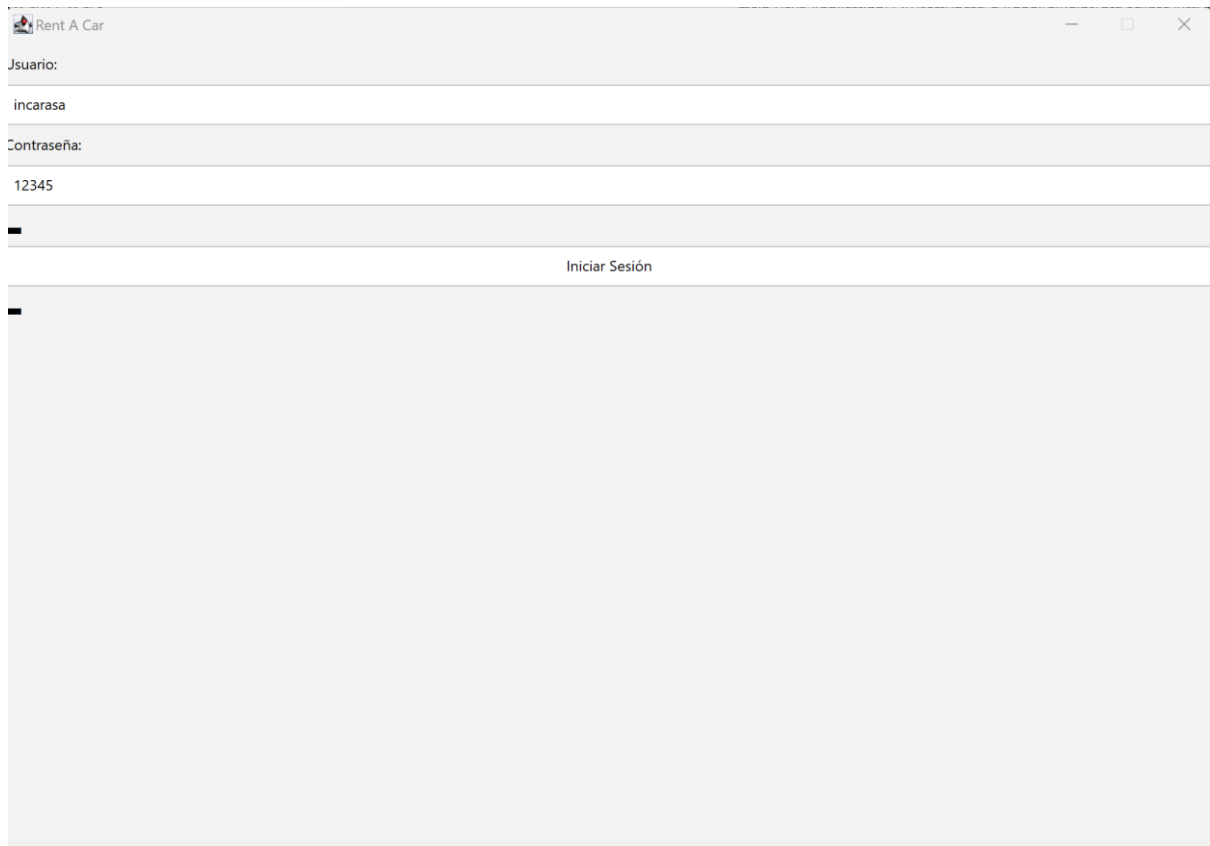


Documento de Diseño: Rent a Car

La aplicación Rent A Car se divide en 4 secciones, cada una con funcionalidades específica para cada uno de los usuarios (cliente, empleado, administrador, administrador principal). Se mostrarán a continuación las funciones de cada uno de estos.

Cliente:



The image shows a web browser window titled "Rent A Car". Inside the window, there is a login form with the following elements:

- A label "Usuario:" followed by a text input field containing the text "incarasa".
- A label "Contraseña:" followed by a text input field containing the text "12345".
- A button labeled "Iniciar Sesión" located below the password field.

The form is styled with a light gray background and white input fields. The browser window has standard OS controls (minimize, maximize, close) in the top right corner.

Imagen 1: Interfaz de inicio de sesión. Se ingresan los datos de cliente1.

Lo primero que se encuentra cualquier usuario es una interfaz de inicio de sesión como la que se observa en la imagen 1. Una vez se inicia la sesión se abre una ventana que le permite a un cliente reservar un vehículo.

Cliente

Reserva tu vehículo

Tipo vehículo:

Carro

Categoría del vehículo

A

Recogida

Sede de Recogida

NORMANDIA

Fecha de recogida (dd/mm/aaaa)

Hora de recogida (hh:mm)

Devolución

Sede de Devolución

NORMANDIA

Fecha de entrega (dd/mm/aaaa)

Hora de entrega (hh:mm)

Cancelar

Reservar

Imagen 2: Interfaz cliente.

La interfaz de cliente le permite a un cliente reservar un vehículo. Para esto el cliente debe ingresar la categoría del vehículo y la sede de recogida y de devolución junto con la respectiva hora. El sistema revisará si las sedes se encuentran abiertas en la fecha y la hora seleccionadas. En caso de no ser así se mostrará un mensaje como el que se observa en la siguiente imagen.

Fecha de entrega (dd/mm/aaaa)

11/12/2023

Hora de entrega (hh:mm)

23:00

La sede de devolucion no está abierta en la hora seleccionada

Cancelar

Reservar

Imagen 3: La sede está cerrada

Otro posible problema que maneja la interfaz es que el usuario ingrese una hora inválida. Esto se muestra en el mensaje a continuación.

10/12.2020
Hora de recogida (hh:mm)
23:00
Devolución
Sede de Devolución
NORMANDIA
Fecha de entrega (dd/mm/aaaa)
Hora de entrega (hh:mm)
23:00
Ha ingresado la fecha sin el backslash /
Cancelar
Reservar

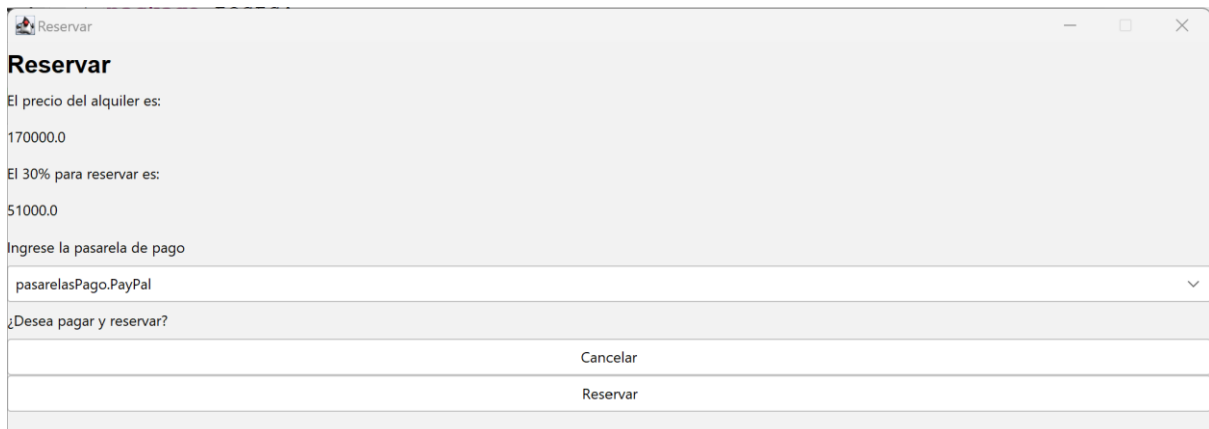
Imagen 4: Fecha con un formato inválido

En caso de no haber vehículos disponibles (pueden estar alquilados todos los de la sede en la categoría seleccionada o superior) se mostrará el siguiente mensaje:

12:00
No hay carros disponibles con las características seleccionadas
Cancelar
Reservar

Imagen 5: No hay vehículos disponibles

En caso de que si haya un vehículo disponible con las características seleccionadas se abrirá una nueva ventana que le permitirá al usuario reservar su vehículo. Esta ventana calculará el precio de la reserva que es el 30% y le permite al usuario la opción de reservar el vehículo. También se incluyeron las pasarelas de pago para que el cliente seleccione con cual de estas desea realizar el pago de la reserva.



Reservar

El precio del alquiler es:
170000.0

El 30% para reservar es:
51000.0

Ingrese la pasarela de pago

pasarelasPago.PayPal

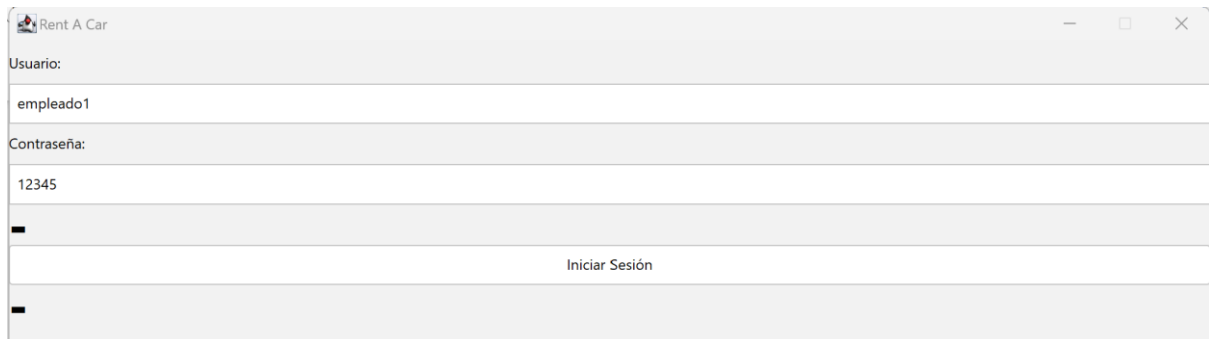
¿Desea pagar y reservar?

Cancelar

Reservar

Imagen 6: Confirmar reserva

Empleado:



Rent A Car

Usuario:
empleado1

Contraseña:
12345

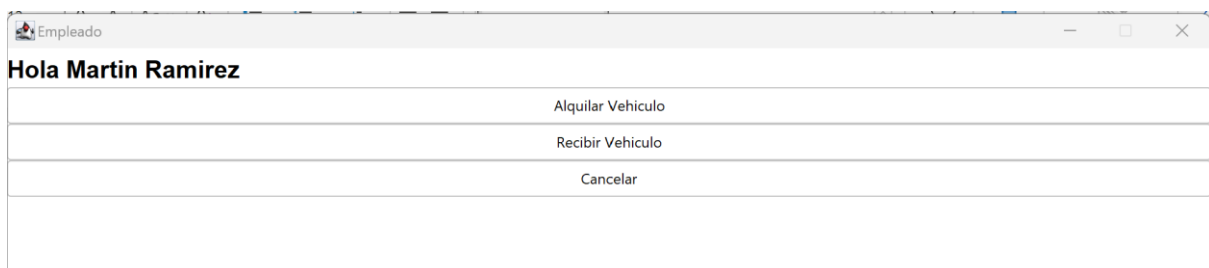
■

Iniciar Sesión

■

Imagen 7: Inicio sesión empleado

Al iniciar sesión como un empleado se muestra la interfaz que le permite al trabajador alquilar un vehículo o recibir un vehículo que se encontraba alquilado.



Empleado

Hola Martin Ramirez

Alquilar Vehiculo

Recibir Vehiculo

Cancelar

Imagen 8: Interfaz empleado

Si el empleado desea alquilar un vehículo se abrirá la interfaz para alquilar vehículos (mostrada abajo).



Imagen 9: Ventana alquilar vehículo

Para esto el empleado deberá ingresar la cedula del cliente y presionar el botón consultar. Aquí pueden ocurrir tres cosas diferentes. Si el cliente no existe, se activará el botón crear cliente para que el empleado pueda registrar un nuevo cliente en el sistema y así que este pueda alquilar un vehículo.

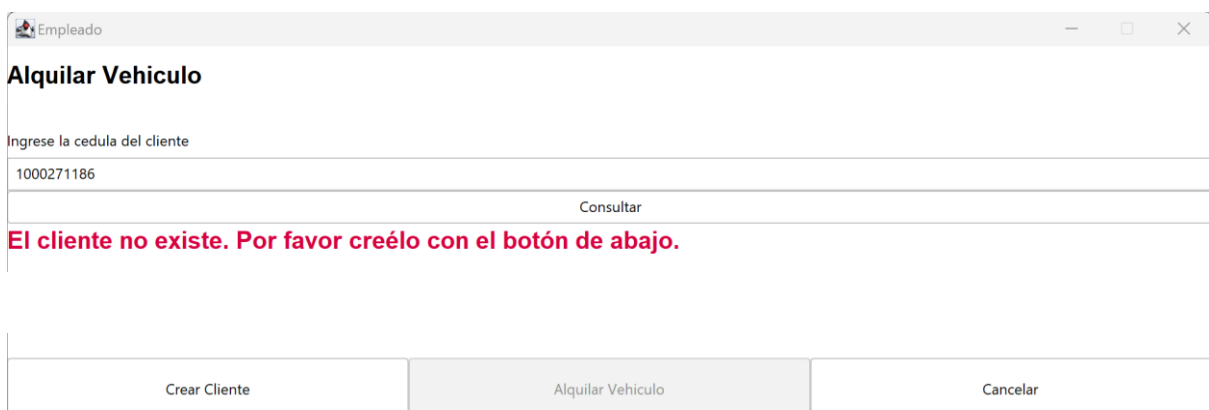


Imagen 10: Se activa el botón crear un cliente

Si el empleado presiona la opción crear cliente, se abrirá una ventana nueva para que este pueda registrar al nuevo cliente.

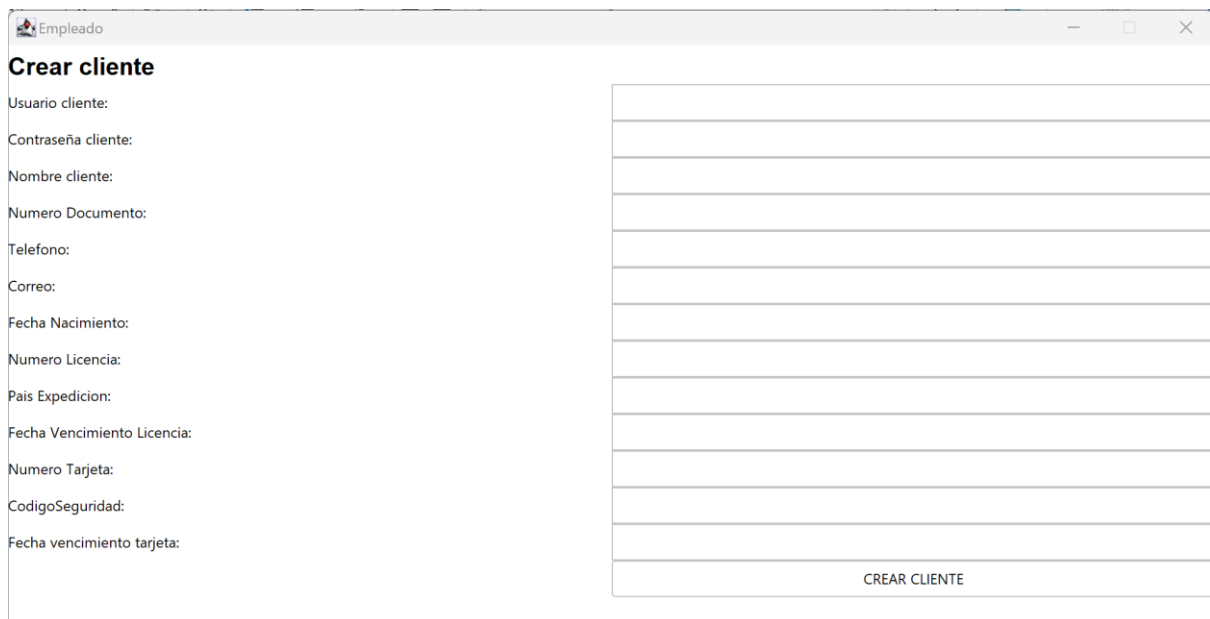


Imagen 11: Ventana crear cliente

Una vez se llenan los datos y se presiona crear cliente el nuevo cliente hará parte del sistema y podrá iniciar sesión para hacer reservas. Por otra parte, si el cliente existe en el sistema, pero no ha reservado un vehículo para el día en que se está consultado el sistema este indicará eso con un mensaje y habilitará la opción de alquilar vehículo.

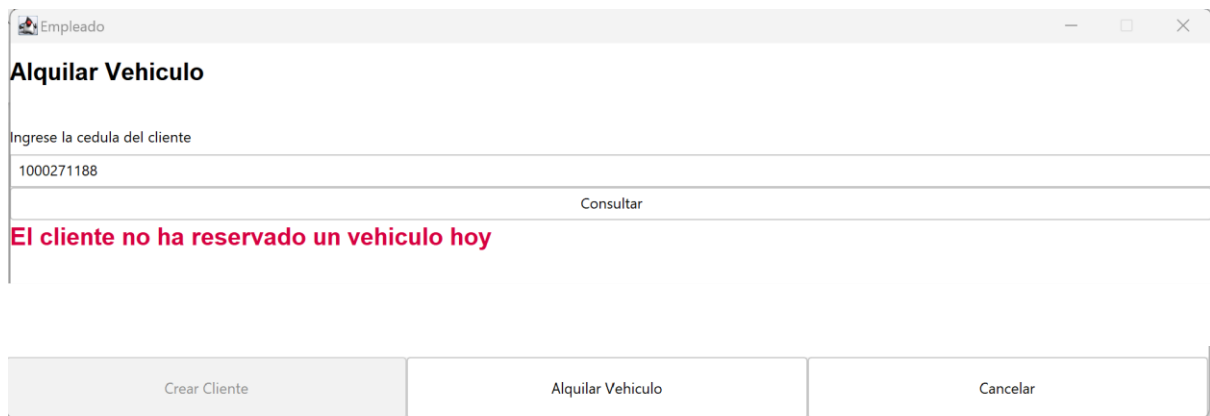


Imagen 12: El cliente no tiene reserva

Si el cliente no tiene una reserva para el día en que el empleado consulta el sistema el empleado entonces deberá crear una reserva que inicie ese mismo día hasta el día que el cliente desee alquilar el carro, para así verificar si hay vehículos disponibles para alquilar. Es por este motivo que al presionar el botón de Alquilar Vehículo se abrirá una ventana para que el empleado haga una reserva a nombre del cliente.

The screenshot shows a window titled 'Reserva tu vehículo' with a sidebar labeled 'Cliente'. The main content area contains the following fields:

- Categoría del vehículo:** A dropdown menu with 'A' selected.
- Recogida:**
 - Sede de Recogida:** A dropdown menu with 'NORMANDIA' selected.
 - Fecha de recogida (dd/mm/aaaa):** An empty text input field.
 - Hora de recogida (hh:mm):** An empty text input field.
- Devolución:**
 - Sede de Devolución:** A dropdown menu with 'NORMANDIA' selected.
 - Fecha de entrega (dd/mm/aaaa):** An empty text input field.
 - Hora de entrega (hh:mm):** An empty text input field.
- Buttons:** 'Cancelar' and 'Reservar' buttons at the bottom right.

Imagen 13: Se abre la ventana para reservar primero

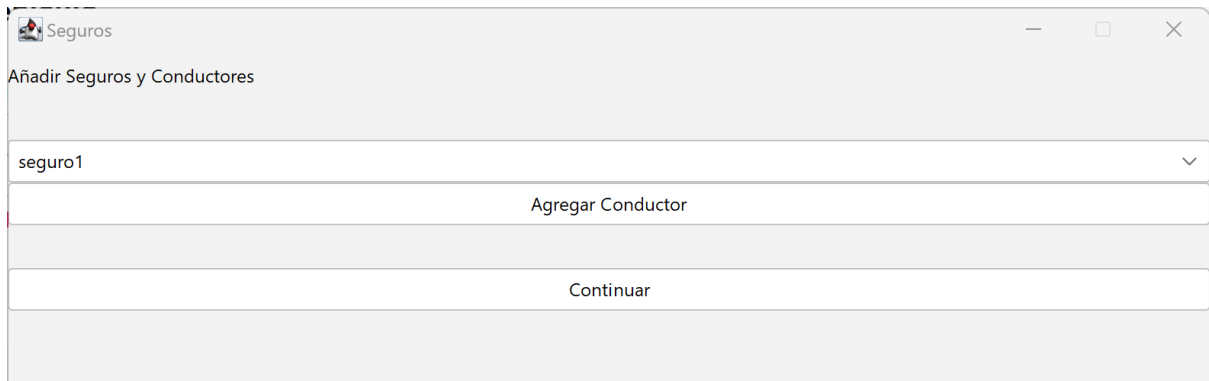
Finalmente, una vez ya exista una reserva a nombre del cliente que desea realizar el alquiler en el día en que el empleado consulte el sistema, se abrirá automáticamente una ventana para continuar con el proceso de alquiler.

The screenshot shows a window titled 'Opciones Alquiler' with a sidebar labeled 'Alquilar vehículo'. The main content area contains the following fields:

- Sede de Devolución:** A dropdown menu with 'NORMANDIA' selected.
- Fecha de entrega (dd/mm/aaaa):** An empty text input field.
- Buttons:** A 'Continuar' button at the bottom center.

Imagen 14: Se abre la ventana para alquilar si hay una reserva ese día

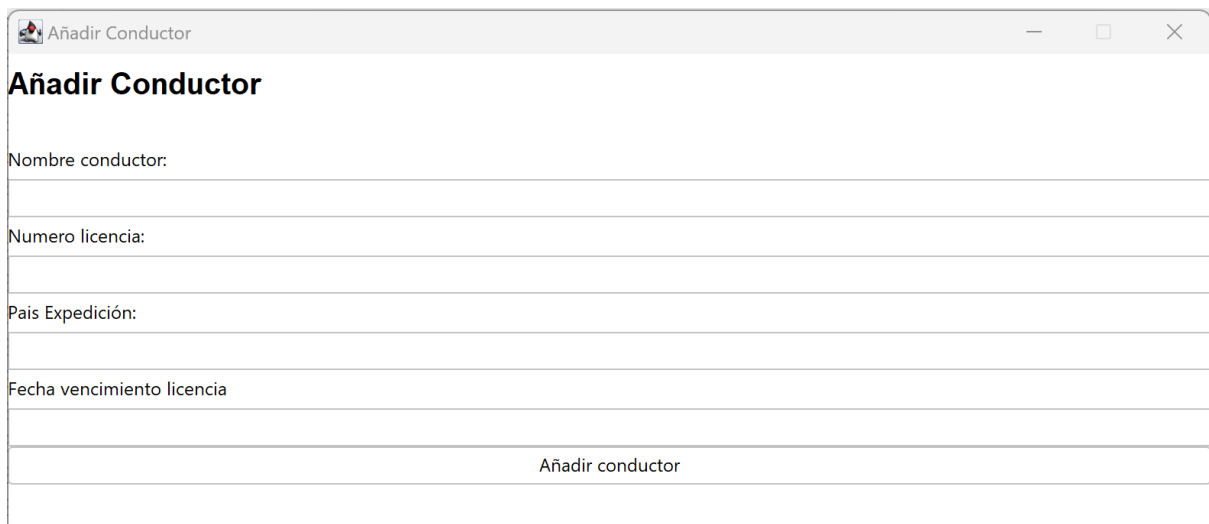
Esta ventana que se abre ÚNICAMENTE si existe una reserva para el día en que se consulta el sistema, permite concretar los detalles finales del alquiler, como la sede en donde se entregará el vehículo y la fecha en la cual se entregará.



The screenshot shows a window titled 'Seguros' with a subtitle 'Añadir Seguros y Conductores'. It features a dropdown menu with 'seguro1' selected, a button labeled 'Agregar Conductor', and a button labeled 'Continuar'.

Imagen 15: Añadir seguros y conductores

Una vez se ajusta la fecha y la sede se abre una nueva ventana para añadir seguros y conductores. Aquí se puede seleccionar un seguro de los disponibles por la empresa y añadir conductores adicionales que manejen el vehículo.



The screenshot shows a window titled 'Añadir Conductor'. It contains several input fields for 'Nombre conductor:', 'Numero licencia:', 'Pais Expedición:', and 'Fecha vencimiento licencia'. At the bottom, there is a button labeled 'Añadir conductor'.

Imagen 16: Ventana añadir conductor

Si se desean añadir conductores adicionales, se abrirá una ventana como esta en la que se debe llenar la información de los conductores adicionales (se pueden añadir el numero de conductores adicionales que se deseé).

Una vez se halla terminado de hacer esto se abre una nueva ventana que indica el precio del alquiler que deberá ser cancelado por el cliente.

Confirmar alquiler

Reservar

El precio del alquiler es:

540000.0

- el 30%:

69000.0

El valor final del alquiler es:

471000.0

Ingrese la pasarela de pago

pasarelasPago.PayPal

¿Desea pagar y alquilar el vehículo?

Alquilar

Imagen 17: Confirmar alquiler

Como se observa en la imagen se muestra el costo total menos el 30% que el cliente debió haber cancelado cuando realizó la reservación del vehículo. En caso de que el cliente no haya hecho reservación el empleado cobrará el total del alquiler en este punto.

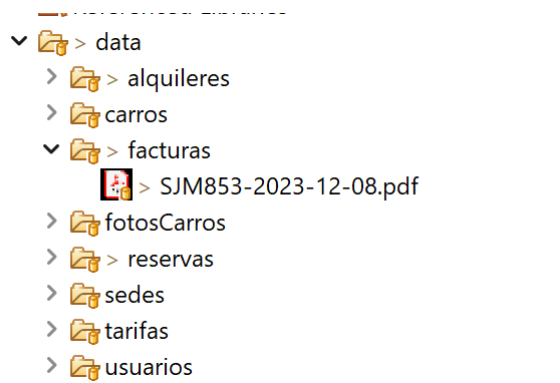


Imagen 18: Ubicación de la nueva factura

Una vez se realiza el alquiler se generará una factura con la identificación del vehículo y la fecha en que se realizó el alquiler.

Recibo de alquiler

Fecha: 2023-12-08

Información Cliente

Nombre
Raul Insuasty
Documento
1000271188
Telefono
3142715800
Correo
rasanicw@gmail.com

Producto	Días	Precio
Vehiculo de categoria B	3	180000
seguro2	3	45000
Cond. adicionales xl		50000
TOTAL		275000

Imagen 19: Ejemplo del recibo

La segunda función del empleado es recibir vehículos por este motivo si se selecciona la opción devolver vehículo se desplegará el siguiente menú en la ventana.

Ingrese la placa del vehículo:

☐ Lavar
☐ Mantenimiento
Fecha disponible nuevamente

Imagen 20: Recibir vehículo

Aquí el empleado debe ingresar la placa del vehículo, seleccionar si el vehículo se lavará o estará en mantenimiento y finalmente poner la fecha en que el vehículo estará disponible nuevamente.

Administrador Sede:

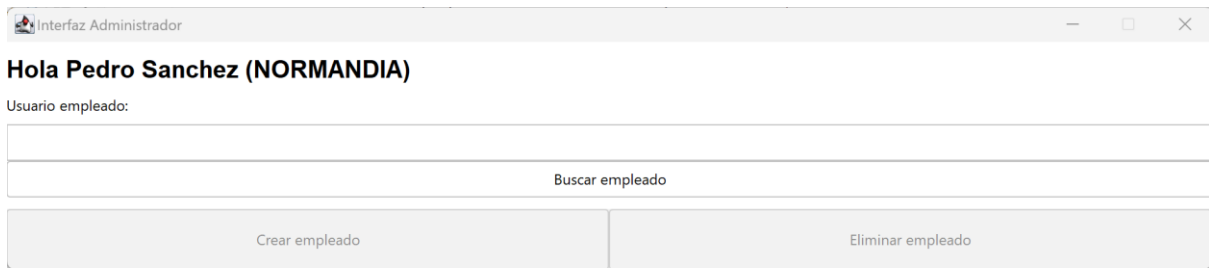
Rent A Car

Usuario:

Contraseña:

Imagen 21: Interfaz administrador

Al iniciar sesión como un administrador de sede se mostrará la siguiente interfaz.



Interfaz Administrador

Hola Pedro Sanchez (NORMANDIA)

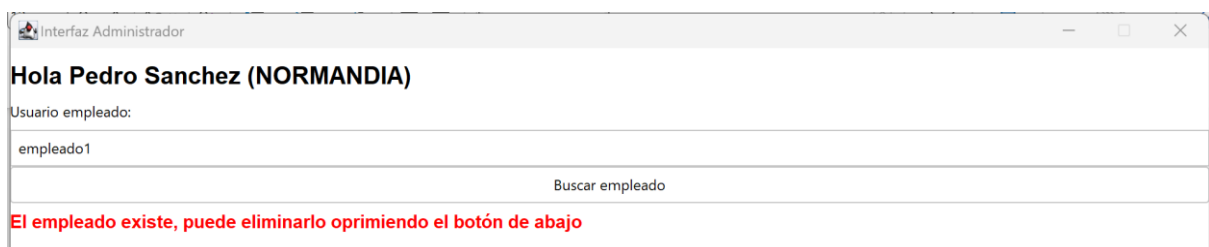
Usuario empleado:

Buscar empleado

Crear empleado Eliminar empleado

Imagen 22: Buscar empleado por usuario

Acá el administrador de la sede puede crear o eliminar empleados de su sede, para hacer esto debe ingresar el documento del empleado, si este existe se habilitará la opción de eliminar el empleado.



Interfaz Administrador

Hola Pedro Sanchez (NORMANDIA)


Usuario empleado:

Buscar empleado

El empleado existe, puede eliminarlo oprimiendo el botón de abajo

Imagen 23: Un empleado existe

Por el contrario, si el empleado no existe se desplegará un menú para que el administrador de la sede pueda crear este nuevo empleado.



Interfaz Administrador

Hola Pedro Sanchez (NORMANDIA)

Usuario nuevo:

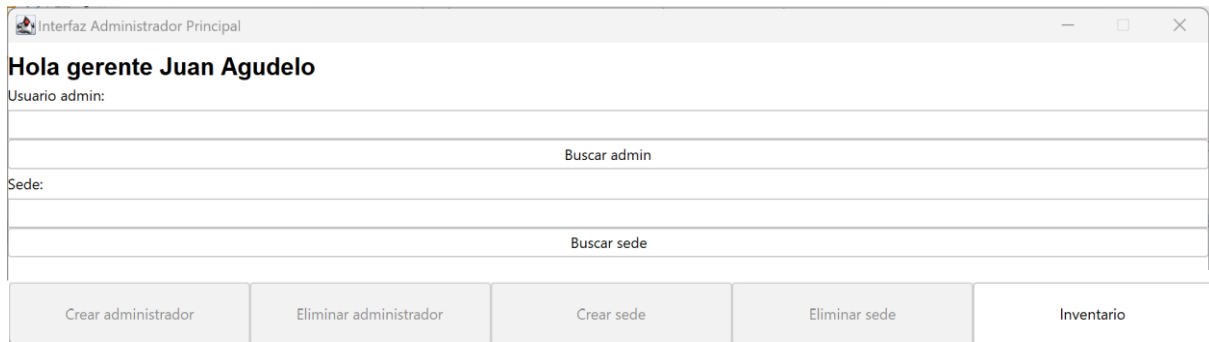
Contraseña:

Nombre empleado:

Crear

Imagen 24: Crear un empleado si no existe

Administrador Principal:



Interfaz Administrador Principal

Hola gerente Juan Agudelo

Usuario admin:

Buscar admin

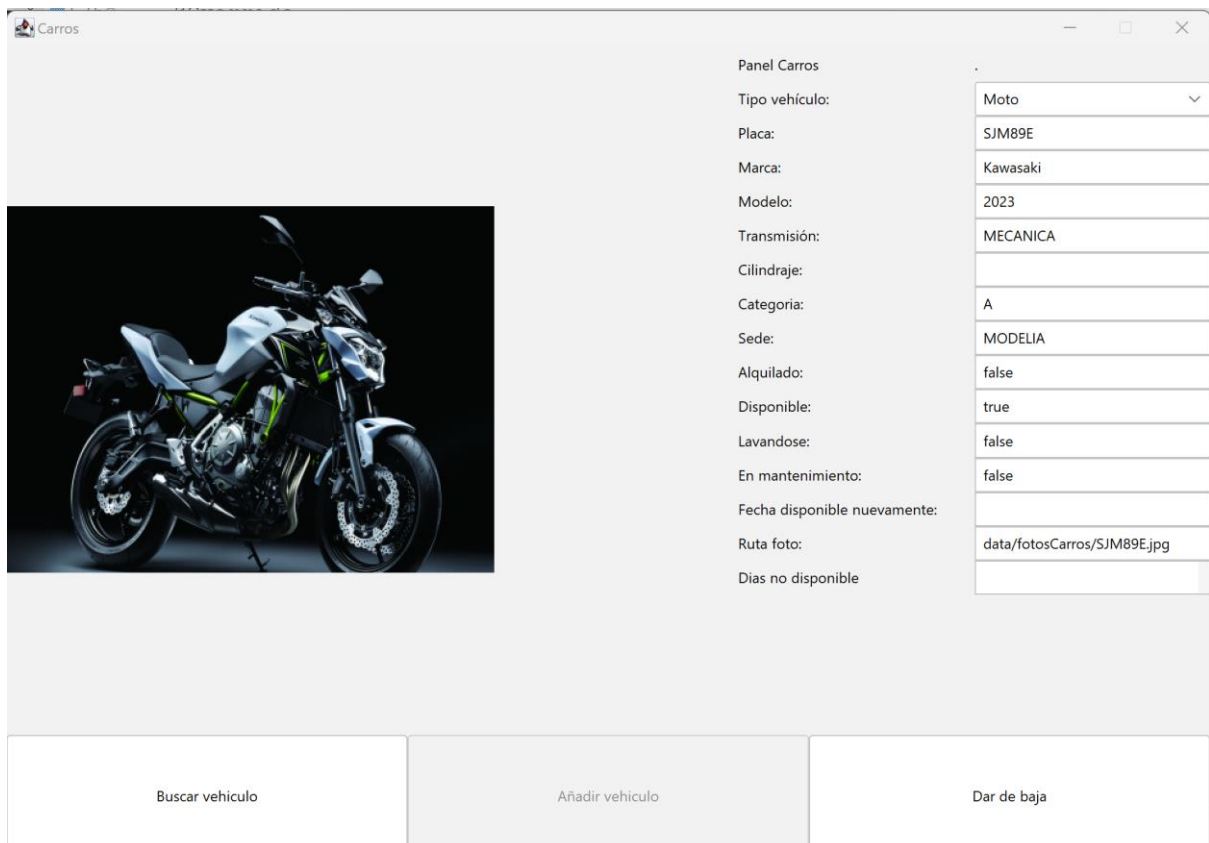
Sede:

Buscar sede

Crear administrador Eliminar administrador Crear sede Eliminar sede Inventario

Imagen 25: Interfaz administrador principal

La interfaz del administrador principal funciona de forma parecida al la de un administrador de sede, en el sentido que puede añadir sedes y administradores y eliminarlos de una forma muy similar a la que ocurría con la interfaz del administrador de sede, no obstante, hay una funcionalidad interesante que diferencia a este administrador del resto y es el inventario.



Carros

Panel Carros

Tipo vehículo: Moto

Placa: SJM89E

Marca: Kawasaki

Modelo: 2023

Transmisión: MECANICA

Cilindraje:

Categoría: A

Sede: MODELIA

Alquilado: false

Disponible: true

Lavandose: false

En mantenimiento: false

Fecha disponible nuevamente:

Ruta foto: data/fotosCarros/SJM89E.jpg

Dias no disponible

Buscar vehiculo Añadir vehiculo Dar de baja

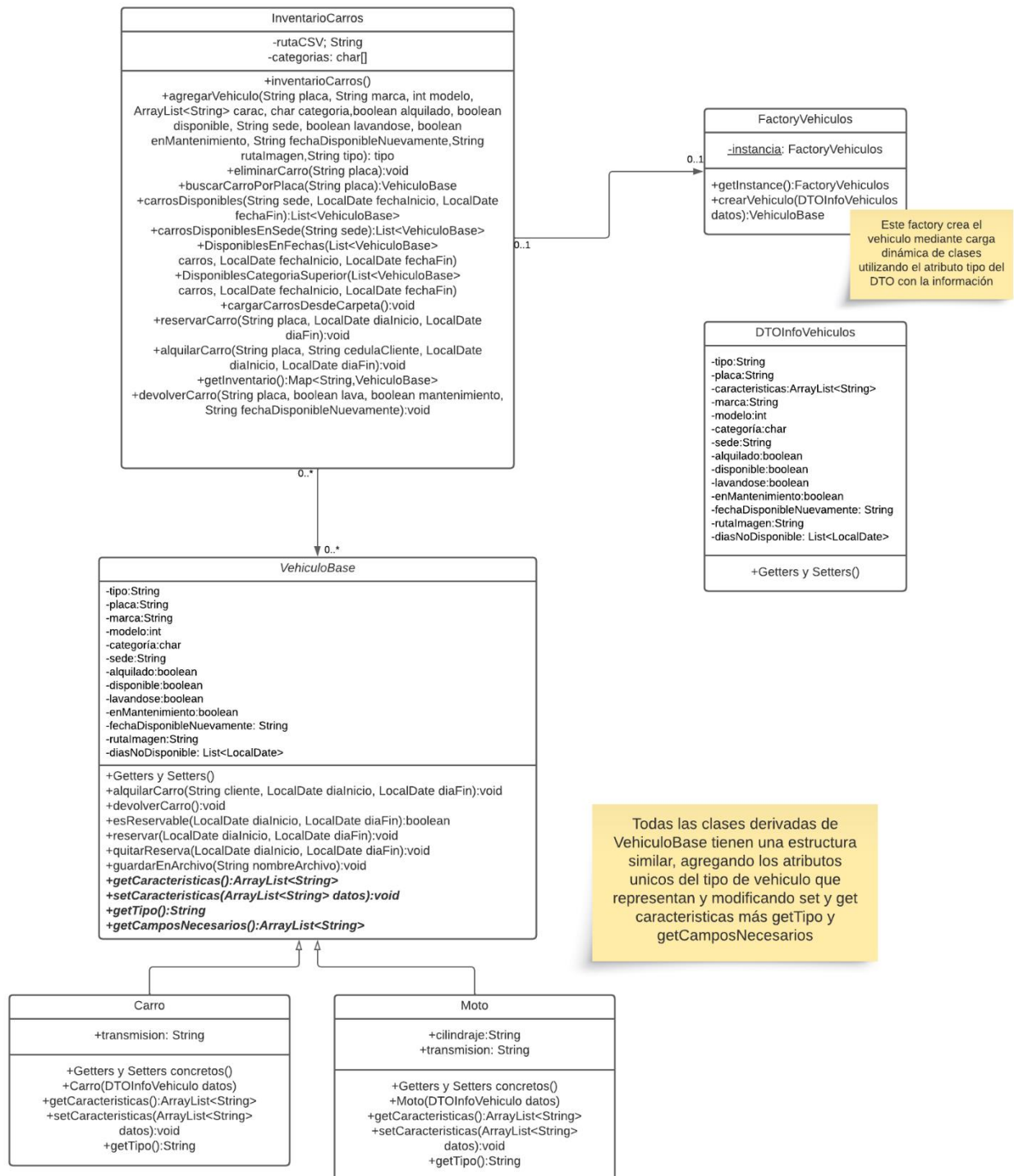
Imagen 26: Ventana inventario

En el inventario el administrador principal podrá entonces buscar un carro por su placa o en caso de no existir, llenar los campos necesarios para crear el vehículo.

Cambios en el funcionamiento interno para el proyecto 3

Gestión del inventario de vehículos:

Con el proyecto 3 se agregó el requisito de que la aplicación pueda extender los tipos de vehículos que maneja y se adapte a los disponibles. Para cumplir este requisito utilizamos un patrón factory junto a una carga dinámica de clases, para que al intentar agregar nuevos tipos solo sea necesario crear una clase que herede de VehiculoBase, llenar los métodos necesarios (relacionados principalmente a las características adicionales que pueda tener el tipo de vehículo) y actualizar un archivo txt (data/tipos/tipos.txt) que tiene los nombres de las clases.



Para esto manejamos una clase principal, VehiculoBase, de la cual heredarán todos los tipos concretos de vehículos y que tiene los atributos y métodos necesarios para manipular los vehículos en las reservas sin problema, porque son características que comparten todos los vehículos sin importar su tipo. A partir de esta clase heredan los demás tipos de vehículos, como ejemplos para la implementación, utilizamos Moto y Carro, que ambos se diferencian en que para carro solo manejamos una característica

adicional, que es la transmisión y para moto utilizamos cilindraje y transmisión, pero a pesar de esto, la carga de datos y el resto del programa los identifica como VehiculoBase, a excepción del factory, que busca la clase concreta y su constructor para generar la instancia a partir de un DTO (DTOInfoVehiculos)

```
@SuppressWarnings({ "unchecked", "rawtypes" })
public VehiculoBase crearVehiculo(DTOInfoVehiculo datos) throws ClassNotFoundException, InstantiationException,
,Exception
{
    VehiculoBase retorno = null;
    Class vehiculo = Class.forName("Inventario."+ datos.getTipo());
    Class dto = DTOInfoVehiculo.class;
    retorno = (VehiculoBase) vehiculo.getDeclaredConstructor(dto).newInstance(datos);

    return retorno;
}
```

Entonces, con este método el factory retorna la nueva instancia según el tipo de vehículo.

¿Como funcionan las características concretas de cada tipo?

Al momento de llamar el DTO y pasarlo al factory, cargamos un ArrayList de Strings en el DTO, el cual posee las características que se deben asignar en cierto orden, que depende de la implementación de cada clase en concreto. Por este motivo, es fácilmente moldeable para que tenga varios strings de información o incluso alguna dirección para que lea un archivo, todo dependiendo de la implementación que desee utilizar la persona que deba extender el código. A continuación, ejemplos de la implementación en Moto:

```
@Override
public void setCaracterísticas(ArrayList<String> datos) {
    this.cilindraje = datos.get(0);
    this.transmision = datos.get(1);
}
```

```
@Override
public ArrayList<String> getCaracterísticas() {
    ArrayList<String> datos = new ArrayList<String>();
    datos.add(cilindraje);
    datos.add(transmision);

    return datos;
}
```

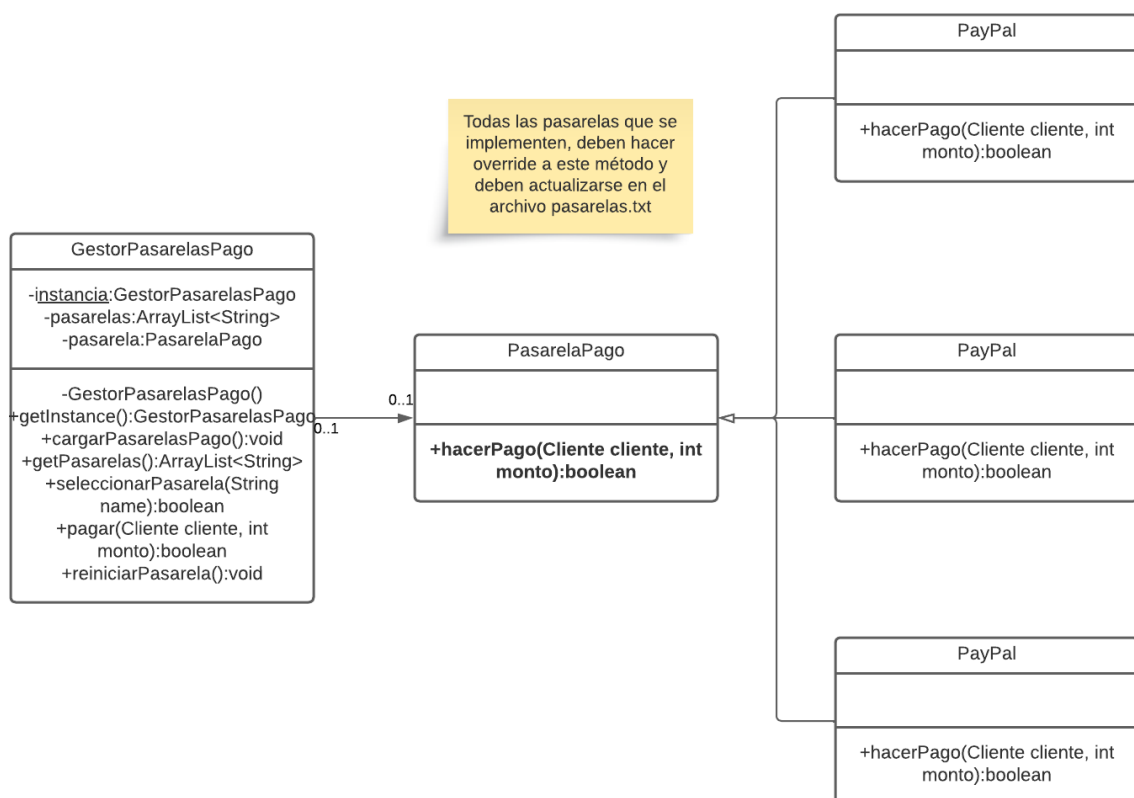
Gestión de errores de inventario:

La gestión de errores en las partes relacionadas al cargar datos del inventario y guardarlos son sencillas. Al momento de tratar de crear un nuevo vehículo con un tipo inexistente (puesto que no exista la clase), no lo crea y retorna una excepción la cual hace que se le notifique al usuario de que no es un tipo correcto (algo que no debería pasar porque al crear un nuevo vehículo, limitamos a que tipos puede crearlo en la interfaz).

Respecto a cuando carga datos y si alguien modifico los archivos con información para cambiarle el tipo a un vehículo (el cual es el último campo de cada archivo de vehículo), no lo carga, retorna la excepción para notificarla en consola, pero el programa sigue corriendo sin mucho problema.

Pasarelas de Pagos:

Otro de los requerimientos del proyecto 3 es la implementación de pasarelas de pago como PayPal o PayU, para esto manejamos el siguiente esquema de clases:



En esta implementación utilizamos un patrón strategy mediante carga dinámica de clases. Para hacer la carga dinámica, GestorPasarelasPago (implementado como un singleton) lee un archivo (data/pasarelasPago/pasarelasPago.txt), donde lee que

pasarelas deberían estar y guarda esta información para suministrarla a las clases que la necesiten al llamarla. Entonces cuando se ejecuta el método seleccionarPasarela, elige una y la carga para guardarla como atributo, con el cual se puede llamar al método pagar, que lo tiene GestorPasarelaPagos y todas las pasarelas, que heredan de la clase PasarelaPago.

Respecto a cómo se ejecuta el pago, utilizamos un método local debido a que en realidad no están conectadas a las API de las pasarelas, por lo que, si un pago es exitoso o no, depende de la información de cliente, dos atributos de esta clase para ser exactos, bloqueado y cupo. Con bloqueado sabemos si la tarjeta del cliente está bloqueada (dentro del programa) y con cupo sabemos el cupo máximo para una compra con la misma.

Finalmente, retorna un booleano dependiendo de si el pago fue exitoso o no, o una excepción en dos casos particulares.

Gestion de errores pasarelas de pago:

Para gestionar los errores, utilizamos dos clases que heredan de Exception, TarjetaBloqueadaException y TarjetaSinCupoException, como sus nombres indican, una se retorna cuando la tarjeta está bloqueada y la otra cuando la tarjeta no tiene el cupo suficiente para el pago, el cómo se gestionan es recibiendo las en la interfaz e informando al cliente de la situación.

El Proceso durante los proyectos

Los Problemas:

Inicialmente, íbamos de forma tranquila al desarrollar el proyecto 1 ya que creíamos que no había mucha diferencia entre Java y Python, que era el lenguaje que habíamos manejado anteriormente. Nos fue bien abordando el primer proyecto en su mayoría, excepto por un integrante que no hizo mucho, desconocemos si fue porque no sabía mucho del lenguaje o porque no quiso. Además de este, que era el gran problema, también tuvimos otros problemas menores al desarrollar el código, principalmente cosas como utilizar el “==” o el “!=” en lugar de equals() en varios casos y el más importante, que fue no darnos cuenta del acoplamiento de nuestro diseño y que nos traería grandes problemas en el siguiente proyecto.

En el segundo proyecto, el problema de nuestro compañero se agravó, porque directamente no participó y a esto se le sumó que tuvimos que corregir y disminuir el acoplamiento de nuestro diseño anterior para poder adaptarlo a los posibles nuevos requerimientos del proyecto 3 y diseñar desde cero la interfaz gráfica. La implementación de la interfaz fue difícil porque éramos menos trabajando en algo que requería mucho tiempo para desarrollarse, sin contar el tiempo gastado en pruebas para verificar que todas las ventanas y paneles se renderizaran de forma adecuada.

Finalmente, para el proyecto 3 no tuvimos tantos problemas, ocurrió la reestructuración del grupo y finalmente pudimos trabajar adecuadamente. Gracias a los cambios que hicimos durante el proyecto 2, fue más fácil adaptar el código ya existente para facilitar la extensión de este, que era el objetivo del proyecto 3. A pesar de esto, para este momento tan avanzado tuvimos un problema algo básico con Java, que fue incluir librerías para hacer varias cosas. Esto ocurrió principalmente con la librería itextpdf, que tiene una situación peculiar respecto a sus versiones y cuales son utilizables, pero que luego de un tiempo pudimos solucionar para poder generar las facturas del programa.

Las decisiones tomadas:

Respecto a las decisiones que tomamos al diseñar la aplicación, hubo varias que demostraron ser muy efectivas, como separar los vehículos en distintos archivos, separar adecuadamente las clases para gestionar la persistencia, utilizar una clase para conectar la interfaz con la lógica, entre otras. Decisiones que nos permitieron facilitar el modificar el código para agregar, quitar y modificar partes de este, pero también tuvimos decisiones que nos costaron tiempo porque no las pensamos bien en su tiempo, entre las cuales estaban el hacer tan específica la clase para los vehículos, porque gracias a esto tuvimos que cambiar una gran parte del programa para que funcionara con cualquier tipo de vehículo.

¿Qué salió bien y que salió mal?

En general, la mayoría de los aspectos del proyecto salieron bien, pero en cada entrega tuvimos uno u otro problema gracias al tiempo y/o falta de organización, motivos que no permitieron entregar un proyecto completo al 100%, faltando funcionalidades por implementar o documentación incompleta. Fuera de estos problemas, logramos la mayoría de los objetivos e implementamos una gran parte de

lo que aprendimos durante la clase, sobre todo en este último proyecto con el diseño de pruebas y el uso de patrones de diseño, que nos ayudaron a solucionar varios problemas de una forma más fácil.

¿Qué haríamos diferente si lo hiciéramos otra vez?

Inicialmente, buscaríamos implementar un diseño mucho más modular, utilizando desde el principio una mayor cantidad de interfaces y clases abstractas, que nos permitan agregar funcionalidades adicionales a los objetos que ya tenemos, pero sin gastar la cantidad de tiempo que utilizamos en implementar esto sobre un sistema ya creado. Otra cosa que cambiaríamos sería el desarrollo de la interfaz gráfica, etapa que podríamos hacer en un menor tiempo y con un resultado más eficaz y adaptable gracias a los conocimientos que adquirimos. Pasa algo similar con la inclusión de librerías y las pruebas unitarias, las incluiríamos desde el principio no solo por ahorrar tiempo en las últimas etapas, sino también por las utilidades que traen para el desarrollo (principalmente por la utilidad de las pruebas unitarias para verificar rápidamente aspectos del código).