PRÁCTICA FINAL

DESARROLLO DE SOFTWARE

Sonsoles Molina Abad – 100432073

Lorenzo Largacha Sanz – 100432129

Doble Grado en Ingeniería Informática y ADE – Grupo 50 – Equipo 11

Desarrollo de Software

Universidad Carlos III de Madrid, Campus de Colmenarejo

23 mayo, 2022

Tabla de contenido

1. INTRODUCCIÓN

El objetivo de esta última entrega es poner en práctica todos los conocimientos adquiridos a lo largo de la asignatura de Desarrollo de Software. Para ello, nos centraremos en especificar, diseñar, programar y probar un componente de software simple. Para mantener la continuidad con las anteriores prácticas, se pretende incluir nuevas funcionalidades y modificar las existentes de un sistema de gestión de vacunas y citas llamado UC3MCare. Además de la presente memoria, también hemos elaborado un documento Excel que incluye la definición de los casos de prueba.

2. FUNCIÓN OBTENER CITA PARA VACUNARSE

Con este primer requisito funcional se pretende añadir la posibilidad de indicar una fecha propuesta para generar la cita de vacunación. Para ello se incluirá un parámetro *date* en el método get_vaccine_date, de la clase VaccineManager, con la fecha en formato "YYYY-MM-DD". En caso de que esta fecha tenga el formato esperado y sea posterior a la fecha actual se registrará la cita para vacunarse, y en otro caso se lanzará una excepción. A continuación, explicaremos el proceso para incluir la nueva funcionalidad en el método get_vaccine_date.

En primer lugar, definimos los casos de prueba necesarios para evaluar la nueva funcionalidad (en la clase TestGetVaccineDate), que nos servirán para implementar posteriormente el código necesario para que dichos test se validen.

2.1. Modificación test_get_vaccine_date_ok

En el método test_get_vaccine_date_ok, añadimos el parámetro date a la llamada al método get_vaccine_date para probar que recibe y utiliza la fecha para generar la cita. También añadimos el parámetro en el resto de las llamadas al método get_vaccine_date desde los otros test, tanto desde esta clase como desde las otras clases de test que lo utilizan. El parámetro *date* será una fecha válida (string de 10 caracteres) con formato correcto "YYYY-MM-DD" y siendo la fecha superior a la actual. Además, comprobaremos que la fecha almacenada en el fichero store_date se corresponde con la fecha de entrada del método. Parámetro *date*: "2022-03-18".

2.2. Modificación test_get_vaccine_date_no_ok_parameter

En el método test_get_vaccine_date_no_ok_parameter, añadimos en la lista de parámetros param_list_nok que recibe el método, la fecha propuesta para los test ya existentes; y además implementamos los siguientes nuevos test para comprobar que la fecha recibida sigue el formato esperado y que es posterior al día actual. En todos los nuevos test pasaremos como *input_file* el fichero test_ok.json y lo que cambiará será la fecha de la cita.

- Test no válido, en el que el parámetro *date* es un string con más de 10 caracteres. *date*: "2022-03-188", excepción esperada: "wrong date format".
- Test no válido, en el que el parámetro *date* es un string con menos de 10 caracteres. *date*: "2022-03-1", excepción esperada: "wrong date format".
- Test no válido, en el que el parámetro *date* no sigue la estructura año-mes-día. *date*: "18-03-2022", excepción esperada: "wrong date format".
- Test no válido, en el que el parámetro *date* utiliza separadores distintos de "-". *date*: "2022/03/18", excepción esperada: "wrong date format".
- Test no válido, en el que el parámetro *date* incluye un mes fuera de rango. *date*: "2022-13-18", excepción esperada: "wrong date format".
- Test no válido, en el que el parámetro *date* incluye un día fuera de rango. *date*: "2022-03-32", excepción esperada: "wrong date format".
- Test no válido, en el que el parámetro *date* incluye un año fuera de rango. *date*: "0000-03-18", excepción esperada: "wrong date format".
- Test no válido, en el que el parámetro *date* es un string con una fecha igual o inferior a la fecha actual. *date*: "2022-03-07", excepción esperada: "date should be after today". La fecha actual fijada para los test mediante freezetime es 2022-03-08.

2.3. Código implementado

Tras generar los test anteriores, procedemos a implementar el código que haga buenos nuestros test. Para ello, en primer lugar, añadimos el parámetro date al método get_vaccine_date que recibe un string con la fecha propuesta para la cita. Asimismo, le pasamos este parámetro al método create_appointment_from_json_file de la clase VaccinationAppointment, que a su vez llamará al constructor de la clase, por lo que también debemos introducir en él el parámetro. Creamos una nueva clase AppointmentDate, hija de Attribute, que se encargará de validar la fecha, y así evitamos tener que refactorizar después. En esta clase, sobrescribimos el método _validate para que genere un objeto tipo datetime con el método fromisoformat, y así verificar el formato de la fecha, y después comprobamos que la fecha sea posterior al día actual (pasando de timestamp a número de días). Si no se cumple alguna de estas condiciones, lanzamos una excepción.

Además, hemos tenido que modificar el método get_appointment_from_date_signature, también dentro de la clase VaccinationAppointment, para que, al llamar al constructor de la clase, envíe como parámetro la fecha de la cita en formato string.

3. FUNCIÓN ANULAR UNA CITA

El segundo requisito funcional consiste en implementar la posibilidad de anular una cita de vacunación. Para ello, crearemos un nuevo método llamado cancel_appointment, que recibirá como entrada un fichero Json con una clave de la cita (*date_signature*), un tipo de cancelación (*cancelation_type*) pudiendo ser temporal o final, y una razón para cancelar la cita (*reason*). Consideramos que será oportuno añadir un nuevo atributo a la clase VaccinationAppointment para poder registrar si una cita ha sido cancelada. A continuación, explicaremos las técnicas de Test Driven Development empleadas para incluir la nueva función en la clase VaccineManager.

3.1. Análisis de clases de equivalencia y valores límite

Para definir correctamente los casos de prueba, en primer lugar, haremos uso de la técnica de análisis de clases de equivalencia y valores límite, considerando todos los posibles rangos de valores de entrada y salida, además de su semántica.

Clases de equivalencia

El número de valores de entrada en todas las clases de equivalencia será uno, puesto que nuestros métodos solo se prueban con un paciente o cita de vacunación cada vez. A continuación, explicamos las clases de equivalencia identificadas:

CE_V_01 → Se trata de una clase de equivalencia válida, en la que la ruta del fichero recibido como parámetro (*input_file*) existe.

CE_NV_02 → Se trata de una clase de equivalencia no válida, en la que la ruta del fichero recibido como parámetro (*input_file*) no existe.

CE_V_03 → Clase de equivalencia válida en la que el fichero (*input_file*) tiene una sintaxis Json correcta.

CE_NV_04 → Clase de equivalencia inválida en la que el fichero (*input_file*) tiene una sintaxis Json incorrecta.

CE_V_05 → Clase de equivalencia válida, dónde todas las entradas tienen un único campo *date_signature* con una cadena hexadecimal de 64 caracteres.

CE_NV_06 → Clase inválida, donde todas las entradas tienen una *date_signature* inválida, ya que no se encuentra en formato hexadecimal.

CE_NV_07 → Se trata de una clase de equivalencia no válida, donde todas las entradas tienen una *date_signature* inválida, ya que no tiene 64 caracteres.

CE_NV_08 → Se trata de una clase de equivalencia no válida, donde el fichero no cuenta el campo *date_signature*.

CE_NV_09 → Se trata de una clase de equivalencia no válida, donde el fichero cuenta con más de un campo *date_signature*.

CE_V_10 → Se trata de una clase válida, donde todas las entradas tienen un único campo *cancelation_type* con el string "Temporal".

CE_V_11 → Se trata de una clase válida, donde todas las entradas tienen un único campo *cancelation_type* con el string "Final".

CE_NV_12 → Se trata de una clase inválida, donde todas las entradas tienen como *cancelation_type* cualquier otro string.

CE_NV_13 → Se trata de una clase de equivalencia no válida, donde el fichero no cuenta el campo *cancelation_type*.

CE_NV_14 → Se trata de una clase de equivalencia no válida, donde el fichero cuenta con más de un campo *cancelation_type*.

CE_V_15 → Se trata de una clase válida, donde todas las entradas tienen un único campo *reason* con un string de 2 a 100 caracteres.

CE_NV_16 → Se trata de una clase inválida, donde todas las entradas tienen como *reason* un string con con más de 100 caracteres.

CE_NV_17 → Se trata de una clase inválida, donde todas las entradas tienen como *reason* un string con menos de 2 caracteres.

CE_NV_18 → Se trata de una clase de equivalencia no válida, donde el fichero no cuenta el campo *reason*.

CE_NV_19 → Se trata de una clase de equivalencia no válida, donde el fichero cuenta con más de un campo *reason*.

En cuanto a las salidas, el método devuelve una cadena que representa el *date_signature* de la cita que acaba de ser anulada, por lo que el número de valores de salida también será uno. Además, los datos se irán almacenando en un fichero, por lo que también lo tendremos que comprobar.

CE_V_20 → Se trata de una clase válida, donde todas las salidas son el *date_signature* de la cita que acaba de ser anulada (cadena en formato SHA-256).

CE_NV_21 → Se trata de una clase inválida, donde todas las salidas serían una cadena que no sigue el formato hexadecimal SHA-256. No es posible crear un test que contemple esta clase de equivalencia, puesto que el método no generará ninguna cadena "incorrecta", simplemente mostrará una excepción si algún dato de la entrada era incorrecto.

CE_V_22→ Se trata de una clase válida, donde el fichero que almacena los datos de las citas (store_date) ha sido modificado correctamente.

CE_NV_23 → Se trata de una clase inválida, donde los datos introducidos no son correctos, y por tanto no se modifican los datos de la cita en el fichero, es decir, el fichero store_date no cambia.

CE_NV_24 → Se trata de una clase inválida, donde los datos introducidos son correctos, pero no se modifica el fichero. Este caso solo se dará si la cita ya se había cancelado anteriormente como Temporal y se intenta volver a cancelar con ese mismo tipo, o si se había cancelado como Final y se intenta anular con cualquier tipo de cancelación.

CE_NV_25 → Se trata de una clase inválida, donde los datos introducidos no son correctos, pero se modifica el fichero. Este caso no se va a dar nunca, ya que, si los datos son incorrectos, no llegarán a almacenarse, por lo que no es posible comprobarlo con un test (funcionamiento definido en el propio método).

CE_NV_26 → Clase inválida en la que se lanza una excepción cuando el fichero de entrada (*input_file*) no se encuentra.

CE_NV_27 → Clase inválida en la que se lanza una excepción cuando el fichero de entrada (*input_file*) no tiene una estructura correcta, es decir, los campos no son los esperados.

CE_NV_28 → Clase inválida en la que se lanza una excepción cuando el fichero de entrada (*input_file*) incluye campos con valores incorrectos.

CE_NV_29 → Clase inválida en la que se lanza una excepción cuando el fichero de entrada (*input_file*) no tiene una sintaxis Json correcta.

Valores límite

Puesto que la mayoría de los errores se producen en los valores límite de las clases de equivalencia, analizamos cuales son los límites de los intervalos de cada una de las clases identificadas.

date_signature → Dado que debe ser un string de 64 caracteres, los valores límite serán 63, 64 y 65 caracteres. Y respecto al número de campos en el fichero, los valores límite serán 0, 1 y 2 campos.

cancelation_type → Respecto al número de campos en el fichero, los valores límite serán 0, 1 y 2 campos. No existen más valores límite ya que solo puede ser un string del tipo “Temporal” o “Final”.

reason → Los valores límite serán 1, 2, 3, 99, 100 y 101 caracteres, ya que el número debe estar entre 2 y 100. Y respecto al número de campos en el fichero, los valores límite serán 0, 1 y 2 campos.

A partir de estas clases de equivalencia y valores límite desarrollamos los test, que hemos incluido en la primera y segunda hoja del documento Excel.

3.2. Análisis sintáctico

Continuando con las técnicas de TDD, haremos uso de la técnica de análisis de análisis sintáctico, de modo que podamos analizar con el menor número de pruebas posible todas las alternativas de los datos de entrada.

Gramática

En primer lugar, procedemos a modelar la gramática de las entradas. La función cancel_appointment recibe como parámetro un fichero Json (*input_file*) con los datos para cancelar una cita de vacunación. Dicho archivo debe seguir el siguiente formato:

```
{
    "date_signature":"5a06c7bede3d584e934e2f5bd3861[…]c",
    "cancelation_type":"Temporal",
    "reason":"motivo para cancelar"
}
```

Definición de la gramática de entrada

fichero ::= llave_ini datos llave_fin

llave_ini ::= {

llave_fin ::= }

datos ::= campo_1 separador campo_2 separador campo_3

separador ::= ,

campo_1 ::= etiqueta_dato1 igualdad valor_dato1

campo_2 ::= etiqueta_dato2 igualdad valor_dato2

campo_3 ::= etiqueta_dato3 igualdad valor_dato3

igualdad ::= :

etiqueta_dato1 ::= comillas valor_etiqueta1 comillas

valor_etiqueta1 ::= date_signature

valor_dato1 ::= comillas valor1 comillas

valor1 ::= [0-9A-Fa-f] {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

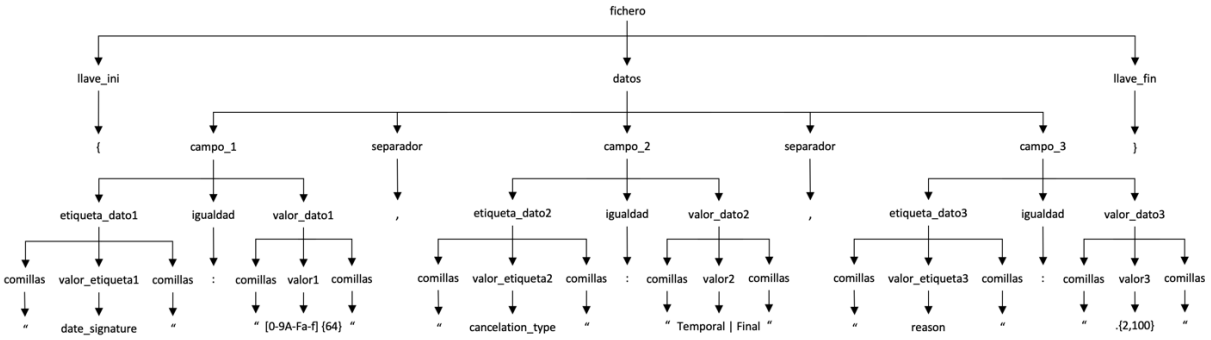
valor_etiqueta3 ::= reason

valor_dato3 ::= comillas valor3 comillas

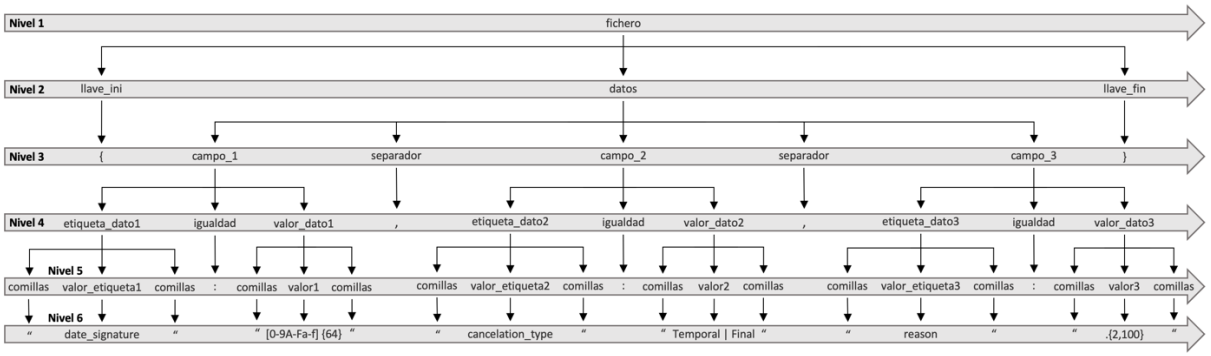
valor3 ::= .{2,100}

Árbol de derivación

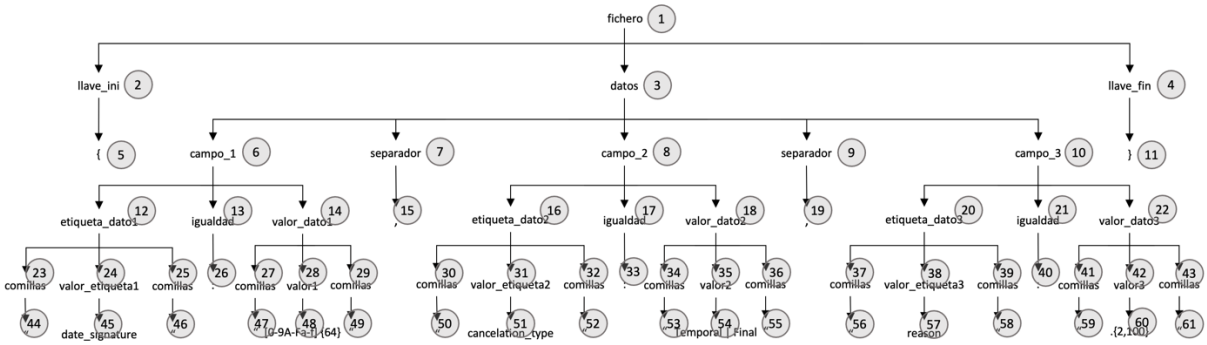
Mediante el siguiente árbol de derivación recogemos toda la gramática definida anteriormente, ordenada en diferentes niveles, y con los nodos numerados.



Niveles



Nodos numerados



A partir de este árbol de derivación, definimos los test de la hoja 3 del documento Excel.

3.3. Casos de prueba adicionales

test_nok_108 (parametrizado)

Añadimos un test para comprobar si se lanza una excepción tipo "Estructura JSON incorrecta" cuando el fichero de entrada tiene un campo más separado por una coma, con una etiqueta y un valor diferentes, y siguiendo un formato Json correcto. Fichero: test_nok_108.json

test_nok_109_no_store_date

Añadimos un test para comprobar que se produce un error de procesamiento interno cuando el fichero store_date no existe. La excepción esperada será "date_signature is not found", puesto que cuando se intenta buscar la clave de la cita, el *_data_list* del store_date estará vacío. Para ello borramos los stores, solicitamos una cancelación de la cita, comprobamos la excepción recibida y que el fichero store_date no exista.

test_ok_110_temporal_final_cancels

Añadimos un test para comprobar que se puede cancelar una cita de forma Temporal en primer lugar, y posteriormente de forma Final. Para ello hacemos dos solicitudes de cancelación, comprobamos la date_signature y que se modifican los datos en store_date.

test_nok_111_final_temporal_cancels

Añadimos un test para comprobar que se lanza una excepción tipo "Cita ya cancelada de forma Final" cuando intentamos cancelar una cita de forma Temporal, que ya se había cancelado antes de forma Final. Para ello hacemos dos solicitudes de cancelación, y comprobamos la excepción recibida.

test_nok_112_final_final_cancels

Añadimos un test para comprobar que se lanza una excepción tipo "Cita ya cancelada de forma Final" cuando intentamos cancelar una cita de forma Final, que ya se había cancelado antes de la misma forma. Para ello hacemos dos solicitudes de cancelación, y comprobamos la excepción recibida.

test_nok_113_appointment_cancelled

Añadimos un test para comprobar que, cuando un paciente que ha cancelado su cita se intenta vacunar, no se puede vacunar. Se lanzará la excepción de tipo "La cita para la que se intenta vacunar ha sido cancelada" y no se llega a crear el fichero store_vaccine.

3.4. Implementación del método cancel_appointment

En primer lugar, añadimos un nuevo atributo *appointment_status* a la clase VaccinationAppointment para poder registrar si una cita ha sido cancelada. Este podrá tener los valores "Active", "Cancelled Temporal" o "Cancelled Final". Para asegurar que solo puede tomar esos valores incluimos una clase AppointmentStatus, hija de Attribute, que lo valide.

El nuevo método cancel_appointment se encontrará en la clase VaccineManager y recibirá como parámetro un fichero Json con los datos de cancelación. Este método llamará al método modify_appointment_from_json_file de la clase VaccinationAppoinment.

Creamos la clase CancelationJsonParser, hija de JsonParser, para abrir el fichero, guardar el diccionario de datos, y validar sus keys, que serán los nombres de cada etiqueta.

Después comprobamos la estructura del fichero Json, para asegurar que haya una solicitud de cancelación que no contenga más de tres campos.

A continuación, validamos los valores de cada campo. Para ello, utilizamos el método ya existente de la clase DateSignature para validar date_signature, y añadimos las clases CancelationType y Reason como las clases hijas de Attribute, para validar el valor del tipo de cancelación y el motivo.

Una vez validado el fichero de entrada, llamamos al método get_appointment_from_date_signature para buscar si la cita introducida existe y crear un objeto tipo VaccinationAppoinment. Si es así, modificamos el campo *appointment_status* del objeto y del fichero store_date para esa cita. Para ello, creamos un método llamado modify_appointment_status en la clase VaccinationAppointment.

El método modify_appointment_status comprueba que la fecha de la cita no haya pasado y que aún no se haya administrado la vacuna. Posteriormente, si la cita está activa, modifica el *appointment_status* a "Cancelled Temporal" o a "Cancelled Final". Si la cita ya se había cancelado de forma Temporal solo permitirá cancelarla

de forma Final, y si ya se había cancelado de forma Final no permitirá volverla a cancelar (lanzando una excepción).

Por último, llamamos al método modify_store_date para modificar el fichero store_date. Creamos un método llamado update_item en la clase JsonStore, que primero carga los datos del fichero en *_data_list* mediante el método load, después busca el item antiguo que queremos borrar con el método find_item, elimina dicho item de *_data_list* y añade el nuevo item recibido como parámetro, y guarda *_data_list* en el fichero con el método save.

Además, hemos realizado un cambio en el método get_appointment_ from_date_signature para que cuando genere un objeto tipo Vaccination Appoinment, pase como parámetro al constructor de la clase el *appointment_status* actual de la cita.

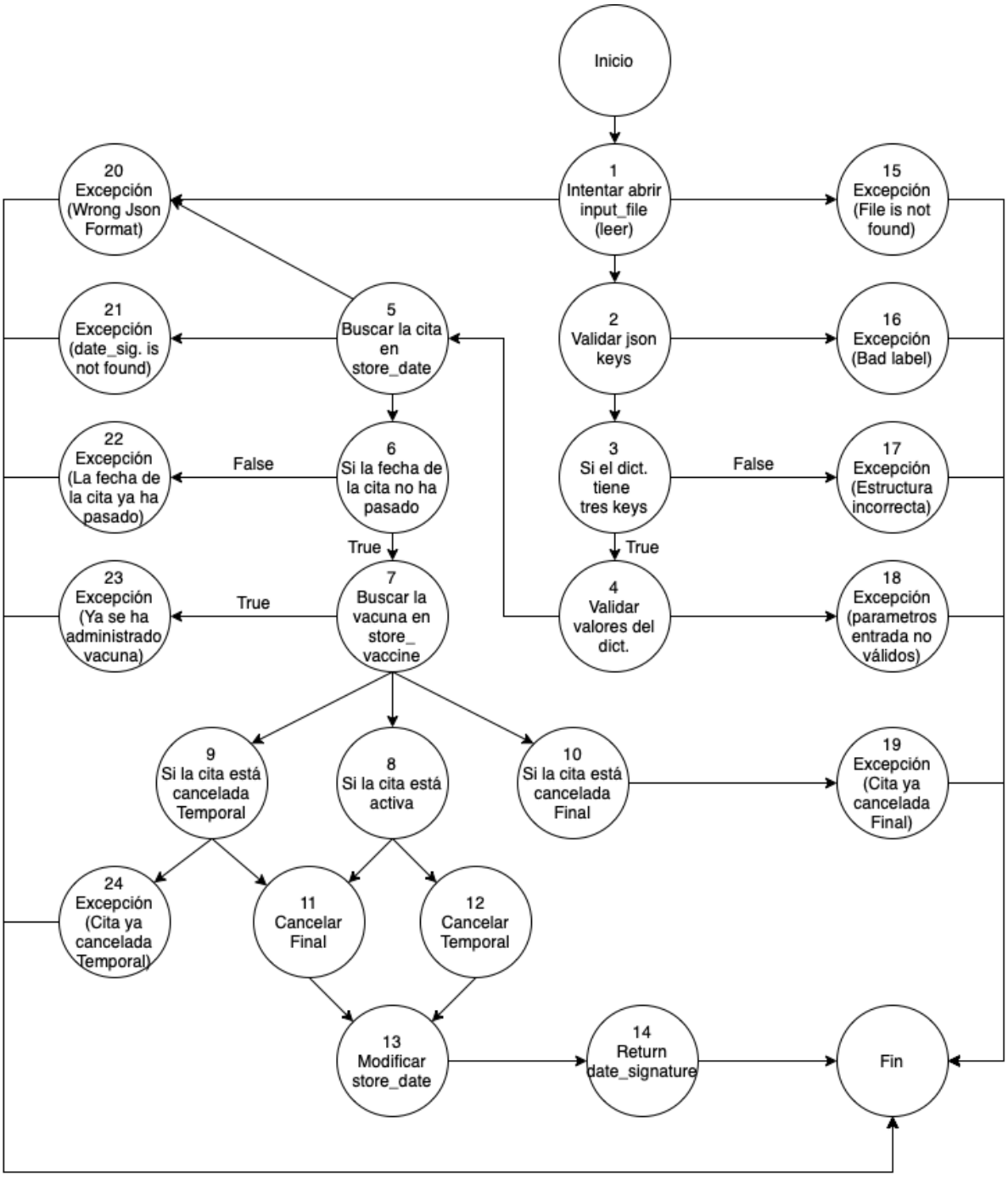
También hemos hecho un cambio en el método register_vaccination de la clase VaccinationAppointment para que, antes de registrar la vacunación de un paciente, compruebe si la cita con la que se quiere vacunar ha sido cancelada, en cuyo caso lanzamos una excepción.

Puesto que el status de la cita se puede modificar, en el futuro se podría implementar la funcionalidad de reactivar una cita que ha sido cancelada de forma temporal.

3.5. Pruebas estructurales

Diagrama de flujo de control

Método cancel_appointment



Complejidad ciclomática:

Enlaces = 38

Nodos = 26

V(G) = E – N + 2 = 37 – 26 + 2 = 14

Rutas básicas

Método cancel_appointment

1, 2, 3, 4, 5, 6, 7, 8, 12, 13, 14 → Abrir el fichero *input_file* y leer los datos, validar las claves, validar que haya 3 campos, validar los valores de los parámetros de entrada, buscar la cita en store_date, verificar que la fecha de la cita no haya pasado, buscar la vacuna en store_vaccine, y si la cita está activa modificar el status a cancelar Temporal, modificar store_date y devolver date_signature **(camino básico)**.

1, 15 → Ruta del fichero *input_file* incorrecta.

1, 20 → Fichero *input_file* con formato incorrecto.

1, 2, 16 → Abrir *input_file* y leer los datos, clave de algún campo del fichero incorrecta.

1, 2, 3, 17 → Abrir *input_file* y leer los datos, claves válidas, estructura del *input_file* incorrecta.

1, 2, 3, 4, 18 → Abrir *input_file* y leer los datos, claves válidas, estructura del *input_file* correcta (3 campos), valores de los parámetros de entrada no válidos.

1, 2, 3, 4, 5, 20 → Abrir *input_file* y leer los datos, claves válidas, estructura del *input_file* correcta (3 campos), valores de los parámetros válidos, formato del store_date incorrecto.

1, 2, 3, 4, 5, 21 → Abrir *input_file* y leer los datos, claves válidas, estructura del *input_file* correcta (3 campos), valores de los parámetros válidos, formato del store_date correcto, la cita recibida no existe.

1, 2, 3, 4, 5, 6, 22 → Abrir *input_file* y leer los datos, claves válidas, estructura del *input_file* correcta (3 campos), valores de los parámetros válidos, cita encontrada en store_date, ver que la fecha de la cita ya ha pasado.

1, 2, 3, 4, 5, 6, 7, 23 → Abrir *input_file* y leer los datos, claves válidas, estructura del *input_file* correcta (3 campos), valores de los parámetros válidos, cita encontrada en store_date, la fecha de la cita no ha pasado, buscar la vacuna en store_vaccine, ver que ya se ha administrado la vacuna.

1, 2, 3, 4, 5, 6, 7, 8, 11, 13, 14 → Abrir *input_file* y leer los datos, claves válidas, estructura del *input_file* correcta (3 campos), valores de los parámetros válidos, buscar la cita en store_date, la fecha de la cita no ha pasado, aún no se ha vacunado, y si la cita está activa modificar el status a cancelar Final, modificar store_date y devolver date_signature.

1, 2, 3, 4, 5, 6, 7, 9, 11, 13, 14 → Abrir *input_file* y leer los datos, claves válidas, estructura del *input_file* correcta (3 campos), valores de los parámetros válidos, buscar la cita en store_date, la fecha de la cita no ha pasado, aún no se ha vacunado,

y si la cita está cancelada Temporal modificar el status a cancelar Final, modificar store_date y devolver date_signature.

1, 2, 3, 4, 5, 6, 7, 9, 24 → Abrir *input_file* y leer los datos, claves válidas, estructura del *input_file* correcta (3 campos), valores de los parámetros válidos, cita encontrada en store_date, la fecha de la cita no ha pasado, aún no se ha vacunado, y si la cita está cancelada Temporal y el *cancelation_type* solicitado es Temporal, la cita ya se ha cancelado de forma Temporal.

1, 2, 3, 4, 5, 6, 7, 10, 19 → Abrir *input_file* y leer los datos, claves válidas, estructura del *input_file* correcta (3 campos), valores de los parámetros válidos, cita encontrada en store_date, la fecha de la cita no ha pasado, aún no se ha vacunado, y si la cita está cancelada Final y se solicita cancelar, la cita ya se ha cancelado de forma Final.

3.6. Cumplimiento del estándar PEP8

Tras implementar todo el nuevo código y sus correspondientes pruebas, hemos refactorizado y, por último, hemos ido corrigiendo los errores que nos mostraba la herramienta Pylint para ajustarnos al estándar PEP8. Hemos modificado el fichero pylintrc para evitar que nos muestre errores por el número mínimo de public methods, el máximo de atributos, el máximo de argumentos de un método, y la advertencia por cyclic imports, ya que debemos ajustarnos a la estructura base del código que se nos aporta.

4. CONCLUSIÓN

Con la realización de esta práctica, hemos comprendido la importancia de llevar a cabo un desarrollo dirigido por pruebas cuando se diseña y programa un componente de software; no solo por ser la mejor forma de contemplar todos los casos de prueba que puede tener nuestro código, sino también por la sencillez que aporta a la hora de diseñar el código definitivo, puesto que al empezar diseñando las pruebas no dejas funcionalidad del componente sin cubrir. Con esta entrega final hemos podido llevar a cabo todas las técnicas aprendidas en la asignatura, y gracias a la realización previa de los ejercicios guiados, obtuvimos la destreza necesaria para poder implementarla de una forma más rápida. Consideramos que hemos tenido un gran rendimiento a lo largo de la asignatura y que está práctica demuestra la cantidad de conocimientos que hemos adquirido.