Trabajo Práctico Final

Taller de Programación I

Profesores:
Leonel Guccione
Adolfo Tomás Spinelli
Guille Lazurri

Regina Entrocassi
Rocío Gigliotti
Paloma de Aza
Tomás Salig

Fecha de entrega: 13/11/2024

Facultad de Ingeniería
U.N.M.d.P

Índice

Índice 2
Introducción 2
Testeo de caja negra 3
Testeo de persistencia 4
Test de integración 4
Diagramas de secuencia del test de integración 5
Testeo de GUI 14
Lista de errores 19
Conclusión 22

Introducción

En el siguiente informe se testeara un sistema y se indicarán los resultados de:

- Caja negra
- Test de integración
- Test de persistencia
- Test de GUI

El test de caja negra busca verificar la funcionalidad de un sistema sin conocer los detalles internos de su implementación, es decir, que evalúa el comportamiento externo del software sin tener acceso al código fuente. En este trabajo se testean las clases:

- en modelo de datos: Administrador, Auto, Chofer, ChoferTemporario, ChoferPermanente, Cliente, Combi, Moto, Pedido, Usuario, Vehiculo, Viaje.
- en modelo de negocio: Empresa.

El test de persistencia se centra en verificar la capacidad del sistema para almacenar y recuperar datos de manera correcta y confiable a lo largo del tiempo, es decir, que este tipo de prueba se enfoca en asegurar que los datos persisten de manera duradera en el sistema, incluso después de reinicios, actualizaciones o fallos. En esta fase del trabajo se testean las clases:

- EmpresaDTO, UtilPersistencia, PersistenciaBIN.

El test de integración es la fase en la que se combinan y prueban los diferentes módulos o componentes del sistema para asegurar que funcionan juntos de manera correcta y coherente. Busca detectar posibles problemas de interacción entre las partes del sistema y garantizar que la integración de esos componentes cumpla con los requisitos establecidos.

El test de GUI se centra en evaluar la usabilidad, la apariencia y la funcionalidad de la interfaz visual del sistema. Este tipo de prueba se realiza para garantizar que la interfaz de usuario cumpla con los requisitos de diseño y usabilidad.
En este trabajo el test de GUI abarca :

- Panel login, panel Registro, panel Cliente, panel Administrador

Testeo de caja negra

IMPORTANTE
Las tablas de particiones y baterías de pruebas se encuentran detalladas en los excel del repositorio titulados: “modeloDatos” y “modeloNegocio”. Para la ejecución de las pruebas fue utilizado JUnit 4.
Decidimos subir las tablas de particiones y baterías de pruebas como excels en vez de agregarlas al documento ya que consideramos que se pueden leer más fácilmente en este formato, aparte de estar separadas por método y clase.

La realización de las pruebas de caja negra fue en base al javadoc provisto por la cátedra, cumpliendo el contrato de cada método para cada prueba.

Los dos paquetes principales a testear mediante el método de caja negra son el modelo de Datos y el modelo de Negocio.
Dentro del modelo de Datos, testeamos cada uno de los métodos y constructores de cada una de las clases. Si cualquier método lanzaba excepciones, se testearon todas ellas, al igual que sus atributos de lanzarlos y sus atributos de tenerlos.
Dentro del modelo de Negocio, la única clase a testear era Empresa. Para asegurarnos de testear todos los métodos y todos los caminos posibles de cada método (incluyendo las excepciones), creamos tres escenarios. Un primer escenario vacío, un segundo escenario lleno pero estático (creamos objetos clientes, choferes, etc, pero ningún pedido era generado ni tomado) y un tercer escenario lleno y dinámico (con los objetos de la empresa creados, pedidos siendo tomados, etc). Tan solo utilizando estos tres escenarios pudimos testear todos los métodos de la clase Empresa con todos los caminos posibles de cada método.

Testeo de persistencia

Para el testeo de persistencia, se utilizó la clase PersistenciaBIN para abrir y cerrar los inputs/outputs, además de leer y escribir.

Con respecto a la clase UtilPersistencia, se trata de la clase que traduce los elementos a persistir de un objeto Empresa a uno EmpresaDTO y

viceversa. El objetivo de estas pruebas es corroborar que puede copiar correctamente los elementos de una clase y pasarlo a otra.
Como Empresa es una clase que no se puede instanciar nuevamente, se utilizará el método Empresa.getInstance(). En ambos métodos las salidas esperadas eran que los objetos copiados sean idénticos a los originales y efectivamente así fue.

Test de integración

En el test de integración, el objetivo era probar que el controlador funcione correctamente. Se probaron las interacciones entre los métodos de controlador y los métodos de la capa de modelo, además de que los actionEvent y actionPerformed funcionen correctamente.
Para testear el método de actionPerformed

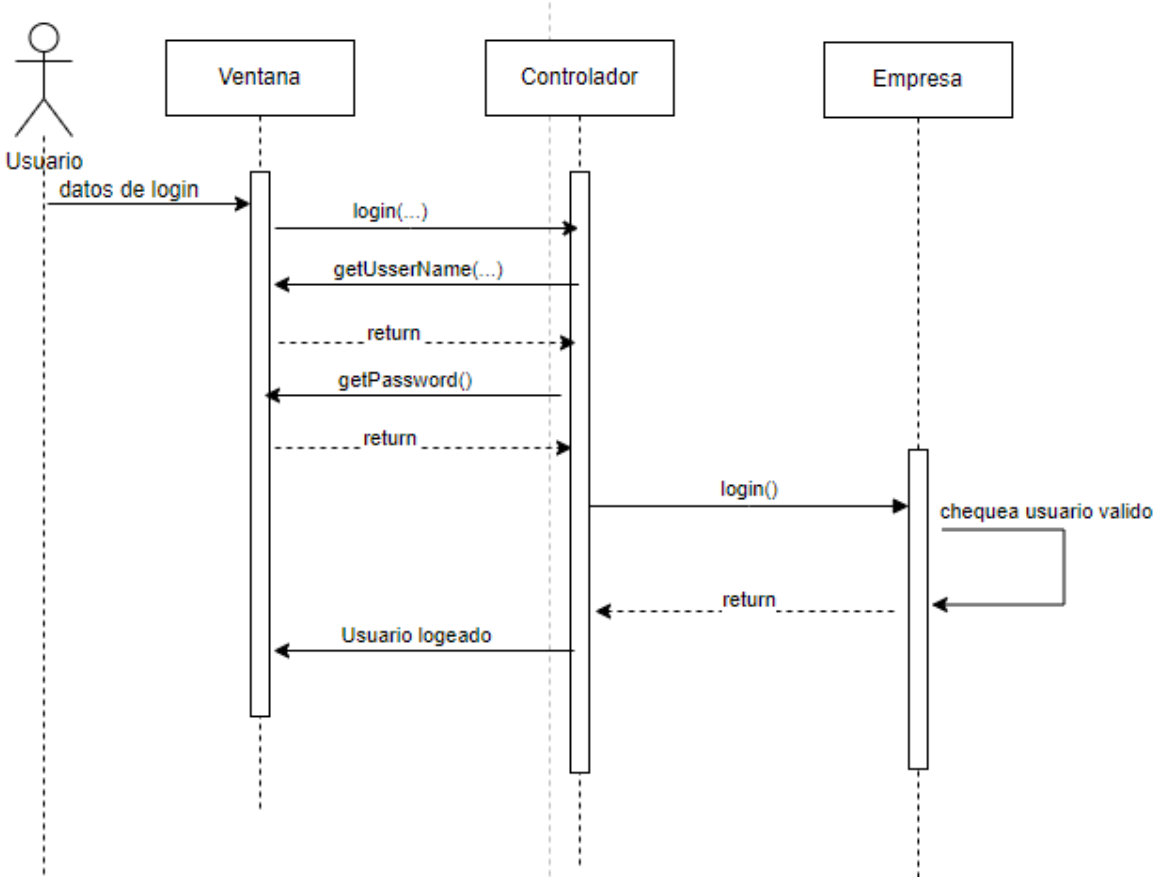
Para la realización de las pruebas de integración de otros métodos del controlador, se realizaron los siguientes casos de uso:

1. **login().** Donde se consideró el caso exitoso, el caso de un usuario no existente, y el caso de una contraseña incorrecta. En caja negra se testeo el logueo de un administrador y de un usuario normal.
2. **logout().** En este caso de uso no distinguimos entre usuario administrador y no administrador. Se prueba únicamente que el logout se haya realizado correctamente desde el controlador.
3. **registrar().** Se consideró un caso exitoso y un caso donde la confirmación de contraseña era incorrecta. Esta prueba no se realizó con el administrador, este no requiere registrarse en el sistema.
4. **nuevoPedido()**. Se probó un caso exitoso y uno donde no existe un vehículo apto para el pedido.
5. **calificarPagar().** Se probó un caso exitoso y un caso donde el cliente no tenía un viaje iniciado.
6. **nuevoVehiculo().** Se probó para cada tipo de vehículo, verificando que el controlador obtenga los valores iniciales correctos para cada uno de estos.
7. **nuevoChofer()**. Se realizaron pruebas para los dos tipos existentes de chofer, verificando que los datos ingresados con el uso de mocks coincidan con los datos de las instancias de chofer correspondientes.
8. **nuevoViaje()**. Se probó que el viaje se haya creado con el pedido, chofer y vehículo seleccionados. También se probó el caso donde no hay un vehículo disponible para el pedido.
9. **escribir() y leer()**. Se probaron las interacciones entre el controlador y la capa de persistencia.

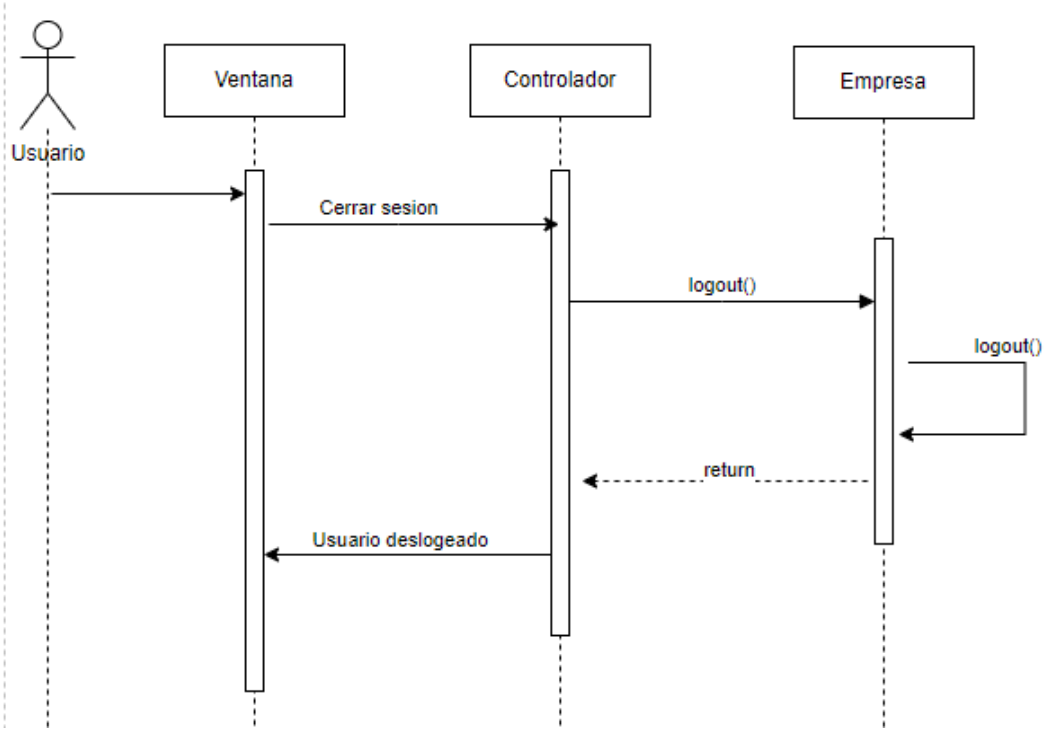
Diagramas de secuencia del test de integración

A continuación se pueden visualizar los diagramas de secuencia para estos métodos:

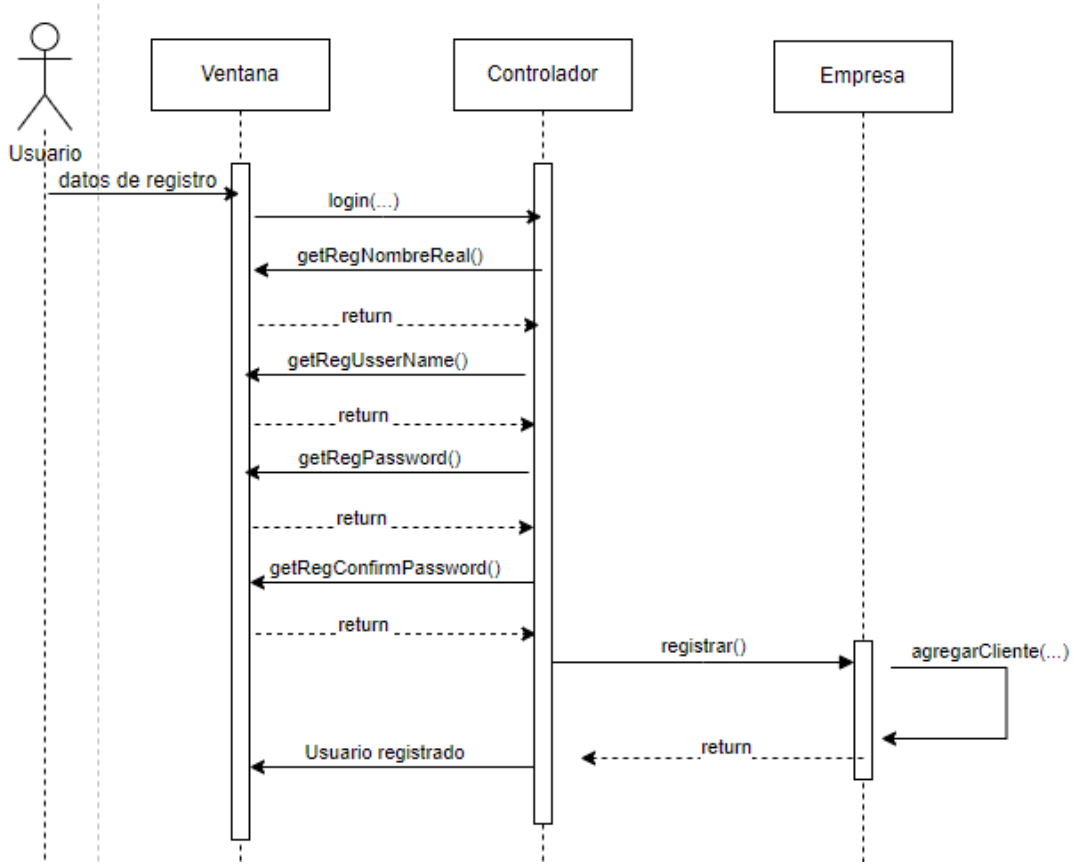
1. **`login()`**



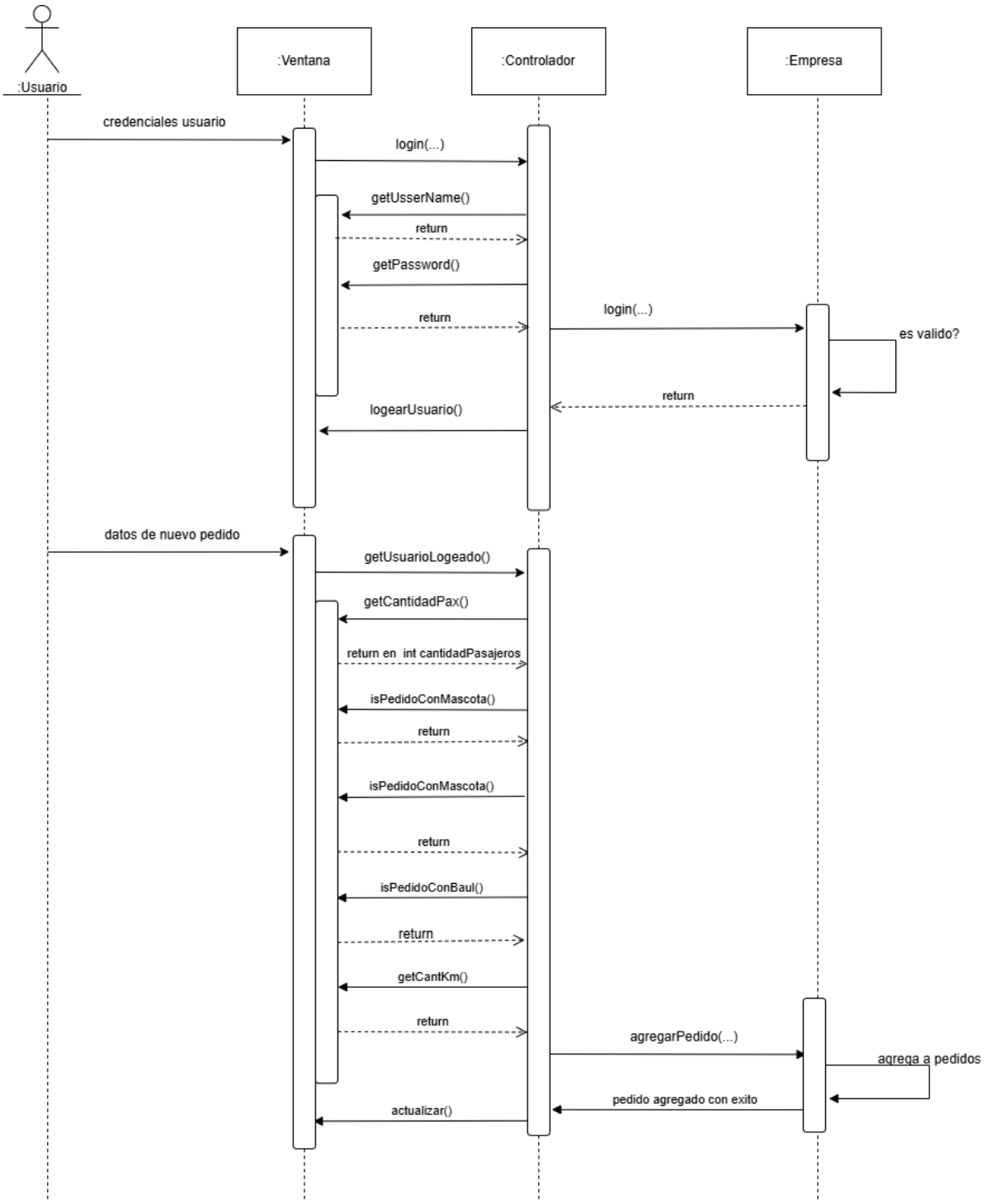
2. **`logout()`**



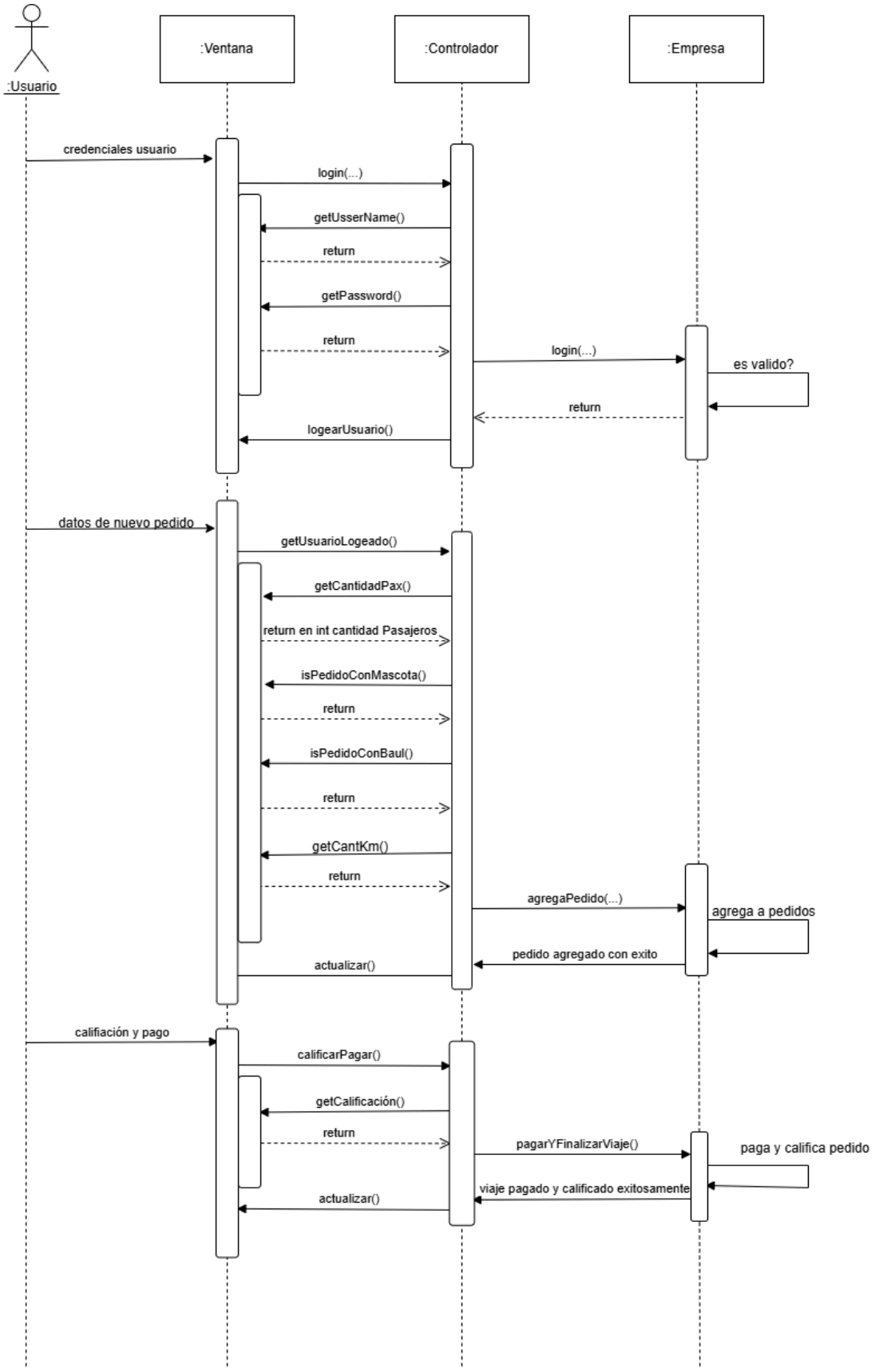
3. **`registrar()`**



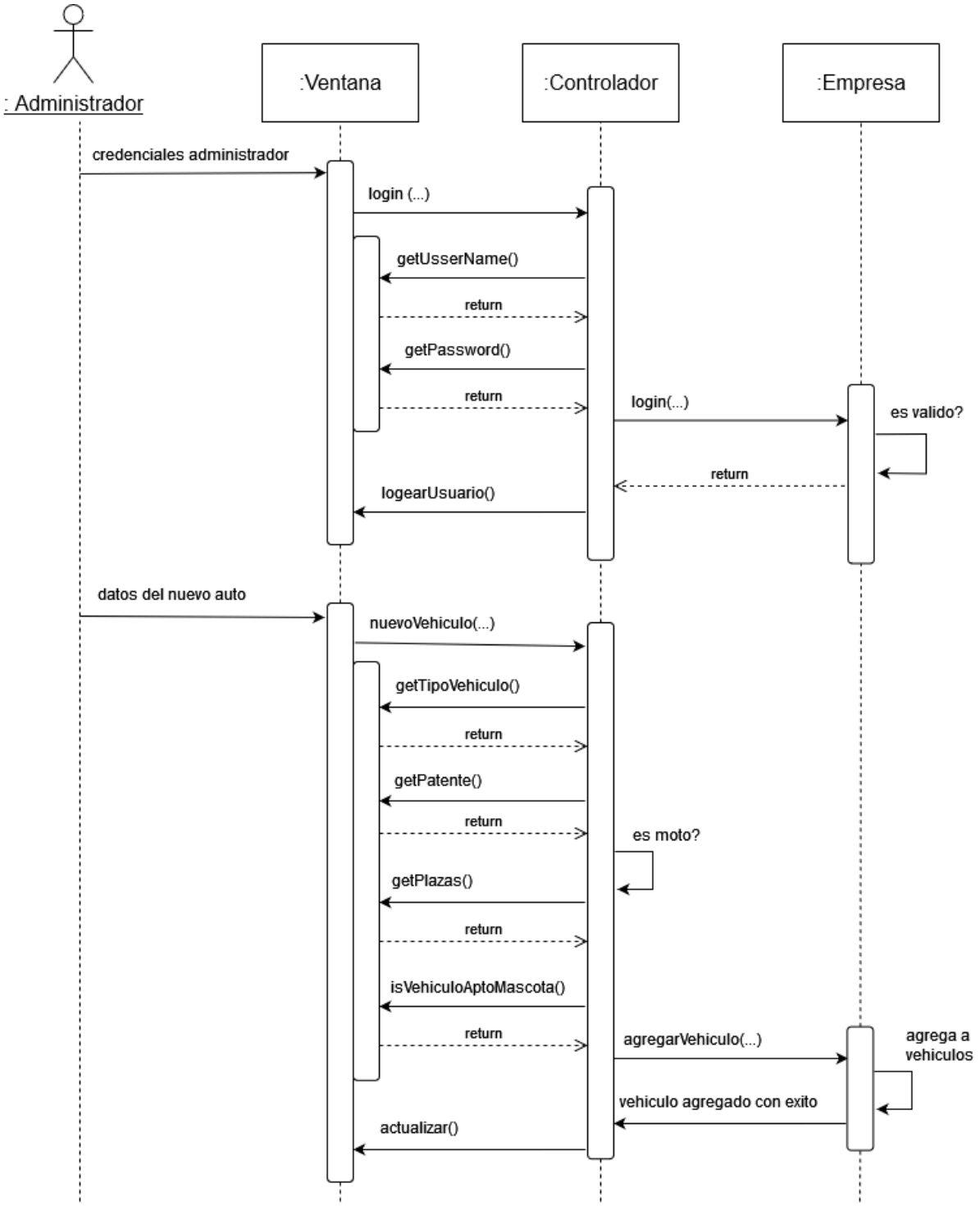
4. **nuevoPedido()**



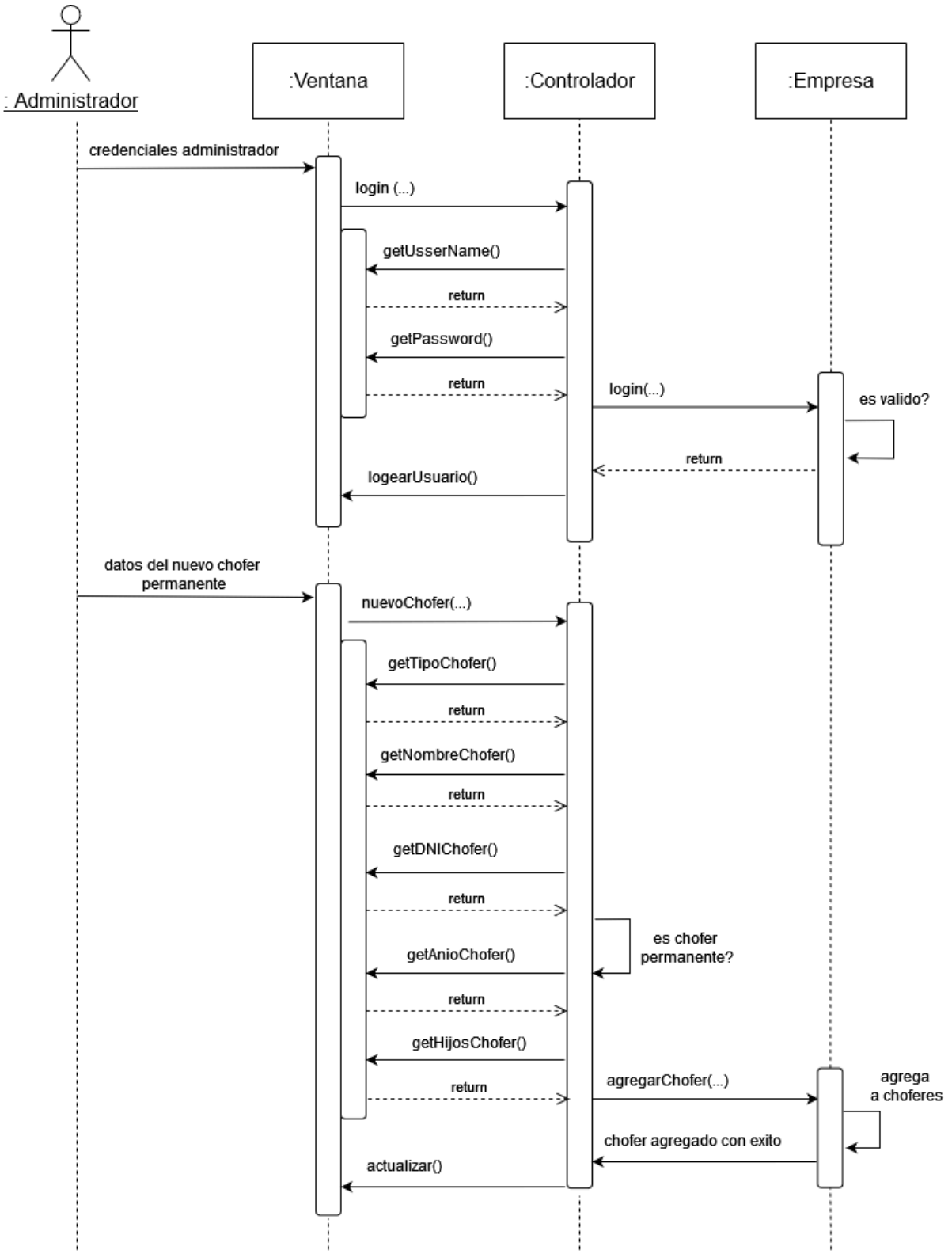
5. **`calificarPagar()`**



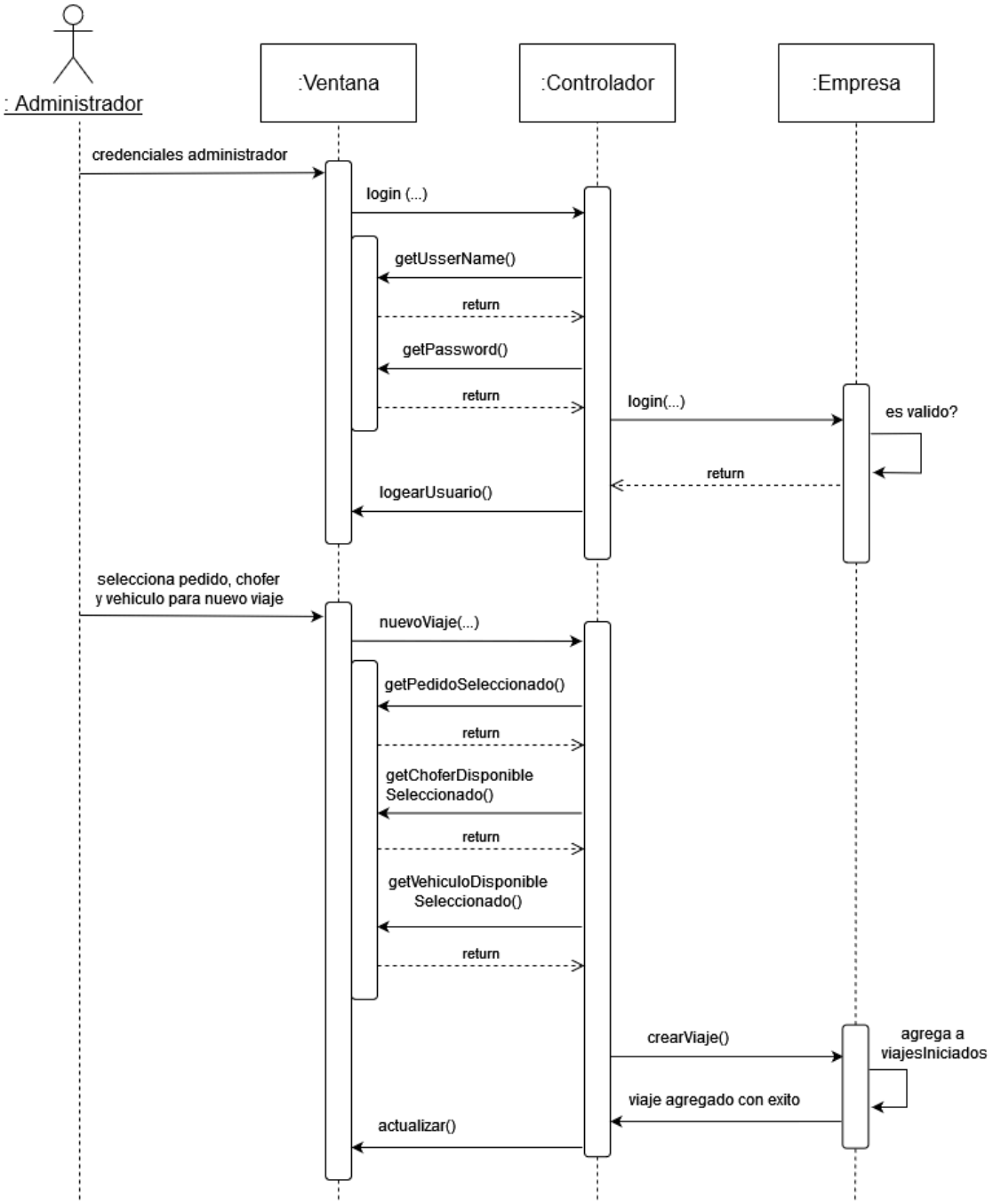
6. **nuevoVehiculo()**



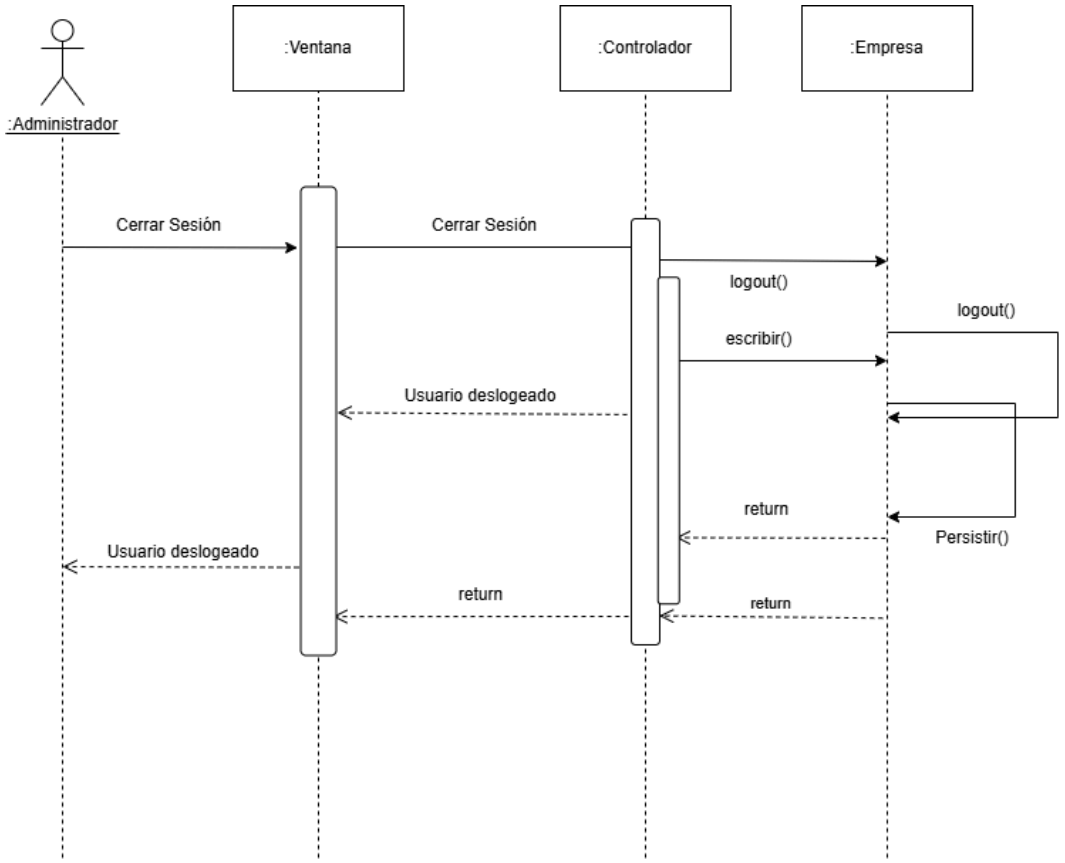
7. **nuevoChofer()**

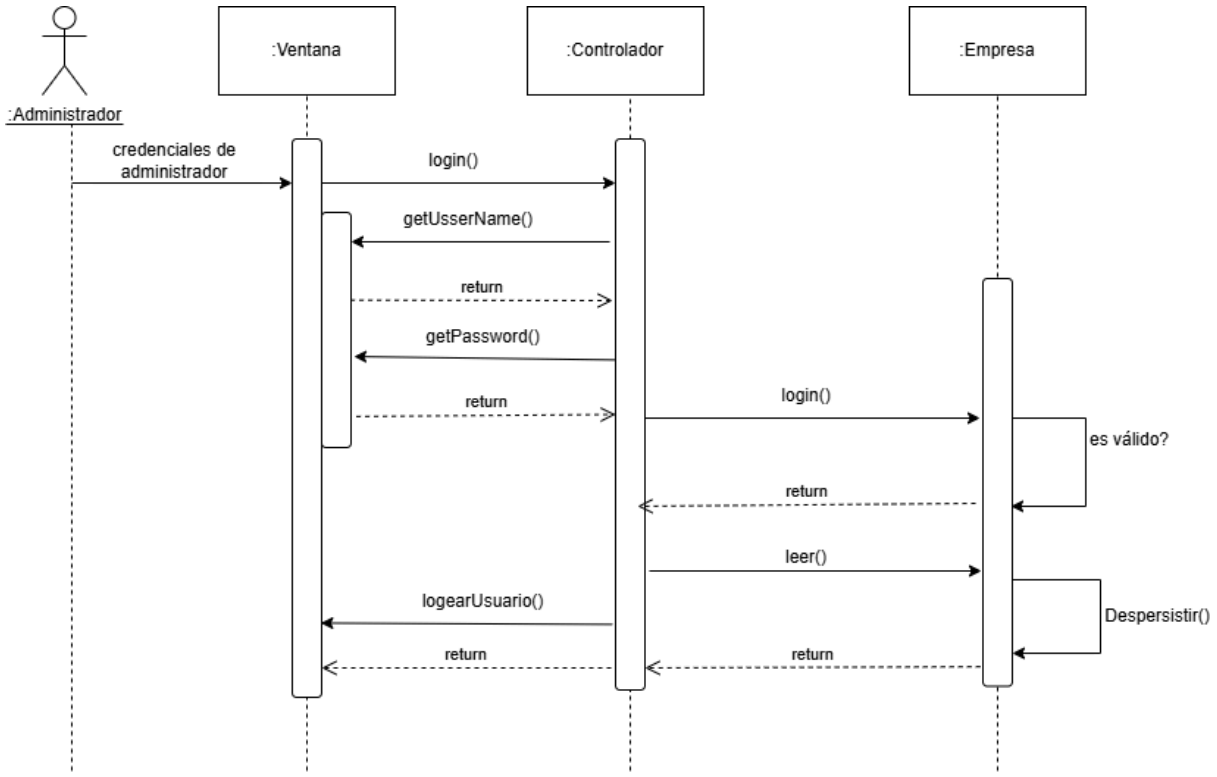

: Administrador
:Ventana
:Controlador
:Empresa
credenciales administrador
login (...)
getUsserName()
return
getPassword()
return
login(...)
es valido?
return
logearUsuario()
datos del nuevo chofer permanente
nuevoChofer(...)
getTipoChofer()
return
getNombreChofer()
return
getDNIChofer()
return
es chofer permanente?
getAnioChofer()
return
getHijosChofer()
return
agregarChofer(...)
agrega a choferes
chofer agregado con exito
actualizar()

8. **`nuevoViaje()`**



9. **escribir() y leer()**





Testeo de GUI

Para realizar el testeo de la interfaz, se dividieron las pruebas en cuatro, una para cada panel:

Login Test:

testBotonRegistro	Se chequea que el boton de registro este activado y active el panel de registro
testVacio	Se chequea que el botón de login no esté activado
testSoloContrasena	Se llena el campo de contraseña y se chequea que el botón login no esté activado
testSoloNombre	Se llena el campo de nombre de usuario y se chequea que el botón login no esté activado
testAmbosLlenos	Se llena el campo de contraseña y nombre de usuario y se chequea que el botón login esté activado
testLoginUsuarioDesconocido	Se ingresa un usuario desconocido y se comprueba que salga el mensaje correcto
testAdminLog	Se comprueba que funcione el logueo de administrador
testLogCorrecto	Se comprueba que funcione el logueo de un usuario guardado en el sistema
testLogContraIncorrecta	Se ingresa una contraseña correcta y se comprueba que salga el mensaje correcto

Registro Test:

testBotonCancelar	Prueba si el botón "Cancelar" abre el PaneldeRegistro
testRegVacio	Verifica que el botón de registro esté deshabilitado cuando todos los campos están vacíos
testRegSoloNombre	Verifica que el botón de registro esté deshabilitado cuando solo se llena el campo "Nombre de usuario"
testRegSoloContrasenia	Verifica que el botón de registro esté deshabilitado cuando solo se llena el campo "Contraseña"
testRegSoloConfirmarContrasenia	Verifica que el botón de registro esté deshabilitado cuando solo se llena el campo

	"Confirmar contraseña"
testRegSoloNombreReal	Verifica que el botón de registro esté deshabilitado cuando solo se llena el campo "Nombre real"
testRegSinUsuario	Verifica que el botón de registro esté deshabilitado cuando falta el campo "Nombre de usuario"
testRegSinConstrasenia	Verifica que el botón de registro esté deshabilitado cuando falta el campo "Contraseña"
testRegSinConstraseniaConf	Verifica que el botón de registro esté deshabilitado cuando falta el campo "Confirmar contraseña"
testRegSinNombreReal	Verifica que el botón de registro esté deshabilitado cuando falta el campo "Nombre real"
testRegCompleto	Verifica que el botón de registro esté habilitado cuando todos los campos están correctamente llenados y realiza el registro
testRegContraIncorrecta	Verifica que no se pueda registrar cuando la contraseña y la confirmación de la contraseña no coinciden
testRegUsuarioRepe	Verifica que no se pueda registrar un usuario si ya existe en el sistema

Cliente Test:

testBotonCerrarSesion	Verifica que el botón "Cerrar sesión" lleva al usuario al panel de inicio de sesión.
testPedidoSinPax	Verifica que el botón "Nuevo pedido" esté deshabilitado si la cantidad de pasajeros es cero.
testPedidoSinKm	Verifica que el botón "Nuevo pedido" esté deshabilitado si no se ha ingresado la cantidad de kilómetros.
testPedidoPaxNeg	Verifica que el botón "Nuevo pedido" esté deshabilitado si la cantidad de pasajeros es negativa.
testPedidoPaxBig	Verifica que el botón "Nuevo pedido" esté deshabilitado si la cantidad de pasajeros excede el límite permitido.
testPedidoKmNeg	Verifica que el botón "Nuevo pedido" esté deshabilitado si la cantidad de kilómetros es negativa.
testPagarYCalificarOff	Verifica que los campos de calificación y valor de viaje estén deshabilitados/inactivos al inicio.

testPedidoSinVehiculo	Verifica que se muestre el mensaje adecuado si no hay vehículo disponible para el pedido.
testPedidoConVehiculo	Verifica que, tras asignar un vehículo al pedido,todos los campos de nuevo pedido están deshabilitados y vacíos.
testCalificarYPagar	Prueba la funcionalidad de calificación y pago, asegurándose de que el pedido y la calificación se vacíen tras el pago y que el viaje se registre en el historial.
testCalificarMaxYPagar	Verifica que el botón de pago se deshabilita si se intenta calificar con un valor fuera del rango permitido.
testCalificarNegYPagar	Verifica que el botón de pago se deshabilita si se intenta calificar con un valor negativo.

Admin Test:

Testeo de viajes:

testBotonCerrarSesion	Verifica que el botón "Cerrar sesión" redirige al usuario al panel de inicio de sesión.
testNuevoViajeSinPedido	Verifica que el botón "Nuevo viaje" esté deshabilitado cuando no hay pedidos activos.
testNuevoViajeSinChofer	Asegura que el botón "Nuevo viaje" esté deshabilitado si se intenta iniciar un viaje sin un chofer asignado.
testNuevoViajeSinVehiculo	Verifica que la lista de vehículos disponibles esté vacía si no hay un vehículo disponible para el viaje.
testNuevoViajeCorrecto	Realiza una simulación completa para verificar que, tras crear un viaje correctamente, las listas de pedidos, choferes y vehículos queden vacías y el botón de "Nuevo viaje" esté deshabilitado.

Testeo de choferes:

testChoferTempSoloNombre	Verifica que, al ingresar solo el nombre, el botón de aceptar esté deshabilitado.
testChoferTempSoloDNI	Verifica que, al ingresar solo el DNI, el botón de aceptar esté deshabilitado.
testChoferTempRepe	Intenta registrar un chofer temporal ya registrado y comprueba si el mensaje indica que el chofer ya está registrado.
test Chofer Temp Correcto	Verifica que se pueda registrar un chofer temporal correctamente, que el chofer

	aparezca en la lista y que los campos se limpien tras el registro.
testChoferPermSoloNombre	Verifica que, al ingresar solo el nombre, el botón de aceptar esté deshabilitado.
testChoferPermSoloDNI	Verifica que, al ingresar solo el DNI, el botón de aceptar esté deshabilitado.
testChoferPermSoloAño	Verifica que, al ingresar solo el año, el botón de aceptar esté deshabilitado.
testChoferPermSoloHijos	Verifica que, al ingresar solo la cantidad de hijos, el botón de aceptar esté deshabilitado.
testChoferPermSinHijos	Verifica que, sin especificar la cantidad de hijos, el botón esté deshabilitado.
testChoferPermSinAño	Verifica que, sin especificar el año, el botón esté deshabilitado.
testChoferPermSinDNI	Verifica que, sin especificar el DNI, el botón esté deshabilitado.
testChoferPermSinNombre	Verifica que, sin especificar el nombre, el botón esté deshabilitado.
testChoferPermHijosNeg	Verifica que, al ingresar un número negativo de hijos, el botón esté deshabilitado.
testChoferPermAñoMax	Verifica que, al ingresar un año superior a 3000, el botón esté deshabilitado.
testChoferPermAñoMin	Verifica que, al ingresar un año inferior a 1900, el botón esté deshabilitado.
testChoferPermRepe	Intenta registrar un chofer permanente ya registrado y verifica si el mensaje indica que el chofer ya está registrado.
testChoferPermCorrecto	Verifica que se pueda registrar un chofer permanente correctamente, que el chofer aparezca en la lista y que los campos se limpien tras el registro.

Testeo de vehículos:

testMotoSinPatente	Verifica que, al no ingresar una patente para la moto, el botón esté deshabilitado.
testMotoConPlazas	Verifica que, al ingresar una patente y plazas válidas para la moto, el botón esté habilitado.
testMotoRepe	Intenta registrar una moto ya registrada y

	verifica si el mensaje indica que el vehículo ya está registrado.
testMotoCorrecto	Verifica que se pueda registrar una moto correctamente, que aparezca en la lista y que el campo de patente se limpie tras el registro.
testAutoSinPatente	Verifica que, al no ingresar una patente para el auto, el botón esté deshabilitado.
testAutoPlazaMin	Verifica que, al ingresar un valor mínimo no válido en plazas para el auto, el botón esté deshabilitado.
testAutoPlazaMax	Verifica que, al ingresar un valor máximo no válido en plazas para el auto, el botón esté deshabilitado.
testAutoRepe	Intenta registrar un auto ya registrado y verifica si el mensaje indica que el vehículo ya está registrado.
testAutoCorrecto	Verifica que se pueda registrar un auto correctamente, que aparezca en la lista y que los campos de patente y plazas se limpien tras el registro.
testCombiSinPatente	Verifica que, al no ingresar una patente para la combi, el botón esté deshabilitado.
testCombiPlazaMin	Verifica que, al ingresar un valor mínimo no válido en plazas para la combi, el botón esté deshabilitado.
testCombiPlazaMax	Verifica que, al ingresar un valor máximo no válido en plazas para la combi, el botón esté deshabilitado.
testCombiRepe	Intenta registrar una combi ya registrada y verifica si el mensaje indica que el vehículo ya está registrado.
testCombiCorrecto	Verifica que se pueda registrar una combi correctamente, que aparezca en la lista y que los campos de patente y plazas se limpien tras el registro.

Testeo de visualizacion de informacion:

testSueldosVacio	Verifica que no haya sueldos mostrados cuando se espera que estén vacíos. Confirma que el campo de texto del total de sueldos esté en "0,00".
testSueldos	Agrega un chofer temporal y verifica que aparezcan sueldos en el campo de texto correspondiente.
testViajesVacio	Comprueba que la lista de viajes históricos

	esté vacía, validando que no haya viajes cuando debería estar sin datos.
testVehiculosTotalesVacio	Verifica que la lista de vehículos esté vacía.
testVehiculosTotales	Agrega un vehículo y verifica que la lista de vehículos no esté vacía.
testClientesTotalesVacio	Comprueba que la lista de clientes esté vacía cuando no se ha registrado ningún cliente.
testClientesTotales	Registra un cliente, y luego confirma que la lista de clientes no esté vacía.
testChoferesTotalesSinSeleccionar	Verifica que al no seleccionar un chofer, no aparezcan viajes, calificación ni sueldo en los campos correspondientes.
testChoferesTotalesSeleccionado	Selecciona un chofer y verifica que aparezcan sus viajes, calificación y sueldo en la interfaz.
testListaSinPedidos	Confirma que la lista de pedidos pendientes esté vacía cuando no hay pedidos.
testListaConPedidos	Agrega un pedido y verifica que la lista de pedidos pendientes tenga el pedido agregado.
testListaSinChoferes	Comprueba que la lista de choferes libres esté vacía cuando no hay choferes disponibles.
testListaConChoferes	Agrega un chofer temporal y verifica que la lista de choferes libres tenga al menos un chofer.
testListaVehiculosSinTocarPedido	Verifica que la lista de vehículos disponibles esté vacía cuando no se ha tocado un pedido.
testListaVehiculosSinPedido	Verifica que no haya vehículos si no cumplen para un pedido.
testListaVehiculosConPedido	Agrega un cliente, vehículo y pedido, selecciona el pedido y verifica que la lista de vehículos disponibles no esté vacía.
testListaViajesHistoricos	Verifica que la lista de viajes históricos no esté vacía.

Lista de errores

A continuación se listan los diferentes errores encontrados en cada tipo de test realizado.

1. Datos/Caja negra
    a. ViajeTest

        public void getValorStandarMascota(): No se aplican las bonificaciones correspondientes a la Zona Standard.

        - Valor esperado: <3100>.
        - Valor real: <2300>.

        public void getValorSinAsfaltarBaulTest(): No se aplican las bonificaciones correspondientes por uso de baúl.

        - Valor esperado:<2900>
        - Valor real:<2350>

        public void getValorStandarTest(): No se aplican nuevamente las bonificaciones por Zona Standard. No se permite baúl ni mascota en este caso de prueba, por lo cual el valor obtenido sin la bonificación es el precio base.

        - Valor esperado:<1800>
        - Valor real:<1000>

        public void getValorStandarBaulTest(): No se aplican las bonificaciones por el uso de baúl ni por Zona Standard. El valor real fue el valor base del viaje.

        - Valor esperado: <2350>.
        - Valor real: <1000>.

    b. CombiTest

        public void getPuntajePedidoTest(): No se suma la bonificación por uso de baúl (+100).

        - Valor esperado: <170>.
        - Valor real: <70>.

    c. AutoTest

        public void getPuntajePedidoSinBaulTest(): Se aplica la bonificación de baul aun cuando el auto no usa baul.

        - Valor esperado: <90>.
        - Valor real: <120>.

2. Modelo/Caja negra
    a. EmpresaVaciaTest
       todo bien
    b. Empresa Test
       todo bien
    c. Empresa test 2 (escenario 3)
        - public void crearViajeTestClienteConViajePendiente(): Lanza una excepción de cliente Con Pedido Pendiente Exception. Esta excepción se lanza al agregar los pedidos, esto imposibilita que se pueda lanzar esta excepción en crearViaje().

- public void agregarPedidoConClienteViajePendienteTest(): nunca lanza la excepción ya que agregarPedido() va a lanzar primero la excepción cliente Con Pedido Pendiente Exception. Ambas excepciones mencionadas en crearViaje() son inalcanzables.

3. GUI
   a. AdminTestChoferesVehiculos
      i. testCombiCorrecto (la patente debería estar vacía)(No se vacían los TextFields)
      ii. testChoferTempRepe (expected chofer ya registrado was null)
      iii. testMotoCorrecto (No se vacían los TextFields)
      iv. testAutoCorrecto (No se vacían los TextFields)
      v. testChoferPermRepe (No se lanza la excepción)
      vi. testChoferPermCorrecto (No se vacían los TextFields)
      vii. testChoferTempCorrecto (No se vacían los TextFields)

4. Persistencia
   No se encontraron errores. Era un resultado esperado debido a que la capa de persistencia utiliza en su mayoría librerías de java.

5. Integración
   a. ActionPerformedTest
      i. public void actionPerformedNuevoPedidoTest() en assertNotNull(pedidoAgregado). Los pedidos no se agregan correctamente a la lista de pedidos desde el panel de usuario.
         - Valor esperado: pedidoAgregado != null
         - Valor real: pedidoAgregado == null
      ii. public void actionPerformedNuevoVehiculoTest(){ assertEquals(ventana.getPlazas(), vehiculoAgregado.getCantidadPlazas()). La cantidad de plazas se inicializa con el valor ingresado para la cantidad de kilómetros.
         - Valor esperado: <4>.
         - Valor real: <3>.
   b. NuevoChoferTest
      i. public void nuevoChoferFailTest(){
         Tendría que haber lanzado ChoferRepetidoException. Se permite ingresar choferes repetidos, es decir, con el mismo dni. Este error se extiende del test de GUI.
         - Valor esperado: ChoferRepetidoException.
         - Valor real: no se lanzan excepciones.

   c. NuevoPedidoTest

   i. public void nuevoPedidoExitosoTest(){ en assertNotNull(pedidoAgregado). Este error ya fue detectado en ActionPerformedTest, no se agrega el pedido correctamente.

d. NuevoViajeTest
   i. public void nuevoViajeFallidoTest(){
      Se debe lanzar la excepción VehiculoNoValidoException, que es lanzada cuando ningún vehículo puede satisfacer el pedido, pero se lanza la excepción VehiculoRepetidoException en su lugar.

Conclusión

Utilizando las técnicas de testing vistas en la materia, se identificaron varios errores en diferentes partes del software, el cual, a simple vista, parecía funcionar sin fallas evidentes. Uno de los tests más interesantes fue el de GUI, ya que no es común en nuestras prácticas realizar pruebas sobre interfaces gráficas.

Durante el test de caja negra, se detectaron múltiples errores, lo cual valida la utilidad de esta técnica para encontrar fallas mediante buenas baterías de prueba.

En el test de GUI, se evidenció que el panel de administración contenía errores. Estos fallos fueron descubiertos con el apoyo de la clase *Robot*, la cual imita el comportamiento de un usuario, simulando la interacción con los campos de la interfaz como si fuera una persona, lo que permite una cobertura más exhaustiva.

En el test de persistencia, se utilizó el archivo *empresa.bin*, y toda la implementación se realizó sin alterar el patrón de diseño *singleton* empleado por la clase *Empresa*.

Finalmente, durante el test de integración se comprobó el funcionamiento correcto del controlador y que sus interacciones con las otras capas del sistema sean las esperadas.

Como conclusión, aunque la tarea de probar este software no es compleja en sí misma, resulta laboriosa; una vez identificados los datos a verificar, la codificación de los tests se vuelve sencilla y repetitiva. Este trabajo resultó ser una experiencia nueva e interesante para nuestro desarrollo académico.