uc3m | Universidad Carlos III de Madrid

SOFTWARE DEVELOPMENT

FINAL PROJECT

(Test Driven Development, Refactoring, Simple Design)

AUTHORS

Emmanuel Onyekachukwu Irabor
(100390297)
Cosmin Octavian Petre
(100428943)

INDEX

1 INTRODUCTION

This practice focuses on the development of a new component for the secure_all token management system (revoke_key method), following the Test Driven Development process, specifically Syntax Analysis (Black Box Testing) and Structural Testing (White Box Testing), as well as Refactoring techniques and Simple Design applied both on the new component and on existing components (request_access_code, get_access_key and open_door methods)

2 DOCUMENT DESCRIPTION

This document is divided into two important sections (modifications and newly added revoke functionality). All sections contain specific testing techniques that are used. The document contains details about the codes used and decisions taken while designing the test cases and the code.
Some sections contain a subsection describing the problems encountered and how they are solved.
Diagrams and schematics are added in the functions that require them.
At the end of the document, there is also a Conclusion and references section.
The conclusion section contains comments about the project and some decisions taken by the developers.
The references section contains links to all information used in the development of the project.

3 MODIFICATIONS TO PREVIOUS SOURCE CODE

3.1 AccessLog Functionality

We created the AccessLog class to store the key that has been used in the *open_door* method along with the timestamp of the access in a JSON file. It has an attribute for the key and another one for the access time. It has a *store* method that returns a dictionary with its attributes and a *store_log* method to actually store the access log.

3.2 Store Log Functionality

We created the AccessLogJsonStore class as a child class of the JsonStore parent class to store the access log itself. It has been implemented through Simple Design.

3.3 AccessKey class modifications

We added three attributes related to the revoke key functionality: a boolean (revoked) to keep track of whether this key has been revoked or not, a string (revocation) for the type of revocation and another string (reason) for the revocation reason. We added a *emails_to_str* method that returns a string with the emails from the *notification_emails* attribute for the revoke key functionality; a *get_revoked* method to read from the *storeKeys.json* file where this key is stored and return whether said key has been revoked or not.

3.4 Other modifications

We added a *store* method to return a dictionary with the attributes in classes AccessRequest, AccessKey and AccessLog .

4 MODIFICATIONS TO PREVIOUS TEST CASES

4.1 Request Access Code changes

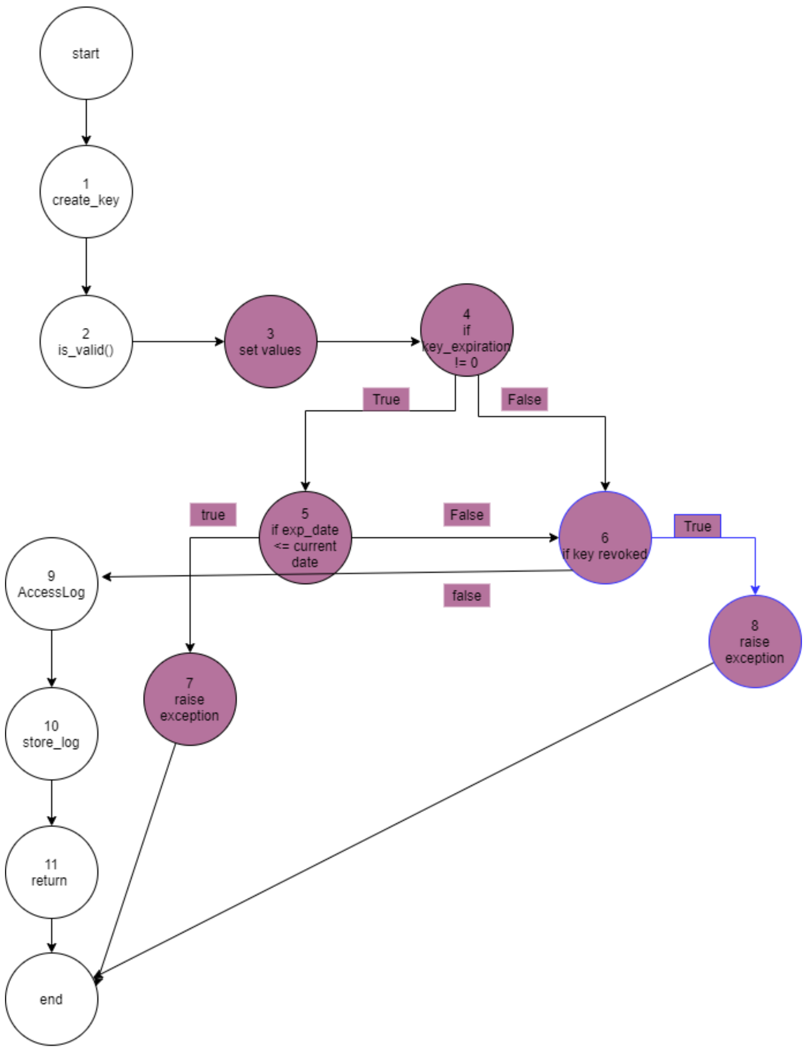
All searches regarding the access request in the previous source code were done using the DNI as the search key. However, for the new implementation of the access_request, the search key has been changed to the Access Code. In the modified method *create_request_from_code*, if the access code has not been found it raises an AccessManagementException; Access code not found in the store. Additionally, the DNI is no longer used as an argument for the aforementioned method since it no longer takes such a parameter. In the *key_nok.json* file, the given Access Code does not exist, but because the previous method searches for the values using the DNI it did not detect this problem, but rather it raises an AccessManagementException saying the DNI is not valid for this Access Code. In order for this test to be successful, we had to modify the expected value in the *testingCases_RF2.csv* for this file. The new expected value is "Access code not found in the store".

4.2 Get Open Door changes

New test for get_open_door

In the previous implementation of the code, a door can only be opened if the access key is valid and not expired. However, since we have added the revoke functionality, we prevent the door from being opened if the key has already been revoked.

To test this we had to redraw the control flow graph:



The part in colour represents the is_valid()method, where the changes were applied. The circle with blue borders is the newly added condition. Two more tests that check if a key has already been revoked were added to the test_open_door. This is because the rest of the cases were already tested.

5 REVOKE FUNCTIONALITY

5.1 Functionality Description

This method is used to deactivate or revoke a key. The conditions required for key revocation are:

- The key has been stored
- The key has not expired
- The key has not already been revoked

The method takes an argument a json file path that has the following structure;
{ "Key": "<String having 64 hex. characters>",
"Revocation":"<Temporal | Final>",
"Reason":"<String having 100 characters max >" }.
It also returns the list of emails corresponding to the given access key as a string.

5.2 Test Definitions

This method takes a JSON file as input (a string with the filename), which contains 3 fields: Key (an access key as generated by get_access_key method, i.e. 64 hexadecimal digits), Revocation (a string, either "Temporal" or "Final") and Reason (a string of up to 100 characters).

5.2.1 Syntax Analysis

5.2.1.1 Grammar

We define an input grammar to test different inputs that the input JSON file that the method takes as a parameter contains.

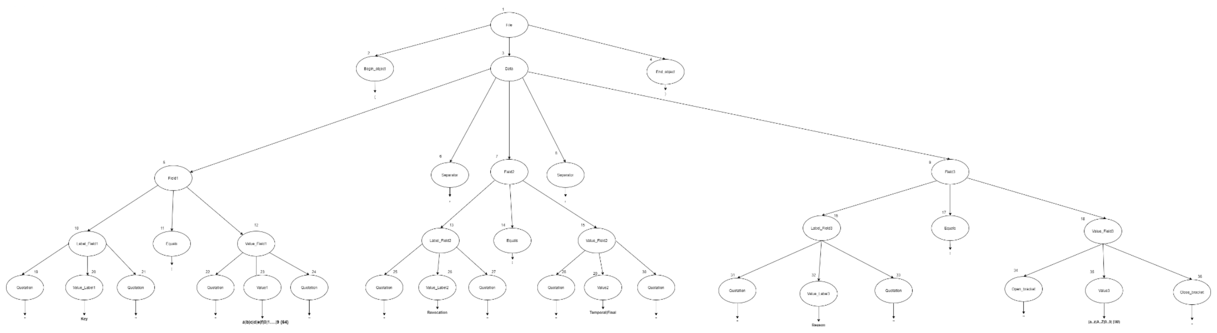
File ::= Begin_object Data End_object
Begin_object ::= {
End_object ::= }
Data ::= Field1 Separator Field2 Separator Field3
Field1 ::= Label_Field1 Equals Value_Field1
Field2 ::= Label_Field2 Equals Value_Field2

Field3 ::= Label__Field3 Equals Value__Field3
Separator ::= ,
Equals ::= :
Quotation ::= "
Label__Field1 ::= Quotation Value__Label1 Quotation
Value__Label1 ::= Key
Value__Field1 ::= Quotation Value1 Quotation
Value1 ::= a|b|c|d|e|f|0|1....|9 {64}
Label__Field2 ::= Quotation Value__Label2 Quotation
Value__Label2 ::= Revocation
Value__Field2 ::= Quotation Value2 Quotation
Value2 ::= Temporal|Final
Label__Field3 ::= Quotation Value__Label3 Quotation
Value__Label3 ::= Reason
Value__Field3 ::= Quotation Value3 Quotation
Value3 ::= (a..z|A..Z|0..9| |\W)

Note: \W represents all non-alphanumeric characters

5.2.1.2 Derivation Tree

This is the derivation tree that results from the above grammar. All Syntax Analysis tests found in the Excel spreadsheet (FP 2021) are derived from this tree, using standard node deletion, duplication and modification. See the referenced Excel spreadsheet for more detail on these tests. Below is a low resolution image of the derivation tree. See the references section for a more detailed view thereof.



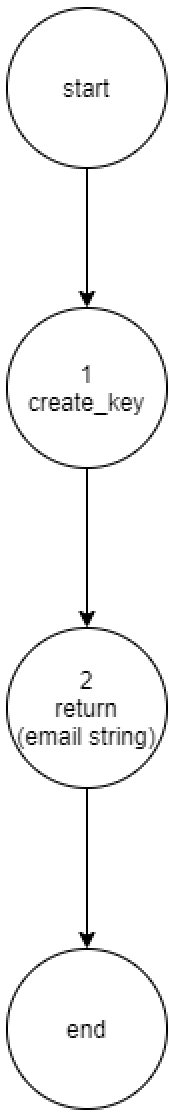
5.2.2 Structural Testing

We defined some test cases following the Structural Testing methodology for the revoke key component. They are found in the referenced Excel file in the “FP 2021 Control Flow” sheet.

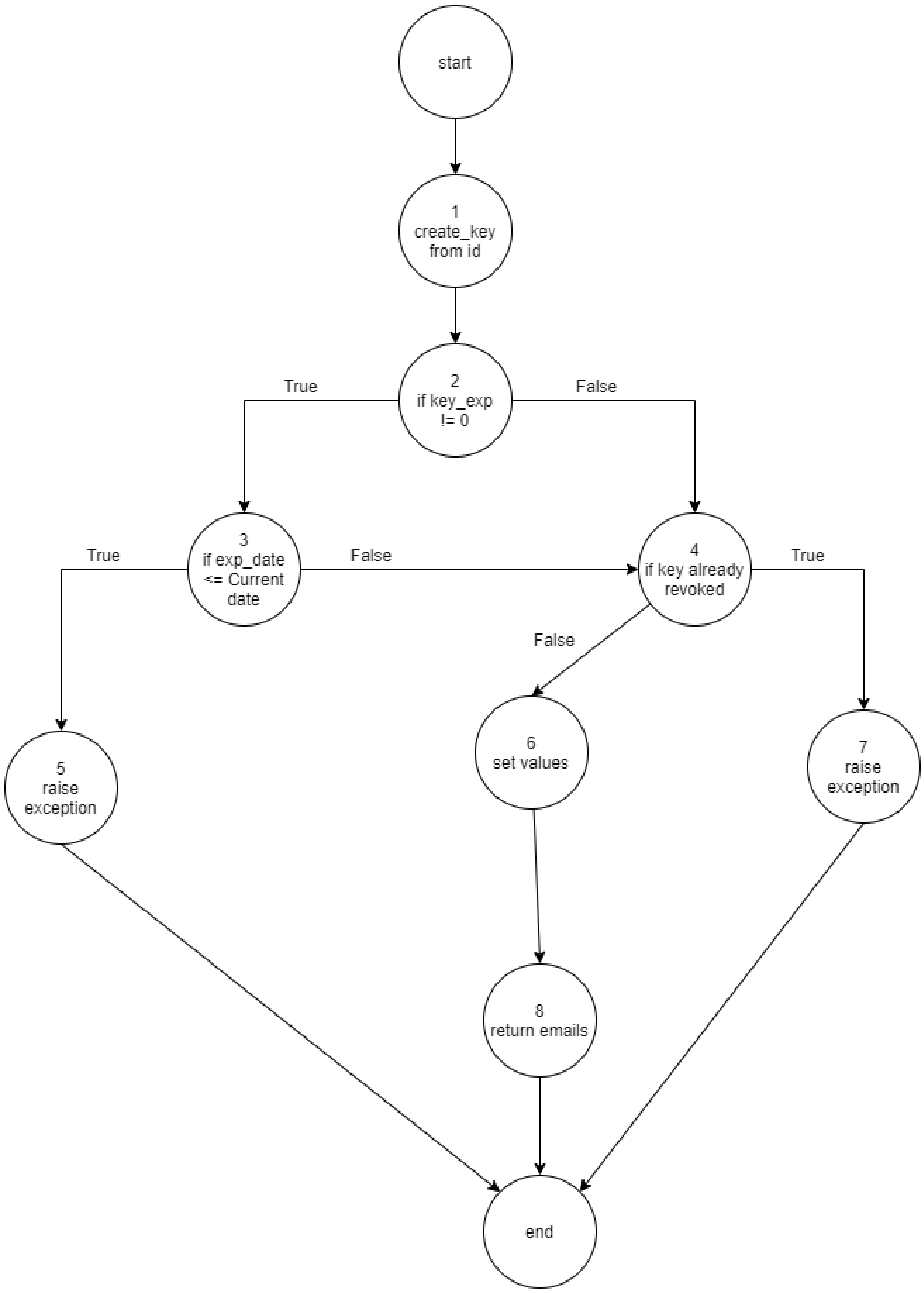
5.2.2.1 Control Flow Graphs

This subsection contains Control Flow Graphs for several methods that have been tested using Structural Testing techniques.

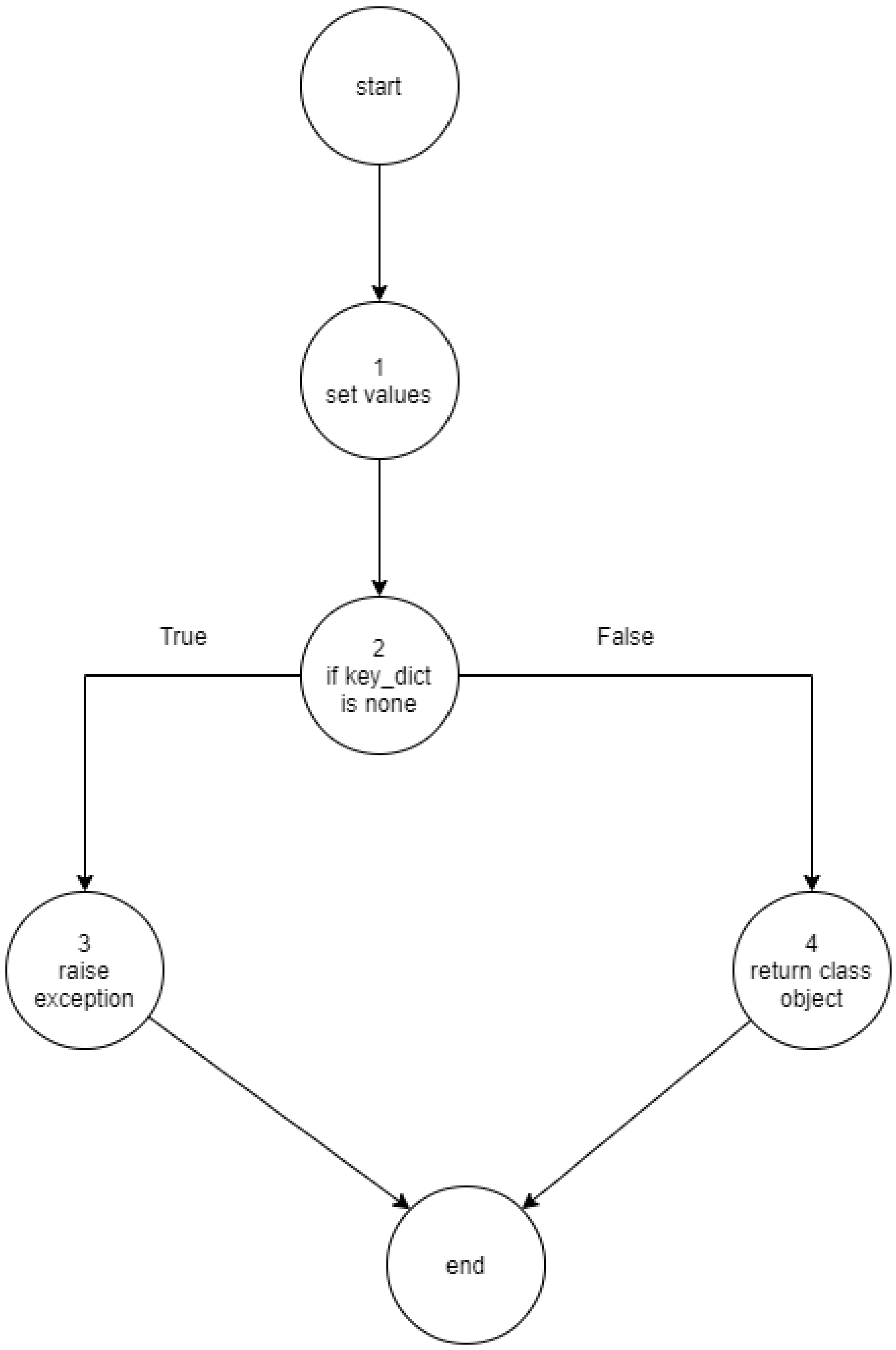
- revoke_key method from AccessManager class:



- revoke_key method from RevokeKey class:



- create_key_from_file_for_revoke from RevokeKey class:



5.2.2.1 Path Selection Algorithms

Below are the paths and test cases chosen for each of the Control Flow Graphs.
In order to know the number of possible paths to be used, the McCabe complexity measure is used. It represents the maximum paths that must be tested to ensure that all statements have been executed at least once and every condition will be evaluated on its true and false side. This is calculated by V(G) = |EDGES| - |NODES| + 2

revoke_key method from AccessManager class: The number of test cases generated using the McCabe complexity is 1. The test paths generated are:

- 1_2 (Key found, not expired and not revoked)

revoke_key method from RevokeKey class: The number of test cases generated using the McCabe complexity is 4. The test paths generated are:

- 1_2_3_5 (Key found but expired)
- 1_2_3_4_6_8 (Key found, not expired and not revoked)
- 1_2_4_7 (Key found, not expired but revoked)
- 1_2_4_6_8 (Key found, not expired and not revoked)

create_key_from_file_for_revoke from RevokeKey class: The number of test cases generated using the McCabe complexity is 2. The test paths generated are:

- 1_2_3 (Key not found)
- 1_2_4 (Key found)

5.2.3 Additional Tests

We included a few more test cases to test some interesting possibilities:

- test_rev_no_exist: test case where the given key is not found in the storeKeys.json file.
- test_rev_expired: test case where the given key is already expired, and thus cannot be revoked.
- test_rev_revoked: test case where the given key is already revoked, and thus cannot be revoked again.

5.3 Code Description

This section describes the implemented code to fulfil the requirements.
Created Classes for this functionality:

- RevokeKey: This class has two methods;
    - create_key_from_file_gfor_revoke(), it is a class method that creates and returns an instance of itself after taking as an argument a file with a specific structure and contains a key.
    - RevokeKey(self); A method that finds the key and checks if it is expired or already revoked, if it has, an Exception is raised. If it hasn't it resets the revoked, revocation and reason attributes of the key to the new ones, and replaces the existing dictionary of

the key with the new one in the store file. And lastly returns the list of emails as a string.

- RevokeJsonStore: A class that inherits from the JsonStore class, with variables specific for the revocation of the key.
- RevokeJsonParser: A class that inherits from the JsonParser class, with variables specific for the revocation of the key.
- Revocation: A class that inherits from the Attribute class, with variables specific for the validation of the “revocation”.
- Reason: A class that inherits from the Attribute class, with variables specific for the validation of the “reason”.

Finally, a method called revoke_key is created in the Access Manager, this method takes as input the file containing an access key, then an object of the RevokeKey class is created. After this object is created, we call the revoke method and it returns its value.

5.4 Design Decisions

This subsection describes the decisions taken to implement this functionality.

5.4.1 Code Design Decisions

To follow the proper coupling and cohesion techniques, we had to create various classes and methods to implement this functionality. As explained in the previous section, new classes were created. This was done in order to follow the structure of the original source code. Each class has its own functionality.

The singleton method was used to make sure various instances of specific classes cannot be created. A method get_revoked() is created in the Accesskey class, this method checks the file of stored keys and returns the revoked value(True or False).

To check the time, the datetime is imported.

Since we were not told how to store the revocation values of a key, we decided to store them in the key_store file. So three new attributes were created in the AccessKey;

- Self._revoked: True or false value that represents if a key is already revoked or not.
- self.___revokation: Type of revocation. Can be Temporal or Final.
- self.___reaseon: a string of characters that represents the reason for the revocation.

5.4.2 Test Design Decisions

As explained in the previous section, two test methods were used for testing this functionality: Syntax analysis and structural analysis.
For the correct and proper functioning of the test, a setupClass is included, it empties the file that will be used and stores new data that is important for the test.

5.5 Problems encountered and Solutions

We had a few issues with the implementation of the functionality that is in charge of storing the Access Code so it can be used as the key to search for Requests in the storeRequests.json file: we tried to add an attribute to the AccessRequest class to store the Access Code itself, but this caused issues with the generated dictionary that is stored in the JSON file. In order to solve this problem, we created a method within AccessRequest called *store* that returns a dictionary with the attributes of the class and an additional entry for the Access Code and we modified the *add_item* method of the JsonStore class to use the *store* method to get the dictionary that has to be stored in the JSON file. We added a *store* method to the other classes that store things in JSON files in order to accommodate this change.

6 CONCLUSIONS

The project was relatively easy to complete, because of the knowledge we had accrued from the guided exercises we did previously. However, we did have some difficulties while choosing the appropriate testing mechanism. Due to the structure of the source code and the new functionality we thought it best to use the Structural and Syntax Analysis methods to test the different components. While testing the code, we tried as much as possible to avoid repeated test cases and tests dependencies.
Before making a commit and push, we made sure the source code was properly tested and had no problems. Therefore, all versions of the source code were committed along with the test logs.
The Test Driven Development methodology was strictly followed for the project, that is the tests were created before the implementation of the code.
In conclusion, this project and all the guided exercises taught us how to work in pairs and write clean and good-quality code.

7 REFERENCES

Links to important documents used in this project.

- Revoke_key Derivation Tree
- Excel Spreadsheet for Test Case Definitions (also included in the repository)
- Function3 get_open_door Control Flow Graph
- revoke_key from AccessManager Control Flow Graph
- revoke_key from RevokeKey Control Flow Graph
- create_key_from_file_for_revoke from RevokeKey Control Flow Graph