



Desarrollo del Software, curso 2020-21

# SecureAll

## Práctica final

Sergio Gil Nova	100429089
Mario García Rodríguez	100428982

# Índice

Índice	2
Introducción	3
Requistio 1	4
Requisito 2	5
Requisito 3	6
Conclusiones	7

# Introducción

El presente documento pretende dar cuenta de la ampliación de la aplicación SecureAll. Esta ampliación consiste en la adición de varios requisitos relacionados con los códigos de acceso, la apertura de puertas y la gestión de permisos.

Para estructurar este documento, hemos puesto el foco en el hecho de que la práctica versa sobre desarrollo guiado por pruebas. Por tanto, hemos dividido el documento en las siguientes secciones:

- Introducción: visión general del propósito y la estructura del documento.
- Clases de equivalencia: contiene una subsección para cada componente que precise este método.
- Valores límite: ídem
- Gramáticas y árboles de derivación: ídem
- Gráficos de flujo de control: ídem
- Conclusiones: esta sección cierra el documento recopilando las acciones realizadas sobre el código. Además, expresamos las dificultades enfrentadas y proporcionamos nuestra opinión sobre la práctica.

Esta memoria se complementa con los informes de cobertura de pybuilder y el documento excel que recoge todas las pruebas.

Para facilitar la comprensión del documento, describiremos a continuación los requisitos que se van a añadir a la aplicación:

1. El componente debe almacenar el valor `access_code` para cada `access_request`. Todas las búsquedas en la tienda de Solicitudes deben basarse en el código de acceso.
2. 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.
3. Antes de su fecha de vencimiento, será posible desactivar la clave correspondiente a una persona específica especificando determinados datos en el archivo de entrada JSON.

## Requisito 1

Este requisito consiste en cambiar la forma de guardar solicitudes y alterar el comportamiento del almacén de solicitudes. El objetivo es que el código de acceso se guarde en la propia solicitud y actúe como identificador de la misma.

Para verificar su funcionamiento, hemos utilizado los tests propuestos, además de otros de clases de equivalencia que recogemos en la hoja de cálculo adjunta.

Lo más remarcable de la fase de pruebas es que hemos podido reutilizar todos los tests propuestos, ya que los cambios se han realizado en el archivo de unittest. Estos cambios consisten en verificar que las solicitudes válidas contienen el campo “\_AccessRequest\_\_acess\_code” y que el valor asociado a ese campo coincide con el código generado.

Gracias a esto y sin añadir más casos de prueba, comprobamos que el requisito 1 efectivamente se ha implementado correctamente según las instrucciones remarcadas en el enunciado de esta práctica.

Por tanto, damos por implementada correctamente la primera funcionalidad que se nos ha propuesto.

## Requisito 2

Este requisito exige la creación de un registro de acceso. En este registro se deben introducir, por cada acceso, el código SHA-256 utilizado y la hora de acceso.

Para lograr este objetivo, ha sido necesario crear un nuevo almacén, esta vez de accesos, y una clase dedicada a OpenDoor (tal y como venimos haciendo con RequestAccessCode y AccessKey).

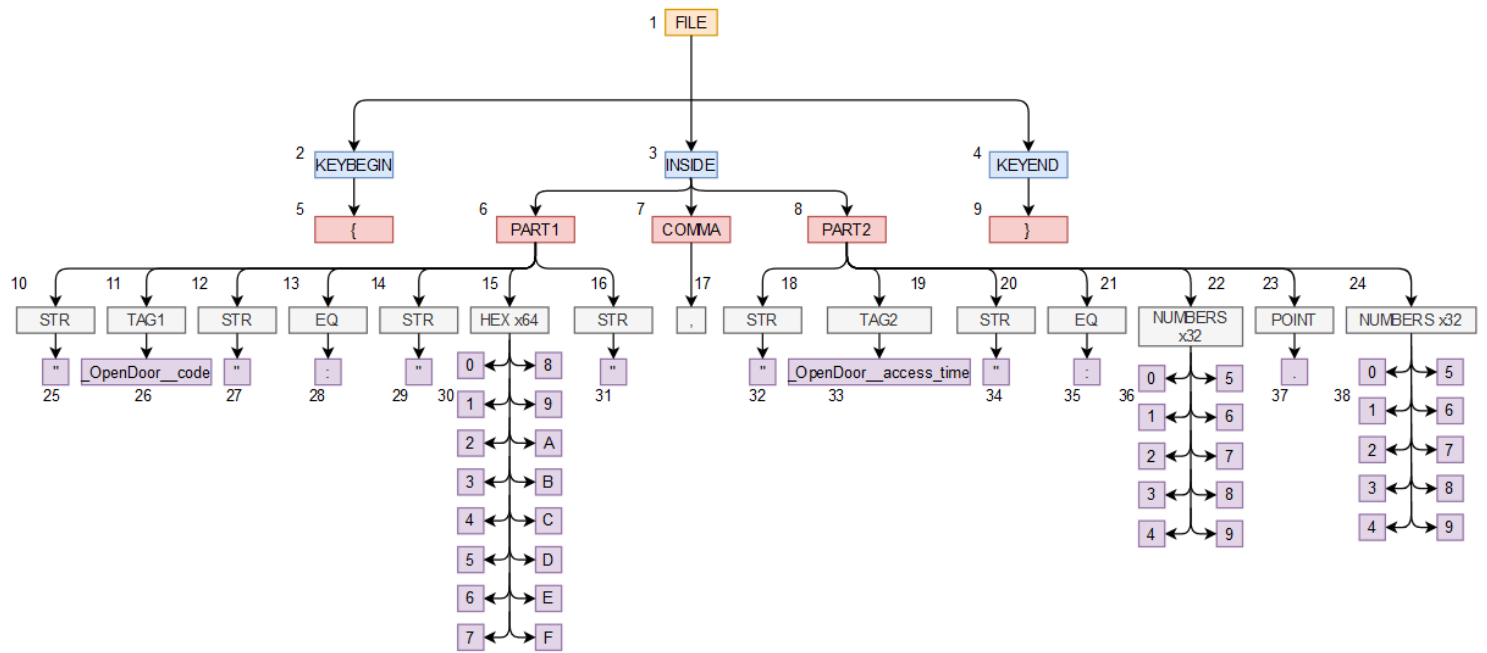
Para el desarrollo de los casos de prueba, hemos decidido implementar estas pruebas mediante un análisis sintáctico completo a través de una gramática. Esta gramática y su árbol de derivación son descritos respectivamente en los siguientes subapartados:

- Gramática completa:

```
FILE          ::= KEYBEGIN INSIDE KEYEND,
KEYBEGIN      ::= {,
KEYEND        ::= },
INSIDE        ::= PART1 COMMA PART2,
PART1 ::= STR TAG1 STR EQ STR HEX STR,
STR           ::= ",
COMMA         ::= , ,
EQ            ::= :,
HEX           ::= a|b|c|d|e|f|0|1|2|3|4|5|6|7|8|9 {64},
PART2 ::= STR TAG2 STR EQ NUMBERS POINT NUMBERS,
NUMBERS       ::= 0|1|2|3|4|5|6|7|8|9 {32}
TAG1          ::= _OpenDoor__code,
TAG2          ::= _OpenDoor__access_time
```

Nótese que para indicar que se requiere determinado número de símbolos terminales, incluimos esta cantidad entre llaves (“{” y “}”). Por ejemplo, el código necesita 64 caracteres, así que lo indicamos en la producción de HEX.

- Árbol de derivación correspondiente a la gramática anterior:



Como podemos observar tanto en la gramática como en el árbol de derivación, se definen unos nodos con los que trabajaremos para la creación e implementación de los casos de prueba correspondientes a la técnica de desarrollo por pruebas del análisis sintáctico.

El árbol está coloreado por niveles, para facilitar su interpretación. Es importante notar que, a efectos de los tests, consideraremos a los tres bloques de terminales como un solo nodo terminal (nodos 30, 36 y 38).

En estos nodos podemos encontrar dos tipos:

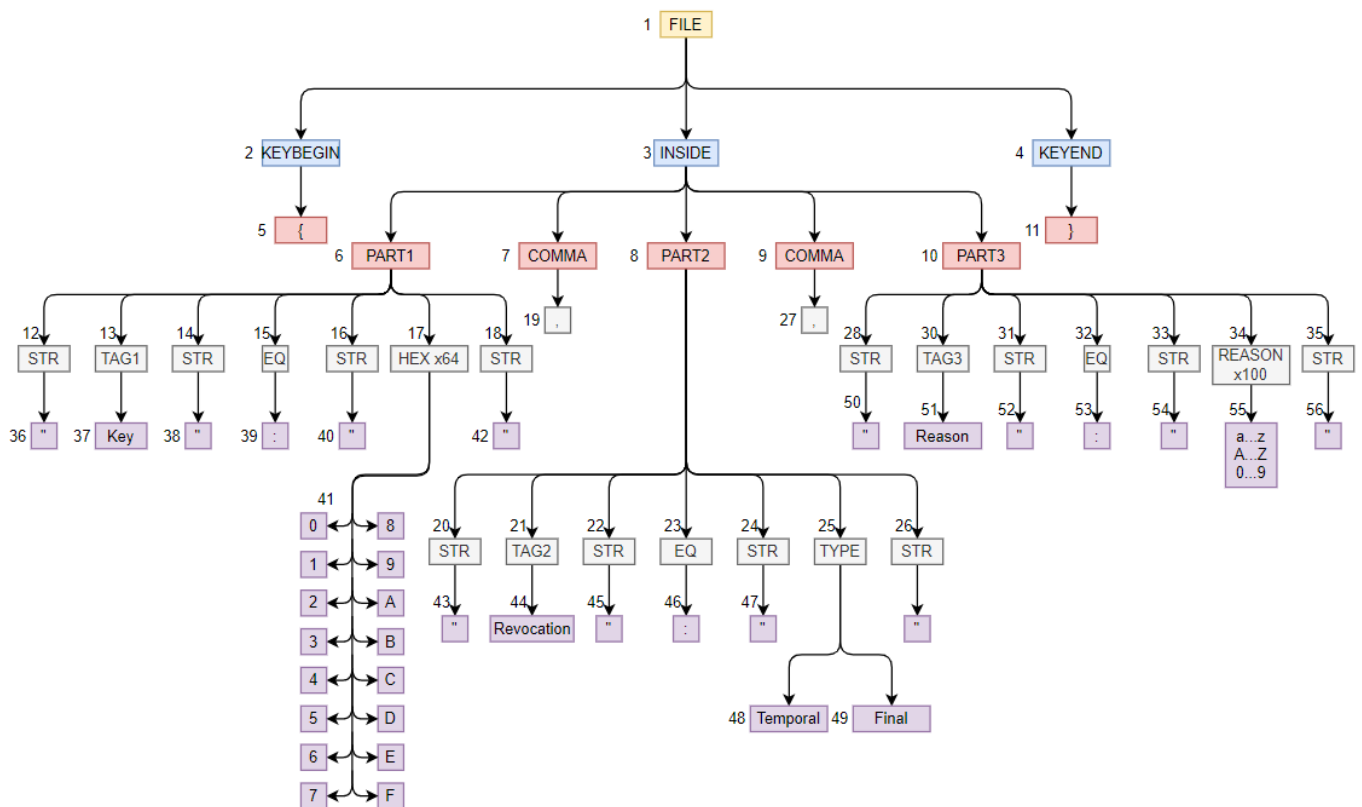
- No terminales: su tipaje es en mayúsculas. El proceso de creación de pruebas para estos nodos recaerá en la eliminación o adición de nodos.
- Terminales: su tipaje no es en mayúsculas. En este caso la creación de pruebas a partir de estos nodos se llevará a cabo modificándolos, ya que la eliminación y la adición ya la cubren los no terminales.

Una vez identificados todos los nodos,

## Requisito 3

Para resolver este problema, hemos optado por la creación de dos almacenes nuevos: uno para las revocaciones definitivas (a modo de histórico) y otro para las revocaciones temporales para que se puedan recuperar en un momento dado (esta decisión de diseño proporciona escalabilidad a la aplicación para futuras mejoras).

Dependiendo del tipo de revocación, las solicitudes van a un almacén o a otro, eliminándose del almacén de solicitudes.



```

FILE          ::= KEYBEGIN INSIDE KEYEND,
KEYBEGIN      ::= {,
KEYEND        ::= },
INSIDE        ::= PART1 COMMA PART2 COMMA PART3,
PART1 ::= STR TAG1 STR EQ STR HEX STR,
STR           ::= " ,
COMMA         ::= , ,
EQ            ::= :,
HEX           ::= a|b|c|d|e|f|0|1|2|3|4|5|6|7|8|9 {64},
PART2 ::= STR TAG2 STR EQ STR TYPE STR,
PART3        ::= STR TAG3 STR EQ STR REASON STR,
TAG1         ::= Key,
TAG2         ::= Revocation,
TAG3         ::= Reason,

```

```
TYPE          ::= Temporal | Final,  
REASON ::= a...zA...Z0...9 {100}
```



## Conclusiones

Para acometer la práctica hemos necesitado activar todos nuestros conocimientos en Desarrollo Guiado por Pruebas. Por tanto, ha sido un trabajo francamente enriquecedor que nos ha permitido condensar todo lo que hemos aprendido a lo largo del curso en estas dos intensas semanas que ha durado la práctica. Nuestra primera aproximación consistió en escribir primero el código y después programar las pruebas, lo cual resultó en numerosos problemas. En ese momento optamos por ponernos serios y desarrollar primero los tests, lo que reportó mejoras en nuestro rendimiento al escribir el código.

Para resolver el primer problema, no necesitamos más que cambiar ligeramente el código (constantes, sentencias de los almacenes, etcétera), por lo que no necesitamos añadir pruebas. Sin embargo, sí fue necesario añadir una aserción más a los tests para comprobar la nueva funcionalidad.

En el segundo problema hemos tenido que añadir módulos al proyecto (uno para la clase `OpenDoor` y un almacén de accesos), además de refactorizar `access_manager` para que guarde el acceso en el almacén. Las pruebas que hemos desarrollado para esta parte complementan a las pruebas proporcionadas por los profesores. La parte nueva está enfocada al análisis sintáctico del JSON que debe guardarse, para comprobar que su formato es correcto.

Por último, la tercera parte ha supuesto un reto en el que hemos tenido que añadir una nueva funcionalidad a `AccessManager`. Sin embargo, gracias a la experiencia acumulada a lo largo del curso nos hemos podido desenvolver satisfactoriamente. Aún así, la parte más costosa fue pasar los tests y corregir las partes que no operaban. Tal fue la dificultad que tuvimos que crear nuevos módulos y almacenes, modificando también la funcionalidad actual de forma profunda.

Esta práctica pone el broche a la asignatura de Desarrollo del Software. El camino recorrido comenzó con la asignatura de Ingeniería del Software, donde aprendimos elicitación de requisitos, diseño conceptual y diseño arquitectónico. Y ahora, hemos profundizado aún más en el mundo del software, conociendo aspectos de deontología y, sobre todo, el diseño dirigido por pruebas. Esta forma de trabajar nos ha enseñado cosas tan básicas pero esenciales como no trabajar sobre el código hasta que pase todas las pruebas.

Gracias a este tipo de lecciones, hemos aumentado nuestros conocimientos en programación y desarrollo de proyectos software, permitiéndonos completar hitos en nuestra carrera como futuros ingenieros informáticos.