SOFTWARE DEVELOPMENT

Computer Science

uc3m | Universidad Carlos III de Madrid

Guided Exercise 3

Collective Code Ownership and Coding Standards

Course 2022/2023

Group 89 -> Inner Group 20

Sergio Barragán Blanco, Eduardo Alarcón Navarro

100472343 , 100472175

100472343@alumnos.uc3m.es, 100472175@alumnos.uc3m.es

Table of Contents:

Function 1: Requesting the shipment

Introduction:

The first part of the guided exercise consisted of generating the necessary tests following the method of Test Driven Development (TDD), as well as developing the implementation of the interface regarding Functional requirement 01, called `register_order`.
To implement this function, and its correct operation, we need to check the input parameters of the function are valid. In this case, the EAN13 number is valid, the order_type is REGULAR or PREMIUM, the address is a string between 20 and 100 characters with at least 2 strings separated by one white space at least, the phone number is valid (9 or 12 digits if we include the "+34") and a valid Spanish zip code.

Code Analysis:

As the exercise requested, we created a function named `register_order`, which takes as input parameters `product_id` which is a string, `order_type` which is a string, `address` which is a string, `phone_number` which is a string, `zip_code` which is a string.
The first thing we did was check that the parameters were correct. To do that, we implemented the functions, validate_order_type(order_type), validate_address(address), validate_phone_number(phone_number), validate_zip_code(zip_code) and validate_ean13(product_id).
We then created an OrderRequest Object and then, with the method to_json we store it in the `order_request.json`

Testing:

To validate that the function would work under any possible scenario, we designed a battery of tests, found in the excel file. To better understand the tests and the nomenclature, we will explain them in detail.

The excel is divided into various columns, which represent different characteristics of the test. The first column indicates if the test is working on the Input or Output aspect of the function. The Field addresses the section of the function that is being tested, for example, the different possibilities of Product_Ids' or the phone numbers. Then, we have the Validity of the test, if it should return a valid or invalid state. To properly identify the tests, we decided to work on a naming system that followed the following format, which will be explained using this example: EC_V_1. The first element may be EC, representing it as an Equivalence Class test or a Boundary Value test, which will be BV. The second element indicates if the test is Valid, with a V, and with an NV if the test is Not Valid. The last element represents the number of tests.

To correctly test our function, we designed 45 tests, which include 7 for the product_id using BV since it's a number (EAN13) and EC, 5 for the OrderType with only EC needed as tests (only upper is valid), 11 for the address including BV (length of the string) and EC for valid and none values, 6 for the phone number with EC has only 9 digits numbers are valid (or a string with +34 and a valid number), 11 for the zip_code, including BV (since its a number in a range of values) and EC (for valid and none cases) and 5 EC for the output file.

The name of the file you may find the tests descriptions is `GE3_TestCasesTemplate_2023`

Testing Excel:

To divide the tests, we are going to have subsections, with regard to each input:

- ProductID: We checked the boundary values for length 13 and The equivalence classes for the string containing only numbers, a number different from an EAN13 number, an EAN13 number, and a None value.
- Order_Type: We only did equivalence classes as there are only two possibilities, "PREMIUM" and "REGULAR". We also thought that it would be convenient to include the tests for "premium" and "regular" as they might be common uses although they aren't valid.
- Address: To check the address, we did both equivalence class tests and boundary values tests. The boundary values were used with two different parameters, the number of spaces and the length of the total string, for no spaces, one space, and two spaces, and regarding the length of the string (19, 20, 21, 99, 100, 101). Regarding the equivalence classes, we did one for the correct address and one for the None
- Phone_number: We did both Equivalence classes and boundary values for the phone number. The equivalence classes are regarding a correct phone number with 9 digits, one with 12 characters, and an empty one. For the boundary values, we did one with a length of 8, another one with a length of 9, and one with a length of 10 numbers
- Zip_Code: For the zip code we went all out. We check the equivalence classes, the correctness, and the None or null type. But for the boundary values, we check the value of the number is 52999 or less, by checking 52998, 52999, and 53000, the minimum value, 1001, testing 1000, 1001, and 1002. We also did tests for the length of the number, for lengths 4, 5, and 6.
- Output_File: We only have equivalence classes, testing if the file does not exist, if the length is not valid, if the file is not modified, if the file is correct, and if the resulting file is not a JSON.

Difficulties:

While implementing this function, following the functional requirements, we found out the most difficult part was sticking to the proposed requirements, as we, being students, have other ideas we would have liked to implement, but could not do as they collided with the project's requirements. We also had problems with the naming convention of the tests.

Correction and Comments regarding the partner's tests:

We were assigned group 16, composed of the students Javier Campos and Javier Madrid, with NIAs 100472233@alumnos.uc3m.es and 100472291@alumnos.uc3m.es.
The document was delivered on the 17th of March, the scheduled date.
The first thing I noticed was that there is no test for the correctness of OrderType, for the value "REGULAR". They are also missing, in our opinion, Boundary Values for the number of white spaces in the address. Test number 27 does not make any sense.

We took different approaches to the testing of the phone number. The other group only accepted Spanish phone numbers, meaning the number must have 9 digits and the string must be composed entirely of digits. We, on the contrary, choose to accept international phone numbers from Spain, thus only accepting the +34 prefix to the phone number.

An aspect we had a similar decision on was the type of the arguments. In both cases, we all chose to have them as strings, including the phone number and the zip code. Regarding the zip code tests, we had the same checkers, regarding not only the magnitude, but also the length, as a code 1001 is not valid, but a 01001 is.

Regarding the output tests, they are missing a test that includes the scenario where the data written to the file is not correct.

As an overall observation, despite some changes, our excels are pretty similar even though we have a bit more. We also have to mention that since we had problems naming the tests, their naming for the tests was interesting, adding a number at the end when testing for boundary values.

Function 2: Sending a Product

Introduction:

In this second function, we were asked to get some information from a JSON file, and with the information from it, generate a new entry on the order_shipping.json, that represents the order that has been shipped. In this input file, we were given the order_id and an email. In the order_shipping.json entry, the elements present are: "product_id", "order_id"," delivery_email", "issued_at", "delivery_day", and "tracking_code".

Code Analysis:

As a first measure, like in the first function, we checked everything has the correct format, starting by loading the input JSON file and searching for the

To validate this function we had to open several JSON files and validate their content, so to implement this, the first thing we do is try to open the input file and check if its content is correct according to our regular expression of how the order should be. In case the content does not match the regular expression, it automatically raises the error of "`JSON file has not the expected structure`", otherwise we will get the ordered by its pattern in the regular expression. Then we try to open the `order_request.json` file and look for each request if there is one that matches the one we are looking for, in case it doesn't, we raise the "`Data has no valid values`" error. Then, we create a dictionary that checks if the dictionary is created correctly, hash it using the md5 algorithm, and, if it differs from the one we already had, raise an error. We are checking that the data has not been modified. Then we get the email from the dictionary, check its validity with the regular expression (although if we have it in this variable, it should exist, we don't know for sure it follows the regular expression), and create our order shipping object with its corresponding tracking code. Write it in the order shipping.json file and return it.

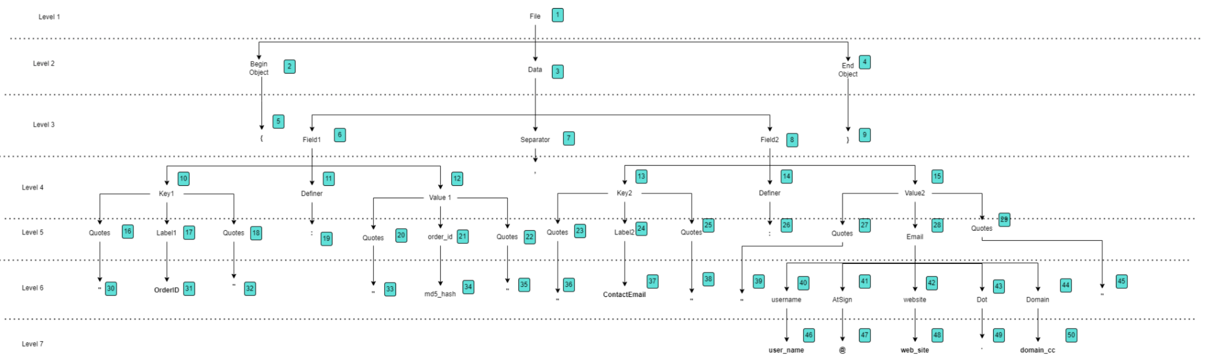
Testing:

For testing this function, (as requested) we made a Derivation tree (see below) with 50 nodes to check the structure of an order request object. Since for almost every node, 2 tests were needed, we ended up with 80 test cases in different JSONs and decided to implement them in different python files for each possible test case of duplicated, deleted, or modified node. For the valid test cases, we create an instance of order manager and call the function, then create our own instance and compare if it's the same, but for the invalid, we assert if the error is the same as the one we expect the function to give.
To check the different test files, we understood we had to create one test file for each JSON, which in hindsight, was a gross error. To comply with the instructions of the professor, we had to merge all the files into one big file, which comprehends all the tests, but the tests got disordered and due to the lack of time provided for this project we deemed it as "lesser devil" but in case of wanting the test ordered, we can upload the folder with the 50 test files anytime.

Grammar and Derivation Tree:

Grammar:

File ::= Begin_Object Data End_Object
Begin_Object ::= {
End_Object ::= }
Data ::= Field1 Separator Field2
Field1 ::= Key1 Definer Value1
Key1 ::= Quotes Laberl1 Quotes
Quotes ::= “
Laberl1 ::= OrderID
Definer ::= :
Value1 ::= Quotes order_id Quotes
order_id ::= md5_hash - r’[a-f0-9]{32}’
Separator ::= ,
Field2 ::= Key2 Definer Value2
Key2 ::= Quotes Label2 Quotes
Label2 ::=ContactEmail
Value2 ::= Quotes Email Quotes
Email ::= username AtSign website Dot Domain
username ::= user_name == - r’[A-z0-9.-]+’
AtSign ::= @
website ::= web_site == r’[A-z0-9]+(\.?[A-z0-9]+)*’
Dot ::= .
Domain ::= domain_cc == r’\.[a-zA-Z]{1,3}’



https://drive.google.com/file/d/1CN4In-RAlortCVSNrGjV2kmACEEV5sCL/view?usp=sharing

Difficulties:

The huge amount of files that were requested to make was one of the main problems but it was more bothersome than difficult since we changed the derivation tree while making the tests but by the end, we managed to automate it. The other main problem was the time, this function is “easier” than the first one since we now have some experience dealing with files but it is way longer and having only one week to implement it (without mentioning that function 3 is delivered 2 days after the explanation) is not pleasant.

To comply with the requirement of the exercise to solve all pylint-related errors, we had to modify the function. Specifically, we had to create the subfunctions, such as `process_data`, `checker_checker`, `find_email`, and `saving_to_file`. These actions were put into production because the pylint checker gave an error regarding having too many branches and too many variables inside a function. To solve this problem, we also had to create some variables of the type lists, to reduce the number of variables used, such as testers, a list of 3 elements, that checks if elements exist.

Function 3:

Introduction:

We developed the function by dividing it into 3 parts: validate_tracking_code, tracking_code_searcher, and deliver_product, aside from one extra part hash_checker, included for checking if the data received through the tracking code was modified or not:

- Validate_tracking code: checks the tracking code has sha256 format (used in tracking_code_searcher)
- tracking code_searcher: Looks in order_shipping for a tracking code that’s equal to the one provided and returns the object where the tracking code is.
- hash_checker: Receives the tracking code and the object, opens the order_request_json and looks for the one with the order_id of the object, checks if the order is regular or premium, and subtracts the corresponding days from the delivery date of the object. Create the tracking code of a new object with the same data (at the time it was supposed to be created) and check if the hash is equal to the one provided. Returns true if everything works smoothly.
- delivery_product: The main body, receives the tracking code and calls the tracking_code_searcher to get the object, then calls the hash_checker function to verify if data was modified, supposing everything works correctly, the current date and the timestamp are written inside a dictionary in order_delivery.json

Code Analysis:

To achieve this functionality, we decided to implement the functionality using 4 different functions:

- Validate_tracking_cide: Uses a regular expression to check the tracking code is an SHA-256 code.
- Tracking code searcher: Tries to validate the tracking code, the previous function, and then opens the order_shipping.json. While iterating through the JSON, it looks for the correct tracking code and returns that entry of the database

- Hash_checker: This function receives the tracking code, and the object from where the tracking code is supposedly generated, and looks for the required information, the order_request.json (to get the type of request and calculate the timestamp), then it freezes the time and with all the information necessary to recompute the hash. Finally, it checks if the tracking code just generated is the same as the one given as an input.

- Deliver product: Lastly, we have the function that integrates all of them, calling to tracking_code_searcher and to hash_checker. Then, if the delivery_day is the same as the actual time, we store in the order_delivery.json file the tracking code and when it has been delivered, the actual time.

Testing:

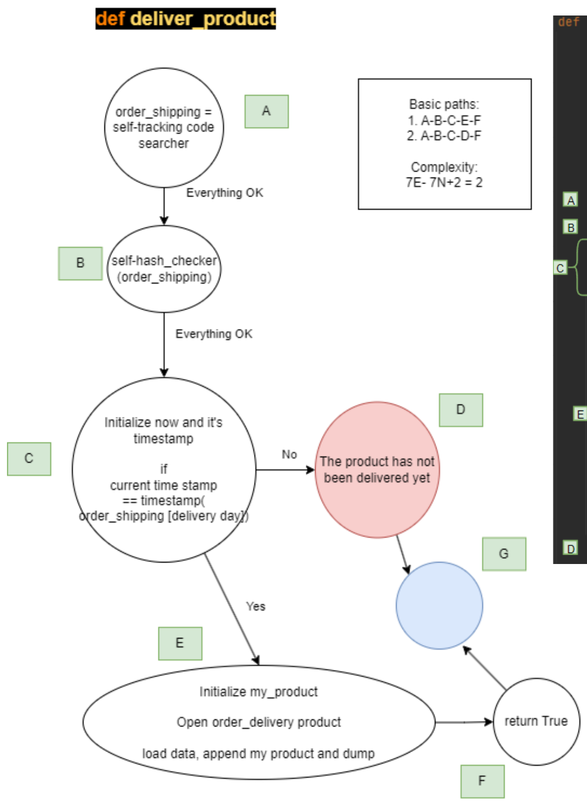
Testing in this function was implemented through the structural testing method, which creates a graph and possible paths to follow. Since if everything were in a single function the graph would be huge, we decided to separate it into 4 more informative graphs, the content of the nodes is a simplification of the code, to avoid misunderstandings using ambiguous grammar like A or B which were also implemented but only for the possible paths. The rules used were:

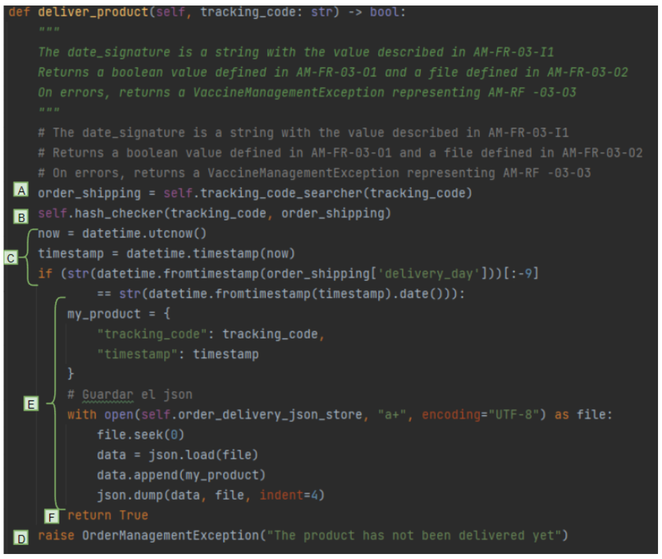
- The first path is the most significant/usual one when applying the code
- The nodes shall only deviate once and stick as much as possible to the first path (in case of a loop where the tracking code or the order id is not found, we can't follow this rule

We do know that some nodes could be a bit more simplified but even though they could have been smaller, we deemed it easier to understand this way. We have also to mention that on the delivery product, an opening of the delivery.json file is done, but we don't consider any errors from it, that's because since we want to write if the file doesn't exist, it will simply make one. The most important change is that the hidden methods of self.__order_request/shipping/delivery.json were unhidden to test the possible path in the graph where the path to the file is not correct and gives a file not found error, for the correct usage of the functions, those variables should be refactored and hide them again.

Diagrams:

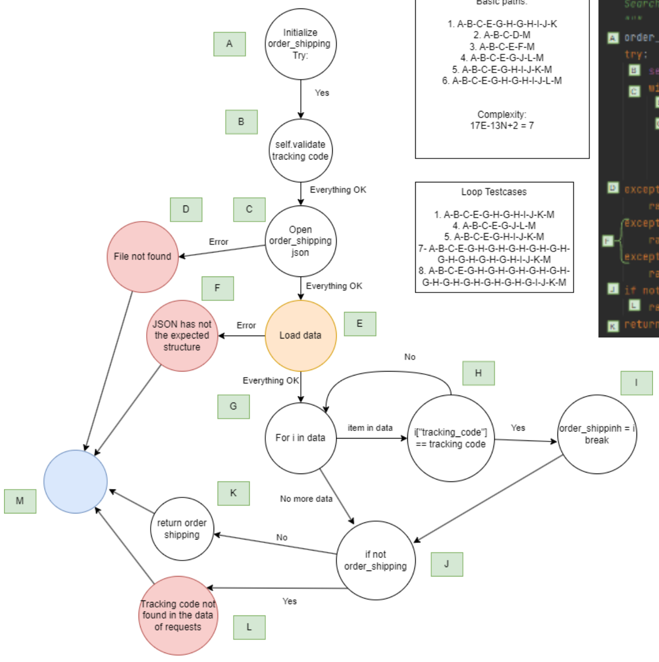
The order is first the main function, and then the order in which the auxiliary functions are called


def deliver_product
order_shipping = self-tracking code searcher
A
Everything OK
B
self-hash_checker (order_shipping)
Everything OK
C
Initialize now and it's timestamp
if
current time stamp == timestamp(order_shipping [delivery day])
No
The product has not been delivered yet
D
G
Yes
E
Initialize my_product
Open order_delivery product
load data, append my product and dump
return True
F
Basic paths:
1. A-B-C-E-F
2. A-B-C-D-F
Complexity:
7E- 7N+2 = 2



```
def deliver_product(self, tracking_code: str) -> bool:
    """
    The date_signature is a string with the value described in AM-FR-03-I1
    Returns a boolean value defined in AM-FR-03-01 and a file defined in AM-FR-03-02
    On errors, returns a VaccineManagementException representing AM-RF -03-03
    """
    # The date_signature is a string with the value described in AM-FR-03-I1
    # Returns a boolean value defined in AM-FR-03-01 and a file defined in AM-FR-03-02
    # On errors, returns a VaccineManagementException representing AM-RF -03-03
A   order_shipping = self.tracking_code_searcher(tracking_code)
B   self.hash_checker(tracking_code, order_shipping)
    now = datetime.utcnow()
C   timestamp = datetime.timestamp(now)
    if (str(datetime.fromtimestamp(order_shipping['delivery_day']))[:-9]
            == str(datetime.fromtimestamp(timestamp).date())):
        my_product = {
            "tracking_code": tracking_code,
            "timestamp": timestamp
        }
E       # Guardar el json
        with open(self.order_delivery_json_store, "a+", encoding="UTF-8") as file:
            file.seek(0)
            data = json.load(file)
            data.append(my_product)
            json.dump(data, file, indent=4)
F       return True
D   raise OrderManagementException("The product has not been delivered yet")
```

def tracking_code_searcher



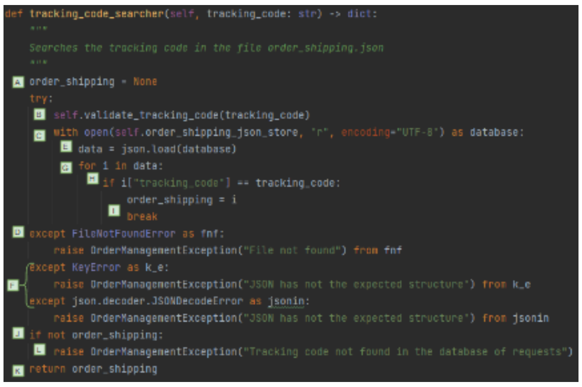
Basic paths:

1. A-B-C-E-G-H-G-H-I-J-K
2. A-B-C-D-M
3. A-B-C-E-F-M
4. A-B-C-E-G-J-L-M
5. A-B-C-E-G-H-I-J-K-M
6. A-B-C-E-G-H-G-H-I-J-L-M

Complexity:
17E-13N+2 = 7

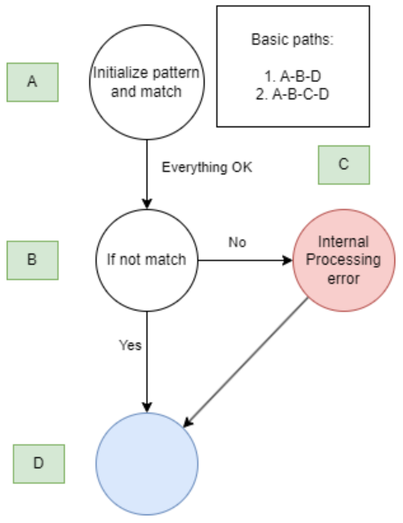
Loop Testcases

1. A-B-C-E-G-H-G-H-I-J-K-M
4. A-B-C-E-G-J-L-M
5. A-B-C-E-G-H-I-J-K-M
7- A-B-C-E-G-H-G-H-G-H-G-H-G-H-G-H-G-H-G-H-G-H-I-J-K-M
8. A-B-C-E-G-H-G-H-G-H-G-H-G-H-G-H-G-H-G-H-G-H-G-H-G-I-J-K-M



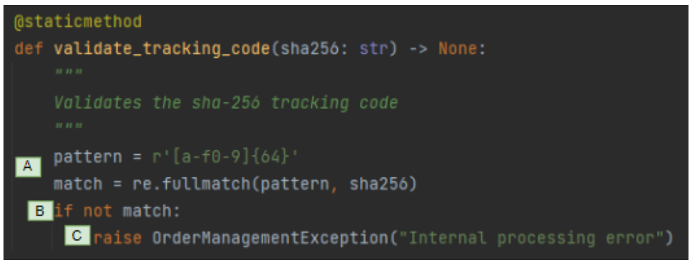
```
def tracking_code_searcher(self, tracking_code: str) -> dict:
    """
    Searches the tracking code in the file order_shipping.json
    """
A   order_shipping = None
    try:
B       self.validate_tracking_code(tracking_code)
C       with open(self.order_shipping_json_store, "r", encoding="UTF-8") as database:
E           data = json.load(database)
G           for i in data:
H               if i["tracking_code"] == tracking_code:
                    order_shipping = i
I                   break
D   except FileNotFoundError as fnf:
        raise OrderManagementException("File not found") from fnf
    except KeyError as k_e:
F       raise OrderManagementException("JSON has not the expected structure") from k_e
    except json.decoder.JSONDecodeError as jsonin:
        raise OrderManagementException("JSON has not the expected structure") from jsonin
J   if not order_shipping:
L       raise OrderManagementException("Tracking code not found in the database of requests")
K   return order_shipping
```

def validate_tracking_code

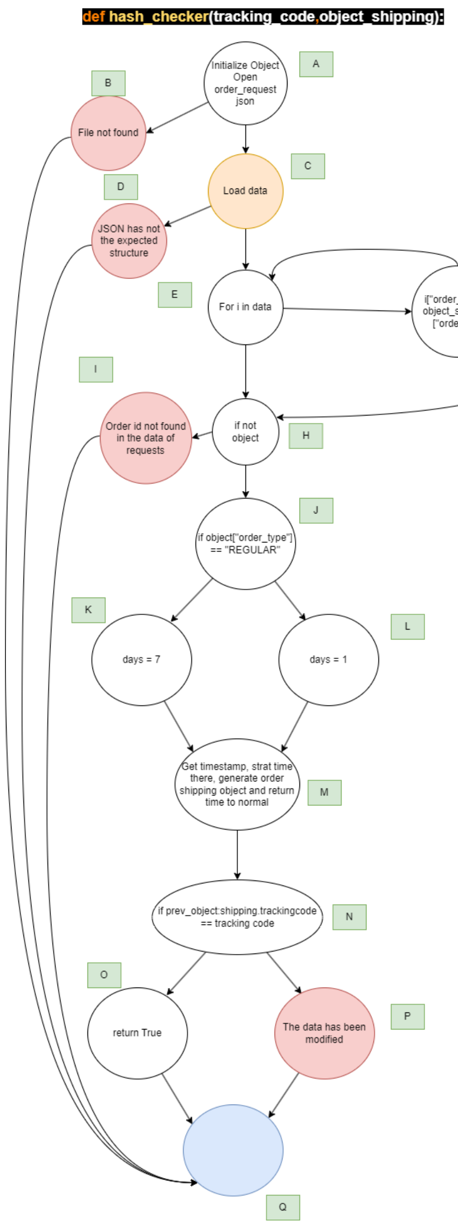


Basic paths:

1. A-B-D
2. A-B-C-D



```
@staticmethod
def validate_tracking_code(sha256: str) -> None:
    """
    Validates the sha-256 tracking code
    """
A   pattern = r'[a-f0-9]{64}'
    match = re.fullmatch(pattern, sha256)
B   if not match:
C       raise OrderManagementException("Internal processing error")
```

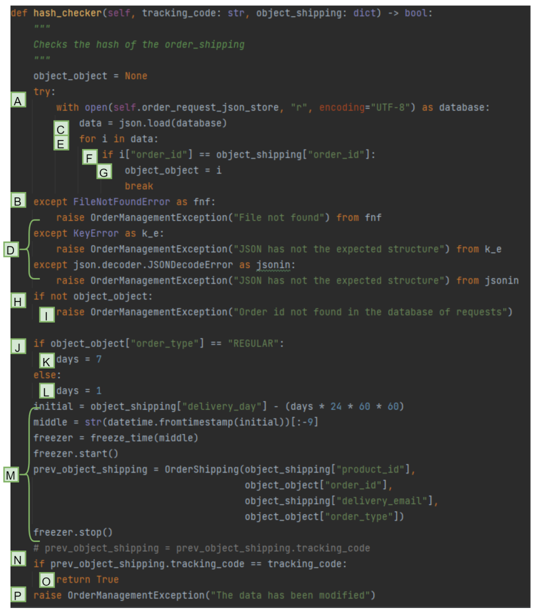

def hash_checker(tracking_code,object_shipping):
Initialize Object Open order_request json
A
B
File not found
Load data
C
D
JSON has not the expected structure
E
For i in data
F
i["order_id"] == object_shipping ["order_id"]
G
object= i break
I
Order id not found in the data of requests
if not object
H
J
if object["order_type"] == "REGULAR"
K
days = 7
L
days = 1
Get timestamp, strat time there, generate order shipping object and return time to normal
M
if prev_object.shipping.trackingcode == tracking code
N
O
return True
The data has been modified
P
Q

Basic paths:
1. A-C-E-F-E-F-G-H-J-L-M-N-O-Q
2. A-B-Q
3. A-C-D-Q
4. A-C-E-H-I-Q
5. A-C-E-F-G-H-J-K-M-N-O-Q
6. A-C-E-F-G-H-I-Q
7.A-C-E-F-E-F-G-H-J-K-M-N-O-Q
8. A-C-E-F-E-F-G-H-J-L-M-N-P-Q

Complexity:
23E-17N+2 = 8

Loop Testcases

1. A-C-E-F-E-F-G-H-J-L-M-N-O-Q
4. A-C-E-H-I-Q
5. A-C-E-F-G-H-J-L-M-N-O-Q
9. A-C-E-F-E-F-E-F-E-F-E-F-E-F-E-F-E-F-E-F-G-H-J-L-M-N-O-Q
10. A-C-E-F-E-F-E-F-E-F-E-F-E-F-E-F-E-F-E-F-E-F-G-H-J-L-M-N-O-Q



```
def hash_checker(self, tracking_code: str, object_shipping: dict) -> bool:
    """
    Checks the hash of the order_shipping
    """
    object_object = None
A   try:
        with open(self.order_request_json_store, "r", encoding="UTF-8") as database:
C           data = json.load(database)
E           for i in data:
F               if i["order_id"] == object_shipping["order_id"]:
G                   object_object = i
                    break
B   except FileNotFoundError as fnf:
        raise OrderManagementException("File not found") from fnf
    except KeyError as k_e:
D       raise OrderManagementException("JSON has not the expected structure") from k_e
    except json.decoder.JSONDecodeError as jsonin:
        raise OrderManagementException("JSON has not the expected structure") from jsonin
H   if not object_object:
I       raise OrderManagementException("Order id not found in the database of requests")

J   if object_object["order_type"] == "REGULAR":
K       days = 7
    else:
L       days = 1
    initial = object_shipping["delivery_day"] - (days * 24 * 60 * 60)
    middle = str(datetime.fromtimestamp(initial))[:-9]
    freezer = freeze_time(middle)
    freezer.start()
M   prev_object_shipping = OrderShipping(object_shipping["product_id"],
                                         object_object["order_id"],
                                         object_shipping["delivery_email"],
                                         object_object["order_type"])
    freezer.stop()
    # prev_object_shipping = prev_object_shipping.tracking_code
N   if prev_object_shipping.tracking_code == tracking_code:
O       return True
P   raise OrderManagementException("The data has been modified")
```

Difficulties:

This function was the “easiest” given the knowledge learned in the previous functions, the only remarkable thing was the time provided to finish it and its length, which is the reason why it was divided into different significant parts. Among which we have to highlight the hash_checker, which, since we didn’t know how to re-create the hash with the correct timestamp, we created our own approach which we understand is not the most efficient one despite making use of the given function in class OrderShipping to create the hash.