DESARROLLO DE SOFTWARE

EJERCICIO GUIADO 3

31 DE MARZO DE 2023

Belén Gómez Arnaldo 100472037

Ignacio Fernández Cañedo 100471955

GRUPO 4

ÍNDICE

FUNCIÓN : register_order

Esta función recibe la información de un pedido y lo registra en el almacén *store_request*. Además devuelve un código hexadecimal de 32 dígitos, *orderID,* que será necesario para el resto de funciones. También se comprueba que los parámetros de entrada y salida son correctos. Para los tests de esta función usamos la técnica de las clases de equivalencia.

Los tests de esta función están en el módulo *test_order_manager_tests*. En estos tests comprobamos que, en caso de que las entradas sean correctas, se genera el *orderID* para el pedido y se almacenan los datos del pedido en un fichero JSON. Si alguno de los datos de entrada no es válido, la función devuelve una excepción y el pedido no se almacena en el fichero JSON. Esto lo comprobamos buscando en el almacén los datos de la entrada incorrecta y comprobando que no hay ningún pedido con esas entradas en el almacén. Además, si se intenta registrar dos veces el mismo pedido, solo se almacena una vez en el fichero.

Clases de equivalencia y análisis de valores límite

- **Product_ID:**
    - Clase válida: número de 13 dígitos que cumple la función de validate_eAn13. Ejemplo: 8421691423220.
    - Clase inválida 1: el product_ID no está compuesto únicamente por números. Ejemplo: 842169142322A.
    - Clase inválida 2: el product_ID no cumple la función de validate_eAn13. Ejemplo: 8421691423225
    - Clase inválida 3: el número tiene más de 13 dígitos. Para realizar el análisis de valores límites hemos comprobado el product_ID con un test introduciendo 14 dígitos. Ejemplo: 84216914232200
    - Clase inválida 4: el número tiene menos de 13 dígitos. Para realizar el análisis de valores límites hemos comprobado el product_ID con un test introduciendo 12 dígitos. Ejemplo: 842169142322
- **Address**

  Número máximo de dígitos: 100

  Número mínimo de dígitos: 20

    - Clases válidas: dirección de entre 20 y 100 caracteres en la que al menos hay dos cadenas separadas por un espacio. Ejemplo: C/LISBOA,4, MADRID, SPAIN.

Para hacer el análisis de valores límite correctamente, hemos comprobado entradas de 20, 21, 99 y 100 caracteres

- Clase inválida 1: la longitud de la dirección es menor de 20. Para hacer el análisis de valores límites usamos una cadena de 19 dígitos. Ejemplo: C/LISBOA, 4, MADRID,.
- Clase inválida 2: la longitud de la dirección es mayor de 100. Para hacer el análisis de valores límites usamos una cadena de 101 dígitos. Ejemplo: C/LISBOA,4, MADRID, SPAINSPAINSPAINSPAINSPAINSPAINSPAINSPAINSPAINSPAINSPAINSPAINSPAINSPAIN,.
- Clase inválida 3: la dirección está formada por una única cadena, sin espacios. Ejemplo: C/LISBOA,4,MADRID,SPAIN.

- **Order_type**

  La lista de valores válidos es [PREMIUM, REGULAR], por lo que hay que hacer una clase válida para cada valor de la lista y una clase no válida para el resto.

  - Clase válida 1: order_type = REGULAR.
  - Clase válida 2: order_type = PREMIUM.
  - Clase inválida: cualquier otro valor que no sea REGULAR o PREMIUM. Ejemplo: PRE.

- **Phone**
  - clase válida: La longitud del número será de 9 sin letras ni dígitos especiales. Ejemplo: 123456789
  - clase inválida 1: El número contiene un caracter que no es un número. Ejemplo: 12345A789
  - Clase inválida 2: el número tiene más de 9 dígitos. Para realizar el análisis de valores límites hemos comprobado el Phone con un test introduciendo 10 dígitos. Ejemplo: 1234567890
  - Clase inválida 3: el número tiene menos de 9 dígitos. Para realizar el análisis de valores límites hemos comprobado el Phone con un test introduciendo 8 dígitos. Ejemplo: 12345678

- **Zip_code**

  En España los códigos postales son cadenas de 5 caracteres numéricos y los dos primeros dígitos son 52 o un número menor.

- Clase válida: una cadena de 5 dígitos que empieza o por 52 o un número menor. Ejemplo: 28005.
- Clase inválida 1: una cadena de menos de 5 caracteres. Para realizar el análisis de valores límites hemos comprobado el código postal con una cadena de 4 caracteres. Ejemplo: 2800.
- Clase inválida 2: una cadena de más de 5 caracteres. Para realizar el análisis de valores límites hemos comprobado el código postal con una cadena de 6 caracteres. Ejemplo: 280055.
- Clase inválida 3: una cadena con algún valor no numérico. Ejemplo: 2800A.
- Clase inválida 4: una cadena donde los dos primeros dígitos son un número mayor de 52. Ejemplo: 67005.

FUNCIÓN 2: send_product

La entrada de esta función es un fichero json que incluye el OrderID y el email de contacto. La función va a crear un fichero store_shipping con los datos del envío y devolverá un código en hexadecimal de 64 dígitos llamado tracking_code. También se comprueba que los parámetros de entrada y salida son correctos.

Los tests de esta función están en el módulo test_order_shipping_tests. En estos tests comprobamos que, en caso de que las entradas sean correctas, se genera el tracking_code para el pedido y se almacenan los datos del pedido en un fichero JSON. Además, si se intenta registrar dos veces el mismo pedido, solo se almacena una vez en el fichero.

Si alguno de los datos de entrada no es válido el pedido no se almacena en el fichero JSON. Esto se comprueba en los test borrando el almacén al inicio del test y comprobando que al final no se ha creado.

Como al empezar los test válidos borramos el almacén y depués llamamos a la función register_order, nos aseguramos de que el almacén que devuelve esta función no ha sido manipulado.

Gramática:

La gramática de esta función es la siguiente.

Fichero ::= Inicio_objeto Datos Fin_objeto
Inicio _objeto ::= {
Fin _objeto ::= }

Datos ::= Campo1 Separador Campo 2
Campo1 ::= Etiqueta_dato1 Igualdad Valor_dato1
Campo2 ::= Etiqueta_dato2 Igualdad Valor_dato2
Separador ::= ,
Igualdad ::= :

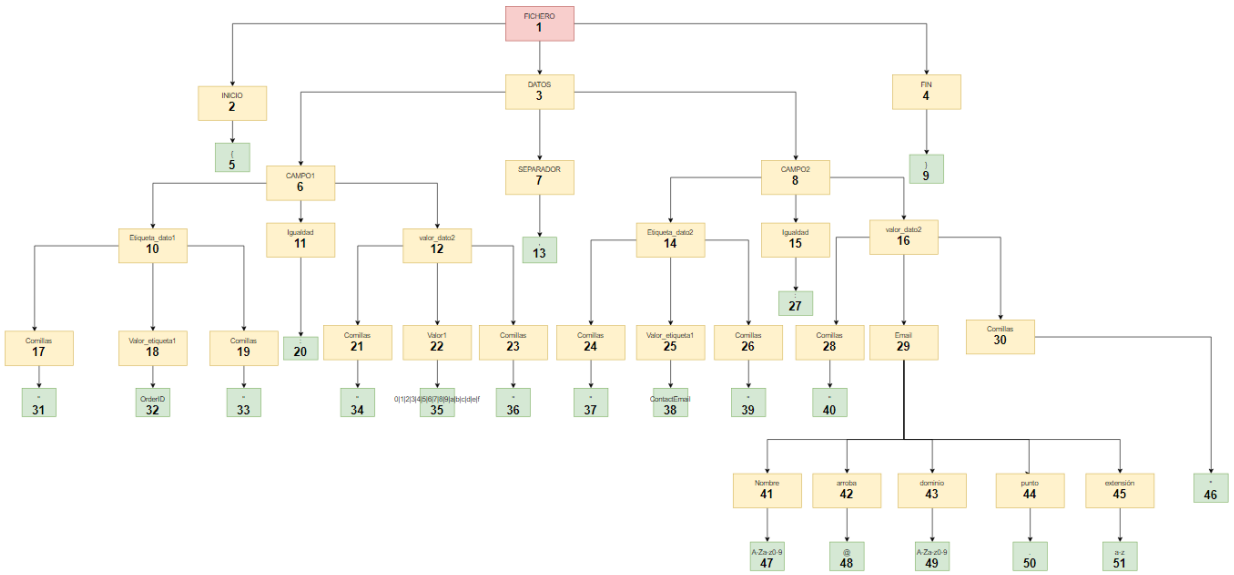
Etiqueta_dato1 ::= Comillas Valor_etiqueta1 Comillas
Valor_etiqueta1 ::= OrderID
Valor_dato1 ::= Comillas Valor1 Comillas
Valor1 ::= a|b|c|e|f|0|1…|9| {32}

Comillas ::= "

Etiqueta_dato2 ::= Comillas Valor_etiqueta2 Comillas
Valor_etiqueta2 ::=ContactEmail
Valor_dato2 ::= Comillas Email Comillas
Email ::= nombre_correo arroba dominio punto extensión
Arroba ::= @
Nombre_correo ::= a|..|z|0|..|9
Dominio ::= a|..|z
Punto ::= .
Extensión ::= a|..|z (máximo 3)

Árbol de derivación:

El árbol de derivación que hemos desarrollado para esta función es el siguiente:



Casos de pruebas:

Los casos de prueba de esta función son los siguientes:

- Para los nodos no terminales, hemos hecho un caso para cuando se duplica el contenido del nodo y otro para cuando se borra el contenido.

- Para los nodos terminales hemos hecho un caso para cuando se modifica el contenido del nodo a un valor no válido. El caso de que se borre o se duplique un nodo terminal ya se contempla en los tests de los nodos no terminales.
- Además, hemos hecho el análisis de valores límites con el orderID, comprobando qué pasa cuando es una cadena de 31 caracteres y de 33 caracteres.

- También se han hecho tests para comprobar las siguientes excepciones: no se encuentra el archivo de datos de entrada, el archivo de entrada no es de formato JSON, no se encuentra el almacén, el pedido no se encuentra entre los pedidos registrados y el fichero de entrada está vacío.

FUNCIÓN 3: delivery_product

La entrada de esta función es un tracking_code. La función va a crear un fichero store_delivery con los datos de la entrega y devolverá True si el código de seguimiento y la fecha son válidas. También se comprueba que los parámetros de entrada y salida son correctos.

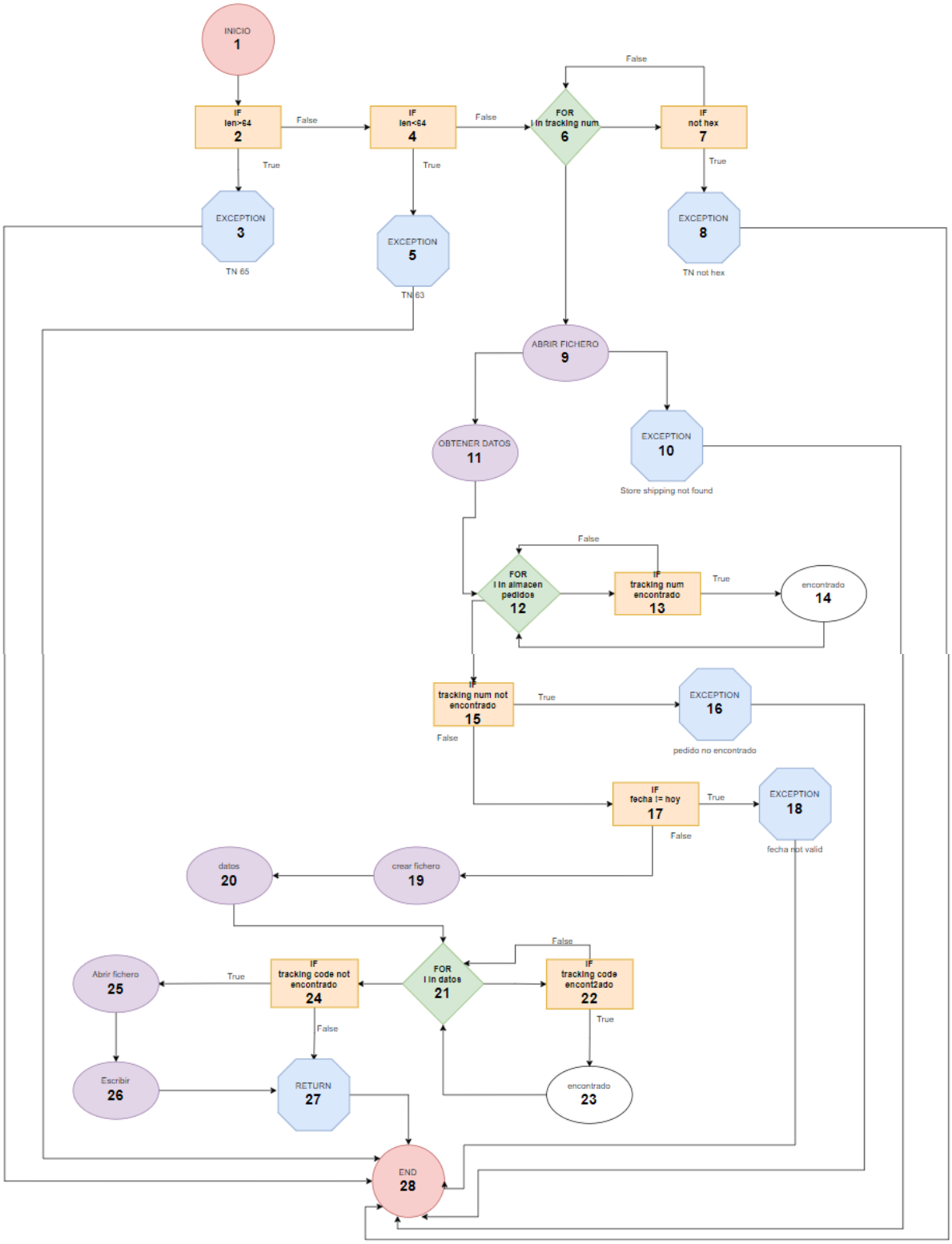
Hemos decidido eliminar las excepciones "*JSON Decode error*" que se ejecutaban en caso de que el archivo abierto con la función with no fuese del tipo .json. Estas excepciones no pueden ocurrir nunca ya que el archivo *store_shipping* se crea en la función RF2 y en ella comprobamos que se crea como un JSON. Además también creamos nosotros el archivo *store_delivery* en esta función como un JSON y por ello tampoco va a saltar este error.

Los tests de esta función están en el módulo *test_order_delivery_tests*. Para optimizar el código hemos creado una función *inciar* al inicio del módulo para borrar los almacenes y llamar a esta función desde cada test. De esta forma no es necesario copiar el código para borrar los almacenes en cada test.

En estos test llamamos a las funciones RF1 y RF2 para crear los almacenes con los valores necesarios. Después comprobamos que, en caso de que las entradas sean correctas, se almacenan los datos del pedido en un fichero JSON y la función devuelve True. Para que la fecha de entrega sea correcta usamos la función *freeze_time*. Utilizamos una fecha al llamar a las funciones RF1 y RF2 y otra fecha (1 día después o 7 después dependiendo del tipo del pedido) al llamar a la función RF3. Si alguno de los datos de entrada no es válido el pedido no se almacena en el fichero JSON.

Los tests de esta función coinciden con los caminos del diagrama de flujo y además se añaden los tests necesarios para probar los bucles. Cabe destacar que en nuestra función no se hace diferencia entre los pedidos PREMIUM y REGULAR, por lo que no hay diferentes caminos ni diferentes tests para el tipo de pedido. Además, todos los tests de las funciones los hemos hecho con los valores "*REGULAR*" y "*PREMIUM*" escritos en mayúsculas, por lo que, para evitar errores, hemos modificado en la clase *order_shipping* el if que comprobaba el *order_type* ya que estaba escrito "Regular" en minúsculas.

Diagrama de flujo



Casos de prueba

Los casos de prueba de esta función son los caminos del diagrama de flujo y los casos necesarios para probar el for.

1. **test_tracking_num_63**

   1_2_3. El *tracking_number* es una cadena de menos de 64 caracteres. Para realizar el análisis de valores límite comprobamos un *tracking_number* de 63 caracteres.

2. **test_tracking_num_65**

   1_2_4_5. El *tracking_number* es una cadena de más de 64 caracteres. Para realizar el análisis de valores límite comprobamos un *tracking_number* de 65 caracteres.

3. **test_tracking_num_not_hex**

   1_2_4_6_7_8. El *tracking_number* es una cadena que no está en hexadecimal.

4. **test_store_shipping_not_found**

   1_2_4_6_7_9_10. El fichero almacén *store_shipping* no existe.

5. **test_pedido_not_found**

   1_2_4_6_7_9_11_12_13_14_15_16. El pedido no se encuentra en el almacén.

6. **test_fecha_not_valid**

   1_2_4_6_7_9_11_12_13_14_15_17_18. La fecha de entrega no es válida.

7. **test_ok**

   1_2_4_6_7_9_11_12_13_14_15_17_19_20_21_24_25_26_27_28. Caso correcto donde el pedido no se había almacenado todavía en *delivery_store*. En este caso no se entra en el for del nodo ya que el almacén *delivery_store* está vacío

8. **test_ok_ya_almacenado**

   1_2_4_6_7_9_11_12_13_14_15_17_19_20_21_22_23_24_25_26_27_28. Caso válido donde el pedido ya estaba almacenado en *delivery_store*, por lo que no se tiene que

volver a almacenar. En este caso se entra 1 vez al bucle for del nodo 21 ya que había un elemento almacenado.

9. **test_ok_for_dic2**

   1_2_4_6_7_9_11_12_13_14_15_17_19_20_21_22_23_24_25_26_27_28. Caso válido en el que el pedido no estaba en el almacén pero sí que existía el almacén con 1 pedido. Este caso lo hacemos para probar el funcionamiento del código cuando se entra en el bucle 1 vez y no se encuentra el pedido. Al finalizar la función en el almacén *delivery_store* están los dos pedidos. Para comprobar este caso hemos llamado a la función *register_order* (RF1) 2 veces para crear los 2 pedidos y después a la función *deliver_product* (RF3) 2 veces también, con las entradas correspondientes

10. **test_ok_for_dic4**

    1_2_4_6_7_9_11_12_13_14_15_17_19_20_21_22_23_24_25_26_27_28. Caso válido en el que el pedido no estaba en el almacén pero sí que existía el almacén con 3 pedidos. Este caso lo hacemos para probar el funcionamiento del código cuando se entra en el bucle 3 veces. Al finalizar la función en el almacén *delivery_store* están los 4 pedidos. Para comprobar este caso hemos llamado a la función *register_orde*r (RF1) 4 veces para crear los 4 pedidos y después a la función *deliver_product* (RF3) 4 veces también, con las entradas correspondientes. Co

    mo el bucle for no tiene un número máximo de iteraciones, no hemos seguido comprobando el bucle para más iteraciones porque el funcionamiento es el mismo y nunca va a llegar a superar un límite.

11. **test_ok_for_almacen2**

    1_2_4_6_7_9_11_12_13_14_15_17_19_20_21_24_27_28. Caso válido que utilizamos para comprobar el funcionamiento del for del nodo 12. En este caso en el almacén de la función RF1 hay 2 pedidos, por lo que el bucle for tiene que realizar dos iteraciones para encontrar el orderID del pedido.

12. **test_ok_for_almacen4**

    1_2_4_6_7_9_11_12_13_14_15_17_19_20_21_24_27_28. Caso válido que utilizamos para comprobar el funcionamiento del for del nodo 12. En este caso en el almacén de la

función RF1 hay 4 pedidos, por lo que el bucle for tiene que realizar cuatro iteraciones para encontrar el *orderID* del pedido.

No hemos realizado un test para comprobar que no se entra el bucle del nodo 12 porque eso significaría que el almacén *store_shipping* no existe, por lo que saltaría antes la excepción del nodo 10 y no se haría este camino.

Tampoco hemos realizado casos para probar el bucle del nodo 6 porque en este bucle siempre se entra 64 veces. En caso de que el *tracking_num* tuviera una longitud distinta saltarían antes las excepciones de los nodos 3 o 5 y no se llegaría al bucle.

PYLYNT

Para que no haya advertencias de pylint hemos cambiado los siguientes parámetros en .pylyntrc:

- Good words: hemos añadido la palabra cm.
- Max module lines 1000->1600.
- Max line length 100->185.
- Max-arg 5-> 6.
- max-locals 15-> 25 (número máximo de variables locales).
- max-statemenst 50->65.
- max-branches 12->25.
- max-attributes 7->8

Hemos ejecutado pylint en la carpeta del proyecto para comprobar que no hay más errores ni warnings.

```
(venv) C:\Users\nacho\PycharmProjects\G80.2023.T04.EG3>pylint src

--------------------------------------------------------------------
Your code has been rated at 10.00/10 (previous run: 10.00/10, +0.00)
```