Práctica Final Desarrollo de Software
Grupo 83
2020/2021

Grupo 5 de Prácticas:

Shikai Ji-100428979

Rodrigo Eduardo Herrera Coto-100400285

Índice

Introducción

Para esta práctica final, se implementaran 3 cambios generales en el programa base proporcionado:

1. **Almacenar el valor access code de cada Access Request dentro de *StoreRequest*, además de basar toda búsqueda dentro de *StoreRequest*, en el access code.**

2. **Almacenar cada key introducida(válida) en open door, junto con el momento en el que se abrió la puerta, en un archivo aparte que es el almacén de accesos.**

3. **Implementar una nueva funcionalidad en la que podamos revocar una key antes de su vencimiento, a través de un fichero json. Esta función devolverá la lista de emails asociado a la llave revocada, y en caso de error provocará un *AccessManager* Exception.**

En el siguiente documento se presentarán las diferentes clases de equivalencia, valores límites, gramática asociada, y gráficos de flujo creado para generar las pruebas de la nueva función a implementar, siguiendo el proceso TDD.

Únicamente se ha explicado todo lo relacionado al cambio número 3 (la nueva funcionalidad a implementar). Esto se debe a que los cambios 1 y 2 solo implican ligeras modificaciones a las funciones ya implementadas, y no generan como tal ningún tipo de test nuevo (exceptuando los test singleton para todas las clases de store).

Este documento no mostrará ningún código como tal ni la estructura de los ficheros utilizados como prueba. Estos estarán implementados en el código subido a github, y en el excel adjunto con el documento respectivamente. El objetivo de este documento es mostrar la base teórica utilizada para poder generar los respectiovs tests.

Clases de Equivalencia y Valores Límite

Puesto que el archivo *json* que recibe la función de *RevoqueKey* posee tres campos a rellenar, las clases de equivalencia que se han generado junto con los respectivos valores límites asociados son los siguientes:

Referentes al campo key:

-Clase key estándar válido

-Clase key Invalido menos de 64 caracteres
    -Valor límite Key con 63 caracteres

-Clase key Invalido más de 64 caracteres
    -Valor límite Key con 65 caracteres

-Clase key Invalido con carácter no hexadecimal
    -Valor límite Key con un carácter no hexadecimal

Referentes al campo Revocation:

-Clase Revocation Temporal válido

-Clase Revocation Final válido

-Clase Revocation Invalido

Referentes al campo de Reason:

-Clase Reason estándar válido
    -Valor límite reason con 0 caracteres
    -Valor límite reason con 99 caracteres
    -Valor límite reason con 100 caracteres

-Clase Reason invalido más de 100 caracteres
    -Valor límite reason con 101 caracteres

Gramática y Árbol de Derivación

Gramática asociada

La gramática asociada a la estructura del archivo json que hemos creado es la siguiente:

JSON : Apertura Info Cierre
Apertura : {
Info: Campo1 separador Campo2 separador Campo3

Campo1: Clave1 igual Valor1
Clave1: Comillas K Comillas
Comillas: "
K: Key
separador: ,
*Valor1: Comillas Ex*64 Comillas (Con Ex repetido 64 veces)*
Ex: 1|2|3|4|5|6|7|8|9|a|b|c|d|e|f

Campo2: Clave2 igual Valor2
Clave2: Comillas R Comillas
R: Revocation
Valor2: Comillas RV Comillas
RV: Temporal | Final

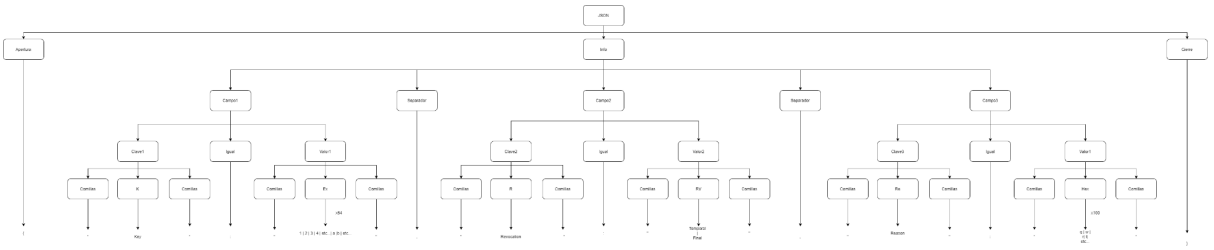
Campo3: Clave3 igual Valor3
Clave3: Comillas Re Comillas
Re: Reason
*Valor3: Comillas Comillas | Comillas Hex Comillas | Comillas Hex*n Comillas con n<=100 (Hex repetido 100 veces)*
Hex: " "|a|b|c|.....(Todos los caracteres posibles, espacio en blanco incluido)
Cierre : }

Dentro del valor Valor 1, entre los dos símbolos no terminales Comillas, hay 64 símbolos *Ex* uno detrás de otro. Solo se han puesto *Ex*64* por economía del lenguaje.

En el caso del valor 3, este puede ser *Comillas Comillas, Comillas Hex Comillas, Comillas HexHex Comillas* | etc.. donde Ex se puede repetir hasta 100 veces, pero solo se ha puesto *Hex*n* por economía del lenguaje.

Árbol de Derivación

El árbol de Derivación que representa la gramática es la siguiente:



**Referirse al Árbol de Derivación anexo para mejor calidad*

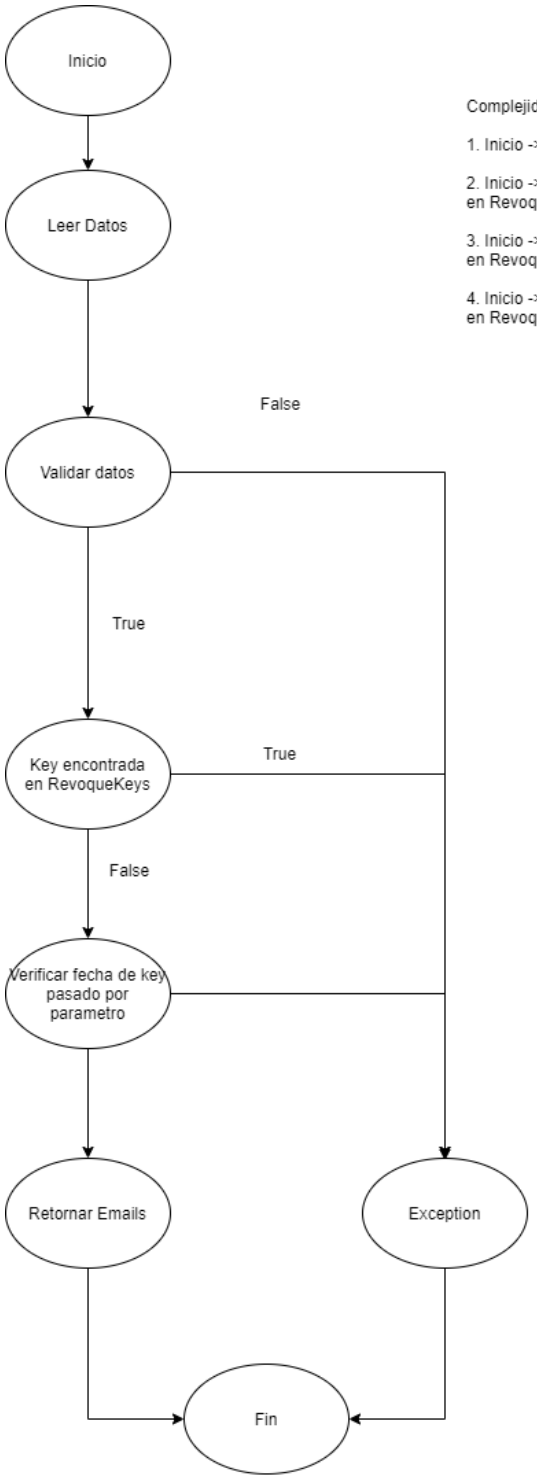
Este árbol de derivación se ha generado gracias a la gramática definida previamente, la cual se usará para generar los casos de prueba posteriores, por cada nodo hoja se podrán generar hasta tres, *Eliminación, Modificación y Duplicación*. Esto significa que al menos se tendrán 72 nodos -por los 24 por hoja- de casos de prueba, pero únicamente sucederá si todos los nodos hoja son distintos, lo cual no es el caso. Esto se debe a que hay varios nodos hoja “Comillas” o “Separador”. Estos nodos repetidos podemos unificarlos en un solo nodo hoja con sus 3 pruebas, es decir, solo se realizarán las 3 pruebas correspondientes al primer nodo hoja no repetido.

A los nodos no hojas les corresponden 2 casos de pruebas: eliminación de nodo hijo y agregación de nodo. Estos consisten , como indica su nombre, en eliminar uno de sus nodos hijo, o agregar un nodo extra que inicialmente no debería estar allí.

Con los test generados gracias al árbol de derivación, se es capaz de predecir todos los posibles errores de formato del archivo *json* pasado como parámetro.

Grafo de Flujo

A continuación, se muestra el gráfico de flujo de la funcionalidad. Con este grafo se podrá predecir todos los distintos posibles escenarios. Este grafo posee 10 nodos, y 8 uniones, por lo que la *Complejidad de McCabe* es 4, siendo 4 los posibles escenarios, los cuales están descritos abajo.



Complejidad de McCabe= 10-8+2 =4

1. Inicio -> Leer Datos -> Validar datos -> Exception

2. Inicio -> Leer Datos -> Validar datos -> Key encontrada en RevoqueKeys -> Exception

3. Inicio -> Leer Datos -> Validar datos -> Key encontrada en RevoqueKeys ->Verificar fecha de key-> Exception

4. Inicio -> Leer Datos -> Validar datos -> Key encontrada en RevoqueKeys ->Verificar fecha de key-> Retornar Email