Trabajo Final

“Subí que te llevo”

Integrantes:
Bianchini, Pilar
Lascar, Marcos
Santiago, Felipe
Stadler, Laura

Grupo 5

Fecha de entrega: 13/11/2024

Facultad de Ingeniería
Taller de Programación 1.

Test de pruebas Unitarias.

Determinación de casos de prueba:

- Test de capa de negocio: Capa de Negocio .
- Test de capa de datos: Capa de Datos .

En estos dos documentos se encuentran los distintos casos de prueba con los que se realizaron los test.

Errores en capa de datos

- **Auto** calcula el puntaje como si fuese baul=True, incluso cuando es False.
- **Combi** no suma 100 puntos por baul al calcular el puntaje.
- **Combi** admite más cantPasajeros que su máximo y que su mínimo.
- **Viaje** : mal calculado el valor del viaje.

Errores en capa de negocio

Para realizar un testeo profundo en los métodos que modifican las listas, nos aseguramos que las mismas estén correctamente seteadas a través de, en el set up, inicializarlas como es esperado que lo sean.

- **agregarPedido():** acepta pedidos con menos de 5 personas en caso de haber únicamente combis en el array de vehículos.
- **agregarChofer()** : permite agregar un chofer con un dni repetido, aunque tenga nombre distinto.
- **calificacionDeChofer(Chofer chofer):** cuando el chofer indicado no tiene viajes realizados no se lanza la excepción, da un resultado. (Escenario 2, clase de equivalencia 2).
- **agregaPedido(), creaViaje()**: se encontró que cuando un cliente tiene un viaje pendiente, su pedido es agregado a la lista de pedidos pendientes, y este mismo podría luego llegar a convertirse en un viaje. (Escenario 2, testAgregaPedido5() y testCreaViaje9())

Testeo de excepciones:

Formato de testeo:

Para testear que las excepciones se lancen bien, con el mensaje y/o argumentos correctos, se realizó con el siguiente formato.

```
public void testAgregaPedido3() {
    Pedido pedido = new Pedido (new Cliente("miUsuario", "12345", "Joel"),  4, true, true, 10, Constantes.ZONA_STANDARD );
    try {
        empresa.agregarPedido(pedido);
        Assert.fail("Deberia haber lanzado excepcion");

    } catch (SinVehiculoParaPedidoException e) {

        Assert.fail("Deberia haber lanzado esta excepcion");

    } catch (ClienteNoExisteException e) {

        String mensaje = e.getMessage();
        Assert.assertEquals(mensaje, Mensajes.CLIENTE_NO_EXISTE.getValor());

    } catch (ClienteConViajePendienteException e) {

        Assert.fail("Deberia haber lanzado esta excepcion");

    } catch (ClienteConPedidoPendienteException e) {

        Assert.fail("Deberia haber lanzado esta excepcion");
    }
```

Se corrobora que el mensaje y los argumentos de la misma sean los esperados. Además de tener en cuenta que si el método lanza más de una excepción, sea lanzada la que se espera.

Errores encontrados:

- **VehiculoRepetidoException**: la excepción se lanza bien, no devuelve el vehículo repetido, devuelve null.
- **ClienteConPedidoPendienteException**: se intenta agregar un viaje a un cliente con pedido pendiente, lanza excepción, pero el mensaje es incorrecto.
- **ClienteConViajePendienteException**: lanza la excepción errónea, lanza ClienteConPedidoPendienteException.
- **SinVehiculoParaPedidoException**: falla, con menos de 5 personas, acepta combi y no lanza la excepción.
- **ChoferNoDisponibleException**: falla, cuando el chofer no está disponible lanza la excepción correcta, pero el mensaje es “pedido no figura en la lista”.
- **SinViajesException :** No se lanza la excepción cuando se pide ver la calificación de un chofer sin viajes realizados.
- **ClienteConViajePendienteException**: no se lanza cuando se intenta agregar un pedido de un cliente con un viaje ya pendiente, ni tampoco cuando se intenta crear el viaje de un cliente con viaje pendiente.

Test de persistencia:

Para el test de persistencia se trataron dos casos:

1. A un archivo .bin existente de la empresa, y se sobre escribe registrando en la empresa: un chofer, un cliente y un vehículo. Posteriormente se le escribe y lee.

2. No existe archivo .bin anterior de la empresa, es decir que se crea uno nuevo. A este se le agrega un chofer, un cliente y un vehículo. Posteriormente se le escribe y lee.

No se encontraron errores.
Cuando el archivo no existe se crea correctamente.

Se persisten y despersisten los datos de la empresa de forma correcta.

Test de GUI:

Para realizar el testeo de interfaces gráficas se utilizó la clase Robot, para automatizar el testeo a través de inputs de teclado y mouse.

Errores en panel Login

No se encontraron errores.

Errores en panel Registrar

No se encontraron errores.

Errores en panel Cliente

No se encontraron errores.

Errores en panel Administrador

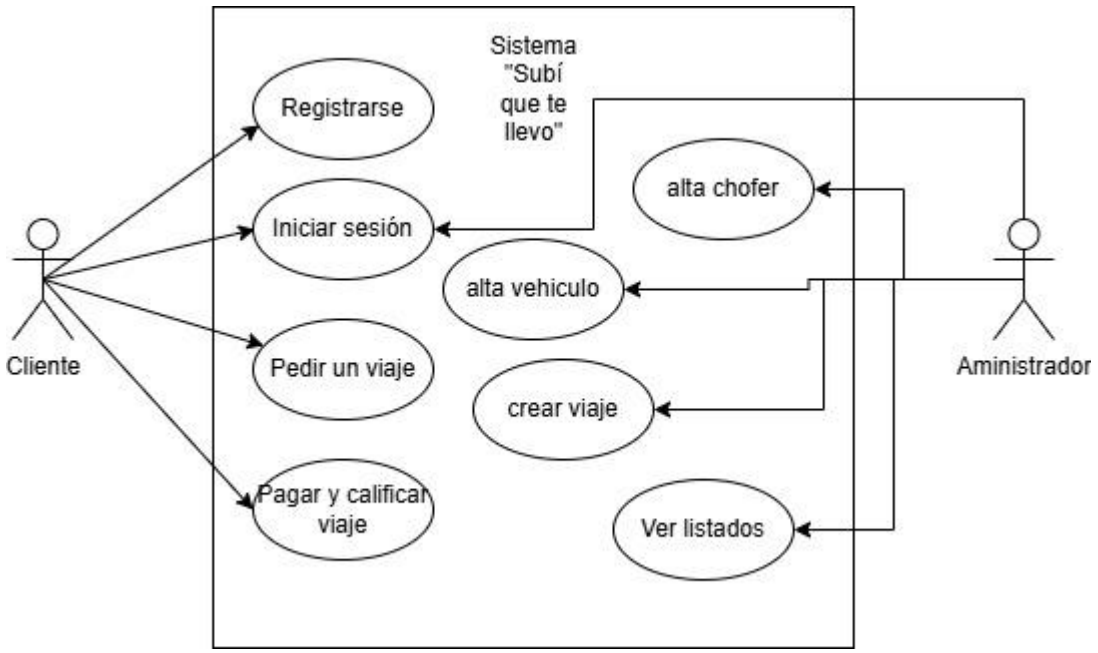
Se encontraron un par de errores amiguitos (que haces adolfo, todo bien?)

Después de agregar un chofer, ya sea permanente o no, los casilleros de DNI, Nombre y año no se vacían correctamente.

Al intentar agregar un chofer que contiene un DNI ya existente en la lista de choferes, no se lanza el mensaje correspondiente. (Creemos que es debido a que ya de por si el método agregarChofer() no lanza la excepción correspondiente)

Test de integración:

Casos de uso determinados: Controlador.

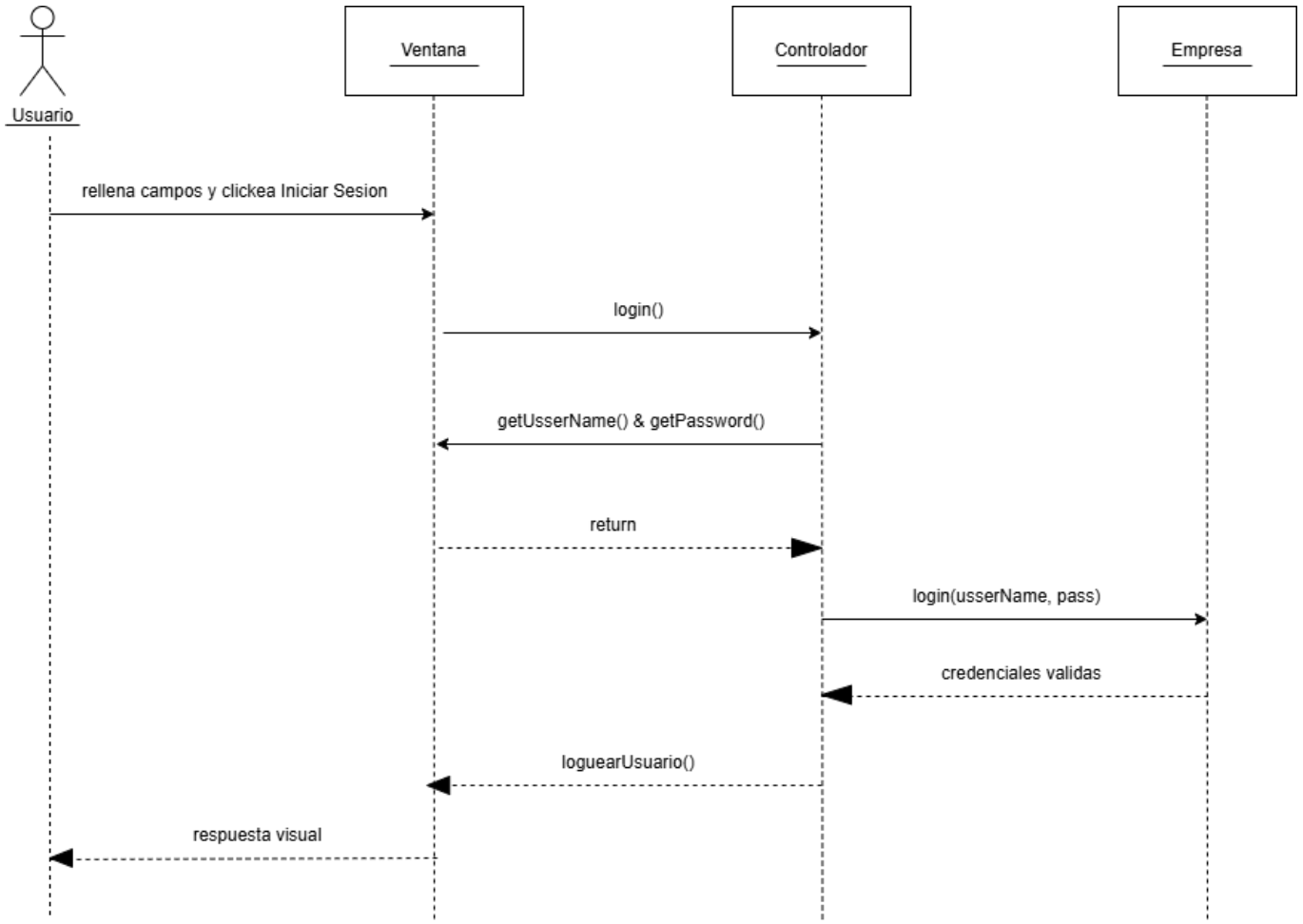


Los casos de prueba detectados para los casos de uso resultaron repetirse con aquellos que ya fueron testeados en las pruebas unitarias de la capa de negocio por lo que decidimos únicamente testear aquellos casos donde el curso de acciones debería resultar ser exitoso.

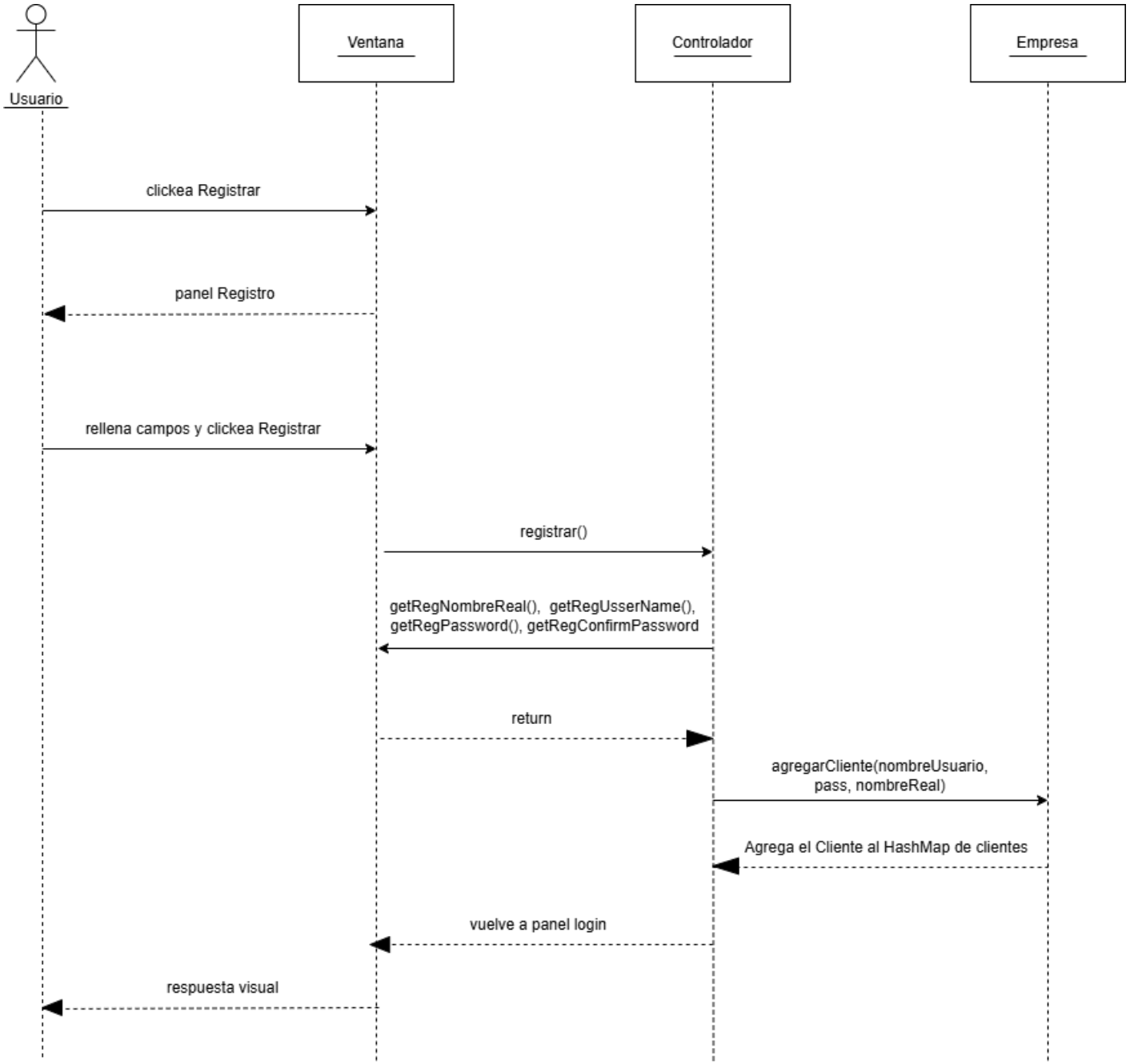
Para el test de integración, se verifica que las interacciones entre todos los módulos funcionen correctamente. Así mismo si hubiera algún error en la capa de datos y/o negocio este se replicará en el test de integración. Vale aclarar que para el caso de la interacción con la ventana se utilizó un mock para poder simular la entrada de datos por la misma

A continuación desarrollamos el diagrama de secuencia de sistema para algunos de los casos de uso más relevantes, entre los testeados.

Usuario se Loguea



Usuario se Registra



Errores del controlador

El controlador determina la cantidad de KM del viaje desde el JTextField CANT_PAX, en vez de CANT_KM, de la vista.
El resto de los errores encontrados en la capa de datos con respecto a excepciones se deberían de replicar en las pruebas de integración para los casos de prueba que no fueron testeados.

Conclusión:

En el transcurso de este trabajo, notamos la importancia que tiene realizar un proceso sistemático de testeo. Ya que en el caso de este no haber sido realizado, y en un entorno real, el software hubiera sido publicado con fallos grandes, que modificaban o imposibilitaban el uso del mismo, los cuales iban a ser descubiertos por los usuarios finales.

También fue importante la parte de sistematizar este proceso, logrando así un chequeo a un nivel generalizado, abarcando gran parte de los usos que puede tener un usuario corriente.

Descubrimos la importancia de una completa y no ambigua documentación, ya que ésta nos fue vital durante el transcurso de todo el proyecto, por ser este a modo de caja negra. Como también la importancia de una buena programación pensada para las pruebas del testing, el uso de las constantes y mensajes fueron de gran utilidad a la hora de testear métodos y excepciones.

Cabe aclarar que nunca se puede decir que el código estará 100% libre de errores sino que no se pudieron encontrar ningún error.