Titulación: GRADO INGENIERÍA INFORMÁTICA
Año Académico: 2020/2021 – 2º cuatrimestre
Asignatura: DESARROLLO SOFTWARE (2º curso)
Memoria práctica final

uc3m | Universidad Carlos III de Madrid

PRÁCTICA FINAL

Adrián Serrano Navarro (100429115); Jorge López Marín (100428977)

Grupo 82 (Ingeniería Informática)

Universidad Carlos III de Madrid

ÍNDICE

1. INTRODUCCIÓN

El siguiente documento informativo, recoge toda la información relativa a la práctica final de desarrollo de software. En ella, a partir de los conocimientos obtenidos, se intentará completar el sistema de acceso a un edificio, de una supuesta empresa SecureAll, desarrollado parcialmente durante los cuatro ejercicios guiados previos. Es decir, se busca terminar de diseñar, implementar y probar, dicho componente de software, usando, tal como se indicará posteriormente, desarrollo dirigido por pruebas, refactorización, singleton, y toda clase de herramientas para conseguir cumplir una serie de requisitos dados.

De esta forma, para perfeccionarlo, primero imponemos que las búsquedas de las peticiones sean por el código de acceso, más tarde incluimos un nuevo archivo almacén json con dicha key final, y una marca de tiempo, y finalmente creamos una nueva funcionalidad encargada de revocar una clave concreta y de devolver la lista de emails asociados, siendo esta sobre la que evaluaremos los casos de prueba previamente dichos.

En definitiva, se busca profundizar, y mejorar la organización del sistema de acceso, mediante los conocimientos adquiridos durante las clases, incluyendo el material de apoyo proporcionado, así como el código necesario para el desarrollo.

2. FUNCIONALIDADES

A continuación, se abordan las funcionalidades a desarrollar e implementar como tal en la práctica, las cuales se corresponden con los tres requisitos ya mencionados.

2.1. REQUISITO 1

El requisito 1 establece lo siguiente: El componente debe almacenar el valor access_code para cada access_request. Todas las búsquedas en el fichero de solicitudes deben basarse en el código de acceso.

- **EXPLICACIÓN:** Para generar correctamente lo pedido, únicamente hemos recurrido a modificaciones en el store relativo al request_code y a la propia clase de las peticiones AccessRequest. De este modo, hemos cambiado las búsquedas del find_item de forma que las que usaban el DNI, toman ahora dicho code como argumento, hemos creado un nuevo método de agregación en el que ahora incluimos el access_code en los requests con el método add_item2, sobreescribiendo el que era el add_item de la clase padre, y todo ello, eso sí, con una previa inclusión en el constructor de la petición, del valor de dicho code, para que pudiera tomarse dicho atributo.

2.2. REQUISITO 2

El requisito 2 establece lo siguiente: En el método open_door, si la clave era válida, el componente debería registrar en un archivo json la marca de tiempo del acceso y el valor de la clave (básicamente un registro de acceso). Esta funcionalidad no está incluida en la solución propuesta y debe incluirse ahora.

- **EXPLICACIÓN:** En este caso, han sido dos los procesos desarrollados. En primer lugar, se ha generado un método encargado de obtener la marca temporal pedida, que, a su vez, llama a un nuevo método de adición que incluirá el diccionario generado con, tanto la key como dicho tiempo en el store asociado.

  Por otro lado, como se pide, también se ha modificado el método open_door dado, incluyendo en esta ocasión, además, la llamada a dicho método mencionado en el párrafo anterior, para provocar el cambio, y no solo la validación pertinente mediante el procedimiento is_valid (key). Es decir, ahora en open_door ya no solo validamos que el objeto key no está caducado, sino que también hacemos que se recoja el diccionario en el store o almacén (Last_access_store)

2.3. REQUISITO 3

El requisito 3 es el más extenso y establece lo siguiente: Antes de su fecha de vencimiento, será posible desactivar la clave correspondiente a una persona específica especificando los siguientes datos en el archivo de entrada JSON:

{ "Key": " ", "Revocation":" ", "Reason":" "}

El método devolverá la lista de correos electrónicos de notificación correspondiente a la tecla que ha sido desactivada, RevokeKey (FilePath).

- **EXPLICACIÓN:** Como se ha comentado al inicio del epígrafe, esta funcionalidad es la más extensa y la que más trabajo ha requerido, esencialmente ya que ha supuesto la generación de todos los casos de prueba de la misma, así como los test pertinentes de comprobación. Por ello, en primer lugar, se plantearon las clases de equivalencia, posteriormente el árbol y gramática asociada, y finalmente el grafo de control de flujo, todas ellas técnicas del desarrollo dirigido por pruebas, que se mencionarán en los puntos siguientes. No obstante, no solo se ha tratado de generar las pruebas, sino que la propia función a desarrollar como tal, ha requerido a su vez de otras modificaciones en el código, suponiendo nuevos métodos y la ya mencionada final refactorización.

  Además, es importante mencionar que como se busca también que la llave no haya sido procesada antes, ha sido necesario generar un nuevo almacén de dichos valores. Así, al no establecerse nada fijado como claves en el diccionario supuesto de llave revocadas a introducir, se ha optado por incluir solamente, el valor de la llave a procesar, ignorando los otros dos strings de clave (reason y revocation)

  Asimismo, de acuerdo con una correcta implementación en Pycharm de los respectivos métodos de validación de las entradas, se ha optado por hacer uso, para la comprobación de dichos valores, de las expresiones regulares, también conocidas como Regex (Para utilizarlas se ha importado la librería re y el método asociado, validate de la clase padre de los atributos). Estas, han permitido definir con facilidad el patrón de búsqueda de los valores válidos de los métodos, de modo que ha resultado mucho más sencillo rechazar aquellos que no lo cumpliesen. De igual forma, como sucedía en casos anteriores, estos dos nuevos atributos, Revocation y Reason, contarán también con sus respectivos ficheros en Attributes, donde la Key ya posee uno, de ahí que no haya sido necesario incluirlo.

2.3.1. CLASES DE EQUIVALENCIA

El sistema recibirá información sobre la Key a revocar, así como del tiempo y motivos asociados a la misma eliminación. Por ello, cada una de las posibles entradas dadas, se acompaña de un tabla que incluye todas las respectivas posibilidades de clases de equivalencia y valores límite, para cuya creación se estudió rango de valores de entrada / salida, cantidad de valores de entrada y semántica, etc. Asimismo, a la hora de generar el testing_cases no ha sido necesario incluir toda la serie de filas propuestas, ya que algunas quedaban definidas mediante otros casos, de ahí que la cardinalidad de la siguiente tabla propuesta no sea igual a la parte asociada del archivo CSV.

TECHNIQUE	FIELD	VALID/INVALID	CRITERIA	DESCRIPTION
CE	KEY	VALIDO	TODO OK	STRING EN FORMATO HEXADECIMAL (64 DÍGITOS)
CE	KEY	INVALIDO	NOT STRING VALUE	SE RECIBE VALOR EN FORMATO INCORRECTO (NOT STRING)
CE	KEY	INVALIDO	INVALID LETTER	SE INCLUYE UNA LETRA NO VÁLIDA EN HEXADECIMAL
VL	KEY	INVALIDO	INVALID LENGHT (-1)	STRING CON 63 CARACTERES
VL	KEY	INVALIDO	INVALID LENGHT (+1)	STRING DE 65 CARACTERES
VL	KEY	VALIDO	TODO OK	STRING CON 64 CARACTERES
CE	REVOCATION	VALIDO	TODO OK	REVOCATION == TEMPORAL
CE	REVOCATION	VALIDO	TODO OK	REVOCATION == FINAL
CE	REVOCATION	INVALIDO	INVALID REVOCATION	REVOCATION DISTINTO DE TEMPORAL O FINAL
CE	REVOCATION	INVALIDO	NOT STRING VALUE	SE RECIBE VALOR EN FORMATO INCORRECTO (NOT STRING)
CE	REASON	VALIDO	TODO OK	STRING COMPRENDIDO ENTRE 0 Y 100 CARACTERES
VL	REASON	VALIDO	TODO OK	STRING CON 0 CARACTERES
VL	REASON	VALIDO	TODO OK	STRING CON 1 CARACTER
VL	REASON	VALIDO	TODO OK	STRING CON 99 CARACTERES
VL	REASON	VALIDO	TODO OK	STRING CON 100 CARACTERES
VL	REASON	INVALIDO	INVALID REASON (+1)	STRING CON 101 CARACTERES
CE	REASON	INVALIDO	NOT STRING VALUE	SE RECIBE VALOR EN FORMATO INCORRECTO (NOT STRING)

2.3.2. ÁRBOL DE DERIVACIÓN Y GRAMÁTICA

Siguiendo el formato del JSON dado en el enunciado, el análisis sintáctico de la entrada de la funcionalidad, da lugar a la consiguiente gramática y árbol de derivación, el cual ha permitido definir los casos de pruebas, para clases inválidas, tomando duplicación y borrado de nodos, y de las válidas, incluyendo además modificación.

Como el título de los nodos en ocasiones resultaba largo, por estética, se ha decidido asignar a cada uno de ellos una expresión más corta para hacer su visualización menos confusa (Los nodos que no aparecen en las tablas mantienen su nombre, si bien es cierto que el fichero de entrada actúa como axioma).

GRAMÁTICA:

NOMBRE	Inicio	Cuerpo	Parte1	Parte2	Parte3	Comillas	Fin
REFERENCIA	I	U	1	2	3	M	N

NOMBRE	Separador	Igualdad	Etiqueta parte1	Etiqueta parte2	Etiqueta parte3
REFERENCIA	S	G	D	H	T

NOMBRE	Dato parte1	Dato parte2	Dato parte3	FICHERO_ENTRADA
REFERENCIA	A	B	C	ENTRADA

NOMBRE	Valor Etiqueta parte1	Valor Etiqueta parte2	Valor Etiqueta parte3
REFERENCIA	E	J	W

NOMBRE	Valor parte1	Valor parte2	Valor parte3
REFERENCIA	F	K	Z

ENTRADA: IUN

I: {

N: }

U: 1S2S3

1: DGF

2: HGK

3: TGZ

S: ,

M: “

G: :

D: MEM

H: MJM

T: MWM

E: Key

J: Revocation

W: Reason

F:MAM

C: {1-9a-zA-Z}{0,100}

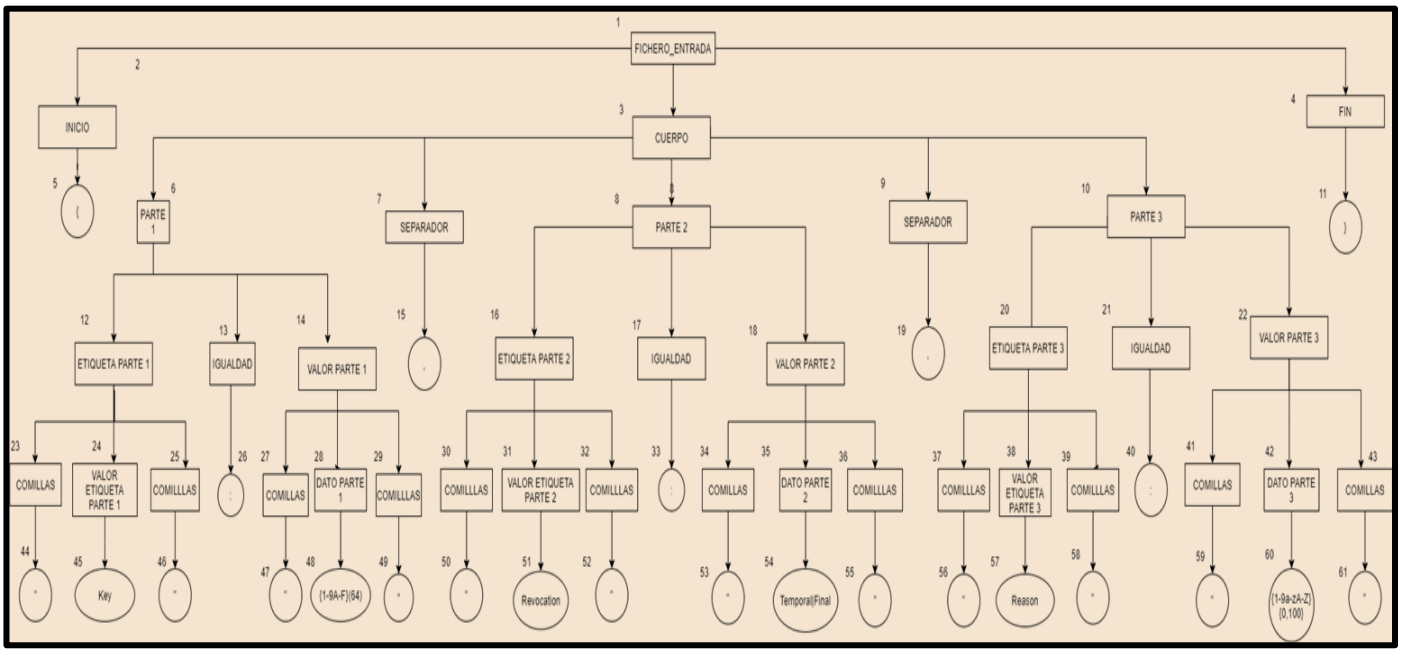
Z: MCM

A: {1-9A-F}(64)

B: Temporal|Final

K: MBM

- **ARBOL DE DERIVACIÓN ASOCIADO**

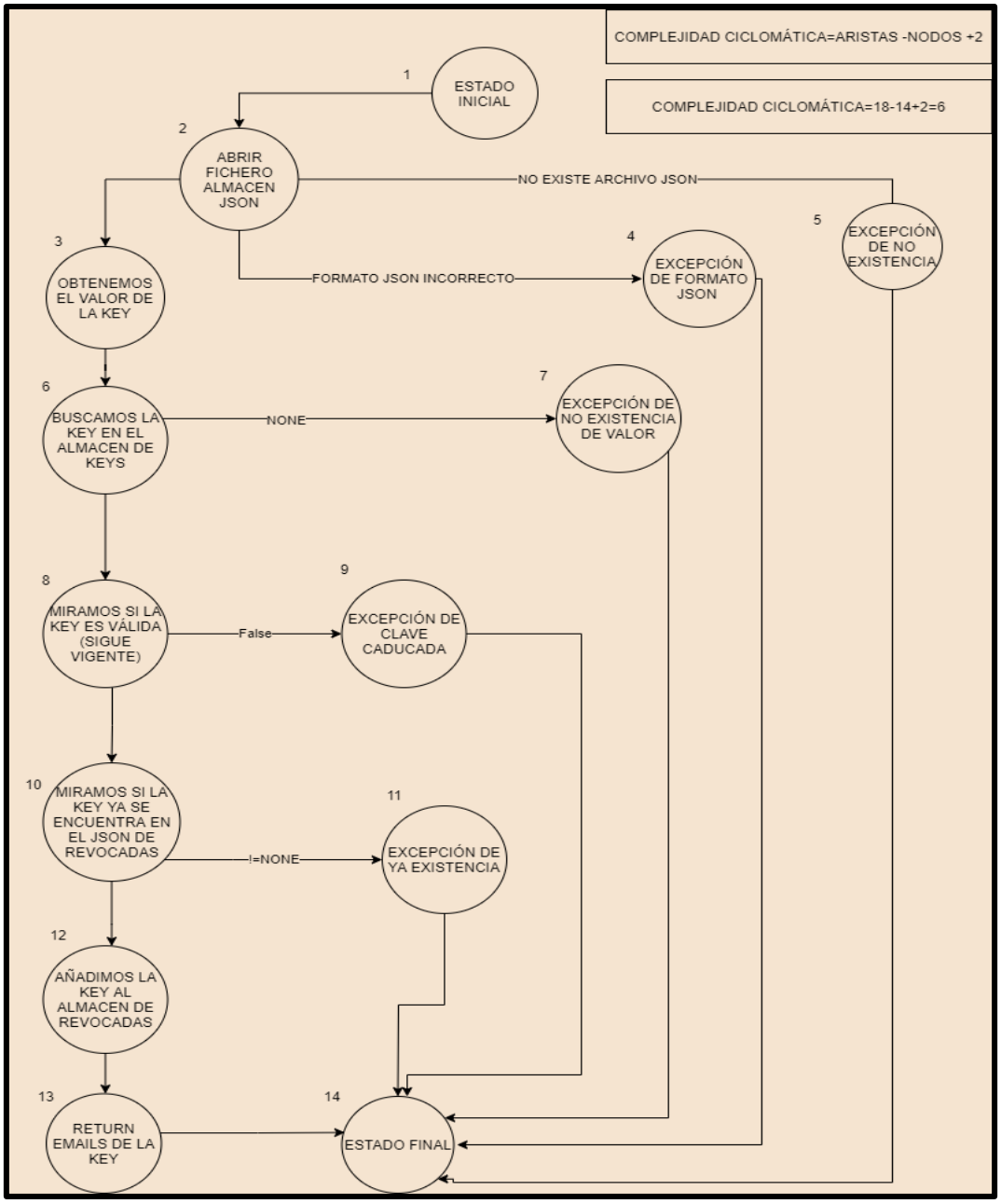


- **CONSIDERACIONES DEL ÁRBOL**

  - Los nodos representados mediante esferas hacen referencia a elementos terminales.
  - Existen un total de 61 nodos divididos en terminales y no terminales, donde la raíz recibe el nombre de fichero de entrada o entrada.
  - Para una mejor visualización, se adjunta la imagen del árbol a su vez en la carpeta docs del proyecto.
  - La tabla asociada al análisis sintactico resultaba demasiado grande, por ello se ha optado por no incluirla en este informe, sino que se podrá distinguir junto al resto en la carpeta Docs del proyecto de Github.

2.3.3 GRAFO DE FLUJO DE CONTROL

Para identificar los casos de prueba, finalmente se ha generado el grafo asociado de flujo de control, base del sistema, y sobre el que ejecutaremos dos técnicas complementarias, en función del nivel de cobertura, decisión y condición, respectivamente. Así, el grafo encargado de documentar la estructura, es el siguiente:



- **Pruebas de camino básico (nivel de cobertura de decisión):** En este apartado, generaremos casos de prueba, a partir de un conjunto dado de caminos independientes, por los que circula el programa.

  El número de dichos casos, se ha estimado mediante la complejidad ciclomática o de McCabe, que se calcula como complejidad=aristas-nodos+2. De este modo, antes de identificar caminos, podemos concluir que, en esta ocasión, al tratarse de una complejidad con un valor de 6, se contemplarán como máximo 6 pruebas para garantizar la cobertura de casos.

  Una vez dicho la anterior, se puede afirmar que, siguiendo el algoritmo de definición de caminos, finalmente se obtiene la consiguiente lista de nodos para cada uno.

  1. **El json de las llaves no puede abrirse:** 1_2_5_14
  2. **El json de las llaves tiene formato erróneo:** 1_2_4_14
  3. **La key buscada no existe:** 1_2_3_6_7_14
  4. **La key existe, pero está caducada:** 1_2_3_6_8_9_14
  5. **La key a revocar ya había sido revocada antes:** 1_2_3_6_8_10_11_14
  6. **Todo correcto y se devuelve la lista de emails:** 1_2_3_6_8_10_12_13_14

PATH	ID TEST	DESCRIPTION	EXPECTED RESULT
1_2_5_14	RF5 1	el json de las llaves no puede abrirse	AccessManagerException
1_2_4_14	RF5 2	el json de las llaves tiene formato erróneo	AccessManagerException
1_2_3_6_7_14	RF5 3	key buscada no existe	AccessManagerException
1_2_3_6_8_9_14	graf_key_expired.json	key existe pero caducada	AccessManagerException
1_2_3_6_8_10_11_14	graf_key_revoke.json	key ya había sido revocada antes	AccessManagerException
1_2_3_6_8_10_12_13_14	RF5 6	Se devuelve la lista de emails	["mail1@uc3m.es", "mail2@uc3m.es"]

3. MODIFICACIONES .PYLINTRC

A lo largo de la implementación en pycharm, se han observado varios problemas a la hora ejecutar pylint debido a la no concordancia entre la convención seguida por dicho entrono de desarrollo, PEP-8, y nuestro código. Por ello, ha sido necesario modificar ciertos campos, los cuales se mencionan a continuación:

- **Maximum number of locals for function / method body;** 15 -> 25

4. CONSIDERACIONES DE LOS TESTS

Para generar la nueva funcionalidad, ha sido también necesario obtener los tests pertinentes de ejecución, para todos los casos propuestos. Para ello, se incluyeron en cada uno de los tres últimos CSV´s presentes de JsonFiles (Carpeta disponible en el código del github), las respectivas situaciones a estudiar (Clases de equivalencia = testingCases_RF3, árbol = testingCases_RF4, grafo= testingCases_RF5). No obstante, por facilidad en la resolución, se optó por mantener el mismo formato de los campos en estos tres CSV´s, de forma que finalmente se colocaron todos los casos de prueba a evaluar en el testingCases_RF4, cada uno con su respectiva referencia de test.

- RF3: Clases de equivalencia y valores límite
- RF4: Árbol de derivación y gramática
- RF5: Grafo de control de flujo

5. CONCLUSIONES

En general, la resolución de los tres requisitos se ha llevado a cabo de manera conveniente, siguiendo en gran medida los pasos estipulados tanto en el enunciado de la práctica, como en las diapositivas informativas, también suministradas.

Sin embargo, también durante el desarrollo de las mismas, se ha observado de primera mano la dificultad de trabajar con casos de prueba, siendo, sin lugar a dudas, el tercer apartado, el que más tiempo a conllevado, y el que por tanto mayor pesadez ha implicado.

Por tanto, pese a la desconfianza inicial que conllevaba enfrentarse a la implementación propia, sin tener diapositivas de apoyo, como en los otros casos, consideramos que el resultado final ha sido perfectamente válido y concluyente tal y como demuestran los informes de pruebas proporcionados y toda la información suministrada a lo largo de esta breve memoria.