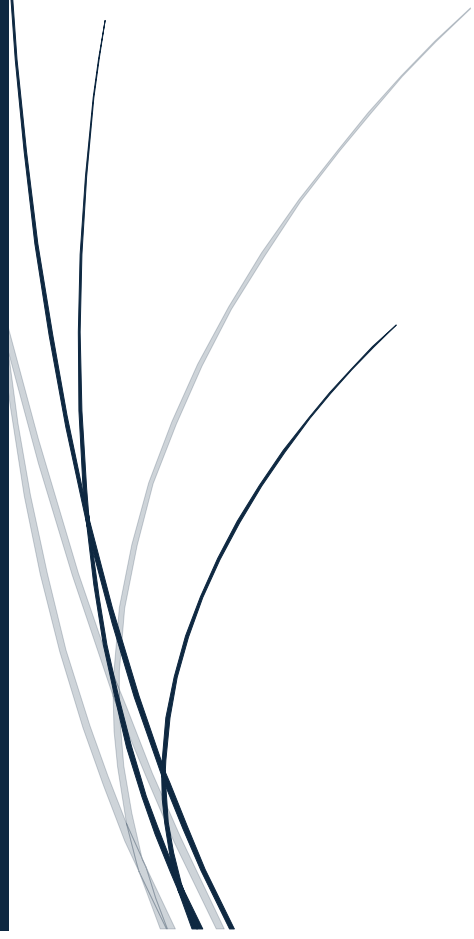
1-4-2024

EJERCICIO GUIADO 2

Eduardo Faro González - 100475239
Mario Salvador Camacho - 100475293
Ignacio Tordable Mestres - 100475359



ÍNDICE

FUNCIÓN 1: room_reservation

CLASES DE EQUIVALENCIA

1.- Número de Tarjeta de Crédito:

- CEV1: Número de tarjeta de crédito válido, cumple el algoritmo de luhn.
- CEV2: Número de tarjeta de crédito válido, son todo números.
- CEV3: Número de tarjeta de crédito válido, tiene una longitud de más de 16 dígitos.
- CEV4: Número de tarjeta de crédito válido, tiene una longitud de menos de 16 dígitos.
- CENV1: Número de tarjeta de crédito inválido, no cumple el algoritmo de luhn.
- CENV2: Número de tarjeta de crédito inválido, no son todo dígitos.
- CENV3: Número de tarjeta de crédito inválido, no tiene una longitud de 16.

2.- Número de DNI:

- CEV4: Número de DNI válido, cumple con el algoritmo necesario para crear un DNI (nift)
- CEV5: Número de DNI válido, tiene una longitud de 9 caracteres.
- CENV5: Número de DNI inválido, no cumple con el algoritmo de nift.
- CENV6: Número de DNI inválido, tiene una longitud de más de 9 caracteres.
- CENV7: Número de DNI inválido, tiene una longitud de menos de 9 caracteres.

3.- Nombre y Apellido:

- CEV6: Nombre y apellido válido, contiene todo caracteres.
- CEV7: Nombre y apellido válido, contiene un espacio en medio de dos palabras.
- CEV8: Nombre y apellido válido, contiene más de 10 y menos de 50 caracteres.
- CENV8: Nombre y apellido inválido, no contiene todo caracteres.
- CENV9: Nombre y apellido inválido, no hay un espacio separando distintas palabras.
- CENV10: Nombre y apellido inválido, hay dos o más espacios seguidos entre palabras.
- CENV11: Nombre y apellido inválido, hay un espacio en la primera posición.
- CENV12: Nombre y apellido inválido, hay un espacio en la última posición.
- CENV13: Nombre y apellido inválido, hay más de 50 caracteres.
- CENV14: Nombre y apellido inválido, hay menos de 10 caracteres.

4.- Número de Teléfono:

- CEV9: Número de teléfono válido, son todo dígitos
- CEV10: Número de teléfono válido, tiene una longitud de 9.
- CENV15: Número de teléfono inválido, no son todo dígitos.
- CENV16: Número de teléfono inválido, contiene más de 9 dígitos.
- CENV17: Número de teléfono inválido, contiene menos de 9 dígitos.

5.- Tipo de Habitación:

- CEV11: Tipo de habitación válida, single.
- CEV12: Tipo de habitación válida, double.
- CEV13: Tipo de habitación válida, suite.
- CENV18: Tipo de habitación inválida, no es ni single ni double ni suite.

6.- Fecha de Llegada:

- CEV14: Fecha de llegada válida, es un día que existe.
- CEV15: Fecha de llegada válida, el formato de la fecha es el correcto.
- CEV16: Fecha de llegada válida, la fecha de reserva es mayor a la actual.
- CENV19: Fecha de llegada válida, un mes de 31 días tiene un día de más de 31.
- CENV20: Fecha de llegada válida, tiene un día menor a 1.
- CENV21: Fecha de llegada válida, un mes de 31 días tiene un día de más de 31.
- CENV22: Fecha de llegada válida, febrero tiene un día mayor que 28, en año no bisiesto.
- CENV23: Fecha de llegada válida, febrero tiene un día mayor que 29, en año bisiesto.
- CENV24: Fecha de llegada válida, la fecha no sigue el formato establecido.
- CENV25: Fecha de llegada válida, la fecha de reserva es menor a la actual.

7.- Número de Días de Estancia:

- CEV17: Número de días válido, un dígito entre 1 y 10
- CEV18: Número de días válido, es un dígito.
- CENV26: Número de días inválido, el dígito es mayor que 10.
- CENV27: Número de días inválido, el dígito es menor que 1.
- CENV28: Número de días inválido, no es un dígito.

8.-Número de reservas:

- CEV19: Solo puede haber una reserva idéntica.
- CENV29: Hay más de una reserva igual.

9.-Localizer

- CEV20: El localizer que sale es una cadena alfanumérica
- CEV21: El localizer que sale tiene longitud 32.

VALORES LÍMITE

1.- Longitud de Número de Tarjeta de Crédito:

- VLV1: 16 caracteres
- VLNV1: 17 caracteres
- VLNV2: 15 caracteres

2.- Longitud de Número de DNI:

- VLV2: 9 caracteres
- VLNV3: 10 caracteres
- VLNV4: 8 caracteres

3.- Longitud de Nombre y Apellido:

- VLV3: 50 caracteres
- VLNV5: 50 caracteres
- VLNV6: 9 caracteres

4.- Longitud de Número de Teléfono:

- VLV4: 9 dígitos
- VLNV7: 10 dígitos
- VLNV8: 8 dígitos

5.- Fecha de Llegada:

- VLV5 día 14
- VLNV9 día 32
- VLNV10 día 0
- VLNV11 día 31
- VLNV12 día 29
- VLNV13 día 30

6.- Longitud de Número de Días:

- VLV6: 9 días
- VLNV14: 0 días
- VLNV15: 11 días

CASOS DE PRUEBA

Hemos realizado 3 casos de prueba válidos y 29 casos de prueba inválidos. Los tres válidos nos han servido para comprobar que todos los campos estén correctos, incluyendo en cada caso un tipo de habitación distinta (single, double o suite), además de comprobar un nombre con un solo apellido y con varios. En estos casos válidos hemos tenido que poner tarjeta de crédito, DNIs, nombre y apellidos y número de teléfono distintos, para que no haya dos o más reservas que sean iguales (caso inválido número 29).

En cuanto a los casos inválidos, hemos realizado cambios en cada uno de los campos para que se compruebe que se están metiendo datos que no son correctos. En el campo de la tarjeta de crédito, hemos variado la tarjeta para que no cumpla el algoritmo de luhn, sea de mayor o menor longitud a la adecuada y no sean todo números. Para el ID CARD, lo hemos variado para que la longitud sea menor o mayor y para que no cumpla el algoritmo que siguen los DNIs. En el nombre y apellidos, lo hemos variado con una longitud menor y mayor, y que no incluya todo caracteres y espacios, incluyendo dobles espacios y espacios al principio y final. En cuanto al número de teléfono se ha variado la longitud en menos y más dígitos, y el tipo de dato poniendo algún carácter. En el tipo de habitación hemos puesto premium, un tipo que no existe en este hotel, y en las fechas de llegada, fechas surrealistas que no existen como un mes con un día mayor que 31, 30, 29, 28 dependiendo del año o mes, también fechas con formatos distintos y una fecha menor a la actual. Y, por último, en el número de días hemos puesto más o menos días de los que alguien se puede alojar y un día que no contiene un dígito sino una palabra.

FUNCIÓN 2: guest_arrival

DEFINICIÓN DE LA GRAMÁTICA

Esta función recibe la ruta del archivo con los datos de la reserva, a diferencia de lo que ocurría con la función 1 que extraía los datos directamente de un fichero JSON. El fichero de entrada tiene el siguiente formato:

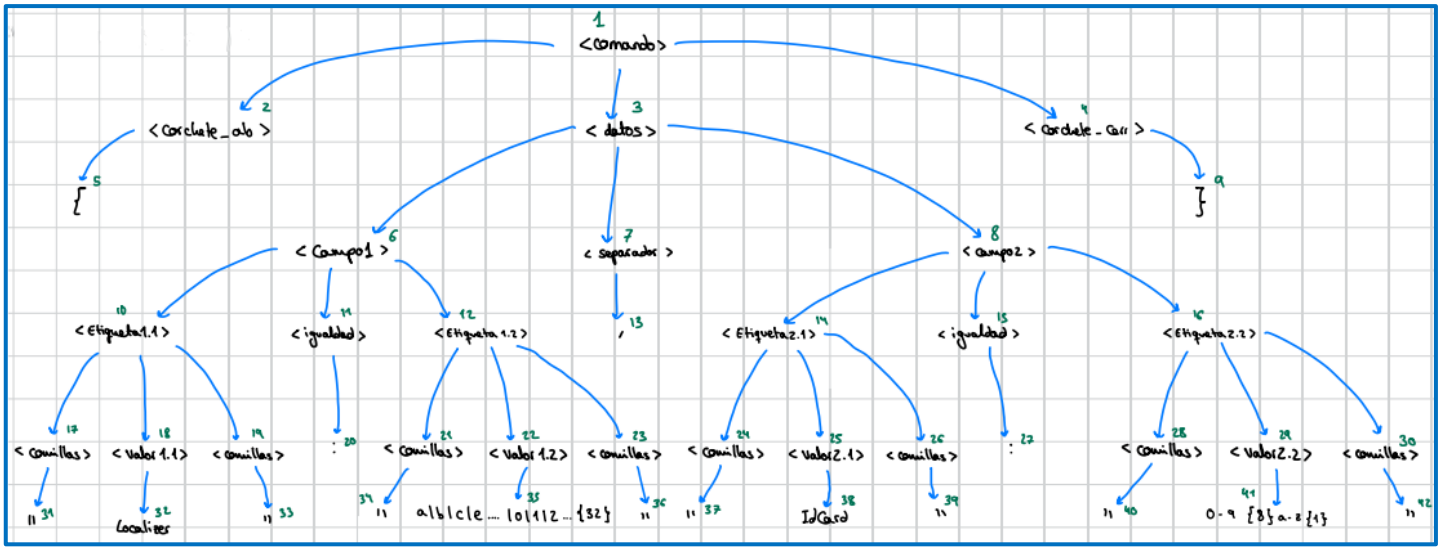
```
{"Localizer" : " String having 32 hexadecimal characters> ", "IdCard" : “valid idCard>”}
```

Para poder verificar el formato del fichero de entrada, se ha descrito la siguiente gramática:

```
<comando> ::= <corchete_ab> <datos> <corchete_cerr>

<corchete_ab>:: = {

<corchete_cerr> ::= }

<datos> ::= <campo1> <separador> <campo2>

<campo1> ::= <Etiqueta1.1> <igualdad> <Etiqueta1.2>

<campo2> ::= <Etiqueta2.1> <igualdad> <Etiqueta2.2>

<separador> :: = ,

<igualdad> ::= :

<Etiqueta1.1> ::= <comillas> <valor1.1> <comillas>

<valor1.1> ::= Localizer

<Etiqueta1.2> ::= <comillas> <valor1.2> <comillas>

<valor1.2> ::= a | b | c | e | ... | 0 | 1 | 2 ...  {32}

<comillas> ::= “

<Etiqueta2.1> ::= <comillas> <valor2.1> <comillas>

<valor2.1> ::= IdCard

< Etiqueta2.2> ::= <comillas> <valor2.2> <comillas>

<valor2.2> ::= 0 – 9 {8}  a – z {1}
```

ÁRBOL DE DERIVACIÓN

Para poder visualizar la estructura de la derivación del fichero de entrada según las reglas definidas en el apartado anterior, dibujamos un árbol de derivación. De este modo, vemos gráficamente cómo se construye la cadena a partir de símbolos terminales y no terminales.



IDENTIFICACIÓN DE LOS CASOS DE PRUEBA CASOS DE PRUEBA

La función *guest_arrival* tiene 71 casos de prueba. Éstos los hemos dividido en 4 tipos: valid, deletion, duplication y modification. Sólo hemos hecho un caso que pruebe la entrada válida. Éste debe contener un hash apto para todos los datos. Para ello, se mantiene el formato JSON y las características de cada campo como los nombres y la longitud de las claves.

```
{"Localizer": "45783ff517743faae67b33ddefa77163", "IdCard": "57899286D"}
```

Para cubrir el tipo *modification*, hemos hecho 20 casos. Estas pruebas sólo afectaban a los nodos terminales (5, 9, 13, 20, 27, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42). En cada TC[1], se modificaba el contenido del nodo, por ejemplo, el TC55 (KO – igualdad en campo1 modificado)

```
{"Localizer" = "45783ff517743faae67b33ddefa77163", "IdCard": "57899286D"}
```

[1] Test Case

A los valores *valor1.2* y *valor 2.2* le dedicamos varias pruebas que no sólo contemplaban la longitud del campo, sino que también el dominio. Es decir, si tenía algún carácter especial diferente de números o letras. Son las siguientes:

TC61 - {"Localizer": "45783ff517743faae67b3", "IdCard": "57899286D"}

TC62 - {"Localizer": "4+783ffg1bbadfaae67bdaddefadadcd", "IdCard": "57899286D"}

TC68 - {"Localizer": "45783ff517743faae67b33ddefa77163", "IdCard": "899286D"}

TC69 - {"Localizer": "45783ff517743faae67b33ddefa77163", "IdCard": "5789928AD"}

TC70 - {"Localizer": "45783ff517743faae67b33ddefa77163", "IdCard": "5789928*D"}

Los test del tipo deletion o duplication sólo afectan a los nodos no terminales (NT). Sin embargo, dependiendo de si tenían un solo hijo terminal, el caso afectaba también al nodo hijo. Se han realizado los mismos casos de prueba a los mismo de prueba. Éstas constaban de duplicar o eliminar el contenido del nodo. Incluimos el caso en el que el archivo estuviera vacío. Se definieron 50 pruebas, aquí hay algún ejemplo:

TC9 - KO(Etiqueta1.1 eliminado) - {: "45783ff517743faae67b33ddefa77163", "IdCard": "57899286D"}

TC34 - KO(Etiqueta1.1 duplicado)- {"Localizer""Localizer": "45783ff517743faae67b33ddefa77163", "IdCard": "57899286D"}

La mayoría de los errores eran tipo *"KO (JsonDecodeError)",* sin embargo, aparecieron otros:

KO (El archivo de entrada está vacío)

KO (El archivo de reservasf2 JSON está vacío (no hay datos entre las llaves)

KO (Error, el JSON contiene una clave vacía)

KO (Error, el valor asociado a la clave Localizer está vacío)

KO(Error, el valor asociado a la clave IdCard está vacío)

KO (El archivo reservasf2 tiene un fallo de escritura en alguna de las claves)

KO (IdCard contiene más de 9 caracteres),

KO(Localizer contiene menos de 32 caracteres),

KO(Localizer no contiene solo números o solo letras),

KO(IdCard contiene menos de 9 caracteres)

KO(El formato de IdCard no cumple el formato 8 dígitos y 1 letra)

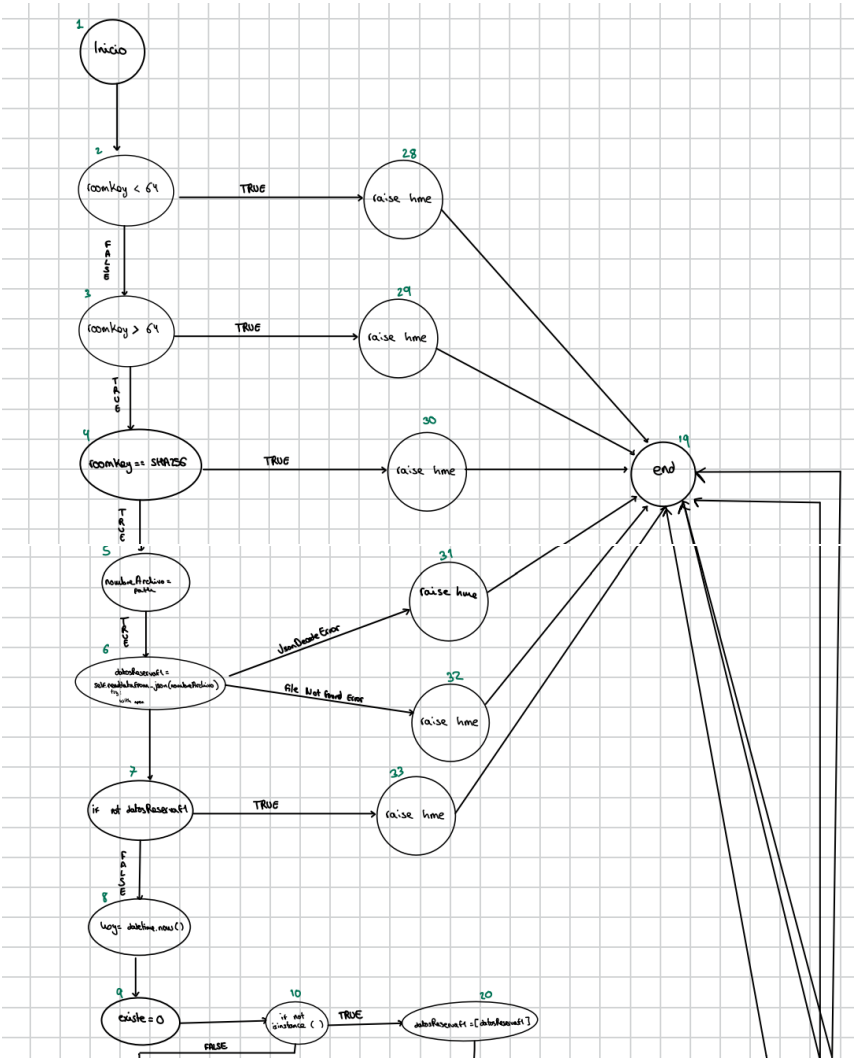
FUNCIÓN 3: guest_checkout

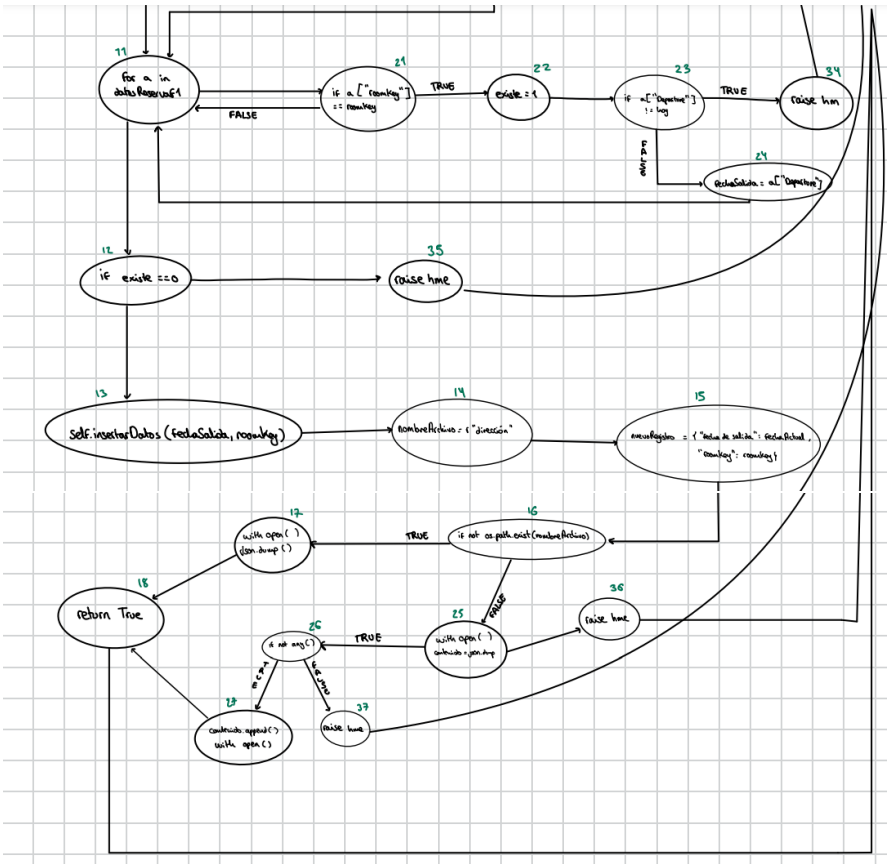
CASOS DE PRUEBA

Hemos realizado un único caso válido para comprobar que funciona la función. Este caso válido tiene que devolver true y un fichero en formato JSON que incluya una RoomKey válida (hexadecimal de 64 dígitos), y una fecha de salida válida (que sea igual a la fecha de hoy).

En cuanto a los casos no válidos hemos realizado 10. El primero comprueba que la RoomKey que se extrae del archivo de reservas2 tiene una longitud menor que 64, si es así salta el error. El 2 hace lo mismo, pero con una longitud mayor a 64, y el 3 lo mismo, pero que cumpla que está en formato SHA256. El siguiente caso comprueba que la RoomKey que nos proporciona el cliente que se va es igual a la que está almacenada en el fichero reservas2. También comprobamos que la fecha de salida sea igual a la de hoy, si no es así da error. Y, además, se comprueba que el archivo de reservas2 esté en formato JSON, no esté vacío y se encuentre en el path proporcionado, sino salta el error. Por último, comprobamos que el archivo que devolvemos (reservas3) esté en formato JSON.

GRÁFICO DE CONTROL DE FLUJO





Complejidad = aristas – nodos + 2 = 50 – 37 + 2 = 15

RUTAS BÁSICAS

Empezamos con un nodo inicial, que pasa al segundo nodo que comprueba si RoomKey es menor de 64, después pasa a si es mayor de 64 y a comprobar si sigue la regla SHA256. Luego le pasamos un archivo JSON con una ruta en el siguiente nodo, y la abre. Si no está vacía continua al siguiente, donde guarda la fecha de hoy (fecha que hemos puesto en el freeze en los test). Más tarde se mete en un if que lo que hay en datosReservaf1 lo mete en una lista y en el for va comprobando si RoomKey que hay en la lista es igual al dado, y si es así si la fecha de salida es igual a la de hoy. Cuando esto pasa guarda las variables que luego se van a imprimir (RoomKey y fecha actual), y ejecuta la función insertarDatos. Por último, esta función crea si no lo está el archivo reservas3 y le inserta los datos, si la RoomKey no es igual que otra que ya está en ese fichero. Más tarde, si todo va bien devuelve return True y va al nodo final.

PROBLEMAS ENCONTRADOS

Al principio de esta práctica tuvimos muchas dificultades con el repositorio. Uno de los integrantes del grupo no tenía el entorno de pycharm instalado correctamente y a otro el pybuilder no le funcionaba. Además, se crearon muchas ramas en el Git. Como queríamos empezar el proyecto con buen pie, decidimos borrar el repositorio y crear uno nuevo, pero esta vez estando más atentos a los *commits* de cada uno y asegurándonos que todos teníamos el entorno bien instalado.

Uno de los problemas al que nos enfrentamos en esta práctica fue el de respetar y cumplir con las reglas del pylint, ya que siempre salían errores. Para optimizar nuestro tiempo y darle la suficiente importancia al código decidimos no prestarle atención hasta que no tuviéramos todos los unitests y las funciones bien.

En el transcurso de la función *guest_arrival*, hicimos un archivo JSON que incluía toda la batería de pruebas. Sin embargo, tras ver que no éramos capaces de correr los unitests sin tener problemas, decidimos hacer 71 archivos JSON (uno para cada caso), de este modo pudimos solucionarlo. Además, nuestra gramática inicial fue demasiado corta, por lo que tuvimos que rehacerla y cambiar también el árbol de derivación. Finalmente, no prestamos atención y en el Excel todos los DNIs tenían un dígito más y todos los localizadores un número menos. Por suerte nos dimos cuenta y aplicamos todos los cambios necesarios.

Para ahorrar tiempo en la corrección de errores y en el desarrollo de los archivos de los unitests, empleamos la función de “Code With Me” de pycharm que nos permitió programar tres personas al mismo tiempo. Así no teníamos que esperar a que uno acabase para hacer un *Pull*.

Cambios finales

Al final del proyecto hemos tenido que cambiar reglas de pylint, para poder desarrollar nuestro código.

```
299  # Maximum number of boolean expressions in an if statement (see R0916).
300  #max-bool-expr=5
301  max-bool-expr=9
```

Hemos aumentado a 9 el número de boolean expressions

```
302  # Maximum number of branch for function / method body.
303  #max-branches=12
304  max-branches=40
```

Hemos cambiado el número de branches a máximo 40

```
305  # Maximum number of locals for function / method body.
306  #max-locals=15
307  max-locals=30
```

Hemos cambiado las variables locales a máximo 30

```
296  # Maximum number of attributes for a class (see R0902).
297  #max-attributes=7
298  max-attributes=9
```

Hemos cambiado el máximo de atributos a 9

Captura final del proyecto

Para poder pasar las pruebas de pybuilder, hemos tenido que cambiar el nombre de las clases de los tests, y ponerle una mayúscula al inicio de cada uno. Por lo tanto, se nos quedan tres errores más de formato camelCase. Además de los errores unable to import.