Software Development

Bachelor's Degree on Computer Science & Engineering

Course 2020-21

Final Practice

Date: **21/05/21**

GROUP: **89** TEAM: **17**

Members: **Luis Daniel Casais Mezquida, Iván Darío Cersósimo & Jose Sánchez Viloria**

100429021@alumnos.uc3m.es // 100392936@alumnos.uc3m.es // 100381351@alumnos.uc3m.es

1 FUNCTIONALITY 1

1.1 Testing

For the testing of this functionality, as it only changes the internal structure of the code, as long as it passed the tests for the other functionalities, it was enough.

1.2 Implementation

For the implementation of this functionality, some changes were made to the AccessRequest and RequestJsonStore classes. As the requests had to store the access code on its JSON for later searching, an attribute __access_code was added to the AccessRequest class, validating an AccessCode attribute with the hash of the json. Also, the classmethod create_request_from_code only takes the access_code and not the dni. In the RequestJsonStore class, an attribute DNI was added, and ID_FIELD was changed to contain the access code. This makes all functions to search using the access code (ID_FIELD).

2 FUNCTIONALITY 2

2.1 Testing

To implement the tests for this functionality we used structural testing using a control flow diagram, the diagram depicts the control structure of the functionality on the method 'open_door' that stores the key in the access log for further use.
As you can see on the control flow graph on the next page, we end up with 5 different paths to test, according to the McCabe complexity this is the right number of paths when the graph has 14 nodes and 17 edges.
The McCabe complexity is calculated with this formula: V(g) = Edges - Nodes + 2 .

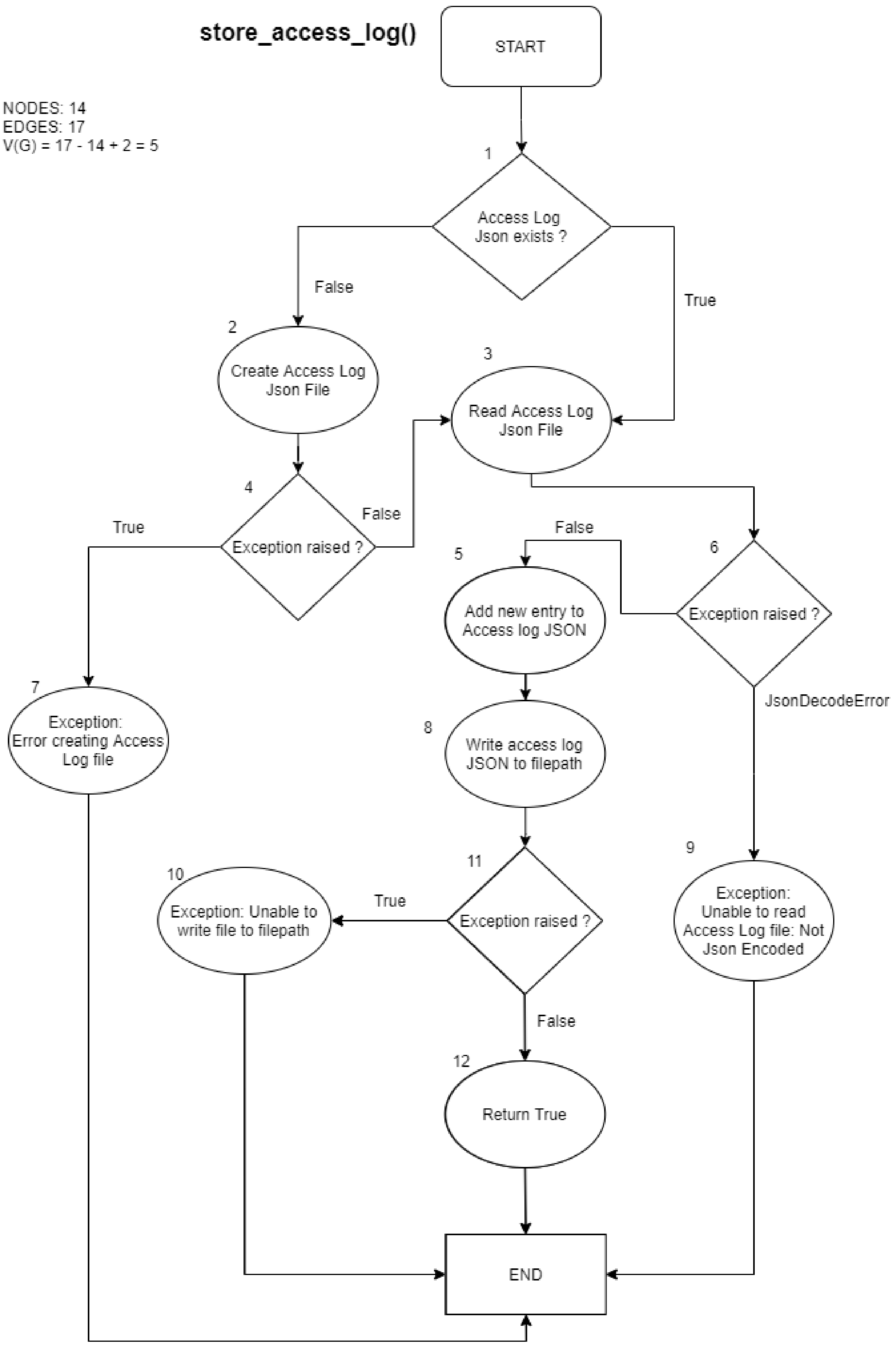
2.2 Implementation

To implement this functionality we created a new python module 'access_log.py' which contains the class in charge to store the key as an access log 'AccessLog'.
The method 'open_door' of our AccessManager class then makes use of 'AccessLog' by creating an instance of this class and calling its 'store_access_log' method which in turn creates a new object of 'AccessLogStore', a class that inherits from JsonStore and overrides some of its methods so that it saves the instance of 'AccessLog' as an item in the access log json file, making sure that the item is not duplicated in the process.

Assumes Key has already been verified in open_door()

store_access_log()

NODES: 14
EDGES: 17
V(G) = 17 - 14 + 2 = 5


START
1
Access Log Json exists ?
False
True
2
Create Access Log Json File
3
Read Access Log Json File
4
Exception raised ?
True
False
5
Add new entry to Access log JSON
6
Exception raised ?
False
JsonDecodeError
7
Exception: Error creating Access Log file
8
Write access log JSON to filepath
9
Exception: Unable to read Access Log file: Not Json Encoded
10
Exception: Unable to write file to filepath
11
Exception raised ?
True
False
12
Return True
END

3 Functionality 3

3.1 Testing

To test the functionality number 3 we created two types of test. The first test checks if the syntax of the json is correct and the second checks if the path of the file is correct. For the first test we used syntax analysis and for the second we used the equivalence classes method. All these cases can be found in the csv file FP_TestCases - FP 2021 FC3-0.csv and FP_TestCases - FP 2021 FC3-1.csv.

Syntax analysis testing

G = (Begin_object, Begin_object, Data, End_object, Field1, Separator, Field2, Separator, Field3, Label_Field1, Equals, Value_Field1, Label_Field2, Equals, Value_Field2, Label_Field3, Value_Field3, Quotation, Value_Label1, Value1, Value_Label2, Value2, Value_Label3, Value3, {, , , }, ", Revocation, Temporal, Final, Reason, a...z, 0...9, ., ,,')

File:= Begin_object, Data, End_object

Begin_object:= {

Data:= Field1, Separator, Field2, Separator, Field3

End_object:= }

Field1:= Label_Field1, Equals, Value_Field1

Separator:= ,

Field2:= Label_Field2, Equals, Value_Field2

Label_Field3:= Quotation, Value_Label3, Quotation

Value_Field3:= Quotation, Value3, Quotation

Quotation:= "

Value_Label1:= Key

Value1:= a|...|f|0|...|9

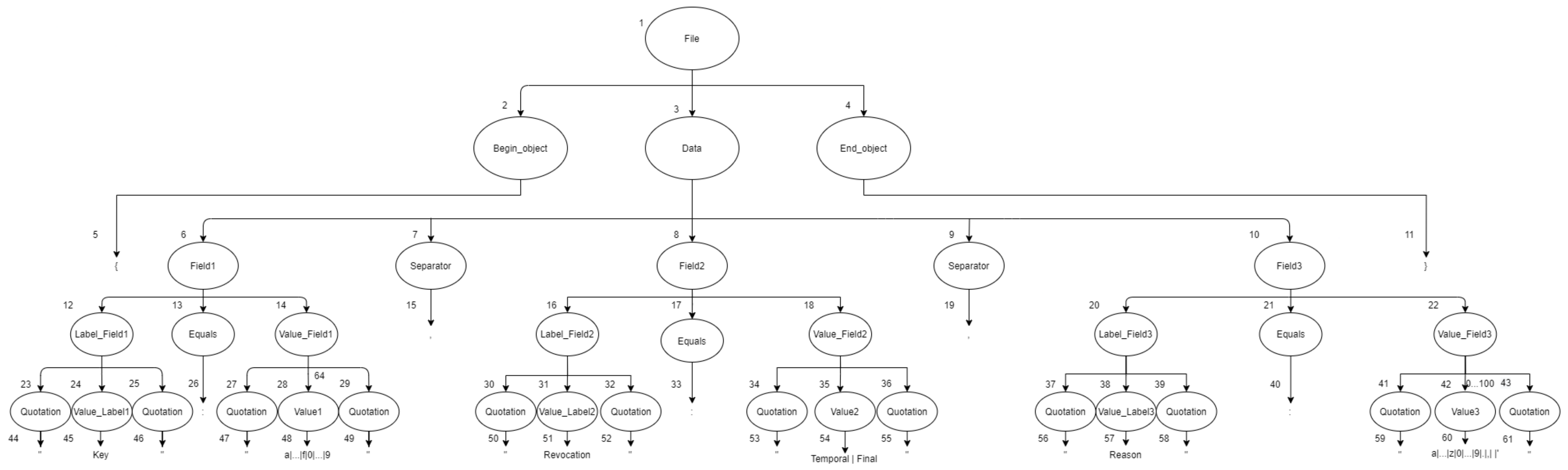
Value_Label2:= Revocation

Value2:= Temporal | Final

Value_Label3:= Reason

Value3:= a|...|z|0|...|9|.|,| |'

Equals:= :


1
File
2
Begin_object
3
Data
4
End_object
5
{
6
Field1
7
Separator
8
Field2
9
Separator
10
Field3
11
}
12
Label_Field1
13
Equals
14
Value_Field1
15
,
16
Label_Field2
17
Equals
18
Value_Field2
19
,
20
Label_Field3
21
Equals
22
Value_Field3
23
Quotation
24
Value_Label1
25
Quotation
26
:
27
Quotation
28
Value1
64
29
Quotation
30
Quotation
31
Value_Label2
32
Quotation
33
:
34
Quotation
35
Value2
36
Quotation
37
Quotation
38
Value_Label3
39
Quotation
40
:
41
Quotation
42
Value3
0...100
43
Quotation
44
"
45
Key
46
"
47
"
48
a|...|f|0|...|9
49
"
50
"
51
Revocation
52
"
53
"
54
Temporal | Final
55
"
56
"
57
Reason
58
"
59
"
60
a|...|z|0|...|9|.|,| |'
61
"

3.2 Implementation

The function revoke_key was implemented in the AccessManager class. To implement the functionality we had to make a json parser specifically for this function. This parse is called RevokeKeyJsonParser. Once we parse the content of the file we found the key on the keyStore. Once we found it we removed it from the store and returned the email or emails.

4 Refactoring

For the refactoring we made use of the base classes like the JsonParser and JsonStore to simplify the AccessManager class as much as we can and avoid having duplicated code, we also made sure to use descriptive names for the variable and argument names of our code and avoided using string literals on the tests to access objects' properties.
Doing this we reduced the AccessManager code lines from almost 200 lines to just 64 lines, resulting in a much cleaner and easy to read code.
Some examples of our refactoring can be seen in the 'RevokeKeyJsonParser' class, which inherits from JsonParser and implements it's own '_validate_json' method, or the 'AccessLogStore' which inherits from JsonStore and implements its own add_item and find_item methods to make sure that there are no duplicated items with the same timestamp and key.

5 Conclusions

We have learnt so much about programming through this course. Using git & github, techniques for writing efficient code with other people by using Test Driven Development, how to refactor… These are all tools that will help us in the future, and we're grateful we learned them now.