EJERCICIO GUIADO 3

DESARROLLO DIRIGIDO POR PRUEBAS

Sonsoles Molina Abad – 100432073

Lorenzo Largacha Sanz – 100432129

Doble Grado en Ingeniería Informática y ADE – Grupo 50

Desarrollo de Software

Universidad Carlos III de Madrid, Campus de Colmenarejo

1 abril, 2022

Tabla de contenido

1. FUNCIÓN SOLICITAR VACUNA 2

CLASES DE EQUIVALENCIA 2
ANÁLISIS DE VALORES LÍMITE 4

2. FUNCIÓN OBTENER UNA CITA PARA VACUNARSE 5

GRAMÁTICA 5
ÁRBOL DE DERIVACIÓN 6
CASOS DE PRUEBA ADICIONALES 7

3. FUNCIÓN ADMINISTRACIÓN DE LA VACUNA 8

IDENTIFICACIÓN DE LOS NODOS 8
DIAGRAMA DE FLUJO DE CONTROL 9
RUTAS BÁSICAS 10
CASOS ADICIONALES 11

4. CORRECCIÓN RF1 DEL EQUIPO 12 12

DEFINICIÓN DE LAS CLASES DE EQUIVALENCIA 12
DEFINICIÓN DE LOS TESTS 12

1. Función Solicitar Vacuna

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 cada vez.

CE_V_01 → Se trata de una clase de equivalencia válida, dónde todas las entradas tienen como patient_id un uuid válido (string hexadecimal de 32 caracteres) con el formato correcto y en versión 4.

CE_NV_02 → Clase inválida, donde todas las entradas tienen como patient_id un uuid válido (string hexadecimal con el formato correcto), pero en versión 1.

CE_NV_03 → Clase inválida, donde todas las entradas tienen como patient_id un uuid válido (string hexadecimal con el formato correcto), pero en versión 2.

CE_NV_04 → Clase inválida, donde todas las entradas tienen como patient_id un uuid válido (string hexadecimal con el formato correcto), pero en versión 3.

CE_NV_05 → Clase inválida, donde todas las entradas tienen como patient_id un uuid válido (string hexadecimal con el formato correcto), pero en versión 5.

CE_NV_06 → Clase inválida, donde todas las entradas tienen como patient_id un uuid inválido ya que no se encuentra en formato hexadecimal o no tiene 32 caracteres.

CE_V_07 → Se trata de una clase válida, donde todas las entradas tienen como registration_type el string "Regular".

CE_V_08 → Se trata de una clase válida, donde todas las entradas tienen como registration_type el string "Family".

CE_NV_09 → Se trata de una clase inválida, donde todas las entradas tienen como registration_type cualquier otro string.

CE_V_10 → Se trata de una clase válida, donde todas las entradas tienen como name_surname un string de 1 a 30 caracteres, en dos o más cadenas separadas por un blanco.

CE_NV_11 → Se trata de una clase inválida, donde todas las entradas tienen como name_surname un string con menos de dos cadenas, sin estar separadas por un blanco.

CE_NV_12 → Se trata de una clase inválida, donde todas las entradas tienen como name_surname un string con más de 30 caracteres.

CE_NV_13 → Se trata de una clase inválida, donde todas las entradas tienen como name_surname un string con menos de 1 caracter.

CE_V_14 → Se trata de una clase válida, donde todas las entradas tienen como phone_number un número entero de 9 dígitos (12 caracteres incluyendo +_ _). El rango de valores de entrada comprende todos los números entre el 00 000 000 000 y el 99 999 999 999.

CE_NV_15 → Se trata de una clase inválida, dónde todas las entradas tienen como phone_number un número entero de menos de 9 dígitos.

CE_NV_16 → Se trata de una clase inválida, dónde todas las entradas tienen como phone_number un número entero de más de 9 dígitos.

CE_NV_17 → Se trata de una clase inválida, donde todas las entradas tienen como phone_number cualquier variable que no sea un número entero.

CE_V_18 → Se trata de una clase válida, donde todas las entradas tienen como age un número entero (int) entre 6 y 125 (siendo este el rango de entrada).

CE_NV_19 → Se trata de una clase inválida, donde todas las entradas tienen como age un número menor que 6.

CE_NV_20 → Se trata de una clase inválida, donde todas las entradas tienen como age un número mayor que 125.

CE_NV_21 → Se trata de una clase inválida, donde todas las entradas tienen como age cualquier variable distinta a un string de números (string con letras).

En cuanto a las salidas, se espera obtener una cadena hexadecimal, 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_22 → Se trata de una clase válida, donde todas las salidas son una cadena en formato hexadecimal obtenidas mediante el algoritmo MD5.

CE_NV_23 → Se trata de una clase inválida, donde todas las salidas serían una cadena que no sigue el formato hexadecimal MD5. 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_24 → Se trata de una clase válida, donde todas las salidas son los datos del paciente que se almacenan en el fichero. No definimos un test concreto para esta clase de equivalencia puesto que ya está definido en todos los test válidos anteriores, al comprobarse simultáneamente si los datos correctos introducidos se han almacenado en el fichero.

CE_NV_25 → Se trata de una clase inválida, donde los datos introducidos no son correctos, y por tanto no se almacenan los datos del paciente en el fichero.

CE_NV_26 → Se trata de una clase inválida, donde los datos introducidos son correctos, pero no se almacenan en el fichero. Este caso solo se dará si los datos ya estaban guardados en el fichero.

CE_NV_27 → Se trata de una clase inválida, donde los datos introducidos no son correctos, pero se almacenan en 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).

Análisis de valores límite

patient_id → Dado que debe ser un string de 32 caracteres, los valores límite serán 31, 32 y 33 caracteres.

registration_type → No existen valores límite ya que solo puede ser un string del tipo "Family" o "Regular".

name_surname → Los valores límite serán 0, 1 y 2 blancos, ya que debe tener 1 como mínimo. Por otra parte 0, 1, 2, 29, 30 y 31 caracteres, ya que debe estar entre 1 y 30.

phone_number → Los valores límite serán 11, 12 y 13 caracteres porque ha de ser un número de 9 dígitos, teniendo en cuenta el prefijo +_ _ (12 caracteres en total).

age → Los valores límite serán 5, 6, 7, 124, 125 y 126, ya que debe ser un número entre 6 y 125.

* Hemos modificado pylintrc para evitar que nos muestre errores por el número de argumentos, el número de variables locales, el número de expresiones booleanas de una función, el número de sentencias, el número de ramas y el número de atributos, ya que debemos ajustarnos a la estructura base del código que se nos aporta.

2. Función Obtener una Cita para Vacunarse

Gramática

La función get_vaccine_date recibe como parámetro un fichero Json con los datos de acceso de un paciente para los que es necesario generar una cita de vacunación y una nueva clave. Dicho archivo debe seguir el siguiente formato:

```
{
    "PatientSystemID":"0a9d6f313a6355f54caf4cd00a21f2d1",
    "ContactPhoneNumber":"+34123456789"
}
```

Definición de la gramática de entrada

fichero ::= llave_ini datos llave_fin

llave_ini ::= {

llave_fin ::= }

datos ::= campo_1 separador campo_2

separador ::= ,

campo_1 ::= etiqueta_dato1 igualdad valor_dato1

campo_2 ::= etiqueta_dato2 igualdad valor_dato2

igualdad ::= :

etiqueta_dato1 ::= comillas valor_etiqueta1 comillas

valor_etiqueta1 ::= PatientSystemId

valor_dato1 ::= comillas valor1 comillas

valor1 ::= alblcldlelfl0l1l…l9l {32}

comillas ::= “

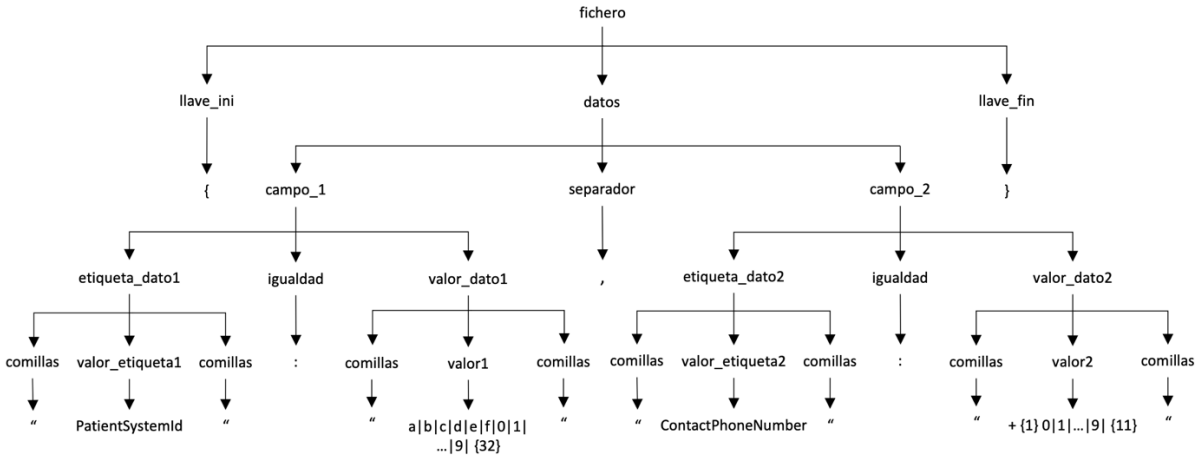
etiqueta_dato2 ::= comillas valor_etiqueta2 comillas

valor_etiqueta2 ::= ContactPhoneNumber

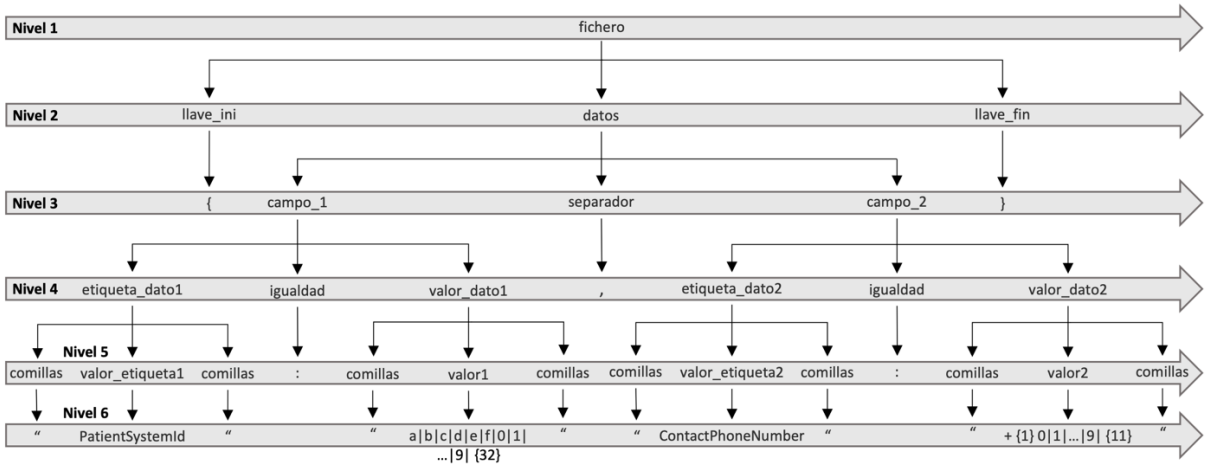
valor_dato2 ::= comillas valor2 comillas

valor2 ::= + {1} 0l1l….l9l {1 1}

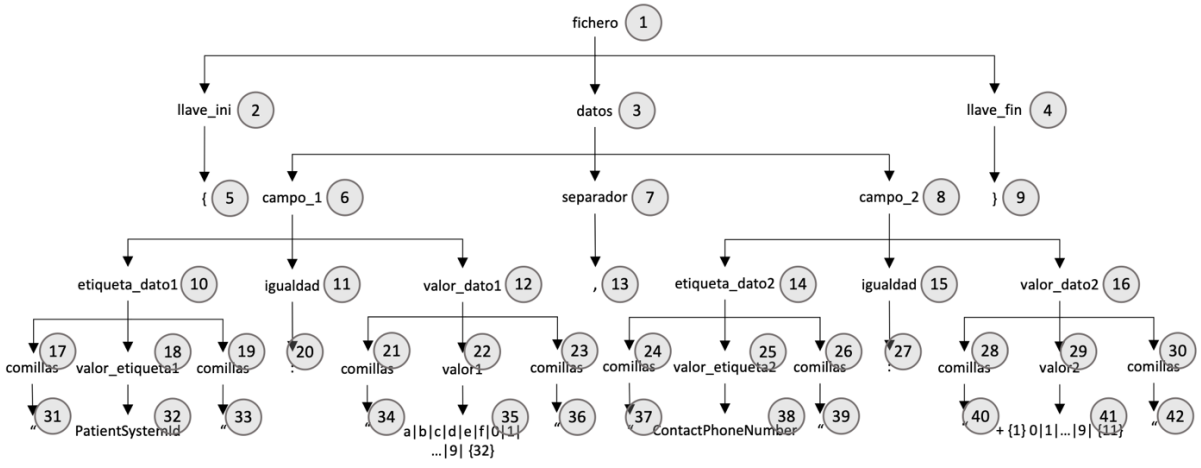
Árbol de derivación



Niveles



Nodos numerados



Casos de prueba adicionales

test_69 (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 etiqueta y valor diferente, y siguiendo un formato json correcto. Fichero: test_campo3.json

test_70 (parametrizado)

Añadimos un test para comprobar si se lanza la excepción "Fichero input_file no creado" cuando la ruta del fichero de entrada es incorrecta. Para ello escribimos la ruta de un fichero que no existe.

test_71_no_store_patient_nok

Añadimos un test para comprobar que se produce un error de procesamiento interno cuando el fichero store_patient no existe. La excepción esperada será "Fichero store_patient no creado".

test_72_get_two_vaccine_date_ok

Añadimos un test para comprobar que se pueden guardar varias citas de distintos pacientes. Para ello registramos dos pacientes diferentes y creamos una cita para cada uno, comprobamos las claves y que se escriben los datos en store_date.

3. Función Administración de la Vacuna

Identificación de los nodos

Método vaccine_patient

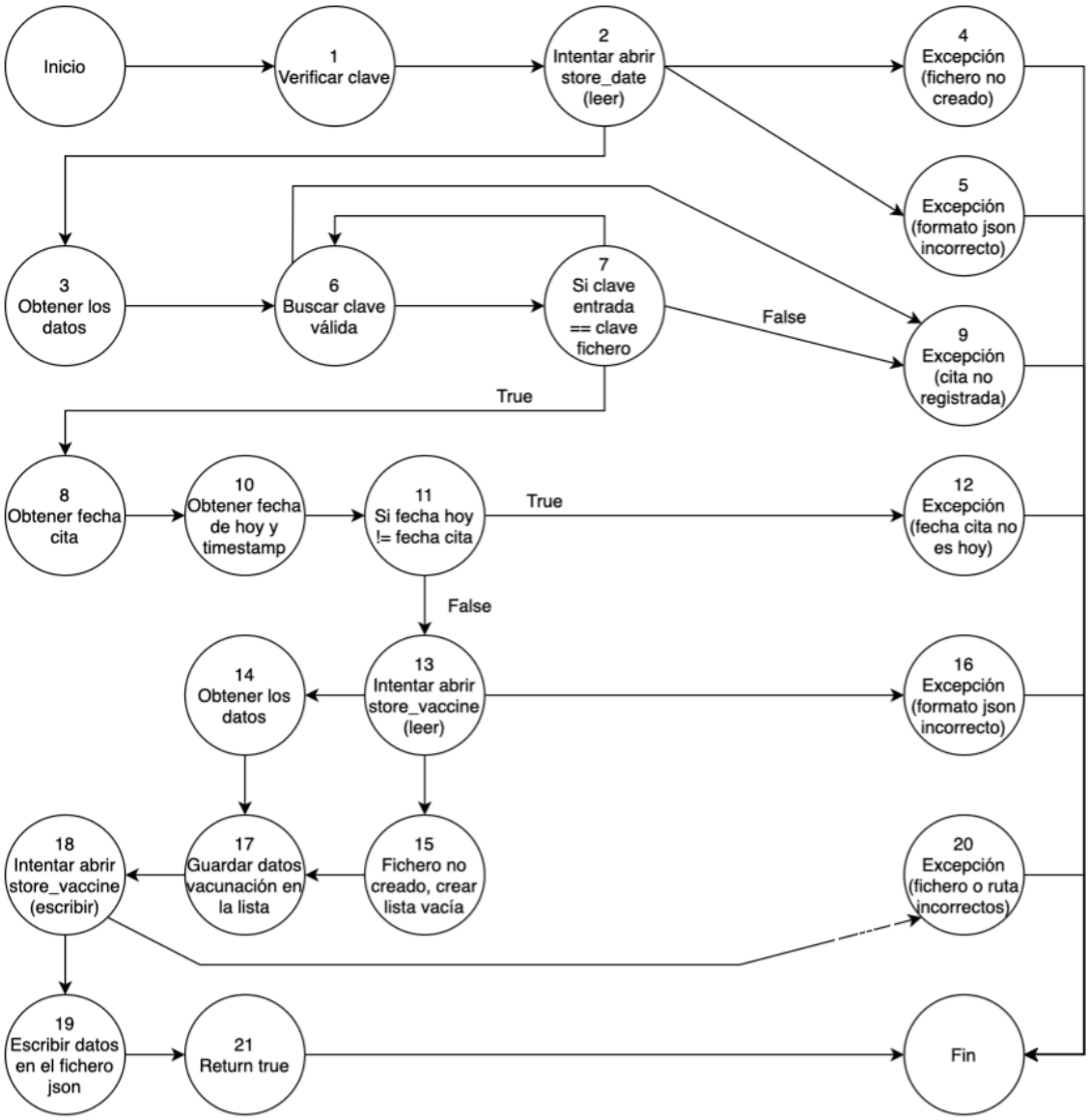
```
def vaccine_patient(self, date_signature):
    """ Comprobamos la clave de entrada, buscamos si la cita esta registrada,
    comprobamos que la fecha es para hoy y guardamos los datos de vacunacion en un fichero """

    # Comprobamos que la firma SHA256 sigue el formato esperado                                  1
    self.validate_date_signature(date_signature)

    # Buscamos la ruta en la que se almacena el fichero store_date                               2
    json_files_path = str(Path.home()) + "/PycharmProjects/G50.2022.T11.EG3/src/JsonFiles/RF2"
    file_store_date = json_files_path + "/store_date.json"

    # Intentamos abrir el fichero store_date y guardar sus datos
    try:
        # Intentamos abrir el fichero JSON
        with open(file_store_date, "r", encoding="UTF-8", newline="") as file:
            # Guardamos los datos del fichero en una lista                                       3
            data_list_date = json.load(file)
    except FileNotFoundError as ex:                                                              4
        # En caso de que el fichero no exista, lanzamos una excepcion
        raise VaccineManagementException("Fichero store_date no creado") from ex
    except json.JSONDecodeError as ex:                                                           5
        # Si los datos no siguen el formato JSON
        raise VaccineManagementException(
            "JSON decode error - formato JSON incorrecto") from ex

    # Creamos una variable para guardar si se encuentra una cita con esa clave                   6
    found = False
    # Recorremos las entradas de fichero
    for item in data_list_date:
        # Si el date_signature se encuentra en el fichero (cita registrada)                      7
        if item["_VaccinationAppoinment__date_signature"] == date_signature:
            found = True
            # Obtenemos la fecha de la cita                                                      8
            date_time = item["_VaccinationAppoinment__appoinment_date"]

    # Si la cita no está registrada, lanzamos una excepcion                                      9
    if found is False:
        raise VaccineManagementException("Cita no registrada")

    # Guardamos la fecha de hoy                                                                  10
    today = datetime.today().date()
    # Convertimos el timestamp de la cita a fecha
    appoinment_date = datetime.fromtimestamp(date_time).date()

    # Si la fecha registrada en la cita no es hoy, lanzamos una excepcion                        11
    if appoinment_date != today:
        raise VaccineManagementException("Cita no registrada para la fecha de hoy")              12

    # Escribimos los datos de vacunacion en el fichero store_vaccine                             13
    # Buscamos la ruta en la que se almacena el fichero store_vaccine
    json_files_path = str(Path.home()) + "/PycharmProjects/G50.2022.T11.EG3/src/JsonFiles/RF3"
    file_store_vaccine = json_files_path + "/store_vaccine.json"

    try:
        # Intentamos abrir el fichero JSON para leer
        with open(file_store_vaccine, "r", encoding="UTF-8", newline="") as file:
            # Guardamos los datos del fichero en una lista                                       14
            data_list = json.load(file)
    except FileNotFoundError:                                                                    15
        # En caso de que el fichero no exista creamos una lista para almacenar los datos
        data_list = []
    except json.JSONDecodeError as ex:                                                           16
        # Si se produce un error al decodificar mostramos una excepcion
        raise VaccineManagementException(
            "JSON decode error - formato JSON incorrecto") from ex

    # Guardamos los datos de vacunación en la lista                                              17
    data_list.append(date_signature.__str__())
    data_list.append(datetime.utcnow().__str__())

    try:
        # Intentamos abrir el fichero JSON para escribir                                         18
        with open(file_store_vaccine, "w", encoding="UTF-8", newline="") as file:
            # Serializamos el objeto y guardamos los datos en el fichero                         19
            json.dump(data_list, file, indent=2)
    except FileNotFoundError as ex:                                                              20
        # Si no existe el fichero lanzamos una excepcion
        raise VaccineManagementException(
            "JSON file not found error - fichero o ruta incorrectos") from ex

    # Si la clave y fecha son válidas, devolvemos True                                           21
    return True
```

Método validate_date_signature

```
@staticmethod
def validate_date_signature(date_signature):
    """ Return True si el date_signature es correcto, en otro caso Excepcion """
    # Comprobamos que el date_signature es una cadena hexadecimal de 64 caracteres   1
    myregex = re.compile(r'^[0-9A-F]{64}$', re.IGNORECASE)                            1
    res = myregex.fullmatch(date_signature)                                           1
    if not res:                                                                       2
        raise VaccineManagementException("Formato del date_signature invalido")       3
    return True                                                                       4
```

Diagrama de flujo de control

Método vaccine_patient



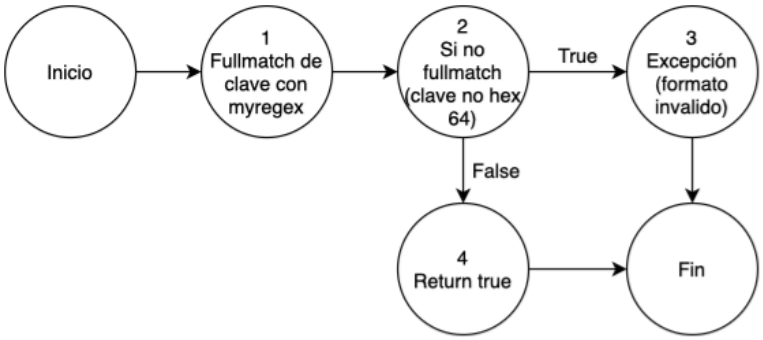
Complejidad ciclomática:

Enlaces = 31

Nodos = 23

V(G) = E – N + 2 = 31 – 23 + 2 = 10

Método validate_date_signature



Complejidad ciclomática:

Enlaces = 6

Nodos = 6

V(G) = E – N + 2 = 6 – 6 + 2 = 2

Rutas básicas

Método vaccine_patient

1, 2, 3, 6, 7, 8, 10, 11, 13, 15, 17, 18, 19, 21 → Abrir store_date y leer los datos, encontrar la clave en el fichero, la fecha de la cita es hoy, fichero store_vaccine no creado, abrir store_vaccine y escribir los datos de vacunación **(camino básico)**.

1, 2, 4 → Fichero store_date no creado.

1, 2, 5 → Fichero store_date con formato incorrecto.

1, 2, 3, 6, 7, 9 → Abrir store_date y leer los datos, no encontrar la clave en el fichero.

1, 2, 3, 6, 9 → Abrir store_date y leer los datos, fichero store_date vacío.

1, 2, 3, 6, 7, 8, 10, 11, 12 → Abrir store_date y leer los datos, encontrar la clave en el fichero, la fecha de la cita no es hoy.

1, 2, 3, 6, 7, 8, 10, 11, 13, 14, 17, 18, 19, 21 → Abrir store_date y leer los datos, encontrar la clave en el fichero, la fecha de la cita es hoy, abrir store_vaccine y leer los datos, abrir store_vaccine y escribir los datos de vacunación.

1, 2, 3, 6, 7, 8, 10, 11, 13, 16 → Abrir store_date y leer los datos, encontrar la clave en el fichero, la fecha de la cita es hoy, fichero store_vaccine con formato incorrecto.

1, 2, 3, 6, 7, 8, 10, 11, 13, 14, 17, 18, 20 → Abrir store_date y leer los datos, encontrar la clave en el fichero, la fecha de la cita es hoy, abrir store_vaccine y leer los datos, fichero o ruta store_vaccine incorrectos. * Si pasa por el nodo 14, no puede pasar por el 20, ya que si ha podido abrir el fichero para leer, la ruta es correcta y el fichero existe.

1, 2, 3, 6, 7, 8, 10, 11, 13, 15, 17, 18, 20 → Abrir store_date y leer los datos, encontrar la clave en el fichero, la fecha de la cita es hoy, fichero store_vaccine no creado, fichero o ruta store_vaccine incorrectos.

Método validate_date_signature

1, 2, 4 → La clave hace fullmatch con la expresión myregex **(camino básico)**.

1, 2, 3 → La clave no hace fullmatch con la expresión myregex.

Casos adicionales

Pruebas bucle for (bucle simple)

No pasar por el bucle → Ya descrita anteriormente (test_5).

Pasar por el bucle 1 vez → Ya descrita anteriormente (test_1, test_4, test_6, test_7, test_8, test_9).

Pasar por el bucle 2 veces → Precargamos 2 citas, y la segunda tiene la clave de entrada. Camino: 1, 2, 3, 6, 7, 6, 7, 8, 10, 11, 13, 14, 17, 18, 19, 21. Abrir store_date y leer los datos, encontrar la clave en el fichero store_date en el segundo item, la fecha de la cita es hoy, abrir store_vaccine y leer los datos, abrir store_vaccine y escribir los datos de vacunación (test_12).

Pasar por el bucle 3 veces → Precargamos 3 citas, y la tercera tiene la clave de entrada. Camino: 1, 2, 3, 6, 7, 6, 7, 6, 7, 8, 10, 11, 13, 14, 17, 18, 19, 21. Abrir store_date y leer los datos, encontrar la clave en el fichero store_date en el tercer item, la fecha de la cita es hoy, abrir store_vaccine y leer los datos, abrir store_vaccine y escribir los datos de vacunación (test_13).

4. Corrección RF1 del equipo 12

Definición de las clases de equivalencia

patient_id → existen otras clases de equivalencia no válidas en las que se podrían contemplar también las demás versiones de los UUID (1,2,3,5).

registration_type → clases de equivalencia bien definidas.

phone_number → clases de equivalencia bien definidas. En la CE_V_07 se podría concretar cuándo es válido (9 dígitos).

age → dividir CE_NV_17 en dos clases de equivalencia, una por encima del rango y otra por debajo.

name → en la CE_V_12 se podría concretar cuándo name es válido (1 a 30 caracteres, en 2 o más cadenas separadas por un blanco). Además, habría que añadir una clase no válida por debajo del rango de caracteres.

MD5 → en la CE_V_15 se podría concretar cuándo es válida (cadena hexadecimal). Además, existiría un caso no válido en el que la cadena es incorrecta, el cual no se llegaría a dar nunca.

Fichero → también se podrían contemplar los siguientes casos:

- Los datos no son correctos, por lo que no se almacenan en el fichero.
- Los datos introducidos son correctos, pero no se almacenan en el fichero, dado que ya estaban guardados anteriormente en el fichero.
- Los datos introducidos no son correctos, pero se almacenan en el fichero. Este caso no se va a dar nunca, ya que, si los datos son incorrectos, no llegarán a almacenarse.

Definición de los tests

En primer lugar, faltaría añadir tests para los valores límite, ya que solo están probando las clases de equivalencia. Para los atributos patient_id, name, phone y age deberían realizar una prueba para cada valor límite correspondiente.

En el caso de phone, simplemente tendrían que considerar 3 valores límite: el número exacto de dígitos que se piden, uno por encima y uno por debajo. Para el caso de los blancos en el atributo name, ocurre lo mismo, ya que este debe contener un espacio como mínimo. También, para la toma de los valores límite del patient_id escogemos los valores de la misma forma, ya que esta debe ser una cadena de excatamente 32 caracteres.

En cambio, para la cadena del name y para el número que corresponde a age, dado que su valor corresponde a un intervalo, deberían identificar los límites superior e inferior del intervalo, y considerar varios valores (límite, justo antes y justo después), con lo que obtendrían 6 valores límite para cada uno.

Por otra parte, también les falta probar las salidas, para validar las clases de equivalencia 15 y 16. Asimismo, creemos que en la prueba 8 se han equivocado y han escrito CE_NV_12, en lugar de CE_NV_11 que es la clase que están probando.

Por último, en el apartado del resultado esperado, se podría concretar qué excepción es la que se espera.