información obtenida de los clientes para que ellos puedan realizar sus reservaciones. | | | | | | |
|
| **TAREAS** | | | | | | **PRIORIDAD** |
| La clase Cliente debe heredar de una clase Persona. Se usará archivos de texto con serializadores para su almacenamiento. | | | | | | **1** |
|
| **T. ESTIMADO** |
| **3 días** |
|
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| **HISTORIA DE USUARIO NRO. 6** | | | | | | |
| **DESCRIPCIÓN** | | | | | | |
| Como gerente de operaciones quiero almacenar la información obtenida de los buses para que los clientes puedan reservar el asiento. | | | | | | |
|
| **TAREAS** | | | | | | **PRIORIDAD** |
| La clase Bus debe heredar de una clase Persona. Se usará archivos de texto con serializadores para su almacenamiento. | | | | | | **1** |
|
| **T. ESTIMADO** |
| **3 días** |
|
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| **HISTORIA DE USUARIO NRO. 7** | | | | | | |
| **DESCRIPCIÓN** | | | | | | |
| Como gestor de destino quiero un registro de viajes para que el cliente pueda reservar su viaje. | | | | | | |
|
| **TAREAS** | | | | | | **PRIORIDAD** |
| En el formulario deben existir los botones ingreso, depuración y salida. El formulario tendrá campos: bus, origen, destino con un buscador automático. Fecha y hora tendrán un Jcalendar. | | | | | | **1** |
|
| **T. ESTIMADO** |
| **4 días** |
|
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| **HISTORIA DE USUARIO NRO. 8** | | | | | | |
| **DESCRIPCIÓN** | | | | | | |
| Como gestor de destino quiero un formulario "Comprar pasaje" para que el cliente obtenga la confirmación del viaje reservado en "Registrar viaje". | | | | | | |
|
| **TAREAS** | | | | | | **PRIORIDAD** |
| En el formulario deben existir los botones ingreso, depuración y salida. La compra de pasaje se realizará de acuerdo a la reserva del cliente en el Registro de viajes. El formulario tendrá campos: fecha, origen, destino, viaje de bus, cliente y numero de asiento. | | | | | | **1** |
|
| **T. ESTIMADO** |
| **3 días** |
|
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| **HISTORIA DE USUARIO NRO. 9** | | | | | | |
| **DESCRIPCIÓN** | | | | | | |
| Como gestor de destino quiero una mejora en el formulario "Comprar pasaje" para evitar errores en el formulario y agregar validaciones. | | | | | | |
|
| **TAREAS** | | | | | | **PRIORIDAD** |
| Se debe validar si los campos están vacíos. Mejorar el diseño del formulario. | | | | | | **3** |
|
| **T. ESTIMADO** |
| **3 días** |
|
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| **HISTORIA DE USUARIO NRO. 10** | | | | | | |
| **DESCRIPCIÓN** | | | | | | |
| Como gerente de destino quiero botones especiales modificar y eliminar en la sección de "Registrar viaje" y "Comprar pasaje" para dar un completo funcionamiento de los formularios | | | | | | |
|
| **TAREAS** | | | | | | **PRIORIDAD** |
| El botón modificar de los dos formularios debe estar con validación y hacer modificaciones mediante el DNI de cliente y Numero de Boleto, respectivamente. El botón eliminar de los dos formularios debe estar con validación y eliminar mediante el DNI de cliente y Numero de Boleto, respectivamente. | | | | | | **4** |
|
| **T. ESTIMADO** |
| **3 días** |
|
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| **HISTORIA DE USUARIO NRO. 11** | | | | | | |
| **DESCRIPCIÓN** | | | | | | |
| Como gestor de destino quiero una mejora en el formulario Registro de viajes para evitar errores en la aplicación y perfeccionar el aspecto del formulario. | | | | | | |
|
| **TAREAS** | | | | | | **PRIORIDAD** |
| En la sección "Fecha de viaje" y "Destino" debe contar con una búsqueda con combobox y dependerán de la disponibilidad de los buses. Se debe validar si los campos están vacíos. Mejorar el diseño del formulario. | | | | | | **3** |
|
| **T. ESTIMADO** |
| **4 días** |
|
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| **HISTORIA DE USUARIO NRO. 12** | | | | | | |
| **DESCRIPCIÓN** | | | | | | |
| Como gerente de operaciones quiero que el sistema posea un inicio de sesión para que los empleados puedan registrar datos y realizar ventas. | | | | | | |
|
| **TAREAS** | | | | | | **PRIORIDAD** |
| Se requiere de un formulario de inicio de sesión que pida usuario y contraseña. De acuerdo al cargo, la aplicación mostrara funciones específicas al empleado. | | | | | | **2** |
|
| **T. ESTIMADO** |
| **4 días** |
|
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| **HISTORIA DE USUARIO NRO. 13** | | | | | | |
| **DESCRIPCIÓN** | | | | | | |
| Como gerente financiero quiero un formulario de registro de ventas para visualizar y analizar todas las ventas realizadas a los clientes. | | | | | | |
|
| **TAREAS** | | | | | | **PRIORIDAD** |
| El formulario mostrara los datos de las compras de pasaje realizadas por el cliente. El formulario mostrara en una etiqueta el total de ventas especificadas en un año, mes, semana, día. | | | | | | **2** |
|
| **T. ESTIMADO** |
| **4 días** |
|
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| **HISTORIA DE USUARIO NRO. 14** | | | | | | |
| **DESCRIPCIÓN** | | | | | | |
| Como gerente financiero quiero un formulario de registro de compras para visualizar y analizar todas las compras realizadas por la empresa. | | | | | | |
|
| **TAREAS** | | | | | | **PRIORIDAD** |
| El formulario tendrá los siguientes campos: compra (combobox), precio, cantidad, marca. El formulario mostrara los datos de las compras de pasaje realizadas por el cliente. El formulario mostrará en una etiqueta el total de ventas especificadas en un año, mes, semana, día. | | | | | | **2** |
|
| **T. ESTIMADO** |
| **4 días** |
|
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| **HISTORIA DE USUARIO NRO. 15** | | | | | | |
| **DESCRIPCIÓN** | | | | | | |
| Como gerente de destino quiero botones especiales modificar y eliminar en la sección de "Reporte" para dar un completo funcionamiento de los formularios | | | | | | |
|
| **TAREAS** | | | | | | **PRIORIDAD** |
| Se mostrará los datos de los datos del registro de venta y de compra. El botón modificar y eliminar es mediante el número de boleto. | | | | | | **4** |
|
| **T. ESTIMADO** |
| **4 días** |
|
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| **HISTORIA DE USUARIO NRO. 16** | | | | | | |
| **DESCRIPCIÓN** | | | | | | |
| Como gerente de recursos humanos quiero un formulario de ingreso de datos del empleado y el destino para facilitar una compra a futuro. | | | | | | |
|
| **TAREAS** | | | | | | **PRIORIDAD** |
| En el formulario deben existir los campos DNI, cargo, nombres, apellidos, dirección, correo electrónico y fecha de nacimiento. Se podrá buscar, eliminar y modificar la lista de clientes. | | | | | | **2** |
|
| **T. ESTIMADO** |
| **3 días** |
|
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| **HISTORIA DE USUARIO NRO. 17** | | | | | | |
| **DESCRIPCIÓN** | | | | | | |
| Como gerente de operaciones quiero la validación del ingreso de datos del empleado y el destino para evitar errores en el sistema. | | | | | | |
|
| **TAREAS** | | | | | | **PRIORIDAD** |
| El área de DNI tiene que ser numérico y no debe repetirse. La edad debe ser mayor e igual a 18 años. Se debe validar si los campos están vacíos. Mejorar el diseño del formulario. | | | | | | **3** |
|
| **T. ESTIMADO** |
| **3 días** |
|
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| **HISTORIA DE USUARIO NRO. 18** | | | | | | |
| **DESCRIPCIÓN** | | | | | | |
| Como gerente de operaciones quiero la generación del pasaje para que el cliente visualice su compra realizada con éxito. | | | | | | |
|
| **TAREAS** | | | | | | **PRIORIDAD** |
| Se mostrarán los datos del cliente, del viaje, origen departamento, origen provincia, origen distrito, origen dirección, destino departamento, destino provincia, destino distrito, destino dirección, fecha y hora de origen y salida, numero de asiento, total a pagar y el tipo de moneda en una boleta electrónica. | | | | | | **5** |
|
| **T. ESTIMADO** |
| **3 días** |
|
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| **HISTORIA DE USUARIO NRO. 19** | | | | | | |
| **DESCRIPCIÓN** | | | | | | |
| Como gerente financiero quiero la validación del ingreso de datos del registro de compras para evitar el malfuncionamiento del sistema. | | | | | | |
|
| **TAREAS** | | | | | | **PRIORIDAD** |
| Validar el numero de boleto de compra que sea numérico. Se debe validar si los campos están vacíos. Mejorar el diseño del formulario. | | | | | | **3** |
|
| **T. ESTIMADO** |
| **3 días** |
|
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| **HISTORIA DE USUARIO NRO. 20** | | | | | | |
| **DESCRIPCIÓN** | | | | | | |
| Como gerente de flota quiero un listado de programación de buses para ver los buses que están en ruta y ver el historial de los buses. | | | | | | |
|
| **TAREAS** | | | | | | **PRIORIDAD** |
| Se mostrarán los datos del bus y sus movimientos. Con un combobox seleccionar el bus. | | | | | | **3** |
|
| **T. ESTIMADO** |
| **3 días** |
|

**DOCUMENTACIÓN DE SPRINT 1**

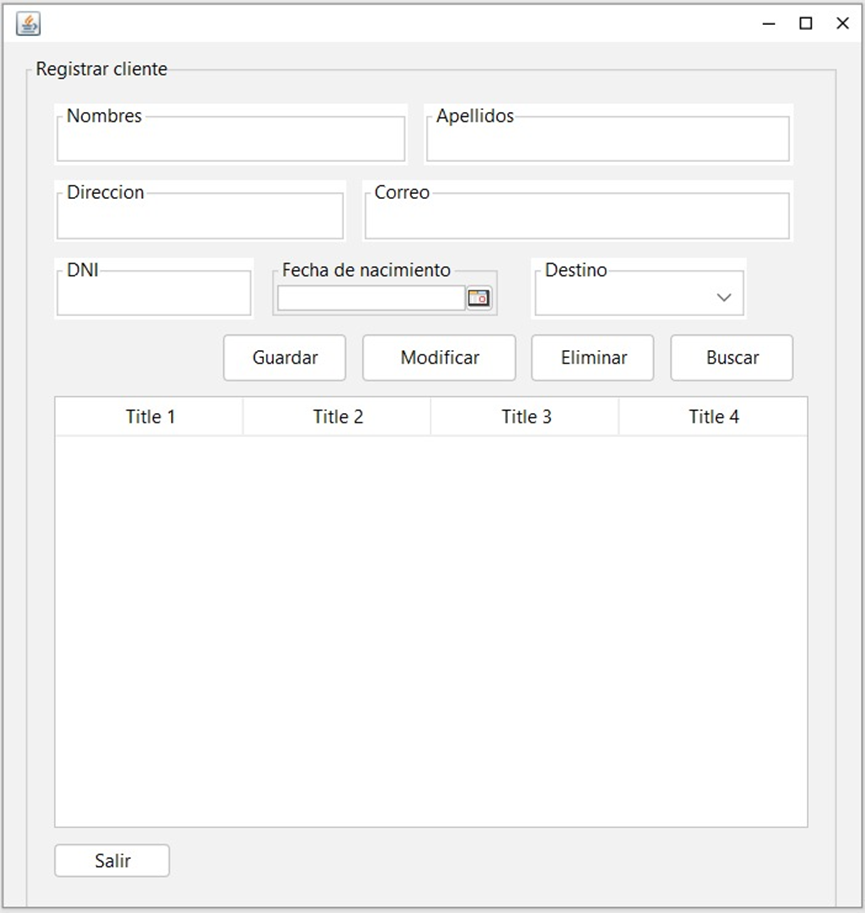
En esta actividad realizamos los requisitos que nos ordenan las Historias de los usuarios que veremos a continuación.

**R1: El sistema requiere un formulario que se pueda ingresar los nombres, apellidos, dirección, correo, fecha de nacimiento del nuevo cliente, fecha de viaje, destino, todo esto dentro de un formulario.**

COMPONENTES QUE SE UTILIZARAN EN EL FORMULARIO

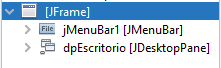


FORMULARIO CON LOS DATOS REQUERIDOS

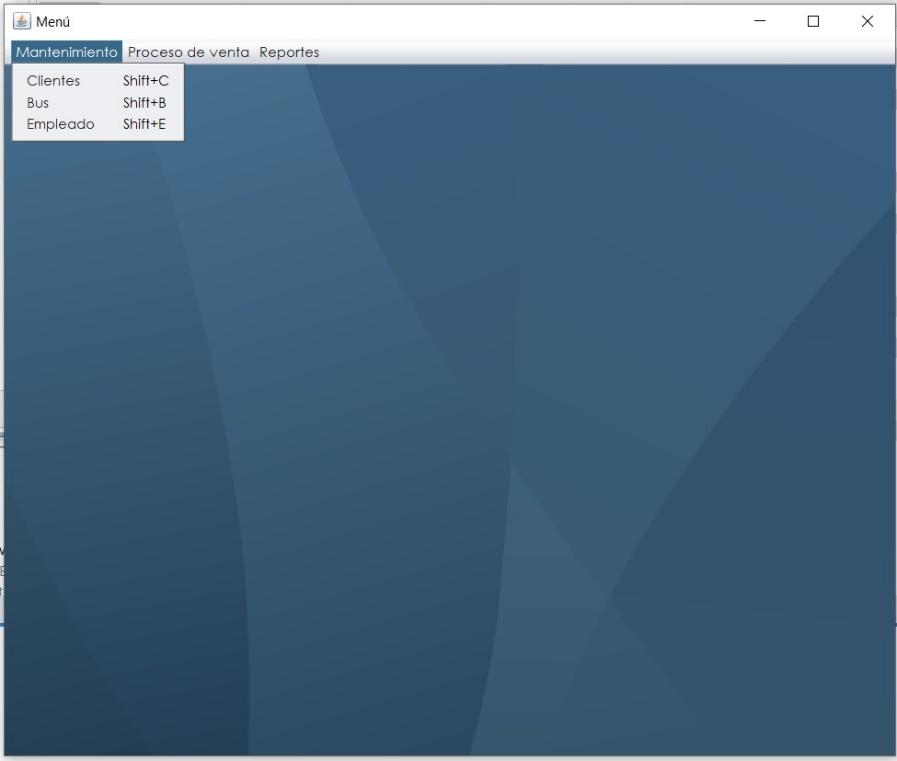


**R2: El sistema necesita un menú de inicio donde se podrá acceder a los formularios.**

COMPONENTES QUE SE UTILIZARAN EN EL FORMULARIO

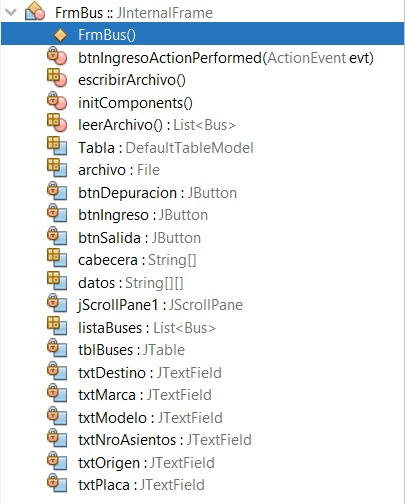


FORMULARIO CON LOS DATOS REQUERIDOS

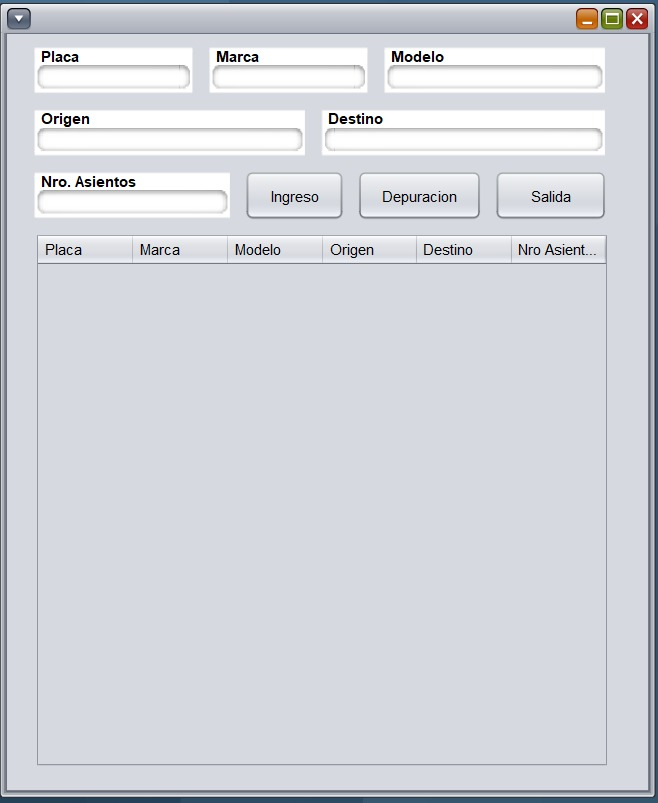


**R4: El sistema requiere que se incluya un formulario bus con las siguientes características: placa, marca, modelo, origen, destino, n# de asientos.**

COMPONENTES QUE SE UTILIZARAN EN EL FORMULARIO

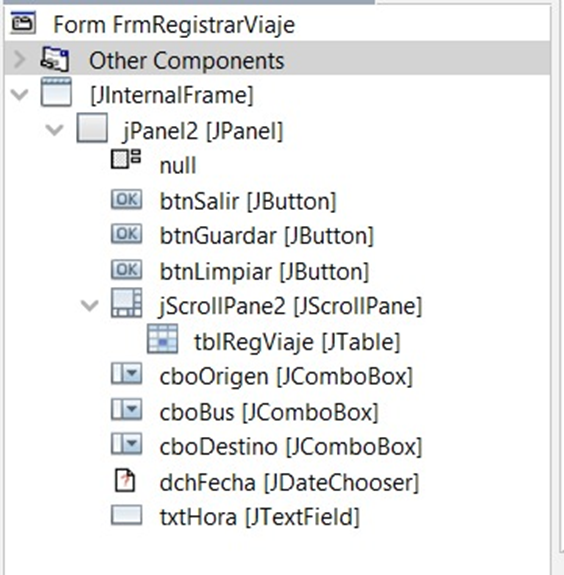


FORMULARIO CON LOS DATOS REQUERIDOS

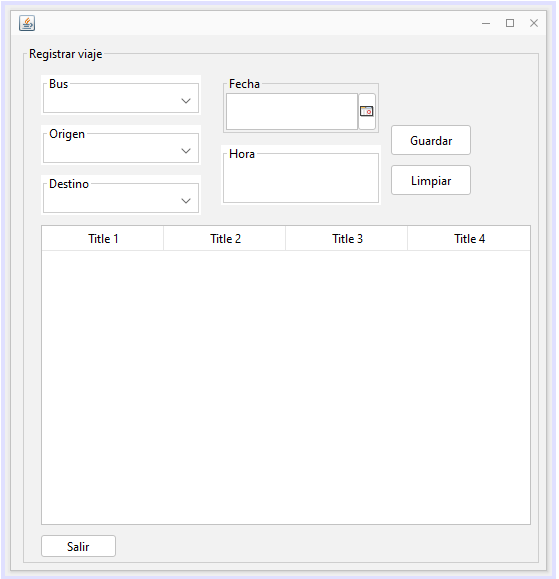


**R7: El sistema requiere un formulario de registro de viajes.**

COMPONENTES QUE SE UTILIZARAN EN EL FORMULARIO

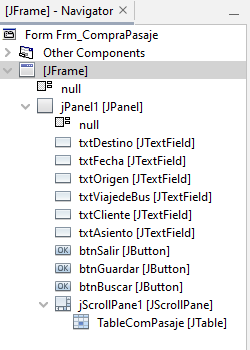


FORMULARIO CON LOS DATOS REQUERIDOS

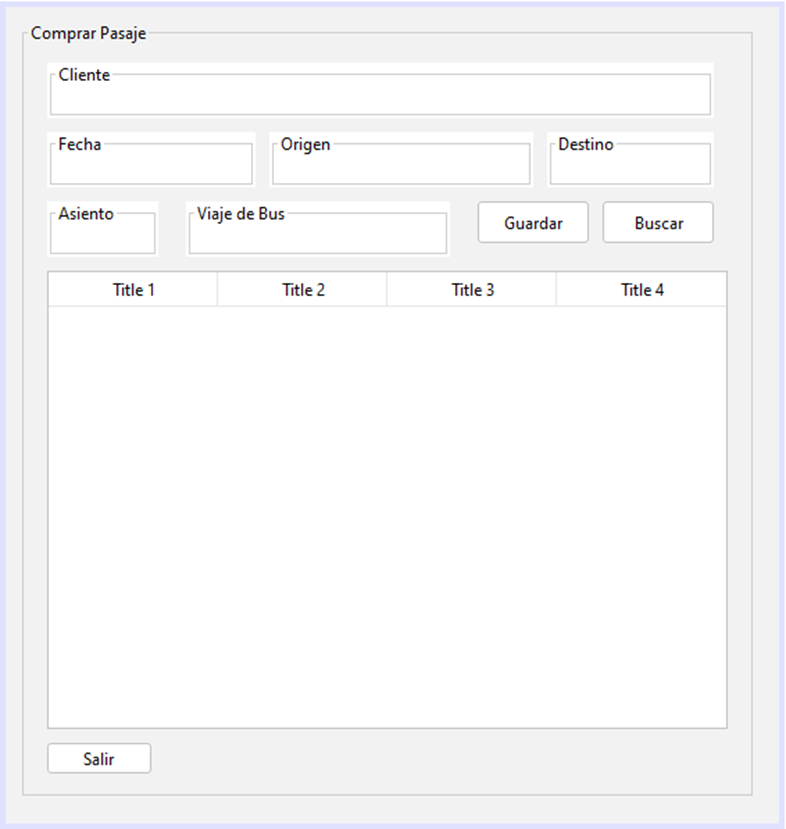


**R8: La aplicación necesita un formulario de compras de pasaje y requerirá las siguientes características: fecha, origen, destino, viaje de bus, cliente y numero de asiento.**

COMPONENTES QUE SE UTILIZARAN EN EL FORMULARIO



FORMULARIO CON LOS DATOS REQUERIDOS



* 1. Requerimientos del proyecto / Historias de Usuario.

A continuación, presentaremos los requerimientos del programa:

- El sistema requiere ingresar nombres, apellidos, dirección, correo, fecha de nacimiento del nuevo cliente, fecha de viaje, destino, todo esto dentro de un formulario.

- Al formulario anterior le corresponde dos botones: "registrar" que hace automáticamente el almacenado de datos a la tabla "clientes" y el botón "cancelar", que debe borrar todos los datos escritos en el formulario, para que esté libre para los futuros datos.

- En las celdas "fecha de viaje" y "destino" deberá contar con una ayuda de búsqueda (combobox) los cuales deben ser alternados de acuerdo a la disponibilidad de buses.

- El sistema requiere que se incluya un formulario con las siguientes características: id\_bus, marca, modelo, placa, origen, destino, n# de asientos. Con el cual se hará la configuración de viajes a los cuales será destinado de acuerdo a su capacidad y lugar donde se encuentre "origen".

- Validar los datos ingresados del nuevo cliente, DNI (que sea numérico = char 8 y no se repita) y edad >= 18 años. En caso el cliente se repita (DNI ya registrado), los datos se duplicarán para el viaje que se le está registrando actualmente.

- Validar datos ingresados del bus, este no debe contar con el mismo origen y destino. La capacidad máxima de los buses no debe exceder a los 90 asientos. el id\_bus debe ser automático, esto se logrará incrementando 1000 seguido de la placa: 10001-placa.

- Al formulario registrar viaje le corresponde tres botones, “registrar”, “limpiar” y “cancelar”, el primer botón tiene la función de guardar en una lista viaje reservado con los datos del cliente, el segundo limpia las casillas en el formulario, y el tercer botón cierra el formulario.

HU 7: Como usuario quiero registrar un viaje para que mi viaje quede reservado y listo para pagar.

* El formulario Comprar pasaje requerirá las siguientes características: fecha, origen, destino, viaje de bus, cliente y número de asiento.

HU 8: Como Administrador del sistema quiero que el formulario tenga las características solicitadas en el requerimiento para que el usuario vea con facilidad el sistema.

- Validar los datos ingresados en comprar pasaje, la fecha debe ser futura, en caso de elegir una fecha anterior mostrará un mensaje de error (Fecha no válida) y validar el origen y destino, estos dos no pueden ser iguales.

- En el campo cliente del formulario Comprar pasaje, validar la longitud del campo cliente, el cual tendrá que ser el DNI, también debe ser numérico y hacer una búsqueda para validar si existe.

HU 9: Como Administrador del sistema quiero validar los datos ingresados en “Comprar pasaje” con mensajes definidos para asegurar el funcionamiento adecuado de la aplicación.

* Se requiere declarar constantes para el precio del pasaje según las siguientes condiciones:

• Buses ECONÓMICOS: 80 soles (PEN)

• Buses VIP: 150 soles (PEN)

HU 10: Como Administrador Financiero quiero dar un precio establecido en el sistema y en caso de que no sea óptimo, actualizar los precios para mantener negociaciones con los clientes.

* Los formularios “Registrar viaje” y “Comprar pasaje” poseen el cuadro de texto fecha, este cuadro debe tener un calendario definido para facilitar el ingreso de fechas.

HU 11: Como administrador del sistema quiero mejorar el sistema mediante los controles del formulario para facilitar la comprensión de este.

* En el formulario “Comprar pasaje” debe mostrar la cantidad de asientos disponibles en el bus, este debe ir disminuyendo de acuerdo a los pasajes que se vayan comprando, es decir, la cantidad de asientos disponibles debe ser menor o igual a los asientos del bus.

HU 12: Como administrador del sistema quiero dar a conocer al cliente la cantidad de asientos disponibles en el bus para evitar que el cliente haga una reserva nuevamente.

- En el formulario Registro Cliente se pide ingresar los siguientes datos: DNI, Nombre, Apellido paterno, materno, fecha de nacimiento, correo y dirección.

- En caso de no ingresar los datos requeridos o no ingresar los datos en los formularios, se mostrará una advertencia de datos incorrectos.

- En el formulario Registrar Viaje se tienen los datos: Bus, Origen, Destino, Fecha y Hora.

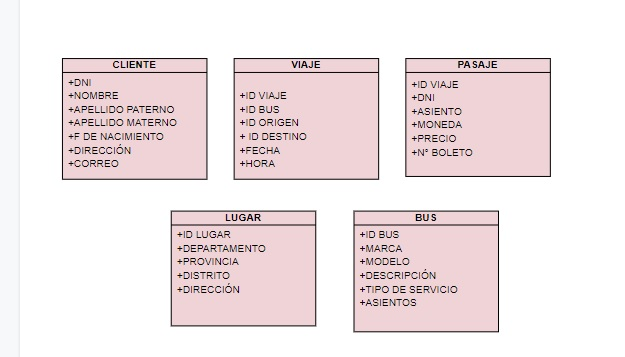
- Según los datos ingresados Bus, Origen y Destino van a derivar dependiendo de qué opciones se van a tomar.

- En el formulario Comprar Pasaje se pide ingresar: Fecha, Origen, Destino, Viaje de Bus, Cliente y Asiento.

- En el formulario Reporte de Viajes se debe mostrar: Bus, Origen, Destino, Fecha, Hora, Cliente, Viaje de Bus.

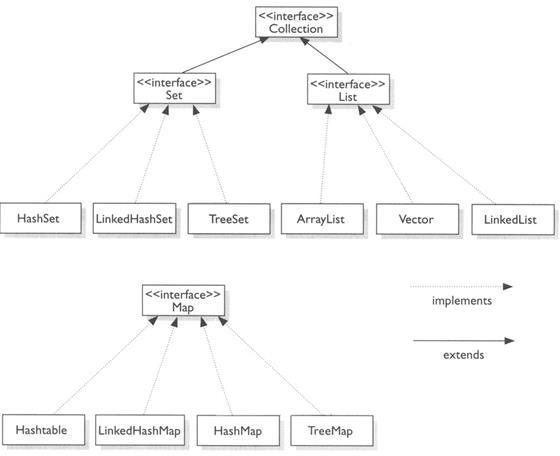
- Dentro del formulario Reporte de Viajes se debe tener un botón que genere el reporte completo con los datos de los datos obtenidos de los formularios.

* 1. Diagrama de clases del proyecto v 0.1



* 1. Resumen de colecciones en Java y forma de aplicación

Una colección representa un grupo de objetos, que son conocidos como elementos. Cuando queremos trabajar con un conjunto de elementos, necesitamos un almacén donde poder guardarlos. En Java, se emplea la interfaz genérica **Collection** para este propósito y con esta podemos almacenar cualquier tipo de objeto y podemos usar una serie de métodos comunes, como pueden ser: añadir, eliminar, obtener el tamaño de la colección. Partiendo de la interfaz genérica **Collection** extienden otra serie de interfaces genéricas. Estas subinterfaces aportan distintas funcionalidades sobre la interfaz anterior.

En el siguiente diagrama se muestran las relaciones de herencia entre los interfaces y clases más importantes en el framework de las colecciones.

**Tipos de colecciones**

1. **SET**

La interfaz Set define una colección que no puede contener elementos duplicados. Esta interfaz contiene, únicamente, los métodos heredados de Collection añadiendo la restricción de que los elementos duplicados están prohibidos. Es importante destacar que, para comprobar si los elementos son elementos duplicados o no lo son, es necesario que dichos elementos tengan implementada, de forma correcta, los métodos equals y hashCode. Para comprobar si dos Set son iguales, se comprobará si todos los elementos que los componen son iguales sin importar el orden que ocupen dichos elementos.

Dentro de la interfaz Set existen varios tipos de implementaciones realizadas dentro de la plataforma Java. Vamos a analizar cada una de ellas:

* **HashSet**: Almacena los elementos en una tabla hash. Es la implementación con mejor rendimiento de todas, pero no garantiza ningún orden a la hora de realizar iteraciones. Es la implementación más empleada debido a su rendimiento y a que, generalmente, no nos importa el orden que ocupen los elementos. Esta implementación proporciona tiempos constantes en las operaciones básicas siempre y cuando la función hash disperse de forma correcta los elementos dentro de la tabla hash. Es importante definir el tamaño inicial de la tabla ya que este tamaño marcará el rendimiento de esta implementación.
* **TreeSet**: Almacena los elementos ordenándolos en función de sus valores. Es bastante más lento que HashSet. Los elementos almacenados deben implementar la interfaz Comparable. Esta implementación garantiza, siempre, un rendimiento de log(N) en las operaciones básicas, debido a la estructura de árbol empleada para almacenar los elementos.
* **LinkedHashSet**: Almacena los elementos en función del orden de inserción. Es, simplemente, un poco más costosa que HashSet.

1. **LIST**

La interfaz List define una sucesión de elementos. A diferencia de la interfaz Set, la interfaz List sí admite elementos duplicados. A parte de los métodos heredados de Collection, añade métodos que permiten mejorar los siguientes puntos:

* Acceso posicional a elementos: manipula elementos en función de su posición en la lista.
* Búsqueda de elementos: busca un elemento concreto de la lista y devuelve su posición.
* Iteración sobre elementos: mejora el Iterator por defecto.
* Rango de operación: permite realizar ciertas operaciones sobre rangos de elementos dentro de la propia lista.

Dentro de la interfaz List existen varios tipos de implementaciones realizadas dentro de la plataforma Java, estas son:

* **ArrayList**: esta es la implementación típica. Se basa en un array redimensionable que aumenta su tamaño según crece la colección de elementos. Es la que mejor rendimiento tiene sobre la mayoría de las situaciones.
* **LinkedList**: esta implementación permite que mejore el rendimiento en ciertas ocasiones. Esta implementación se basa en una lista doblemente enlazada de los elementos, teniendo cada uno de los elementos un puntero al anterior y al siguiente elemento.

**Vector**: Esta es una situación similar al ArrayList, con la diferencia de que el Vector sí que está sincronizado. Este es un caso especial, ya que la implementación básica del resto de tipos de datos no está sincronizada. Esta clase existe desde las primeras versiones de Java, en las que no existía el marco de las colecciones descrito anteriormente. En las últimas versiones el Vector se ha acomodado a este marco implementando la interfaz List..

1. **MAPS**

La interfaz **Map** asocia claves a valores. Esta interfaz no puede contener claves duplicadas y; cada una de dichas claves, sólo puede tener asociado un valor como máximo.

Dentro de la interfaz **Map** existen varios tipos de implementaciones realizadas dentro de la plataforma Java. Veremos algunas de ellas:

* **HashMap**: Almacena las claves en una tabla *hash*. Es la implementación con mejor rendimiento de todas, pero no garantiza ningún orden a la hora de realizar iteraciones. Esta implementación proporciona tiempos constantes en las operaciones básicas siempre y cuando la

función *hash* disperse de forma correcta los elementos dentro de la tabla *hash*. Es importante definir el tamaño inicial de la tabla ya que este tamaño marcará el rendimiento de esta implementación.

* **TreeMap**: Almacena las claves ordenándolas en función de sus valores. Es bastante más lento que *HashMap*. Las claves almacenadas deben

implementar la interfaz **Comparable**. Esta implementación garantiza, siempre, un rendimiento de *log(N)* en las operaciones básicas, debido a la estructura de árbol empleada para almacenar los elementos.

* **LinkedHashMap**: Almacena las claves en función del orden de inserción. Es, simplemente, un poco más costosa que **HashMap**.
  1. Resumen sobre archivos en Java y forma de aplicación

Un archivo o fichero informático es un conjunto de bits que son almacenados en un dispositivo. Un archivo es identificado por un nombre y la descripción de la carpeta o directorio que lo contiene. A los archivos informáticos se les llama así porque son los equivalentes digitales de los archivos escritos en expedientes, tarjetas, libretas, papel o microfichas del entorno de oficina tradicional.

Está compuesto por:

- Nombre: Identificación del archivo.

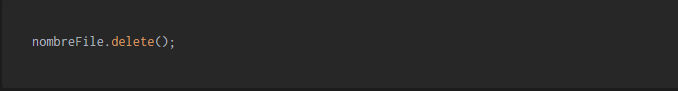
- Extensión: Indica el tipo de archivo.

**Clase File**

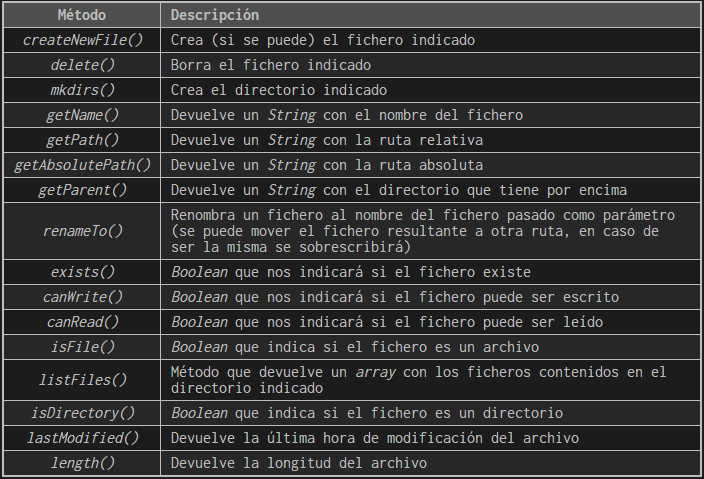
La clase **File** además de proporcionarnos información sobre los archivos y directorios nos permite crearlos y eliminarlos.

Para ello esta clase nos permite crearlos utilizando:



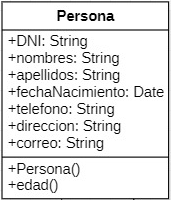
y borrarlos con:

Además de los anteriores disponemos de estos métodos para gestionarlos:



* 1. Manual de clases

**CLASE PERSONA**



**Atributos:**

En esta clase todos los atributos manejan un modificador de acceso público, es decir que son accesibles desde cualquier lugar del programa.

**- DNI:** Almacenará el D.N.I. que identificará a cada persona, siendo de tipo String ya que puede empezar con el dígito “0”.

- **nombres:** Almacenará el nombre de la persona, siendo de tipo String ya que es una cadena de caracteres conformado por letras.

- **apellidos:** Almacenará el apellido de la persona, siendo de tipo String ya que es una cadena de caracteres conformado por letras.

- **fechaNacimiento:** Almacenará la fecha de nacimiento de la persona (día, mes y año), siendo de tipo clase String, ya que nos permite de forma sencilla manejar una fecha, número y símbolos.

- **telefono:** Almacenará el teléfono de la persona, siendo de tipo String ya que puede empezar con el dígito “0”.

- **direccion:** Almacenará la dirección de la persona, siendo de tipo String ya que es una cadena de caracteres conformado por letras, números y símbolos.

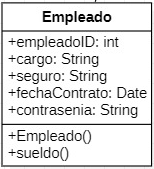
- **correo:** Almacenará el correo de la persona, siendo de tipo String ya que es una cadena de caracteres conformado por letras, números y símbolos.

**Métodos:**

En esta clase tiene todos los métodos, son de modificador de acceso público, para que en otras clases podamos acceder a ellos, sin mayor problema.

* **Persona:** Es el constructor de la clase, el cual permite insertar información al objeto de esta. No tiene un tipo de retorno, pero tiene el mismo nombre de la clase PERSONA.
* **edad:** Método quecaptura el dato que se quiere almacenar dentro del atributo fechaNacimiento y retorna la edad de la persona.

**CLASE EMPLEADO**



**Atributos:**

En esta clase todos los atributos manejan un modificador de acceso público, es decir que son accesibles desde cualquier lugar del programa.

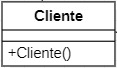
* **empleadoID:** Almacenará el ID que identificará a cada empleado, siendo de tipo String ya que puede empezar con el digito “0”.
* **cargo:** Almacenará el cargo del empleado, siendo de tipo String ya que es una cadena de caracteres conformado por letras.
* **seguro:** Almacenará el seguro del empleado, siendo de tipo String ya que es una cadena de caracteres conformado por letras.
* **fechaContrato:** Almacenará la fecha de contrato del empleado (día, mes y año), siendo de tipo clase Date, ya que nos permite forma sencilla manejar una fecha.
* **contrasenia:** Almacenará la contraseña de acceso, siendo de tipo String ya que una contraseña puede estar compuesta por una cadena de caracteres con letras, números y símbolos.

**Métodos:**

En esta clase tiene todos los métodos, son de modificador de acceso público, para que en otras clases podamos acceder a ellos, sin mayor problema.

* **Empleado:** Es el constructor de la clase, el cual permite insertar información al objeto de esta. No tiene un tipo de retorno, pero tiene el mismo nombre de la clase Empleado.
* **sueldo:** Método que captura el dato y varía dependiendo del cargo que tenga el empleado.

**CLASE CLIENTE**



**Atributos:**

Para esta clase se utilizó Herencia esto nos permite que una clase herede las características (atributos y métodos) de otra clase, en este caso heredará los datos de la clase Persona.

**Métodos:**

En esta clase tiene todos los métodos, son de modificador de acceso público, para que en otras clases podamos acceder a ellos, sin mayor problema.

* **Cliente:** Es el constructor de la clase, el cual permite insertar información al objeto de esta. No tiene un tipo de retorno, pero tiene el mismo nombre de la clase Cliente.

**CLASE VIAJE**

Texto

Descripción generada automáticamente

**Atributos:**

* **viajeID**: Atributo con el que se identificará, buscará, modificará y eliminará un viaje determinado en el sistema, siendo de tipo int para facilitar la generación de ID’s.
* **fechaOrigen**: Atributo en el que se guardará la fecha de origen del viaje, este atributo es de tipo Date porque su formato complementa con la inserción de este dato mediante componentes en el formulario.
* **horaOrigen:** Atributo en el que se guardará la hora de origen del viaje, este atributo es de tipo Date porque su formato complementa con la inserción de este dato mediante componentes en el formulario.
* **fechaOrigen:** Atributo en el que se guardará la fecha de destino del viaje, este atributo es de tipo Date porque su formato complementa con la inserción de este dato mediante componentes en el formulario.
* **horaOrigen:** Atributo en el que se guardará la hora de destino del viaje, este atributo es de tipo Date porque su formato complementa con la inserción de este dato mediante componentes en el formulario.
* **pasajero:** Objeto de tipo cliente donde se le asignará un cliente ya registrado para registrar el viaje de este.
* **nroAsiento:** Atributo que guardará el número de asiento del cliente, de tipo int para mayor facilidad de almacenamiento de datos.
* **movilidad:** Objeto de tipo Bus donde se guardarán los datos del bus para la lectura de asientos disponibles en el bus.
* **origen:** Objeto de tipo Lugar donde se guardará los datos de la posición del bus en el origen del viaje.
* **destino:** Objeto de tipo Lugar donde se guardarán los datos de la posición del bus en el destino del viaje.

**Métodos:**

* **Viaje():** Método constructor para inicializar todos los atributos de un objeto de esta clase.

**CLASE VENTA**

Texto

Descripción generada automáticamente

**Atributos:**

* **ventaID:** Atributo para identificar, buscar, modificar y eliminar una venta determinada en el sistema, siendo de tipo int para la facilitar la generación de ID’s.
* **formaPago:** Atributo que almacena el método de pago del cliente, siendo de tipo String porque los tipos de pago son una cadena de caracteres, donde se podrá seleccionar.
* **descripción:** Atributo que guardará el tipo de servicio que ofrecerá la empresa VIP o económico, de tipo String porque la descripción es una cadena de carcteres.
* **moneda:** Atributo donde se guardará el tipo de moneda con el que el cliente desea pagar. Este atributo es de tipo String porque el tipo de moneda puede ser soles o dólares.
* **viaje:** Objeto de tipo viaje originado por la relación de composición, guardará los datos de un viaje en particular.

**Métodos:**

* **Venta():** Método constructor para inicializar todos los atributos de un objeto de esta clase.
* **costo():** Método para hallar el costo del viaje de acuerdo dependiendo de la descripción y de la moneda de esta clase, es de tipo double porque los costos son decimales.

**CLASE PASAJE**

Tabla

Descripción generada automáticamente

**Atributos:**

* **pasajeID:** Atributo que servirá para identificar, buscar, modificar y eliminar un pasaje determinado en el sistema, siendo de tipo int para facilitar la generación de ID’s.
* **pago:** Objeto de tipo Venta originado por la relación de composición, guardará los datos de una venta en particular.
* **estaAnulado:** Atributo que mostrará el estado del pasaje.

**Métodos:**

* **Pasaje():** Método constructor para inicializar todos los atributos de un objeto de esta clase.

**CLASE ASIENTOBUS**

**Tabla

Descripción generada automáticamente**

**Atributos:**

* **nroAsiento:** Atributo con el que se identificará y buscará un número de asiento determinado en el bus, siendo de tipo String para mejor distinción de los números de asiento.
* **disponibilidad:** Atributo que determina si el asiento está libre (true) u ocupado (false), siendo de tipo boolean porque solo toma en cuenta 2 estados.

**Métodos:**

* **AsientoBus():** Método constructor para inicializar todos los atributos de un objeto de esta clase.

**CLASE BUS**

**Texto

Descripción generada automáticamente**

**Atributos:**

* **nroBus:** Atributo con el que se identificará, buscará, modificará y eliminará un bus determinado en el sistema, siendo de tipo String porque contiene tanto caracteres como números.
* **marca:** Atributo que guardará la marca de bus, siendo de tipo String que almacenará una cadena.
* **modelo:** Atributo que guardará el modelo de un bus, siendo de tipo String que almacenará una cadena de caracteres, números y signos.
* **placa:** Atributo que guardará la placa de un bus, siendo única y de tipo String que almacenará una cadena de caracteres, números y signos.
* **cantidadAsientos:** Atributo que almacenará una cantidad de asientos establecida, siendo de tipo Int porque es el tipo de variable óptimo en esta situación.
* **asientos:** Arreglo de tipo AsientoBusdonde se guardarán los datos de los asientos del bus para conocer la disponibilidad y número de asientos.
* **servicio:** Atributo que guardará el tipo de servicio que ofrecerá la empresa VIP o económico, de tipo String porque la descripción es una cadena de caractéres.

**Métodos:**

* **Bus():** Método constructor para inicializar todos los atributos de un objeto de esta clase.

**CLASE LUGAR**

**Texto

Descripción generada automáticamente**

**Atributos:**

* **departamento:** Atributo que guardará el departamento de un lugar, siendo de tipo String que almacenará una cadena de caracteres.
* **provincia:** Atributo que guardará la provincia de un lugar, siendo de tipo String que almacenará una cadena de caracteres.
* **distrito:** Atributo que guardará el distrito de un lugar, siendo de tipo String que almacenará una cadena de caracteres.
* **direccion:** Atributo que guardará la dirección de un lugar, siendo de tipo String que almacenará una cadena de caracteres.

**Métodos:**

* **Lugar():** Método constructor para inicializar todos los atributos de un objeto de esta clase.

**CLASE VIAJE**

Texto

Descripción generada automáticamente

**Atributos:**

* **viajeID**: Atributo con el que se identificará, buscará, modificará y eliminará un viaje determinado en el sistema, siendo de tipo int para facilitar la generación de ID’s.
* **fechaOrigen**: Atributo en el que se guardará la fecha de origen del viaje, este atributo es de tipo Date porque su formato complementa con la inserción de este dato mediante componentes en el formulario.
* **horaOrigen:** Atributo en el que se guardará la hora de origen del viaje, este atributo es de tipo Date porque su formato complementa con la inserción de este dato mediante componentes en el formulario.
* **fechaOrigen:** Atributo en el que se guardará la fecha de destino del viaje, este atributo es de tipo Date porque su formato complementa con la inserción de este dato mediante componentes en el formulario.
* **horaOrigen:** Atributo en el que se guardará la hora de destino del viaje, este atributo es de tipo Date porque su formato complementa con la inserción de este dato mediante componentes en el formulario.
* **pasajero:** Objeto de tipo cliente donde se le asignará un cliente ya registrado para registrar el viaje de este.
* **nroAsiento:** Atributo que guardará el número de asiento del cliente, de tipo int para mayor facilidad de almacenamiento de datos.
* **movilidad:** Objeto de tipo Bus donde se guardarán los datos del bus para la lectura de asientos disponibles en el bus.
* **origen:** Objeto de tipo Lugar donde se guardará los datos de la posición del bus en el origen del viaje.
* **destino:** Objeto de tipo Lugar donde se guardarán los datos de la posición del bus en el destino del viaje.

**Métodos:**

* **Viaje():** Método constructor para inicializar todos los atributos de un objeto de esta clase.

**CLASE VENTA**

Texto

Descripción generada automáticamente

**Atributos:**

* **ventaID:** Atributo para identificar, buscar, modificar y eliminar una venta determinada en el sistema, siendo de tipo int para la facilitar la generación de ID’s.
* **formaPago:** Atributo que almacena el método de pago del cliente, siendo de tipo String porque los tipos de pago son una cadena de caracteres, donde se podrá seleccionar.
* **descripción:** Atributo que guardará el tipo de servicio que ofrecerá la empresa VIP o económico, de tipo String porque la descripción es una cadena de caracteres.
* **moneda:** Atributo donde se guardará el tipo de moneda con el que el cliente desea pagar. Este atributo es de tipo String porque el tipo de moneda puede ser soles o dólares.
* **viaje:** Objeto de tipo viaje originado por la relación de composición, guardará los datos de un viaje en particular.

**Métodos:**

* **Venta():** Método constructor para inicializar todos los atributos de un objeto de esta clase.
* **costo():** Método para hallar el costo del viaje dependiendo de la descripción y de la moneda de esta clase, es de tipo double porque los costos son decimales.

**CLASE REGISTROVENTAS**

**Tabla

Descripción generada automáticamente**

**Atributos:**

* **listaVentas:** Lista de objetos de tipo Venta que recopilará todas las ventas realizadas.

**Métodos:**

* **calcularTotal():** Método para hallar el total de ventas mediante el costo de cada una, es de tipo double porque los costos son decimales.
* **RegistroVenta():** Método constructor para inicializar todos los atributos de un objeto de esta clase.

**CLASE PASAJE**

Tabla

Descripción generada automáticamente

**Atributos:**

* **pasajeID:** Atributo que servirá para identificar, buscar, modificar y eliminar un pasaje determinado en el sistema, siendo de tipo int para facilitar la generación de ID’s.
* **pago:** Objeto de tipo Venta originado por la relación de composición, guardará los datos de una venta en particular.
* **estaAnulado:** Atributo que mostrará el estado del pasaje y el porque fue cancelado.

**Métodos:**

* **Pasaje():** Método constructor para inicializar todos los atributos de un objeto de esta clase.

**CLASE REGISTROANULADOS**

**Tabla

Descripción generada automáticamente**

**Atributos:**

* **listaPasajes:** Lista de objetos de tipo Pasaje que recopilará todos los pasajes registrados incluido los anulados o cancelados.

**Métodos:**

* **RegistroAnulados():** Método constructor para inicializar todos los atributos de un objeto de esta clase.

**CLASE COMPRA**

**Texto

Descripción generada automáticamente con confianza media**

**Atributos:**

* **facturaID:** Atributo que servirá para identificar, buscar, modificar y eliminar una compra de la empresa en el sistema, siendo de tipo int para facilitar la generación de ID’s.
* **RUC:** Atributo que servirá para identificar el RUC de una factura, siendo de tipo String porque es una variable que no realizará operaciones.
* **Total:** Atributo que guardará el costo de la compra, sineod de tipo float porque los costos son decimales.
* **razonSocial:** Atributo que almacenará el nombre con el cual la empresa registro, siendo de tipo String porque es una cadena.

**Métodos:**

* **Compra():** Método constructor para inicializar todos los atributos de un objeto de esta clase.

**CLASE REGISTROCOMPRAS**

**Tabla

Descripción generada automáticamente**

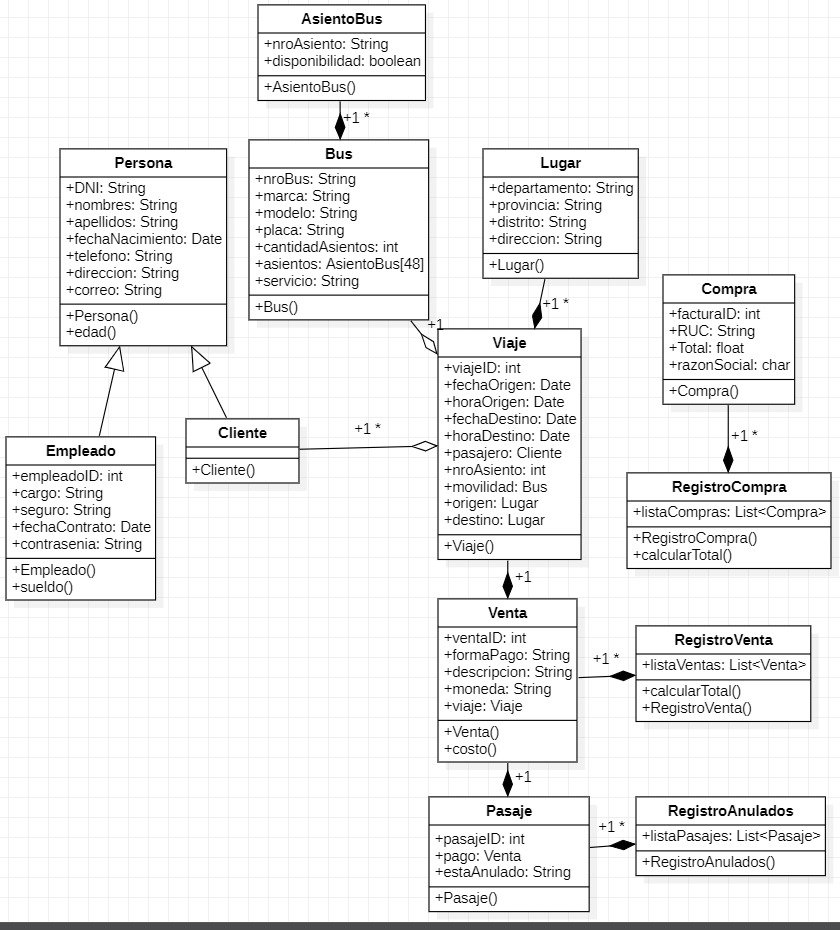
**Atributos:**

* **listaCompras:** Lista de objetos de tipo Compra que recopilará todas las compras realizadas por la empresa.

**Métodos:**

* **RegistroCompra():** Método constructor para inicializar todos los atributos de un objeto de esta clase.
* **calcularTotal():** Método para hallar el total de compras mediante el total de cada una, es de tipo double porque los costos son decimales.
  1. Diagrama de clases actualizado v 0.2

El diagrama mostrado describe la estructura del sistema que se utilizará para el desarrollo del software el cual contiene sus clases, con sus atributos y métodos.



**Fuente:** Diagrama elaborado y verificado por Chumpitaz Pacheco Cristofer Eduardo y De Paz Cotrina David Moises

* 1. Diseño de interfaces

El componente principal de todas las interfaces gráficas en java es una ventana. En java, podemos diferenciar principalmente dos tipos de ventanas:

**JFrame:** Es la ventana principal de nuestra aplicación gráfica. En ella podemos ver los típicos controles de una ventana de Windows como el de cerrar, maximizar o minimizar.

**JDialog:** Este tipo de ventana no tiene los controles de maximizar o minimizar. Además, se encuentran dentro de un JFrame y puede ser de dos tipos

**Modal:** Mientras una ventana de este tipo esté abierta no podremos hacer otra cosa.

**No modal:** En este caso, podemos interactuar con la aplicación incluso aunque se encuentre abierta.

Cuando vayamos a crear una interfaz gráfica de usuario (IGU), debemos tener en cuenta que lo primero que debemos crear será un JFrame.

**Características de un JFrame de las interfaces gráficas en java**

Es importante saber que los JFrame tienen una serie de características que se deben conocer:

Son invisibles, es decir que cuando lanzamos el programa y se ejecuta la ventana, hay que hacerla visible para que se vea.

No tienen ningún tamaño, el tamaño por defecto es de cero píxeles de alto por cero píxeles de ancho.

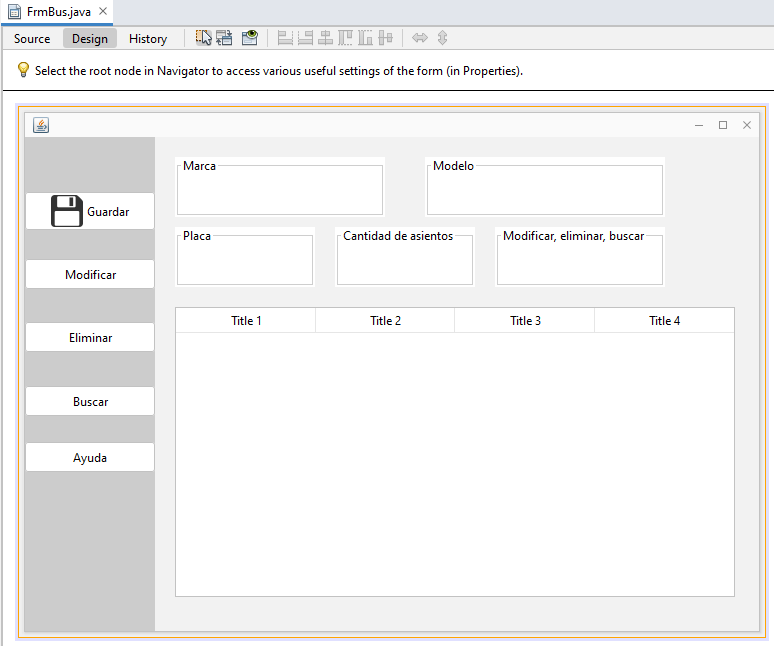
Tenemos que decirle a nuestra ventana lo que queremos hacer cuando se cierre, ya que si no lo hacemos el programa seguirá en ejecución consumiendo recursos.

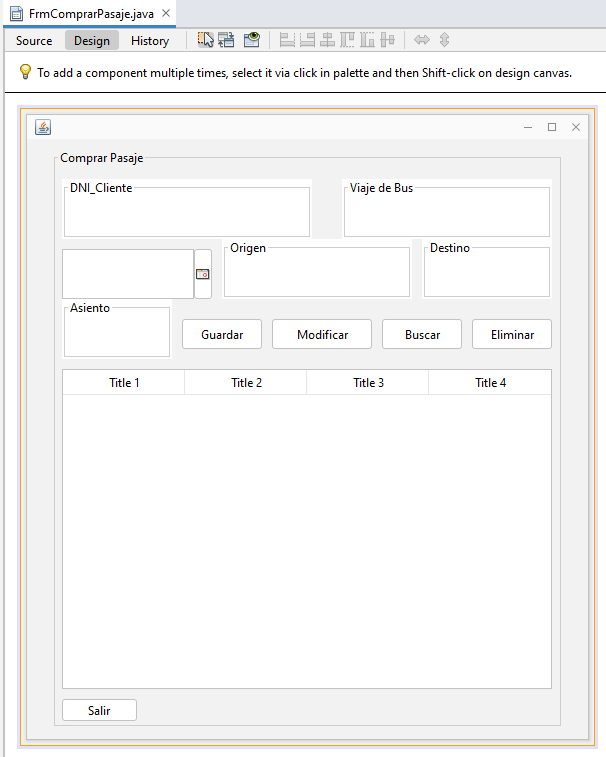
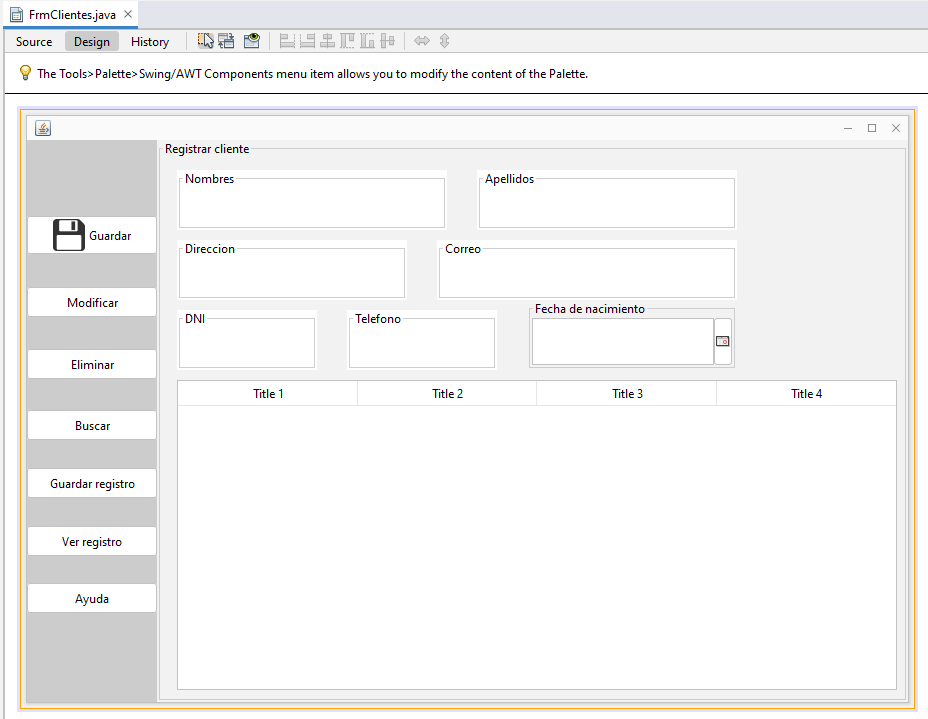
Por otro lado, para construir nuestras aplicaciones gráficas, podemos hacer uso de diferentes elementos swing.

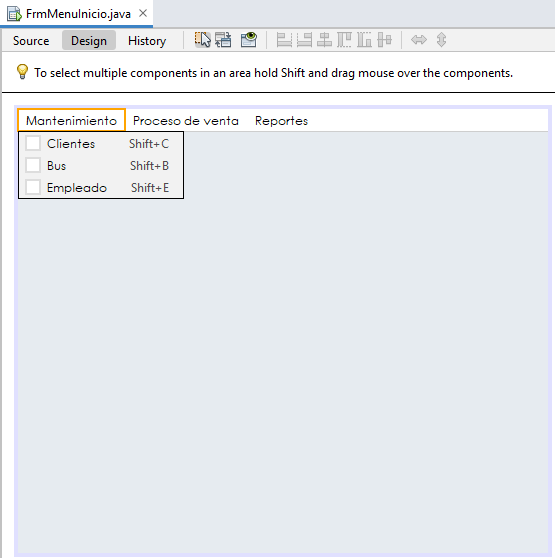
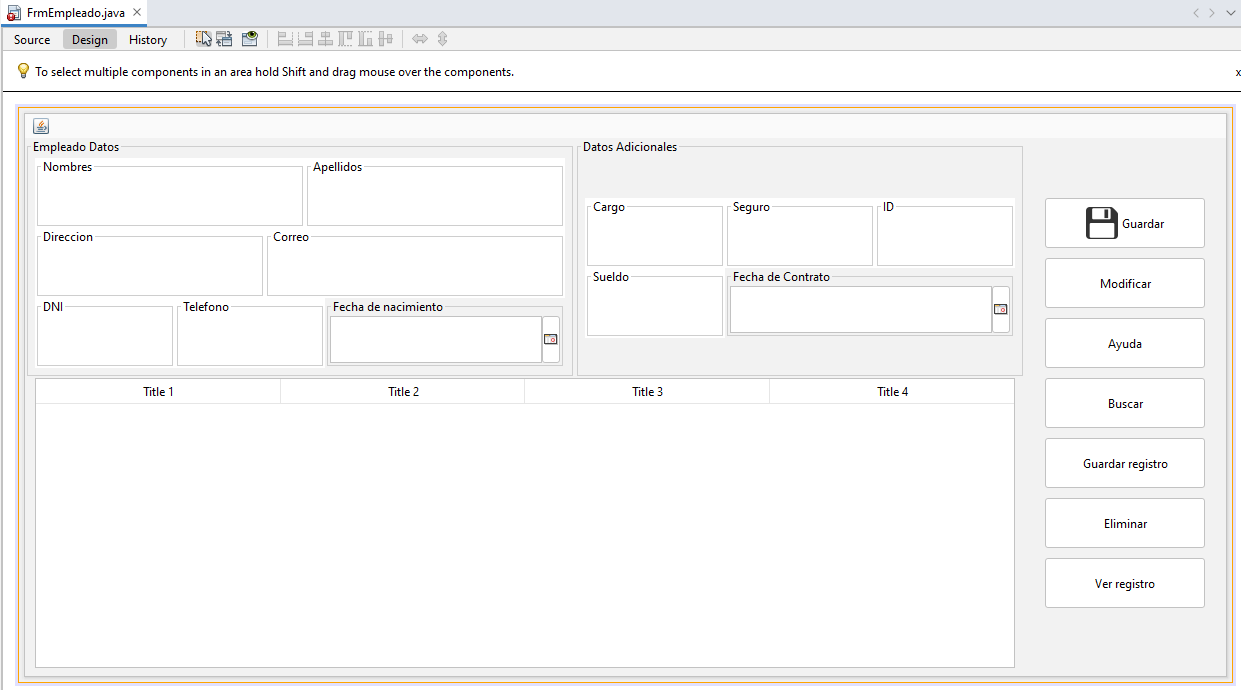
Elementos de una interfaz gráfica con Swing.

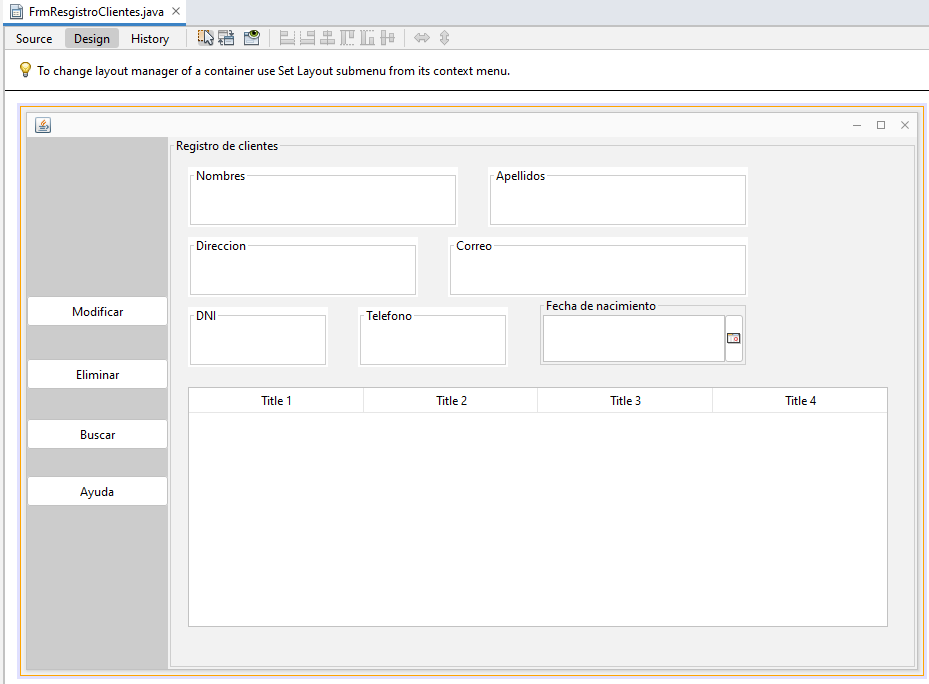
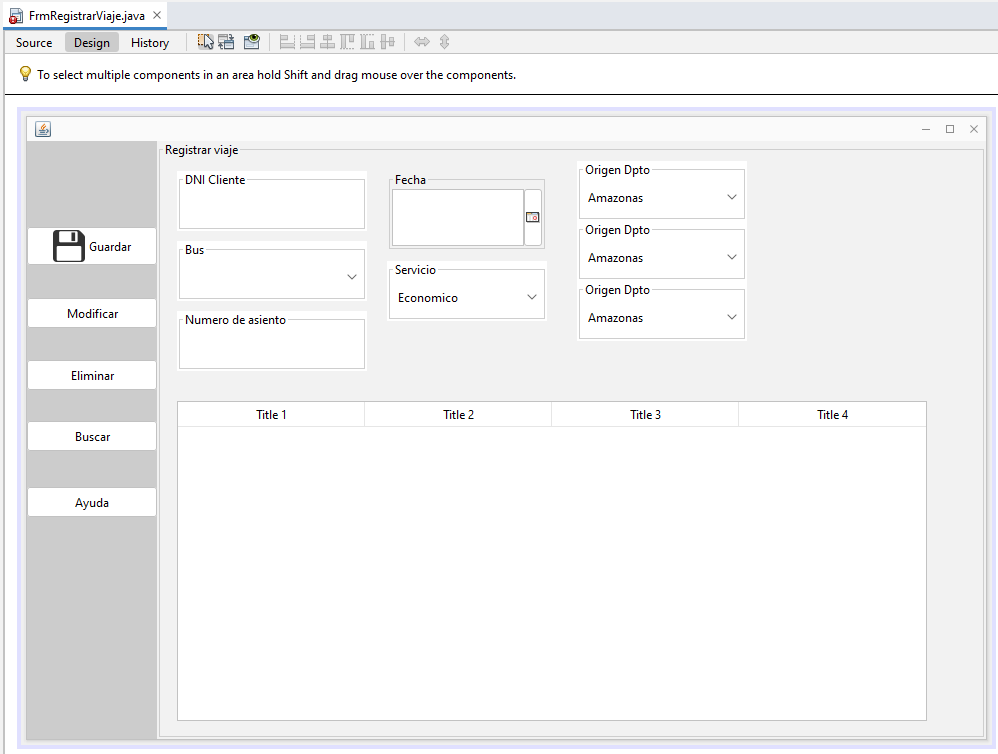
Algunos de los elementos principales que podemos encontrar en una aplicación gráfica son los siguientes.

* **JPanel:** Es una especie de cristal transparente que se coloca encima del JFrame o ventana principal y que puede contener o no una distribución para los diferentes controles. Esta distribución es lo que se llama layout del JPanel. Si por el contrario, no se le asigna ninguna distribución al JPanel, o lo que es lo mismo, se le adjudica un layout nulo, podremos colocar los controles en la posición de la lámina que nosotros queramos.
* **JLabel:** Son etiquetas que muestran información al usuario. Se suele colocar como título o acompañando a los controles de tipo JTextField, JComboBox, etc.
* **JTextField:** Son cuadros de texto donde el usuario puede escribir, pero una linea como máximo.
* **JComboBox:** Son cuadros desplegables. El usuario podrá desplegar este control para seleccionar una opción u otra.
* **JButton:** Estos controles, son botones donde el usuario puede hacer clic para ejecutar un evento.
* **JRadioButton:** Son botones de opción. Generalmente se colocan dentro de un ButtonGroup para que al seleccionar una opción se desmarque otra que pueda estar seleccionada.
* **JCheckBox:** Permiten seleccionar varias opciones a la vez.
* **JTextArea:** A diferencia del JTextField, este control permite escribir varias lineas de texto en el mismo control.
* **JTable:** Este control dibuja una tabla donde podemos alojar por ejemlpo, registros de una base de datos.
* **JScrollPane:** Es un control que suele colocarse dentro de los JTextArea o las JTable, ya que en caso de que haya mucho texto o muchos registros, este control nos permite desplazarnos haciendo scroll hacia abajo o hacia arriba según nuestra necesidad.
* **JPasswordField:** Funciona igual que un JTextField, pero es específico para introducir contraseñas.
* **JList:** Es un control que permite seleccionar varios elementos de una lista.

**INTERFACES GRÁFICAS**



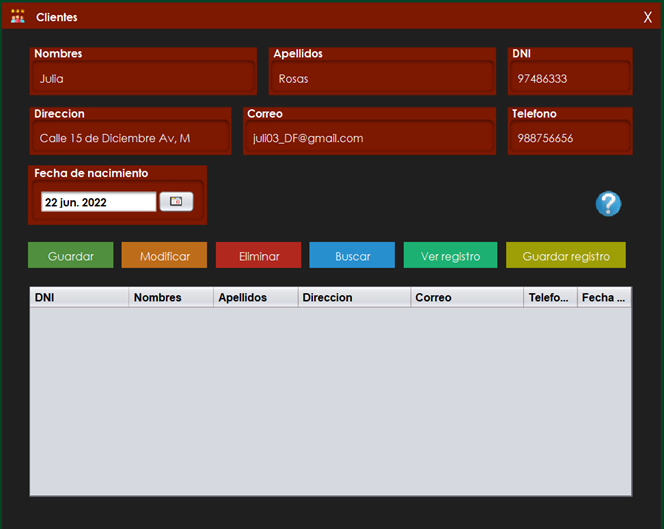


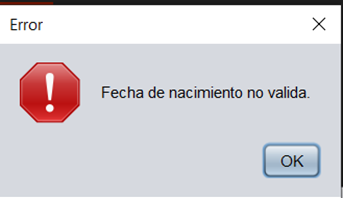
\*Primera fase de diseños

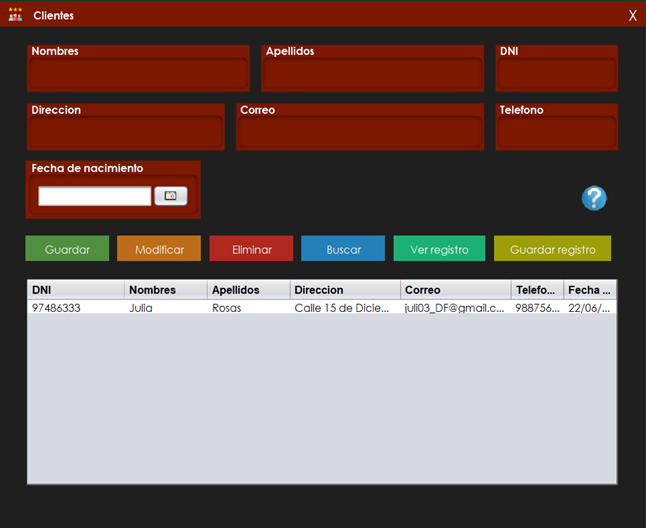
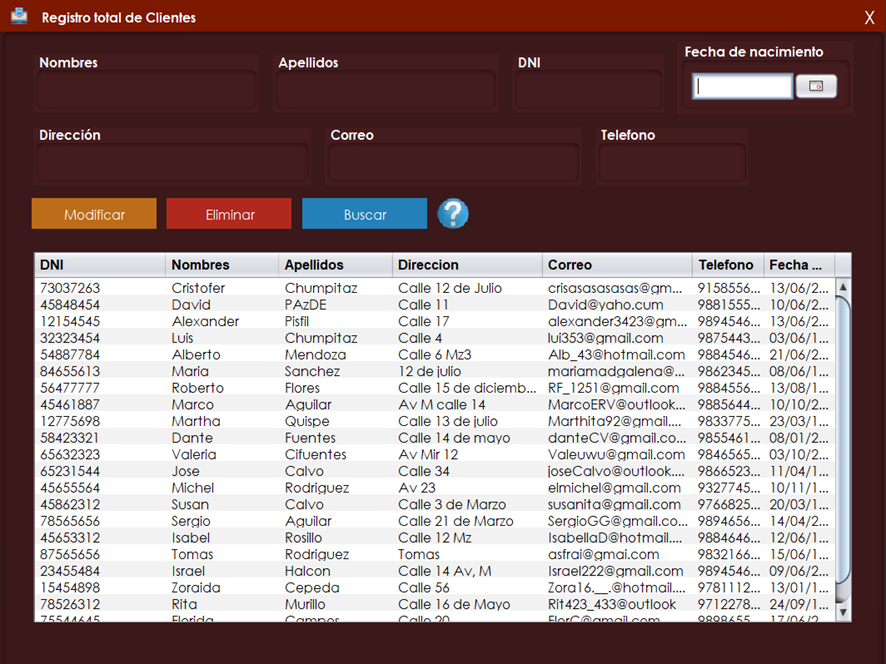
* 1. Pruebas de software

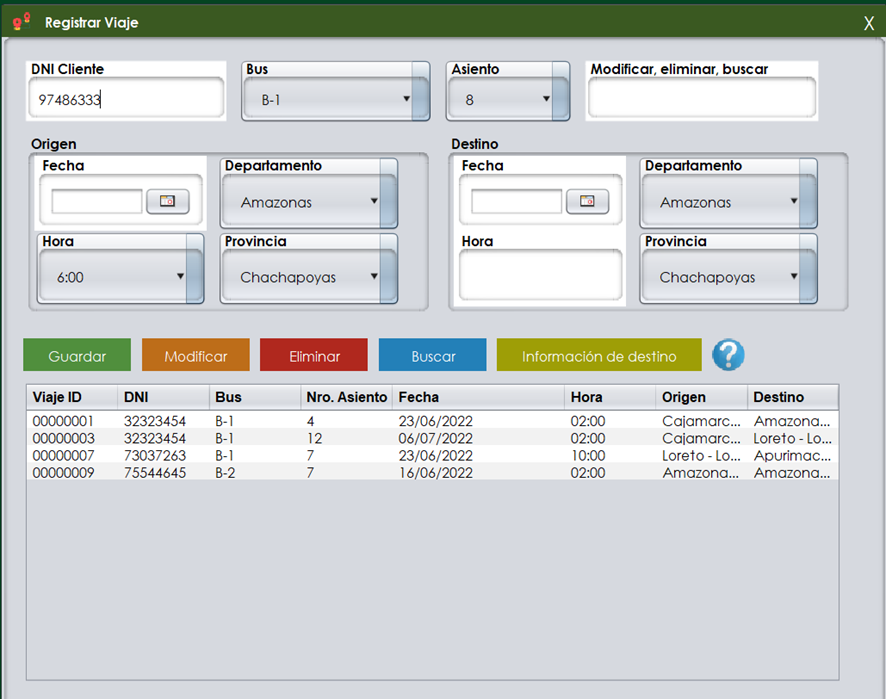
**LOGIN**

**MENÚ DE INICIO**

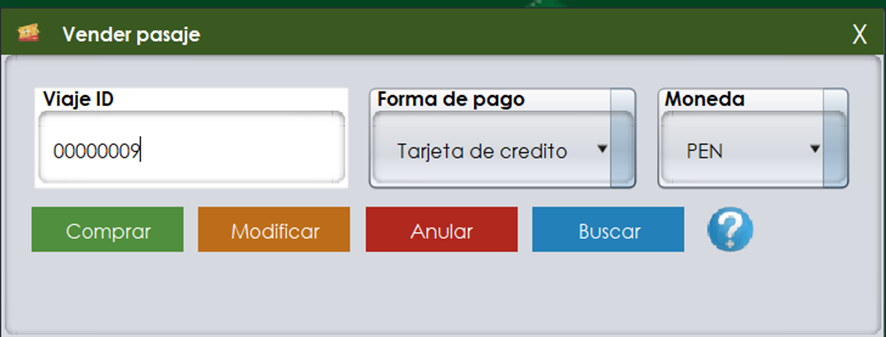
**FORM CLIENTES** 

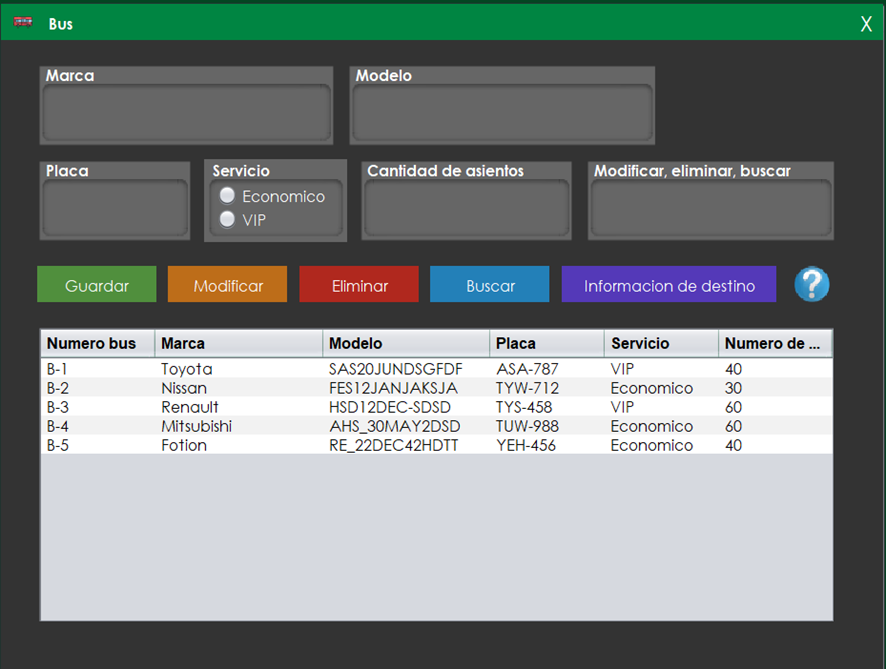


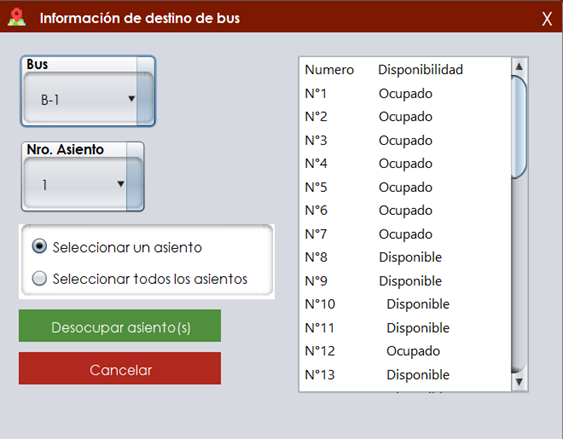
**FORM REGISTRO TOTAL DE CLIENTES**

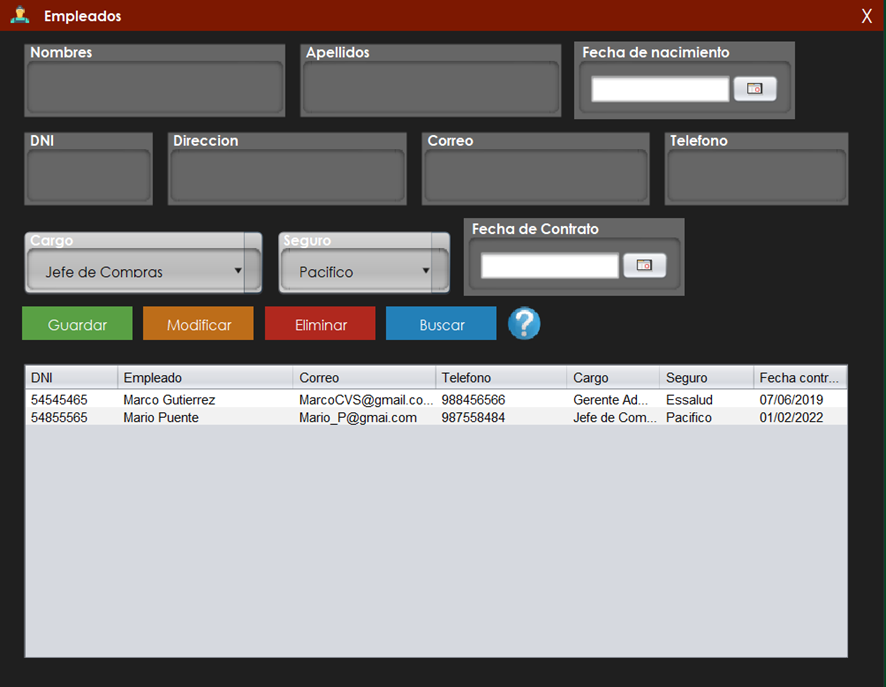
**FORM REGISTRAR VIAJE**

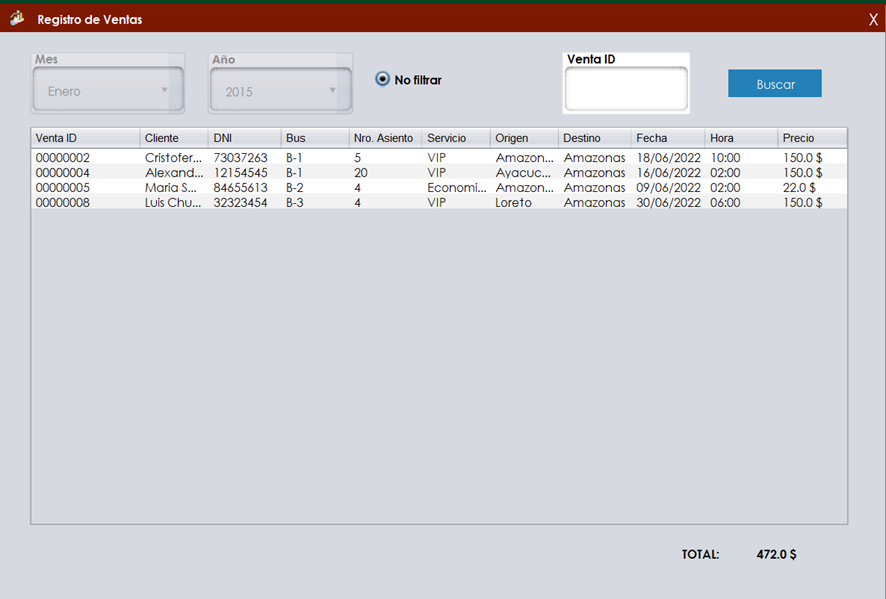
**FORM INFORMACIÓN DE DESTINO**

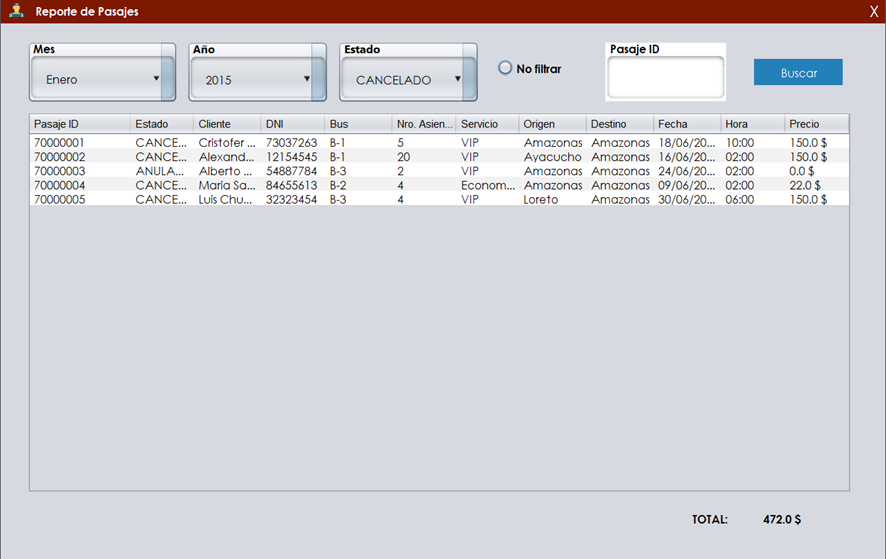
**FORM VENDER PASAJE**

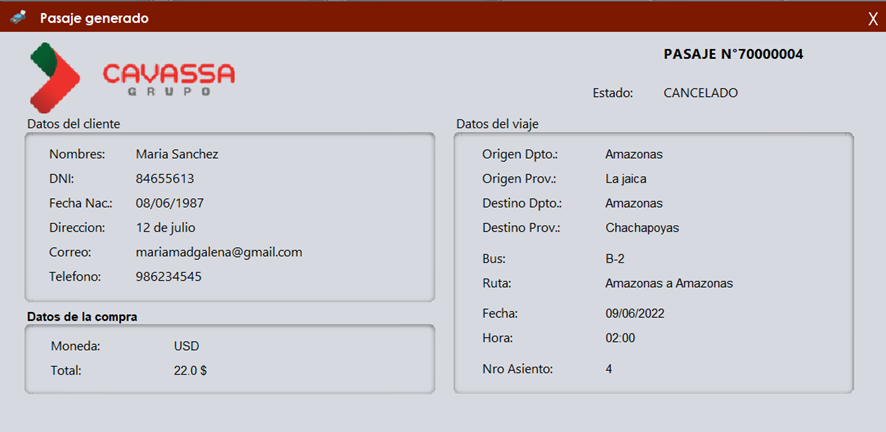
**FORM BUS**



**FORM EMPLEADOS**

**FORM REGISTRO DE VENTAS**

**FORM REPORTE DE PASAJES**



**CAPÍTULO III**

1. **CONCLUSIONES Y RECOMENDACIONES**
   1. Conclusiones

Como conclusión, hemos creado una aplicación para un sistema de registro y venta de boletos en la empresa de transportes Cavassa utilizando como metodología de desarrollo ágil Scrum, además para un manejo de información más eficiente se creó un diagrama de clases y a su vez implementando a la aplicación técnicas de programación orientada a objetos utilizando el lenguaje java.

* 1. Recomendaciones

Como recomendación a la empresa de transportes Cavassa sobre nuestra aplicación es capacitar al encargado del área de registro y venta de pasaje, con el fin de tener un manejo completo del aplicativo esto para evitar alguna complicación o error causado, caso contrario proceder a ponerse en contacto con los desarrolladores.

**CAPÍTULO IV**

**BIBLIOGRAFÍA**

* Morales, J. (2016). *Gestión de archivos en Java.* Recuperado de: <http://jmoral.es/blog/IO-java>
* Universidad Politécnica de Valencia (2017). *Las colecciones en Java.* Recuperado de: <http://www.androidcurso.com/index.php/tutoriales-java-esencial/462-las-colecciones-en-java#:~:text=El%20API%20de%20Java%20nos,definidas%20en%20el%20paquete%20java>.
* Castillo, A. (2016) *Implementación de un sistema de ventas para mejorar la gestión comercial en la empresa Marecast S.R.L., Los Olivos.* Recuperado de <https://repositorio.uch.edu.pe/handle/20.500.12872/90>
* Belmonte, O. (2004) *Introducción al lenguaje de programación Java.* Recuperado de<https://www.academia.edu/download/58971457/Introduccion_a_Java20190420-21890-16e1jjg.pdf>
* Trigas, M. (2012) Recuperado de <http://openaccess.uoc.edu/webapps/o2/bitstream/10609/17885/1/mtrigasTFC0612memoria.pdf>