uc3m | Universidad Carlos III de Madrid

DESARROLLO SOFTWARE

Práctica Final

Grupo 50

Fecha de entrega: 23 de mayo de 2022

Ignacio Marín de la Bárcena Pérez (100432039@alumnos.uc3m.es)

Índice

I) INTRODUCCIÓN

A lo largo de este proyecto se expondrá el proceso realizado para esta práctica final. Se detallarán la modificación pedida al requisito funcional 2 (*get_vaccine_date*), así como también la implementación sobre el nuevo requisito funcional (*cancel_appointment*) en el lenguaje de programación *"Python"* y por medio de la plataforma *"Pycharm"*. En primer lugar, se exponen los cambios realizados a la función *get_vaccine_date* (*mirar punto 2*), más concretamente, validar una fecha para una futura cita, indicada por el usuario, y en caso de ser válida devolver una cita con esa fecha. A continuación, se mostrará el proceso seguido para la creación de la nueva función denominada *cancel_appointment* en la que se permite poder cancelar una cita en base a unos parámetros (*ver punto 3*).En cuarto y quinto lugar, se detalla la refactorización realizada al código (ver *punto 4*), los singleton realizados ver (*punto 4*) y los resultados sobre el *pylint* y el *pybuilder* de la práctica en su conjunto (*ver punto 5*). Finalmente, se expondrá una conclusión sobre el proyecto defendiendo las decisiones tomadas a lo largo del documento, posibles futuras mejoras sobre el mismo y las dificultades encontradas a la hora de completar la práctica.

II) MÉTODO GET_VACCINE_DATE

CÓDIGO

El método *get_vaccine_date* consiste en comprobar el *pattient_system_id* generado por *request_vaccinatio_id*. Una vez comprobado que dicho valor cumple los requisitos crea una cita para ese paciente en base a un fichero pasado como entrada y devolviendo el *date_signature* correspondiente. En base a este nuevo proyecto, se ha pedido modificar el funcionamiento de este mismo método con la intención de que ahora el usuario proponga una fecha para la cita. La fecha propuesta deberá seguir el formato *iso*, aplicando *datetime.fromisoformat* para comprobar que cumple dicho requisito. En caso de que la fecha sea posterior al día de hoy (tanto días anteriores como hoy saltará la excepción *"la fecha propuesta no es válida"*) la fecha propuesta se tomará como válida y se creará una cita para el mismo día indicado devolviendo la función el *date_signature* correspondiente. Para llevar a cabo este funcionamiento se han cambiado los parámetros de entrada del método y algunas de las funciones de la clase *VaccinationAppointment*. Finalmente indicar que se ha establecido que el día de hoy es el *"2022-03-08"* por medio de la sentencia *@freeze_time()*.

PRUEBAS

A las pruebas ya realizadas, se les han realizado una serie de cambios para que se puedan ejecutar en base a los cambios realizados. El cambio principal es el de añadir una fecha cada vez que se llamaba a *get_vaccine_date*. Se ha tomado para los test ya realizados como fecha válida *"2022-10-14"* y en fecha no válida *"2022-03-07"* poniendo cada una de las fechas en base al test. Por último, se han añadido una serie de pruebas necesarias para probar los cambios realizados (las pruebas se encuentran en el fichero *test_get_vaccine_date_tests*), siguiendo el método de clases de equivalencia y valores límite. Algunas de las posibles pruebas a realizar se consideran que ya se han realizado en test anteriores.

III) MÉTODO CANCEL_APPOINTMENT

CÓDIGO

El funcionamiento de la función pedida consiste en comprobar si un cliente posee una cita de vacunación válida para poder cancelarla. Esto se realiza por medio de un fichero de entrada con el *date_signature* del paciente, el tipo de cancelación (*cancelation_type*) y la razón (*reason*) de porqué se cancela. Parámetros que se validan, y en caso de que sean correctos se procede a revisar que el paciente exista, tenga citas asignadas para un día posterior al de hoy (se pueden cancelar también las citas el mismo día) y que no se haya vacunado, es decir, no asistió a la cita dada o a otra. Una vez cancelada la cita el método devuelve el *date_signature* provisto, no obstante en el almacén se guardan los parámetros de entrada, es decir el date_signature, el tipo de cancelación y la razón de la misma

PRUEBAS

Los métodos seguidos para comprobar el correcto funcionamiento de está nueva función implementada (*cancel_appointment*) son los siguientes: para el fichero que se pasa por entrada se ha seguido el método de análisis sintáctico, que verificará que los valores introducidos son correctos. Por lo tanto, para representar los casos de prueba realizados, se mostrará el dominio de los distintos parámetros contenidos dentro de cada gramática (*ver dominio*), así como su árbol de derivación (*ver imagen*). Por otro lado, se ha seguido el método de pruebas estructurales para probar el ciclo de procesamiento de la función y comprobar que todos los posible caminos de la función (*ver diagrama de flujo de control*). Finalmente, se completarán las pruebas con las clases de equivalencia y valores límites necesarios. Todas las pruebas mencionadas se recogen en el fichero de prueba *test_cancel_appointment_tests*.

- DOMINIO

```
{“date_signature”:”Valor1”, ”cancelation_type”:”Valor2”,
”reason”:”Valor3”}
Fichero ::= Inicio_objeto Datos Fin_objeto
Inicio_objeto ::= {
Fin_objeto ::= }
Datos ::= Campo1 Separador Campo2 Separador Campo3
Campo1 ::= Etiqueta_dato1 Igualdad Valor_dato1
Campo2 ::= Etiqueta_dato2 Igualdad Valor_dato2
Campo3 ::= Etiqueta_dato3 Igualdad Valor_dato3
Separador ::= ,
Igualdad ::= :

Etiqueta_dato1 ::= Comillas Valor_etiqueta1 Comillas
Valor_etiqueta1 ::= date_signature
Valor_dato1 ::= Comillas Valor1 Comillas
Valor1 ::= a|b|c|d|e|f|0|1|2|3|4|5|6|7|8|9 {64}
```

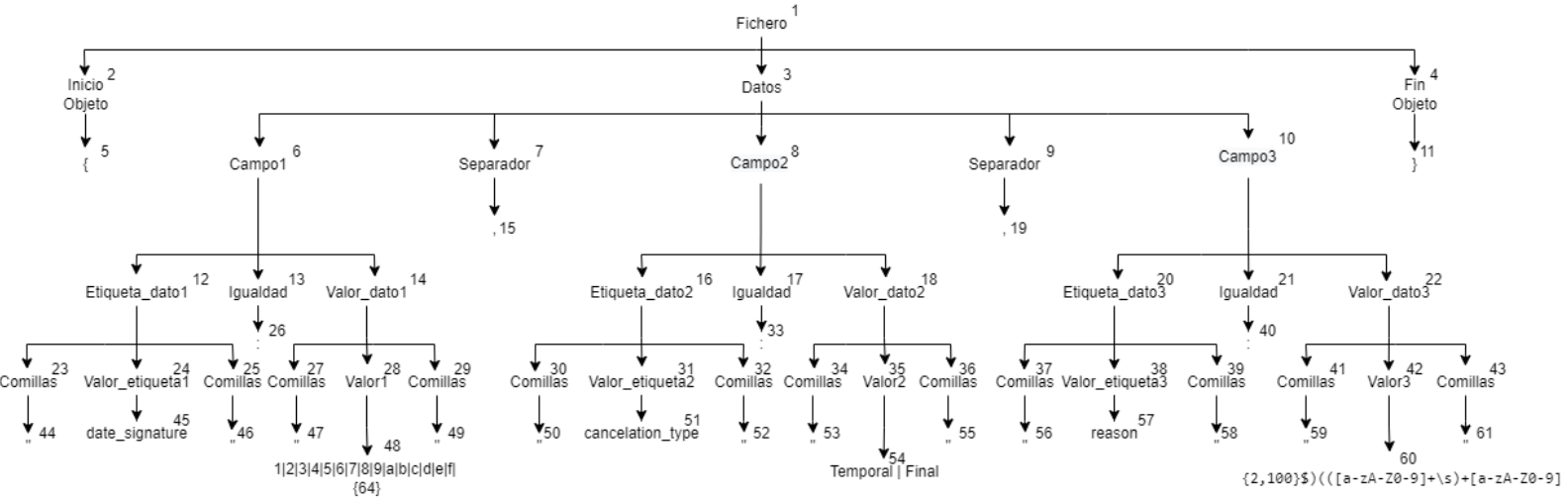
```
Comillas ::= "|"

Etiqueta_dato2 ::= Comillas Valor_etiqueta2 Comillas
Valor_etiqueta2 ::= cancelation_type
Valor_dato2 ::= Comillas Valor2 Comillas
Valor2 ::= Temporal|Final

Etiqueta_dato3 ::= Comillas Valor_etiqueta3 Comillas
Valor_etiqueta3 ::= reason
valor_dato3 ::= Comillas Valor3 Comillas
Valor3 ::= {2,100}$)(([a-zA-Z0-9]+\s)+[a-zA-Z0-9]+)
```

Se expresa el *"Valor3"* por medio de la expresión regex empleada en el código

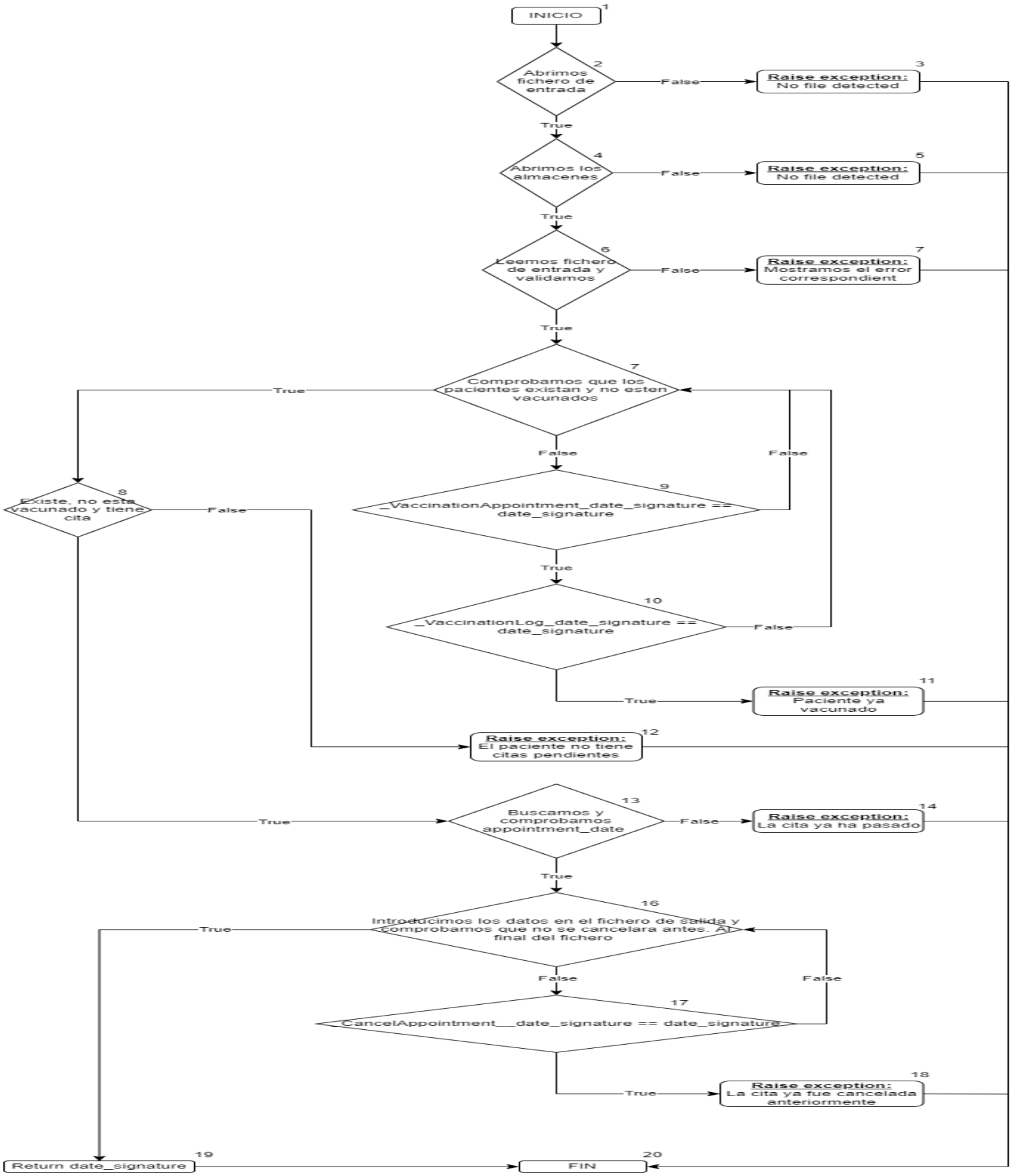
- ÁRBOL



Se expresa el *"Valor3"* por medio de la expresión regex empleada en el código

Una vez realizado el árbol, procedemos a realizar un test asociado a cada nodo (61 nodos en total). Aquellos nodos que denominamos no terminales, nodos que tienen nodos hijos, las pruebas asociadas serán de borrado y duplicado. Mientras que las pruebas para los nodos terminales, serán de borrado, duplicado y modificado. En función del fallo dado, se mostrarán los siguientes mensajes de error por pantalla: *No file detected* (si no se le pasa el fichero por pantalla), *JSON DECODE ERROR - Wrong JSON Format* (si existe un error de formato en el fichero de entrada), una de las etiquetas es inválida o no se encuentra (la explicación es el mismo mensaje de error) y si uno de los valores introducidos no se encuentra o es invalido (*"cancelation_type is not valid"* o *"reason is not included"*). Cabe mencionar que todos aquellos nodos que sigan el mismo esquema, se harán sus pruebas de *DUPLICATION* y *DELETION* juntas. Esto afecta a los *separadores*, a los *etiquetas_datox*, a las *igualdades*, al *valor_datox* y a las *comillas* (separadas en inicio y fin tanto de las etiquetas como de los valores).

- DIAGRAMA DE FLUJO DE CONTROL



A partir del diagrama anterior, las pruebas se destinarán a probar todos los caminos no probados en el resto de pruebas y que se puedan extraer del diagrama de flujo de control. Por cada "*for"* creado se pasará 0, 1 y 2 veces. Se observan diferentes caminos con 20 nodos, dónde algunos de los caminos ya se han probado en pruebas anteriores.

- CLASES DE EQUIVALENCIA Y VALORES LÍMITE

Finalmente, se ha aplicado el procedimiento de valores límites y clases de equivalencia para probar aquellos casos que no se hubieran ya probado con los test del árbol de derivación y del diagrama de flujo. Cabe destacar que muchos de los casos propuestos ya se habían realizado en pruebas anteriores.

IV) REFACTORIZACIÓN Y SINGLETON

REFACTORIZACIÓN

La refactorización sobre la modificación sobre el método *get_vaccine_date*, ha sido mínima debido a la composición del mismo método. La función ya se encontraba factorizada por lo que se ha trabajado sobre esas líneas de código. Respecto a la creación de la nueva funcionalidad se compuso inicialmente en el fichero *vaccine_manager.py*. Una vez comprobada el correcto funcionamiento del método se realizado su refactorización siguiendo los métodos aprendidos a lo largo de la última práctica, creándose para esta nueva funcionalidad dos ficheros (*cancel_appointment.py* y *cancel_appointment_store.py*) para su debida refactorización así como los *attributes* necesarios.

Respecto al resto de ficheros empleados a lo largo del proyecto se han llevado a cabo pequeñas mejoras como indicar el tipo de parámetro que reciben las diferentes funciones, eliminación de posibles líneas de código no usadas y la mejora de las palabras y número mágicos.

SINGLETON

Se ha creado un patrón *singleton* para las siguientes clases: *AppointmentJsonStore*, *CancelAppointmentJsonStore*, *PatientJsonStore*, *VaccinationJsonStore* y *VaccineManager*. A su vez se ha probado el patrón singleton creado. El código empleado es el mismo que se encontraba en el código base tanto a la hora de la implementación como en las pruebas.

Por otro lado, una posible mejora a la hora de crear este mismo patrón y reducir de carga a las clases ya mencionadas. Sería crear un fichero "singleton" al que hicieran referencia las clases y eliminar así posible código duplicado y mejorar la organización del proyecto en si.

El código mencionado sería el siguiente:

```
class SingletonMeta(type):
     _instances {}
     def __call__(cls):
     if cls not in cls._instances:
          instance = super().__call__()
          cls._instances[cls] = instance
     return cls._instances[cls]

class Singleton(metaclass=SingletonMeta):
     def some_business_logic(self):
          pass

if __name__ == "main":
     s1 = Singleton()
     s2 = Singleton()
     if id(s1) == id(s2):
          print("Singleton works, both variables contain the
          same instance")
     else:
          print("Singleton failed, variables contain different
          instances")
```

La razón por la cual no se ha implementado dicho código, y que se podría implementar en futuras versiones del proyecto. Se debe a causa de la cobertura que se realiza sobre el mismo a la hora de probarlo. La parte de código probado sobre este nuevo fichero sería del 62% contrastando con el 90% probado en el resto del código del proyecto (*ver punto 5.1*). Esto supone que al ejecutar el comando *pyb* sobre el proyecto en el terminal resulte en error.

V) PYBUILDER Y PYLINT

PYBUILDER

A lo largo del proyecto se han llevado a cabo diferentes comprobaciones a medida que se implementaba nuevo código. Todas esas pruebas se realizaron por medio de ejecutar los comando *pyb* y *pyb run_unit_tests* en el terminal. Finalmente, tras la realización del código y de los test sobre las distintas partes del código se ejecutaron una vez más para observar el porcentaje de código cubierto. Se obtuvo el siguiente resultado:

```
[INFO]  Executed 102 unit tests
[INFO]  All unit tests passed.
[INFO]  Overall pybuilder.plugins.python.unittest_plugin.run_unit_tests coverage is 92%
[INFO]  Overall pybuilder.plugins.python.unittest_plugin.run_unit_tests branch coverage is 92%
[INFO]  Overall pybuilder.plugins.python.unittest_plugin.run_unit_tests partial branch coverage is 92%
[INFO]  Overall GE3_2022 coverage is 92%
[INFO]  Overall GE3_2022 branch coverage is 92%
[INFO]  Overall GE3_2022 partial branch coverage is 92%
[INFO]  Building binary distribution in c:\users\imbpe\pycharmprojects\g50.2022.t05b.fp\target\dist\ge3_2022-1.0.dev0
[INFO]  Running Twine check for generated artifacts
------------------------------------------------------------
BUILD SUCCESSFUL
------------------------------------------------------------
```

PYLINT

El código empleado a lo largo del proyecto sigue la normativa *PEP-8*, para confirmar dicho cumplimiento se han realizado diferentes escaneos por medio de la sentencia *pylint uc3m_care* en la terminal con la idea de comprobar la correcta redacción del código en los distintos ficheros. Tras la implementación del código se obtuvo el siguiente resultado sobre la correcta expresión del lenguaje usado:

```
--------------------------------------------------------------------
Your code has been rated at 9.40/10 (previous run: 9.40/10, +0.00)
```

Por otro lado, se realizó un escaneo inicial del código, para ver que supuso la modificación e implementación de funcionalidades. El resultado provisionado en esa primera ejecución fue de “9,95/10” (observar imagen). Lo que supone una disminución de 0,55 sobre la puntuación inicial.

```
--------------------------------------------------------------------
Your code has been rated at 9.95/10 (previous run: 9.95/10, +0.00)
```

Para concluir, se mejoró parte del código. Lo más relevante fue la incorporación de las siguientes líneas de código al fichero *.pylintrc* (ver código) para deshabilitar los docstring, así como eliminar código no usado o respetar la longitud de las líneas.

```
# Desactivar docstring modulos, clases y funciones.
disable =
   C0114,   # modulos
   C0115,   # clases
   C0116,   # funciones
```

```
--------------------------------------------------------------------
Your code has been rated at 9.96/10 (previous run: 9.94/10, +0.02)
```

En la imagen anterior se puede observar el resultado final sobre el escaneo del *pylint* del código empleado a lo largo del proyecto.

VI) CONCLUSIÓN

A pesar de implementar la nueva funcionalidad pedida y modificar una de las ya existentes en base a lo indicado en el enunciado. Se considera que dicho proyecto se podría mejorar de cara al futuro añadiendo nuevas funcionalidades o mejorando las ya implementadas, como por medio de separar las cancelaciones en dos posibles almacenes (*"temporal" y "final"*) o que en un mismo almacén se realizará una comprobación tanto del *date_signature* como del tipo de cancelación permitiendo así reactivar en un futuro aquellas que sea temporales. Otra posible mejora sería eliminar del almacén de citas aquellas que ya hayan transcurrido (el paciente se ha vacunado, se ha cancelado de forma final o la fecha de la cita ya ha pasado). Finalmente, sobre el patrón singleton se considera que se podría implementar lo mencionado anteriormente, siempre y cuando se consiga probar al menos el 70% del posible fichero singleton creado.

Esta práctica me ha servido para poner a prueba todos los conocimientos adquiridos a lo largo del curso y que se han explicado en clase. En base al código a implementar se considera que ha seguido el nivel de complejidad de los ejercicios guiados permitiéndome asentar los conocimientos adquiridos además de servirme como una reducida experiencia sobre a lo que me enfrentaré como futuro ingenieros informáticos.