Software Development

Final Practice

Members: Carlos Iborra Llopis / Pablo Brasero Martínez
NIAs: 100451170 / 100451247
Group: #4 from G89

Table of Contents

1. First Method: *get_vaccine_date*

For this first method, we decided to use only the equivalence classes and boundary values technique as it is the only useful one of all the different techniques for testing the given field (in this case, date), with all the different cases it can arrive to.

In this case, as we are going to test the different possibilities for an ISO format date, there will be a lot of different boundary values to test (there are three values inside each date: day, month, and year with different boundaries and format) as well as equivalent classes as you will see below.

1.1. Created Tests

In this first part of the report, we will focus on explaining the different tests that have been created for the first method of the Final Practice.

This part is dissected into the tested attributes as well as the division of them into valid and non-valid tests for those attributes.

1.1.1. Tests for Equivalence Classes and Boundary Values

We have decided to use the parameterized test that created the teacher in the *test_get_vaccine_date_tests.py* file in order to save time and space.

By doing this, with a single test function, we can verify that all the attributes worked as expected.

As we needed to add the date as a parameter for the *get_vaccine_date* method (apart from the JSONs' files), we modified the parameter list for it to contain the dates so that we could also check the new different inputs we created for the date in the parameterized tests.

We will only mention the newly created values in the parameters lists for testing the dates.

1.1.1.1. test_date_signature_ecnv1

This first function takes values from the parameters list *param_list_nok,* in which a lot of invalid parameter inputs that must be tested are stored.

The different date values (along with the *test_ok.json* file so that no error came from the JSON file itself) we have created to test the *get_vaccine_date* method are:

- **"2022-03-07":** This date tests when *vaccination_date* is one day earlier than the frozen current date (2022-03-08). This date is part of the Boundary Values (BV) analysis of the first method.

  Raises the exception "vaccination_date equal or earlier than current_date".

- **"2022-03-08":** This date tests when *vaccination_date* is equal to the frozen current date. This date is part of the Boundary Values (BV) analysis of the first method.

  Raises the exception "vaccination_date equal or earlier than current_date".

- **"2021-03-18":** This date tests when *vaccination_date* is one year earlier but has a valid month and day number. This date is part of the Boundary Values (BV) analysis of the first method.

  Raises the exception "vaccination_date equal or earlier than current_date".

- **"2022-02-18":** This date tests when *vaccination_date* is one month earlier but has a valid year and day number. This date is part of the Boundary Values (BV) analysis of the first method.

  Raises the exception "vaccination_date equal or earlier than current_date".

- **"2022-13-01":** This date tests when *vaccination_date* is one month above the maximum number of months (month number is 13 while the maximum monthly number in a year is 12). This date is part of the Boundary Values (BV) analysis of the first method.

  Raises the exception "Wrong vaccination_date format".

- **"2022-00-04":** This date tests when *vaccination_date* is one month less than the minimum number of months (month number is 0 while the minimum month number in a year is 1). This date is part of the Boundary Values (BV) analysis of the first method.

  Raises the exception "Wrong vaccination_date format".

- **"2022-03-32":** This date tests when *vaccination_date* day number is 32 in a month with a maximum of 31 days. This date is part of the Boundary Values (BV) analysis of the first method.

  Raises the exception "Wrong vaccination_date format".

- **"2022-04-31":** This date tests when *vaccination_date* day number is 31 in a month with a maximum of 30 days. This date is part of the Boundary Values (BV) analysis of the first method.

  Raises the exception "Wrong vaccination_date format".

- **"2022-02-29":** This date tests when *vaccination_date* day number is 29 in a month with a maximum of 28 days. This date is part of the Boundary Values (BV) analysis of the first method.

  Raises the exception "Wrong vaccination_date format".

- **"2024-02-30":** This date tests when *vaccination_date* day number is 30 in a month with a maximum of 30 days (due to leap year). This date is part of the Boundary Values (BV) analysis of the first method.

  Raises the exception "Wrong vaccination_date format".

- **"2022-04-00":** This date tests when *vaccination_date* is one day less than the minimum number of days (day number is 0 while the minimum day number is 1 for every month). This date is part of the Equivalence Classes (EC) analysis of the first method.

  Raises the exception "Wrong vaccination_date format".

- **"04-2022-04":** This date tests when *vaccination_date* has the format: MM-YYYY-DD or DD-YYYY-MM. This date is part of the Equivalence Classes (EC) analysis of the first method.

  Raises the exception "Wrong vaccination_date format".

- **"04-04-2022":** This date tests when *vaccination_date* has the format: MM-DD-YYY or DD-MM-YYYY. This date is part of the Equivalence Classes (EC) analysis of the first method.

  Raises the exception "Wrong vaccination_date format".

- **"22-04-04":** This date tests when *vaccination_date* has the format: YY-MM-DD or YY-DD-MM. This date is part of the Equivalence Classes (EC) analysis of the first method.

  Raises the exception "Wrong vaccination_date format".

- **"2022-4-04":** This date tests when *vaccination_date* month has no zero. This date is part of the Equivalence Classes (EC) analysis of the first method.

  Raises the exception "Wrong vaccination_date format".

- **"2022-04-4":** This date tests when *vaccination_date* day has no zero. This date is part of the Equivalence Classes (EC) analysis of the first method.

  Raises the exception "Wrong vaccination_date format".

- **"2022-4-4":** This date tests when *vaccination_date* day and month have no zero. This date is part of the Equivalence Classes (EC) analysis of the first method.

  Raises the exception "Wrong vaccination_date format".

- **"202203-18":** This date tests when *vaccination_date* has the format: YYYYMM-DD. This date is part of the Equivalence Classes (EC) analysis of the first method.

  Raises the exception "Wrong vaccination_date format".

- **"202203 18":** This date tests when *vaccination_date* has the format: YYYYMM DD. This date is part of the Equivalence Classes (EC) analysis of the first method.

  Raises the exception "Wrong vaccination_date format".

- **"2022-0318":** This date tests when *vaccination_date* has the format: YYYY-MMDD. This date is part of the Equivalence Classes (EC) analysis of the first method.

  Raises the exception "Wrong vaccination_date format".

- **"2022 0318":** This date tests when *vaccination_date* has the format: YYYY MMDD. This date is part of the Equivalence Classes (EC) analysis of the first method.

  Raises the exception "Wrong vaccination_date format".

- **"2022 03 18":** This date tests when *vaccination_date* has the format: YYYY MM DD. This date is part of the Equivalence Classes (EC) analysis of the first method.

  Raises the exception "Wrong vaccination_date format".

- **"20220318":** This date tests when *vaccination_date* has the format: YYYYMMDD. This date is part of the Equivalence Classes (EC) analysis of the first method.

  Raises the exception "Wrong vaccination_date format".

- **20221201:** This date tests when *vaccination_date* is not a string value. This date is part of the Equivalence Classes (EC) analysis of the first method.

  Raises the exception "Wrong vaccination_date format".

1.1.1.2. test_get_vaccine_date_ok

This first function test takes the parameters: *test_ok.json* file and "**2022-03-18**" (as date value) in order to test *get_vaccine_date* with valid parameters.

This date is part of the Equivalence Classes (EC) analysis of the first method.

This test (as it is valid) returns as result the corresponding vaccination appointment date signature: “5a06c7bede3d584e934e2f5bd3861e625cb31937f9f1a5362a51fbbf38486f1c”.

1.1.1.3. test_get_vaccine_date_ok_bv_1

This first function test takes the parameters: *test_ok.json* file and "**2022-12-18**" (as date value) in order to test *get_vaccine_date* with the maximum monthly number in a year.

This date is part of the Boundary Values (BV) analysis of the first method.

This test (as it is valid) returns as result the corresponding vaccination appointment date signature: "8f1ec866e9017e15921649419fd7ddbd1927fcb170906738e1dc3125e2d599ec".

1.1.1.4. test_get_vaccine_date_ok_bv_2

This first function test takes the parameters: *test_ok.json* file and "**2023-01-18**" (as date value) in order to test *get_vaccine_date* with the minimum monthly number in a year.

This date is part of the Boundary Values (BV) analysis of the first method.

This test (as it is valid) returns as result the corresponding vaccination appointment date signature: "1b0887bf18529ee1f12447d49841dc9229cddcc9b10bf54f37566fdd74ecf11d".

2. Second Method: *cancel_appointment*

Regarding this second method, we decided to use two different testing techniques: analysis of the equivalence classes and boundary values and the syntax analysis techniques.

The analysis of the equivalence classes and boundary values technique will be very useful for us to test all the different values and boundaries we can have in each of the JSONs' three different values (*date_signature*, *cancelation_type*, and *reason*).

As for the syntax analysis technique, it will serve for us to analyze and study the different possibilities the JSONs' file structure allows us to modify, create, or duplicate. This is also a very good technique for testing the possible errors and holes in our method we could have not noticed before.

We did not use the structural testing (white box) technique as opposed to the third function of guided exercise 3 (vaccine management), our method is enormous and it would not be reasonable to create a path in it as it would take forever, thus not being efficient.

To sum up, the disadvantages of using this structural testing technique (leading to a huge number of execution paths (which can not be all tested), data sensitivity errors…) weighed more for us than the advantages (checking every path through the code has been identified and tested) as this method is enormous but not it has a not very complex design neither does contain many loops so the different paths can be identified and tested easily without needing of performing the whole structural analysis).

2.1. Created Tests

In this second part of the report, we will focus on explaining the different tests that have been created for the second method of the Final Practice.

This part is dissected into the tested attributes as well as the division of them into valid and non-valid tests for those attributes.

2.1.1. Tests for Equivalence Classes and Boundary Values

We have decided to create the class ***TestCancelAppointmentEcBv*** for containing the five different functions we have created for equivalence classes and boundary values testing purposes, of which two functions perform parameterized tests in order to save time and space, while the other three functions contain special cases which could not be tested within the parameterized functions.

In this way, with a single test function, we can test at the same time whether a bunch of different input files work as we expected.

2.1.1.1. test_parametrized_valid_ec_bv_cancellation_appointment

This first function takes values from the list *param_list_valid_ec_bv,* in which all the valid JSON files that must be tested are stored.

Therefore, we can verify that the valid JSONs are indeed valid by checking the validity of their content.

As a consequence of being valid JSONs', the test function should return the corresponding vaccination appointment date signature which will be used to check the validity of the previously mentioned JSONs'.

- ***test_cancellation_type_ecv1.json*****:** This JSON file for the field *cancelation_type* tests when *cancelation_type* is correct ("Temporal").

  This JSON file is of the Equivalence Classes (EC) analysis of the second method.

- ***test_cancellation_type_ecv2.json*****:** This JSON file for the field *cancelation_type* tests when *cancelation_type* is correct ("Final").

  This JSON file is of the Equivalence Classes (EC) analysis of the second method.

- ***test_date_signature_ecv1.json*****:** This JSON file for the field *date_signature* tests when *date_signature* is correct and has exactly the SHA256 64 characters.

  This JSON file is part of the Boundary Values (BV) analysis of the second method.

- ***test_reason_ecv1.json*****:** This JSON file for the field *reason* tests when *reason* length is exactly 2 characters.

  This JSON file is part of the Boundary Values (BV) analysis of the second method.

- ***test_reason_ecv2.json*****:** This JSON file for the field *reason* tests when reason length is in the range [2, 100], specifically 3 characters.

  This JSON file is part of the Boundary Values (BV) analysis of the second method.

- ***test_reason_ecv3.json*****:** This JSON file for the field *reason* tests when reason length is in the range [2, 100], specifically 99 characters.

  This JSON file is part of the Boundary Values (BV) analysis of the second method.

- ***test_reason_ecv4.json*****:** This JSON file for the field *reason* tests when *reason* length is exactly 100 characters.

  This JSON file is part of the Boundary Values (BV) analysis of the second method.

2.1.1.2. test_parametrized_not_valid_ec_bv_cancellation_appointment

This second function takes values from the list *param_list_not_valid_ec_bv,* in which all the not valid JSON files that must be tested are stored.

As a consequence of being not valid JSONs', the test function should receive the different raised exceptions from the execution of the second method (*cancel_appointment*) having as input the different mentioned JSONs'

- ***test_cancellation_type_ecnv1.json*****:** This JSON file for the field *cancelation_type* tests when *cancelation_type* is not a valid string value ("Cancel").

  This JSON file is of the Equivalence Classes (EC) analysis of the second method.

  As a consequence, it raises the exception: "Invalid cancelation_type".

- ***test_cancelation_type_ecnv2.json***: This JSON file for the field *cancelation_type* tests when *cancelation_type* is not a string (123456).

  This JSON file is of the Equivalence Classes (EC) analysis of the second method.

  As a consequence, it raises the exception: "Invalid cancelation_type: not a string".

- ***test_date_signature_ecnv1.json***: This JSON file for the field *date_signature* tests when *date_signature* has less than the SHA256 64 characters (63 characters).

  This JSON file is part of the Boundary Values (BV) analysis of the second method.

  As a consequence, it raises the exception: "Invalid date_signature".

- ***test_date_signature_ecnv2.json***: This JSON file for the field *date_signature* tests when *date_signature* has more than the SHA256 64 characters (65 characters).

  This JSON file is part of the Boundary Values (BV) analysis of the second method.

  As a consequence, it raises the exception: "Invalid date_signature".

- ***test_date_signature_ecnv3.json***: This JSON file for the field *date_signature* tests when *date_signature* has an invalid SHA256 value (for example, contains g).

  This JSON file is of the Equivalence Classes (EC) analysis of the second method.

  As a consequence, it raises the exception: "Invalid date_signature".

- ***test_date_signature_ecnv4.json***: This JSON file for the field *date_signature* tests when *date_signature* is not a string.

  This JSON file is of the Equivalence Classes (EC) analysis of the second method.

  As a consequence, it raises the exception: "Invalid date_signature: not a string".

- ***test_date_signature_ecnv5.json***: This JSON file for the field *date_signature* tests when *date_signature* is not in the file *store_date.json*.

  This JSON file is of the Equivalence Classes (EC) analysis of the second method.

  As a consequence, it raises the exception: "The appointment received does not exist".

- ***test_reason_ecnv1.json***: This JSON file for the field *reason* tests when *reason* length is below the range [2, 100], specifically 1 character.

  This JSON file is part of the Boundary Values (BV) analysis of the second method.

  As a consequence, it raises the exception: "Invalid reason".

- ***test_reason_ecnv2.json***: This JSON file for the field *reason* tests when *reason* length is above the range [2, 100], specifically 101 characters.

  This JSON file is part of the Boundary Values (BV) analysis of the second method.

  As a consequence, it raises the exception: "Invalid reason".

- ***test_reason_ecnv3.json*****:** This JSON file for the field *reason* tests when *reason* is not a string.

  This JSON file is part of the Boundary Values (EC) analysis of the second method.

  As a consequence, it raises the exception: "Invalid reason: not a string".

2.1.1.3. test_specific_non_valid_ec_bv_encv6_cancellation_appointment

In this third function, we use as input file the JSON file ***test_date_signature_ecnv6.json*** for the field *date_signature* which tests when the date for the *date_signature* in the file *store_date.json* has already passed.

This JSON file is of the Equivalence Classes (EC) analysis of the second method.

As a consequence, it raises the exception: "The appointment date received has already passed".

We have created a specific test for this JSON as we needed to create a test function with a second internal function to freeze the time for us to simulate that the appointment date had passed, and as you would imagine, this was not possible to do within the function *test_parametrized_not_valid_ec_bv_cancellation_appointment*.

2.1.1.4. test_specific_non_valid_ec_bv_encv7_cancellation_appointment:

In this fourth function, we use as input file the JSON file ***test_date_signature_ecnv7.json*** for the field *date_signature* which tests *date_signature* is in the file *store_vaccine.json* (meaning that the given patient has been already vaccinated).

This JSON file is of the Equivalence Classes (EC) analysis of the second method.

As a consequence, it raises the exception: "Vaccine has already been administered".

We have created a specific test for this JSON as we needed to create a test function to test a specific date_signature which is the one already stored in the file *store_vaccine.json*. We also created two different appointments to check if everything worked fine although having two different appointments in the *store_date.json* file, therefore checking that the second method navigated correctly between the different appointments and JSON files. As you may imagine, this was not possible to do within the second created function for this second method.

2.1.1.5. test_specific_non_valid_ec_bv_encv8_cancellation_appointment:

In this fifth function, we use as input file the JSON file ***test_date_signature_ecnv8.json*** for the field *date_signature* which tests when *date_signature* is already in the file *store_cancellation.json*.

This JSON file is of the Equivalence Classes (EC) analysis of the second method.

As a consequence, it raises the exception: "Appointment has already been canceled".

We have created a specific test for this JSON as we needed to create a test function that canceled the same appointment twice (calling twice the second method (*cancel_appointment*)), thus forcing the test to rise the previously mentioned error (as the second method run tries to cancel an appointment which is already stored in *store_cancellation.json)*. As you may imagine, this was not possible to do within the second created function for this second method.

2.1.2. Tests for Syntax Analysis

Our second created class, ***TestCancelAppointmentSyntaxAnalysis***, contains two different parametrized functions we have created for syntax analysis testing purposes (one for valid JSONs' and the other one for invalid JSONs'), this way, we save time and space.

2.1.2.1. test_parametrized_valid_sa_cancellation_appointment:

This first function is responsible for verifying that the JSON file is delivered as input is valid, if it is valid, the *date_signature* will be returned and no error will be raised meaning the *input_file* has both correct values and structure.

For us to perform this test more efficiently, we have created the parameter list *param_list_valid_sa*, which at the moment just contains a single valid JSON file (*json_valid.json*), but which could be extended in the future without needing to create new functions for every new test valid JSONs'.

2.1.2.2. test_parametrized_not_valid_sa_cancellation_appointment:

This second function is responsible for checking all the cases in which the JSON files can raise an error, the function takes values from the parameter list *param_list_not_valid_sa* (containing the name of each JSON file and its associated expected exception) in which all the nodes that must be checked are found.

We made it as a parameterized test as it lets us go through all the nodes and perform the corresponding testing on them in the most efficient way possible (as we can check all the invalid JSON files (97 in total) with just one single function). Each of the nodes will raise its corresponding exception message which will be received by this second function and compared in order to test the validity of the test.

2.1.3. Extra Tests for Non-Tested Files

Apart from the two previous testing classes, we have decided to create a third class, ***TestExtraNonTested,*** in order to test the files that we were unable to verify with the previous two classes' functions so that nothing leaves open the possibility of making our program fail.

2.1.3.1. test_input_file_not_found

The purpose of this first function is to test what would happen if the entered *input_file* is not found.

As result, the function receives the following exception message: "File is not found".

2.1.3.2. test_input_file_not_json

The purpose of this second function is to test what would happen if the entered *input_file* is not a *.json* file.

As result, the function receives the following exception message: "File is not found".

2.1.3.3. test_appointment_file_not_found

The purpose of this second function is to test what would happen if the entered appointment file is not found.

As result, the function receives the following exception message: "File is not found".

2.1.3.4. test_appointment_file_not_json

The purpose of this second function is to test what would happen if the entered appointment file is not a *.json* file, the program raises an error.

As result, the function receives the following exception message: "File is not found".

2.2. Grammar

As a result, our grammar is the following:

```
{
  "date_signature":"<String having 64 hexadecimal characters>",
  "cancelation_type":"Temporal | Final",
  "reason":"<String with 2 - 100 characters>"
}
```

Sample Grammar:

```
{
  "date_signature":"230db695dfb51529e5210244929c3bd61f203d32da20adeaf143a03a312f198f",
  "cancelation_type":"Temporal",
  "reason":"A mistake has taken place when doing the appointment"
}
```

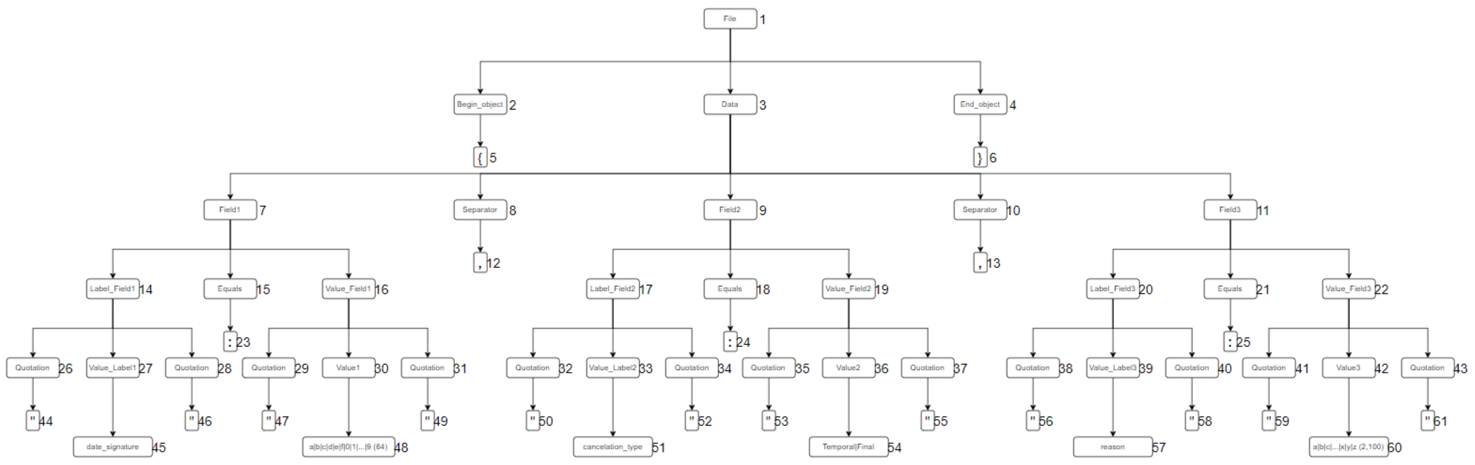
File ::= Begin_object Data End_object
Begin _object ::= {
End _object ::= }
Data ::= Field1 Separator Field2 Separator Field3
Field1 ::= Label_Field1 Equals Value_Field1
Field2 ::= Label_Field2 Equals Value_Field2
Field3 ::= Label_Field3 Equals Value_Field3
Separator ::= ,
Equals ::= :
Quotation ::= "

Label_Field1 ::= Quotation Value_Label1 Quotation
Value_Label1 ::= date_signature
Value_Field1 ::= Quotation Value1 Quotation
Value1 ::= a|b|c|d|e|f|0|1….|9 (64)

Label_Field2 ::= Quotation Value_Label2 Quotation
Value_Label2 ::= cancelation_type
Value_Field2 ::= Quotation Value2 Quotation
Value2 ::= Temporal|Final

Label_Field3 ::= Quotation Value_Label3 Quotation
Value_Label3 ::= PatientSystemID
Value_Field3 ::= Quotation Value3 Quotation
Value1 ::= a|b|c|...|x|y|z (2,100)

2.3. Derivation Tree


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