6.2 Ejercicio de programación 3 y pruebas de unidad

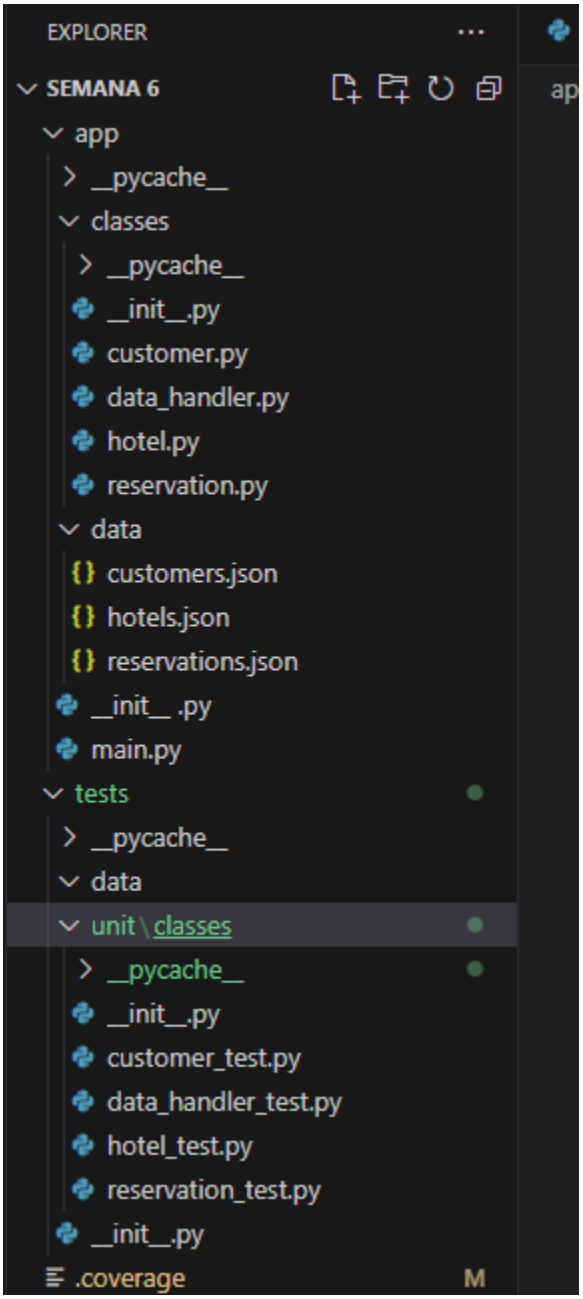
Pruebas de software y aseguramiento de la calidad

Andre M. Hernández Bornn

A01795190

12 febrero 2025

Estructura de aplicación



Clase Hotel

hotel.py | main.py M | customer.py | reservation.py | data_handler.py | ho

app > classes > hotel.py > Hotel > make_reservation

```python
'''Archivo para crear clase de Hotel'''


import os
from app.classes.reservation import Reservation
from app.classes.data_handler import DataHandler


class Hotel():
    '''Clase para manejar hoteles'''

    def __init__(self, data_handler):
        base_path = os.path.dirname(os.path.abspath(__file__))
        file_path = os.path.join(base_path, "..", "data", "reservations.json")
        self.data_handler = data_handler
        reservation_data_handler = DataHandler(file_path)
        self.reservation_instance = Reservation(reservation_data_handler)

    def create_hotel(self, name, country, state, address):
        '''Metodo para crear hotel'''
        try:
            hotel = {
                "name": name,
                "country": country,
                "state": state,
                "address": address,
            }
            if any(
                self.data_handler.is_missing(hotel[key])
                for key in ["name", "country", "state", "address"]
            ):
                raise ValueError("Missing data")
            new_hotel_list = self.data_handler.create_registry(hotel)
            return new_hotel_list
        except ValueError:
            return "Missing data"

    def delete_hotel(self, hotel_id):
        '''Metodo para borrar hotel'''
        try:
            if self.data_handler.is_missing(hotel_id):
                raise ValueError("Missing data")
            all_hotels = self.data_handler.delete_from_file(hotel_id)
            return all_hotels
        except ValueError:
            return "Missing data"

    def get_hotel_info(self, hotel_id):
        '''Metodo para obtener los datos de un hotel'''
        try:
            if self.data_handler.is_missing(hotel_id):
                raise ValueError("Missing data")
            hotel_info = self.data_handler.get_data_by_id(hotel_id)
            return hotel_info
        except ValueError:
            return "Missing data"
```


hotel.py
main.py M
customer.py
reservation.py
data_handler.py
hotel_test.py
app > classes > hotel.py > Hotel > modify_hotel
class Hotel():
def delete_hotel(self, hotel_id):
return "Missing data"
def get_hotel_info(self, hotel_id):
'''Metodo para obtener los datos de un hotel'''
try:
if self.data_handler.is_missing(hotel_id):
raise ValueError("Missing data")
hotel_info = self.data_handler.get_data_by_id(hotel_id)
return hotel_info
except ValueError:
return "Missing data"
def modify_hotel(self, hotel):
'''Metodo para modificar un hotel'''
try:
if any(
self.data_handler.is_missing(hotel[key])
for key in ["id", "name", "country", "state", "address"]
):
raise ValueError("Missing data")
all_hotels = self.data_handler.modify_file(hotel)
return all_hotels
except ValueError:
return "Missing data"
def make_reservation(
self, customer_id, hotel_id, check_in_date, check_out_date
):
'''Metodo para crear
una reservación'''
try:
reservation = {
"hotel_id": hotel_id,
"customer_id": customer_id,
"check_in_date": check_in_date,
"check_out_date": check_out_date
}
reservation_list = self.reservation_instance.create_reservation(
reservation, self
)
if reservation_list == (
"Hotel or Customer with specified ID does not exist"
):
raise ValueError
return reservation_list
except ValueError:
return "Failed creating reservation"
def cancel_reservation(self, reservation_id):
'''Metodo para cancelar una reservación'''
reservation_list = self.reservation_instance.cancel_reservation(
reservation_id
)
return reservation_list

Clase Customer

Clase reservation


hotel.py
main.py M
customer.py
reservation.py
data_handler.py
app > classes > reservation.py > Reservation > get_reservation_by_id
class Reservation():
def create_reservation(
)
if (
customer == "No data found"
or hotel == "No data found"
):
return "Hotel or Customer with specified ID does not exist"
reservation = {
"hotelID": reservation["hotel_id"],
"CustomerID": reservation["customer_id"],
"checkInDate": reservation["check_in_date"],
"checkOutDate": reservation["check_out_date"],
"Status": "confirmed"
}
new_reservation_list = self.data_handler.create_registry(
reservation
)
return new_reservation_list
except ValueError:
return "Missing data"
def cancel_reservation(self, reservation_id):
'''Metodo para canclar reservación'''
reservation_to_modify = self.get_reservation_by_id(reservation_id)
if reservation_to_modify == "No data found":
return "Id does not exist"
reservation_to_modify["Status"] = "canceled"
modified_reservation_list = self.data_handler.modify_file(
reservation_to_modify
)
return modified_reservation_list
def get_reservation_by_id(self, reservation_id):
'''Metodo para obtener info de reservación'''
try:
if self.data_handler.is_missing(reservation_id):
raise ValueError("Missing data")
reservation_info = self.data_handler.get_data_by_id(reservation_id)
return reservation_info
except ValueError:
return "Missing data"

Clase datahandler


```python
'''Archivo para crear clase de DataHandler'''

import json


class DataHandler():
    '''Clase para manejar datos en los archivos'''

    def __init__(self, file):
        self.file = file

    def create_registry(self, data):
        '''Metodo para crear un nuevo registro en el archivo'''
        file_json = self.file_as_json()
        my_id = self.create_id(file_json)
        data["id"] = my_id
        file_json.append(data)
        file_json = self.write_to_file(file_json)
        return file_json

    def modify_file(self, data):
        '''Metodo para modificar un registro en el archivo'''
        my_id = data["id"]
        file_json = self.file_as_json()
        index = self.find_index_of_array(file_json, my_id)
        if index < 0:
            return file_json
        file_json[index] = data
        file_json = self.write_to_file(file_json)
        return file_json

    def get_data_by_id(self, my_id):
        '''Metodo para obtener un registro del archivo'''
        file_json = self.file_as_json()
        index = self.find_index_of_array(file_json, my_id)
        if index < 0:
            return "No data found"
        data = file_json[index]
        return data

    def delete_from_file(self, my_id):
        '''Metodo para borrar un registro del archivo'''
        file_json = self.file_as_json()
        json_file = self.delete_from_json(my_id, file_json)
        self.write_to_file(json_file)
        return json_file

    def file_as_json(self):
        '''Metodo para leer al archivo y convertirlo a json'''
        try:
            with open(self.file, encoding="utf-8") as json_file:
                json_data = json.load(json_file)
                return json_data
        except FileNotFoundError:
            print("File does not exists")
            return "File Does not exists"
        except json.decoder.JSONDecodeError as e:
            print("Empty Json, give propper format to file")
            raise e

    def filter_json_file(self, my_id, json_file):
        '''Metodo para filtrar un json
        y obtener los registros con un id especifico'''
        filtered_data = [x for x in json_file if x['id'] == my_id]
        return filtered_data
```

hotel.py | main.py M | customer.py | reservation.py | data_handler.py | hotel_test.py

app > classes > data_handler.py > DataHandler > create_registry

```python
class DataHandler():
    def delete_from_file(self, my_id):
        return json_file

    def file_as_json(self):
        '''Metodo para leer al archivo y convertirlo a json'''
        try:
            with open(self.file, encoding="utf-8") as json_file:
                json_data = json.load(json_file)
                return json_data
        except FileNotFoundError:
            print("File does not exists")
            return "File Does not exists"
        except json.decoder.JSONDecodeError as e:
            print("Empty Json, give propper format to file")
            raise e

    def filter_json_file(self, my_id, json_file):
        '''Metodo para filtrar un json
        y obtener los registros con un id especifico'''
        filtered_data = [x for x in json_file if x['id'] == my_id]
        return filtered_data

    def delete_from_json(self, my_id, json_file):
        '''Metodo para borrar un elmento especifico del json'''
        index = self.find_index_of_array(json_file, my_id)
        del json_file[index]
        return json_file

    def find_index_of_array(self, json_file, my_id):
        '''Metodo para obtener el indice del
        registro con el id proporcionado'''
        for i, dic in enumerate(json_file):
            if dic["id"] == my_id:
                return i
        return -1

    def write_to_file(self, original_json_file):
        '''Metodo para escribir en el archivo'''
        try:
            with open(self.file, "w", encoding="utf-8") as json_file:
                json.dump(original_json_file, json_file)
            with open(self.file, encoding="utf-8") as json_file:
                return json.load(json_file)
        except TypeError:
            print("Error writing to file")
            return "Error writing to file"

    def create_id(self, json_file):
        '''Metodo para crear un ID'''
        my_id = 1
        if len(json_file) < 1:
            return my_id
        my_id = json_file[len(json_file) - 1]["id"] + 1
        return my_id

    def is_missing(self, value):
        '''Metodo para validar si un dato viene vacio'''
        return value is None or value == ""
```

Código de control

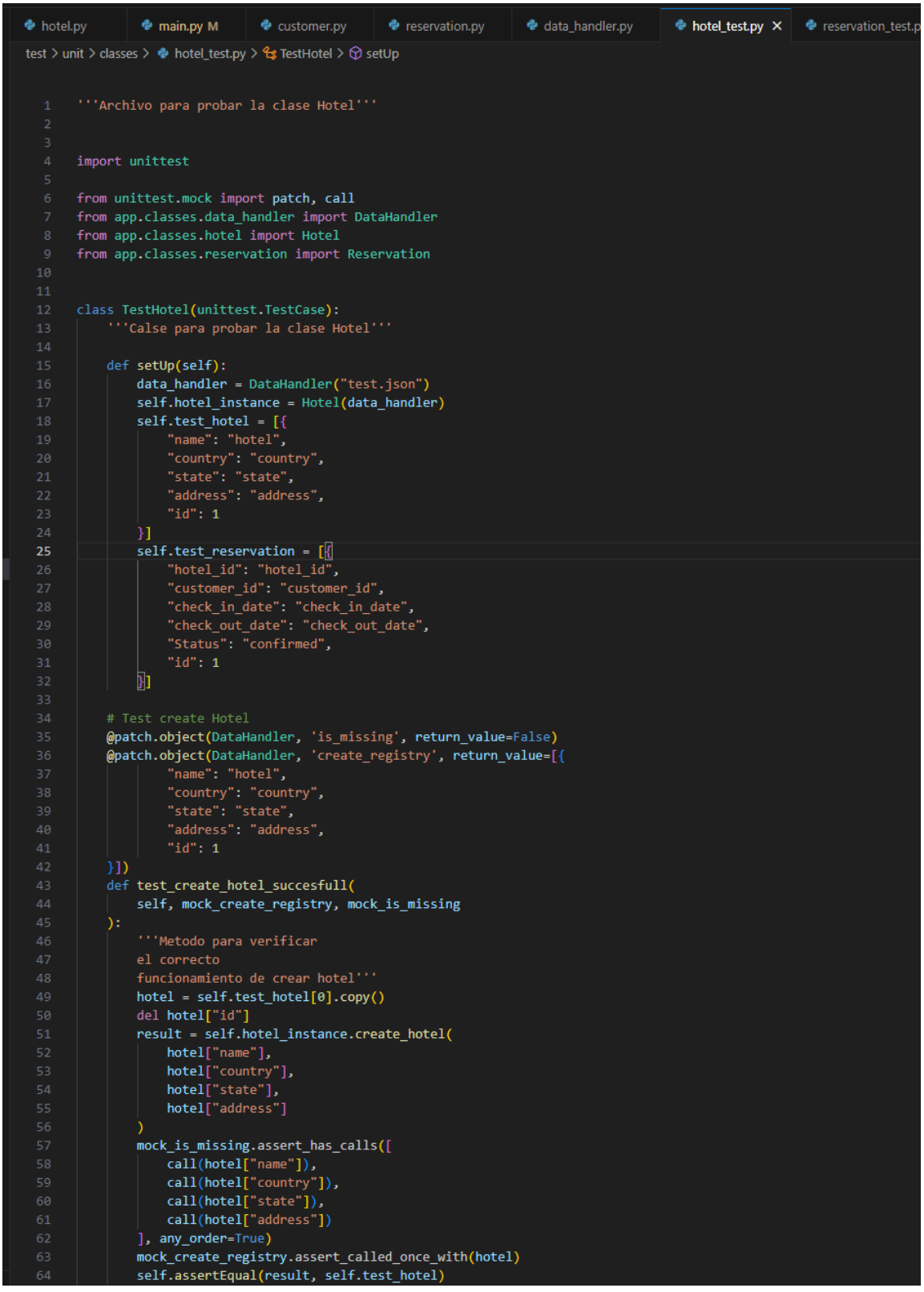
Terminal Help

hotel.py | main.py M | customer.py | reservation.py | data_handler.py

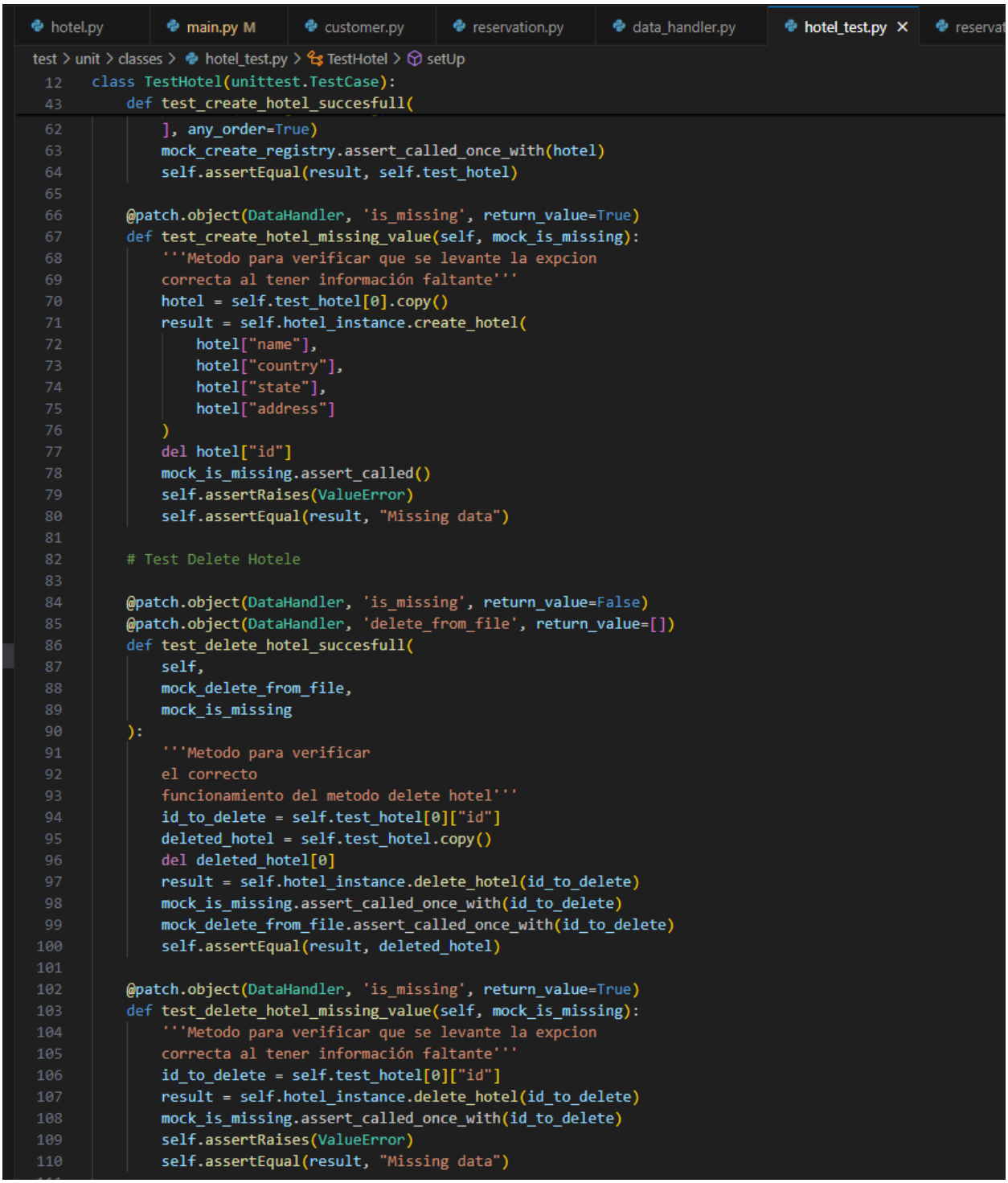
app > main.py > ...

```python
'''Functionality Test'''
import os

from app.classes.hotel import Hotel
from app.classes.data_handler import DataHandler
from app.classes.customer import Customer


BASE_DIR = os.path.dirname(os.path.abspath(__file__))
HOTEL_FILE_PATH = os.path.join(BASE_DIR, "data", "hotels.json")
CUSTOMER_FILE_PATH = os.path.join(BASE_DIR, "data", "customers.json")

hotelDataHandler = DataHandler(HOTEL_FILE_PATH)
hotelInstance = Hotel(hotelDataHandler)
customerDataHandler = DataHandler(CUSTOMER_FILE_PATH)
customerInstance = Customer(customerDataHandler)


createdHotel1 = hotelInstance.create_hotel(
    "name", "country", "state", "address"
    )
createdHotel2 = hotelInstance.create_hotel(
    "name2", "country2", "state2", "address2"
    )
hotelInstance.delete_hotel(2)
hotelToModify = createdHotel2[0]
hotelToModify["name"] = "modifiedName"
modifiedHotels = hotelInstance.modify_hotel(hotelToModify)
hotelInfo1 = hotelInstance.get_hotel_info(1)
print(hotelInfo1)
hotelInfo2 = hotelInstance.get_hotel_info(4)
print(hotelInfo2)


createdCustomer1 = customerInstance.create_customer("name", "lastName")
createdCustomer2 = customerInstance.create_customer("name2", "lastName2")
customerList = customerInstance.delete_customer(2)
print(createdCustomer2)
customerToModify = customerList[0]
customerToModify["name"] = "modifiedName"
modifiedCustomers = customerInstance.modify_customer(customerToModify)
customerIfo1 = customerInstance.get_customer_data(1)
print(customerIfo1)
customerIfo2 = customerInstance.get_customer_data(5)
print(customerIfo2)

reservation = hotelInstance.make_reservation(
    customerIfo1["id"], hotelInfo1["id"], "23/12/2025", "28/12/2025"
    )
reservation = hotelInstance.make_reservation(
    5, hotelInfo1["id"], "23/12/2025", "28/12/2025"
    )
print(reservation)

reservations = hotelInstance.cancel_reservation(1)
print(reservations)
reservations = hotelInstance.cancel_reservation(4)
print(reservations)
```

Pruebas unitarias clase hotel




```python
'''Archivo para probar la clase Hotel'''


import unittest

from unittest.mock import patch, call
from app.classes.data_handler import DataHandler
from app.classes.hotel import Hotel
from app.classes.reservation import Reservation


class TestHotel(unittest.TestCase):
    '''Calse para probar la clase Hotel'''

    def setUp(self):
        data_handler = DataHandler("test.json")
        self.hotel_instance = Hotel(data_handler)
        self.test_hotel = [{
            "name": "hotel",
            "country": "country",
            "state": "state",
            "address": "address",
            "id": 1
        }]
        self.test_reservation = [{
            "hotel_id": "hotel_id",
            "customer_id": "customer_id",
            "check_in_date": "check_in_date",
            "check_out_date": "check_out_date",
            "Status": "confirmed",
            "id": 1
        }]

    # Test create Hotel
    @patch.object(DataHandler, 'is_missing', return_value=False)
    @patch.object(DataHandler, 'create_registry', return_value=[{
            "name": "hotel",
            "country": "country",
            "state": "state",
            "address": "address",
            "id": 1
    }])
    def test_create_hotel_succesfull(
        self, mock_create_registry, mock_is_missing
    ):
        '''Metodo para verificar
        el correcto
        funcionamiento de crear hotel'''
        hotel = self.test_hotel[0].copy()
        del hotel["id"]
        result = self.hotel_instance.create_hotel(
            hotel["name"],
            hotel["country"],
            hotel["state"],
            hotel["address"]
        )
        mock_is_missing.assert_has_calls([
            call(hotel["name"]),
            call(hotel["country"]),
            call(hotel["state"]),
            call(hotel["address"])
        ], any_order=True)
        mock_create_registry.assert_called_once_with(hotel)
        self.assertEqual(result, self.test_hotel)
```

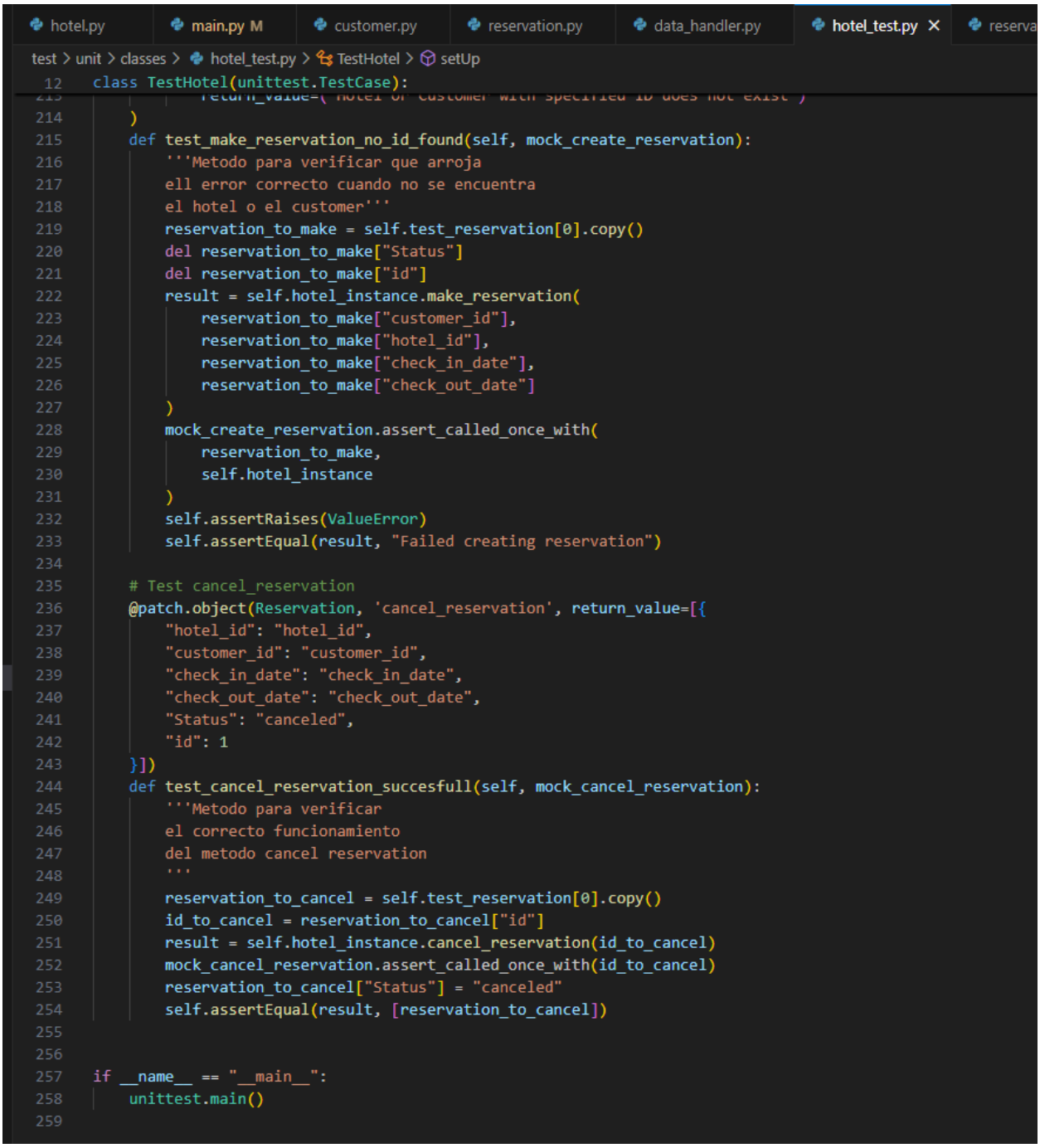

hotel.py
main.py M
customer.py
reservation.py
data_handler.py
hotel_test.py
test > unit > classes > hotel_test.py > TestHotel > setUp
class TestHotel(unittest.TestCase):
def test_create_hotel_succesfull(
], any_order=True)
mock_create_registry.assert_called_once_with(hotel)
self.assertEqual(result, self.test_hotel)
@patch.object(DataHandler, 'is_missing', return_value=True)
def test_create_hotel_missing_value(self, mock_is_missing):
'''Metodo para verificar que se levante la expcion
correcta al tener información faltante'''
hotel = self.test_hotel[0].copy()
result = self.hotel_instance.create_hotel(
hotel["name"],
hotel["country"],
hotel["state"],
hotel["address"]
)
del hotel["id"]
mock_is_missing.assert_called()
self.assertRaises(ValueError)
self.assertEqual(result, "Missing data")
# Test Delete Hotele
@patch.object(DataHandler, 'is_missing', return_value=False)
@patch.object(DataHandler, 'delete_from_file', return_value=[])
def test_delete_hotel_succesfull(
self,
mock_delete_from_file,
mock_is_missing
):
'''Metodo para verificar
el correcto
funcionamiento del metodo delete hotel'''
id_to_delete = self.test_hotel[0]["id"]
deleted_hotel = self.test_hotel.copy()
del deleted_hotel[0]
result = self.hotel_instance.delete_hotel(id_to_delete)
mock_is_missing.assert_called_once_with(id_to_delete)
mock_delete_from_file.assert_called_once_with(id_to_delete)
self.assertEqual(result, deleted_hotel)
@patch.object(DataHandler, 'is_missing', return_value=True)
def test_delete_hotel_missing_value(self, mock_is_missing):
'''Metodo para verificar que se levante la expcion
correcta al tener información faltante'''
id_to_delete = self.test_hotel[0]["id"]
result = self.hotel_instance.delete_hotel(id_to_delete)
mock_is_missing.assert_called_once_with(id_to_delete)
self.assertRaises(ValueError)
self.assertEqual(result, "Missing data")

hotel.py | main.py M | customer.py | reservation.py | data_handler.py | hotel_test

test > unit > classes > hotel_test.py > TestHotel > setUp

```
12  class TestHotel(unittest.TestCase):
111
112     # Test get hotel info
113     @patch.object(DataHandler, 'is_missing', return_value=False)
114     @patch.object(DataHandler, 'get_data_by_id', return_value={
115         "name": "hotel",
116         "country": "country",
117         "state": "state",
118         "address": "address",
119         "id": 1
120     })
121     def test_get_hotel_info_succesfull(
122         self, mock_get_data_by_id, mock_is_missing
123     ):
124         '''Metodo para verificar
125         el correcto
126         funcionamiento del metodo get hotel info'''
127         expected_info = self.test_hotel[0]
128         id_to_get = expected_info["id"]
129         result = self.hotel_instance.get_hotel_info(id_to_get)
130         mock_is_missing.assert_called_once_with(id_to_get)
131         mock_get_data_by_id.assert_called_once_with(id_to_get)
132         self.assertEqual(result, expected_info)
133
134     @patch.object(DataHandler, 'is_missing', return_value=True)
135     def test_get_hotel_info_missing_value(self, mock_is_missing):
136         '''Metodo para verificar que se levante la expcion
137         correcta al tener información faltante'''
138         id_to_get = self.test_hotel[0]["id"]
139         result = self.hotel_instance.get_hotel_info(id_to_get)
140         mock_is_missing.assert_called_once_with(id_to_get)
141         self.assertRaises(ValueError)
142         self.assertEqual(result, "Missing data")
143
144     # Test Modify hotel
145
146     @patch.object(DataHandler, 'is_missing', return_value=False)
147     @patch.object(DataHandler, 'modify_file', return_value=[{
148             "name": "modified Name",
149             "country": "country",
150             "state": "state",
151             "address": "address",
152             "id": 1
153     }])
154     def test_modify_hotel_succesfull(self, mock_modify_file, mock_is_missing):
155         '''Metodo para verificar
156         el correcto
157         funcionamiento del metodo modify hotel'''
158         hotel_to_modify = self.test_hotel[0].copy()
159         hotel_to_modify["name"] = "modified Name"
160         result = self.hotel_instance.modify_hotel(hotel_to_modify)
161         mock_is_missing.assert_has_calls([
162             call(hotel_to_modify["name"]),
163             call(hotel_to_modify["country"]),
164             call(hotel_to_modify["state"]),
165             call(hotel_to_modify["address"]),
166             call(hotel_to_modify["id"])
167         ], any_order=True)
168         mock_modify_file.assert_called_once_with(hotel_to_modify)
169         self.assertEqual(result, [hotel_to_modify])
```

```python
    @patch.object(DataHandler, 'is_missing', return_value=True)
    def test_modify_hotel_missing_value(self, mock_is_missing):
        '''Metodo para verificar que se levante la expcion
        correcta al tener información faltante'''
        hotel_to_modify = self.test_hotel[0].copy()
        hotel_to_modify["name"] = "modified Name"
        result = self.hotel_instance.modify_hotel(hotel_to_modify)
        mock_is_missing.assert_called_once()
        self.assertRaises(ValueError)
        self.assertEqual(result, "Missing data")

    # Test Create reservation
    @patch.object(Reservation, 'create_reservation', return_value=[{
        "hotel_id": "hotel_id",
        "customer_id": "customer_id",
        "check_in_date": "check_in_date",
        "check_out_date": "check_out_date",
        "Status": "confirmed",
        "id": 1
    }])
    def test_make_reservation_succesfull(self, mock_create_reservation):
        '''Metodo para verificar el
        comportamiento correcto
        del metodo make reservation'''
        reservation_to_make = self.test_reservation[0].copy()
        del reservation_to_make["Status"]
        del reservation_to_make["id"]
        result = self.hotel_instance.make_reservation(
            reservation_to_make["customer_id"],
            reservation_to_make["hotel_id"],
            reservation_to_make["check_in_date"],
            reservation_to_make["check_out_date"]
        )
        mock_create_reservation.assert_called_once_with(
            reservation_to_make,
            self.hotel_instance
        )
        self.assertEqual(result, self.test_reservation)

    @patch.object(
            Reservation,
            'create_reservation',
            return_value=("Hotel or Customer with specified ID does not exist")
    )
    def test_make_reservation_no_id_found(self, mock_create_reservation):
        '''Metodo para verificar que arroja
        ell error correcto cuando no se encuentra
        el hotel o el customer'''
        reservation_to_make = self.test_reservation[0].copy()
        del reservation_to_make["Status"]
        del reservation_to_make["id"]
        result = self.hotel_instance.make_reservation(
            reservation_to_make["customer_id"],
            reservation_to_make["hotel_id"],
            reservation_to_make["check_in_date"],
            reservation_to_make["check_out_date"]
        )
        mock_create_reservation.assert_called_once_with(
            reservation_to_make,
            self.hotel_instance
        )
        self.assertRaises(ValueError)
        self.assertEqual(result, "Failed creating reservation")
```

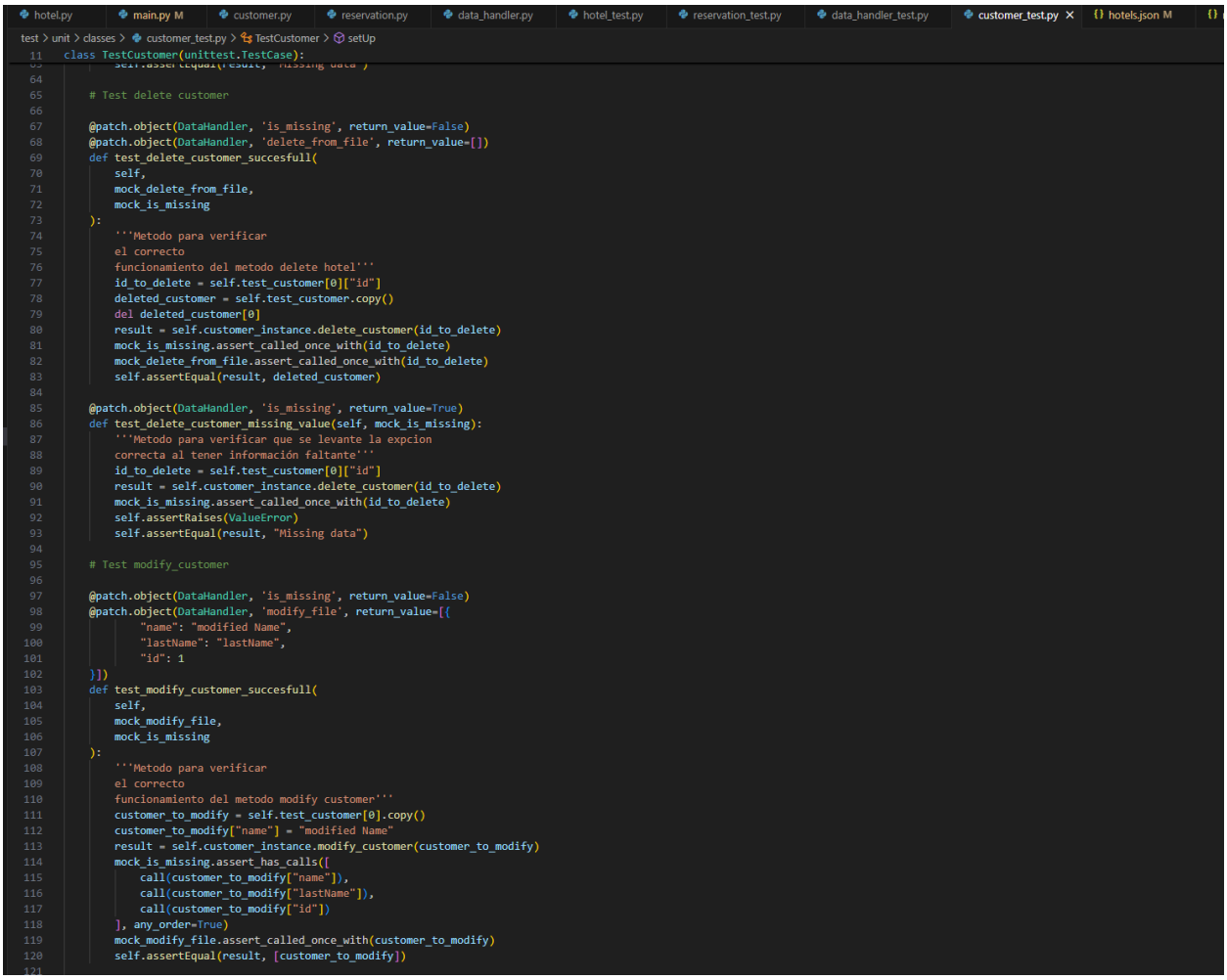


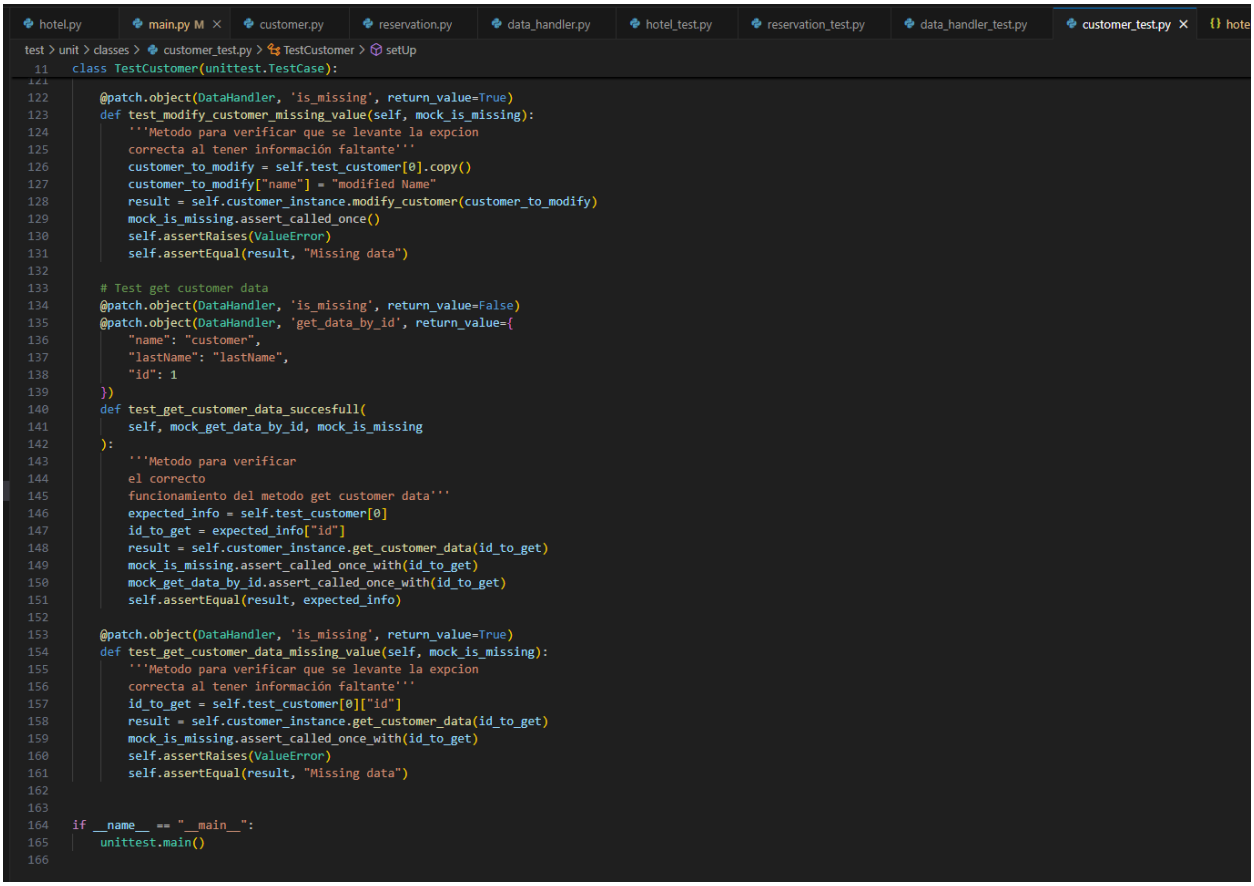
Pruebas Unitarias Clase Customer

hotel.py | main.py M | customer.py | reservation.py | data_handler.py | hotel_test.py | reservation_test.py | data_handler_test.py | customer_test.py | hotels.json M

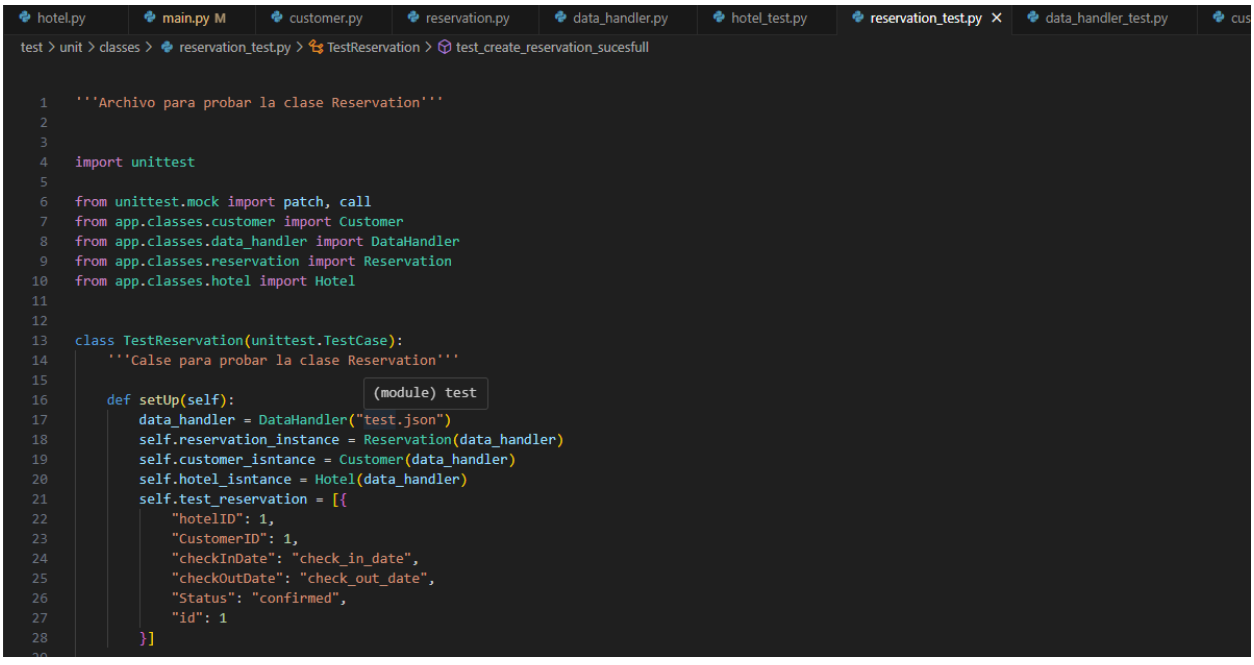
test > unit > classes > customer_test.py > TestCustomer > setUp

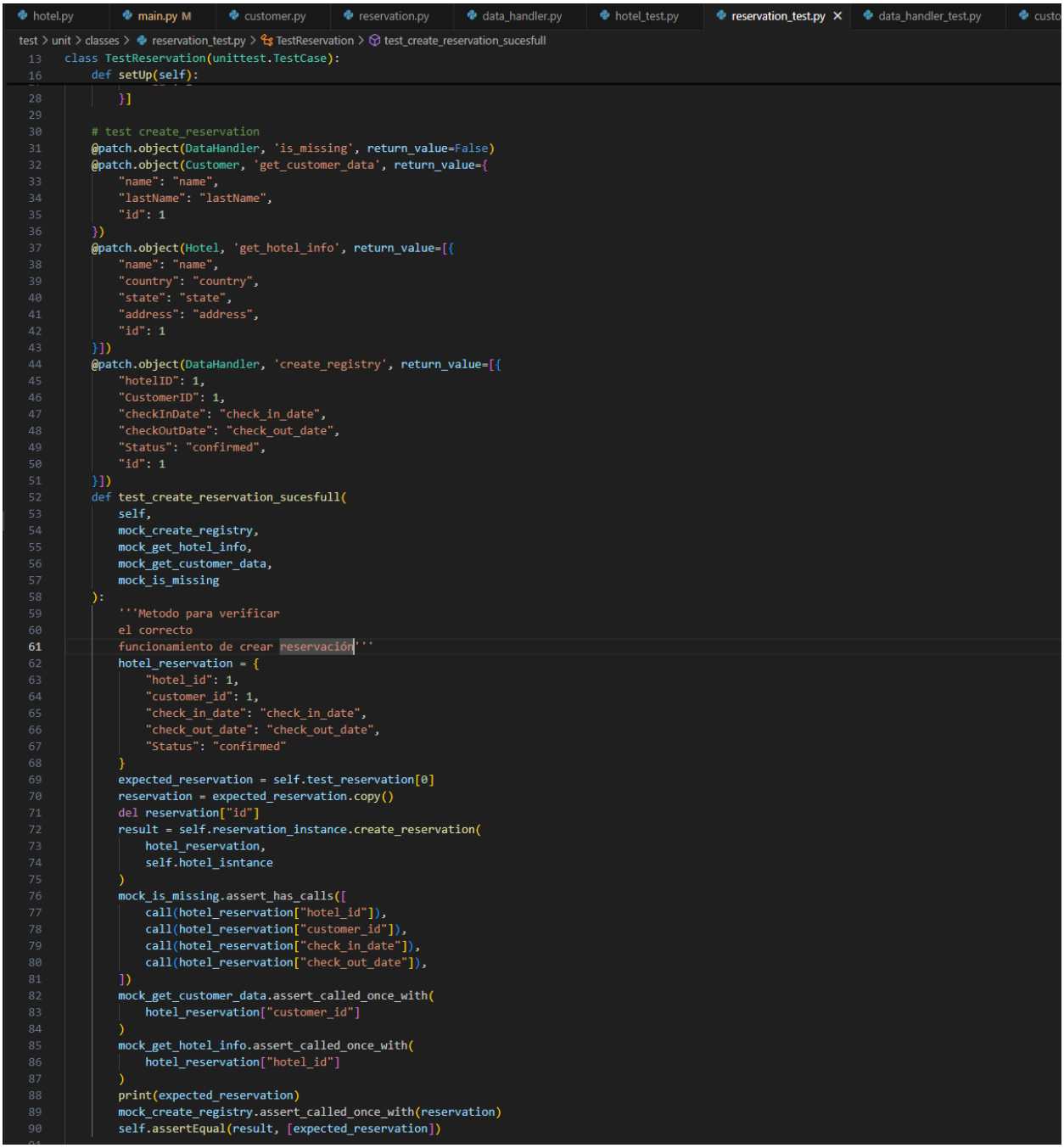
```python
'''Archivo para probar la clase Hotel'''


import unittest

from unittest.mock import patch, call
from app.classes.data_handler import DataHandler
from app.classes.customer import Customer


class TestCustomer(unittest.TestCase):
    '''Calse para probar la clase Customer'''

    def setUp(self):
        data_handler = DataHandler("test.json")
        self.customer_instance = Customer(data_handler)
        self.test_customer = [{
            "name": "customer",
            "lastName": "lastName",
            "id": 1
        }]

    # Test create customer
    @patch.object(DataHandler, 'is_missing', return_value=False)
    @patch.object(DataHandler, 'create_registry', return_value=[{
            "name": "customer",
            "lastName": "lastName",
            "id": 1
    }])
    def test_create_customer_succesfull(
        self,
        mock_create_registry,
        mock_is_missing
    ):
        '''Metodo para verificar
        el correcto
        funcionamiento de crear Customer'''
        customer = self.test_customer[0].copy()
        del customer["id"]
        result = self.customer_instance.create_customer(
            customer["name"],
            customer["lastName"]
        )
        mock_is_missing.assert_has_calls([
            call(customer["name"]),
            call(customer["lastName"]),
        ], any_order=True)
        mock_create_registry.assert_called_once_with(customer)
        self.assertEqual(result, self.test_customer)

    @patch.object(DataHandler, 'is_missing', return_value=True)
    def test_create_customer_missing_value(self, mock_is_missing):
        '''Metodo para verificar que se levante la expcion
        correcta al tener información faltante'''
        customer = self.test_customer[0].copy()
        del customer["id"]
        result = self.customer_instance.create_customer(
            customer["name"],
            customer["lastName"]
        )
        mock_is_missing.assert_called()
        self.assertRaises(ValueError)
        self.assertEqual(result, "Missing data")

```

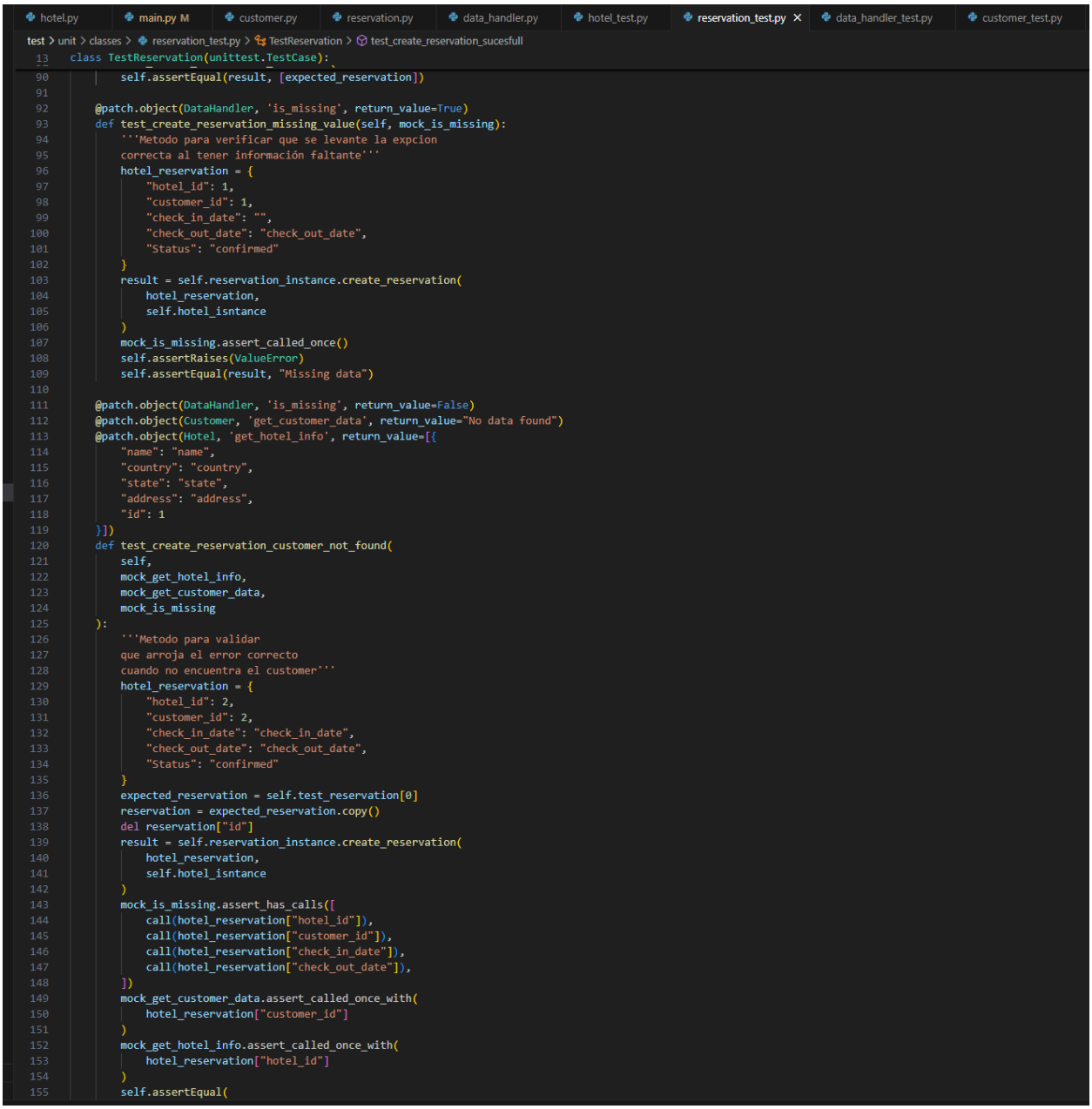

hotel.py
main.py M
customer.py
reservation.py
data_handler.py
hotel_test.py
reservation_test.py
data_handler_test.py
customer_test.py
hotels.json M
test > unit > classes > customer_test.py > TestCustomer > setUp
class TestCustomer(unittest.TestCase):
# Test delete customer
@patch.object(DataHandler, 'is_missing', return_value=False)
@patch.object(DataHandler, 'delete_from_file', return_value=[])
def test_delete_customer_succesfull(
self,
mock_delete_from_file,
mock_is_missing
):
'''Metodo para verificar
el correcto
funcionamiento del metodo delete hotel'''
id_to_delete = self.test_customer[0]["id"]
deleted_customer = self.test_customer.copy()
del deleted_customer[0]
result = self.customer_instance.delete_customer(id_to_delete)
mock_is_missing.assert_called_once_with(id_to_delete)
mock_delete_from_file.assert_called_once_with(id_to_delete)
self.assertEqual(result, deleted_customer)
@patch.object(DataHandler, 'is_missing', return_value=True)
def test_delete_customer_missing_value(self, mock_is_missing):
'''Metodo para verificar que se levante la expcion
correcta al tener información faltante'''
id_to_delete = self.test_customer[0]["id"]
result = self.customer_instance.delete_customer(id_to_delete)
mock_is_missing.assert_called_once_with(id_to_delete)
self.assertRaises(ValueError)
self.assertEqual(result, "Missing data")
# Test modify_customer
@patch.object(DataHandler, 'is_missing', return_value=False)
@patch.object(DataHandler, 'modify_file', return_value=[{
"name": "modified Name",
"lastName": "lastName",
"id": 1
}])
def test_modify_customer_succesfull(
self,
mock_modify_file,
mock_is_missing
):
'''Metodo para verificar
el correcto
funcionamiento del metodo modify customer'''
customer_to_modify = self.test_customer[0].copy()
customer_to_modify["name"] = "modified Name"
result = self.customer_instance.modify_customer(customer_to_modify)
mock_is_missing.assert_has_calls([
call(customer_to_modify["name"]),
call(customer_to_modify["lastName"]),
call(customer_to_modify["id"])
], any_order=True)
mock_modify_file.assert_called_once_with(customer_to_modify)
self.assertEqual(result, [customer_to_modify])

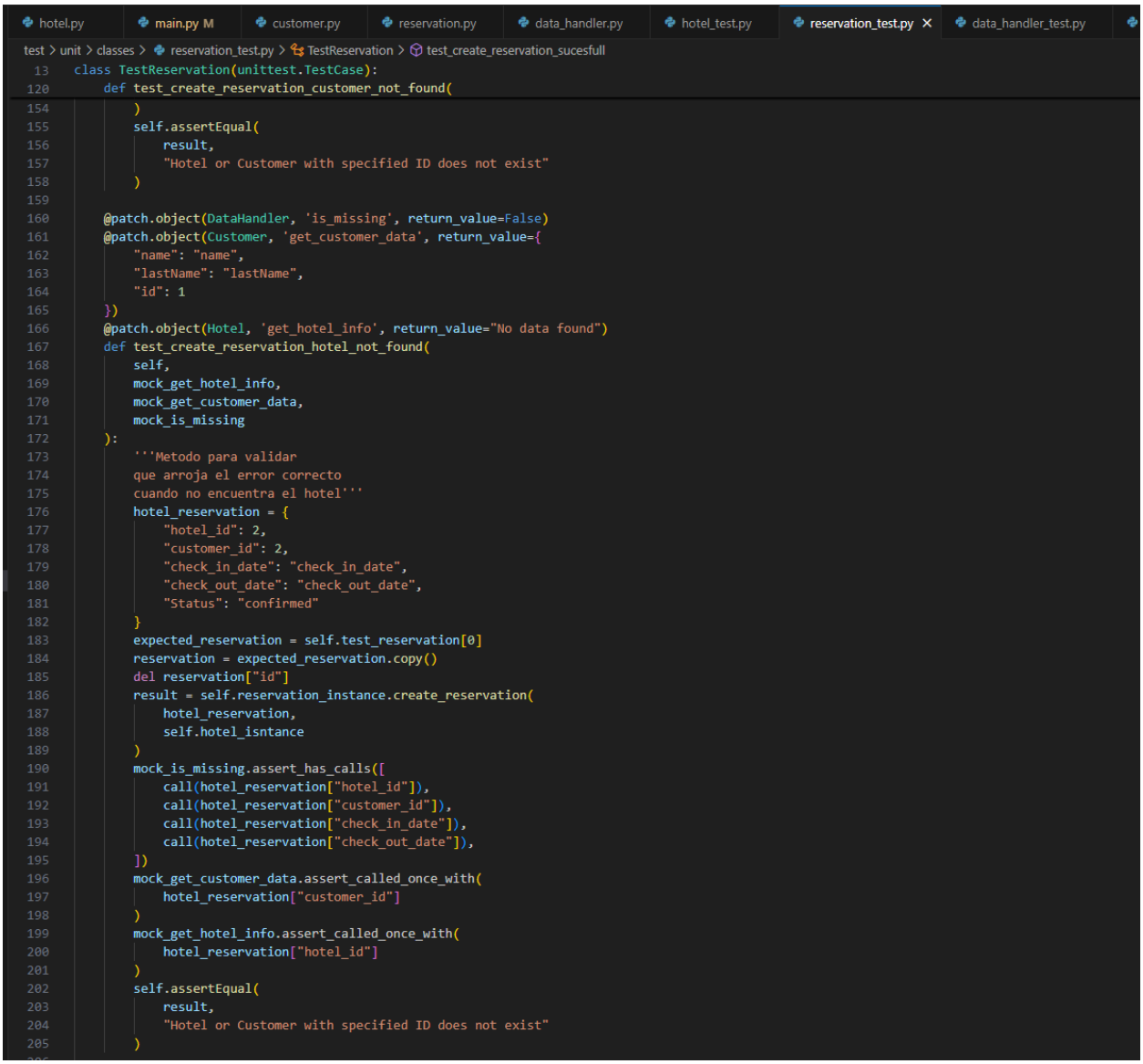


Pruebas unitarias clase reservation

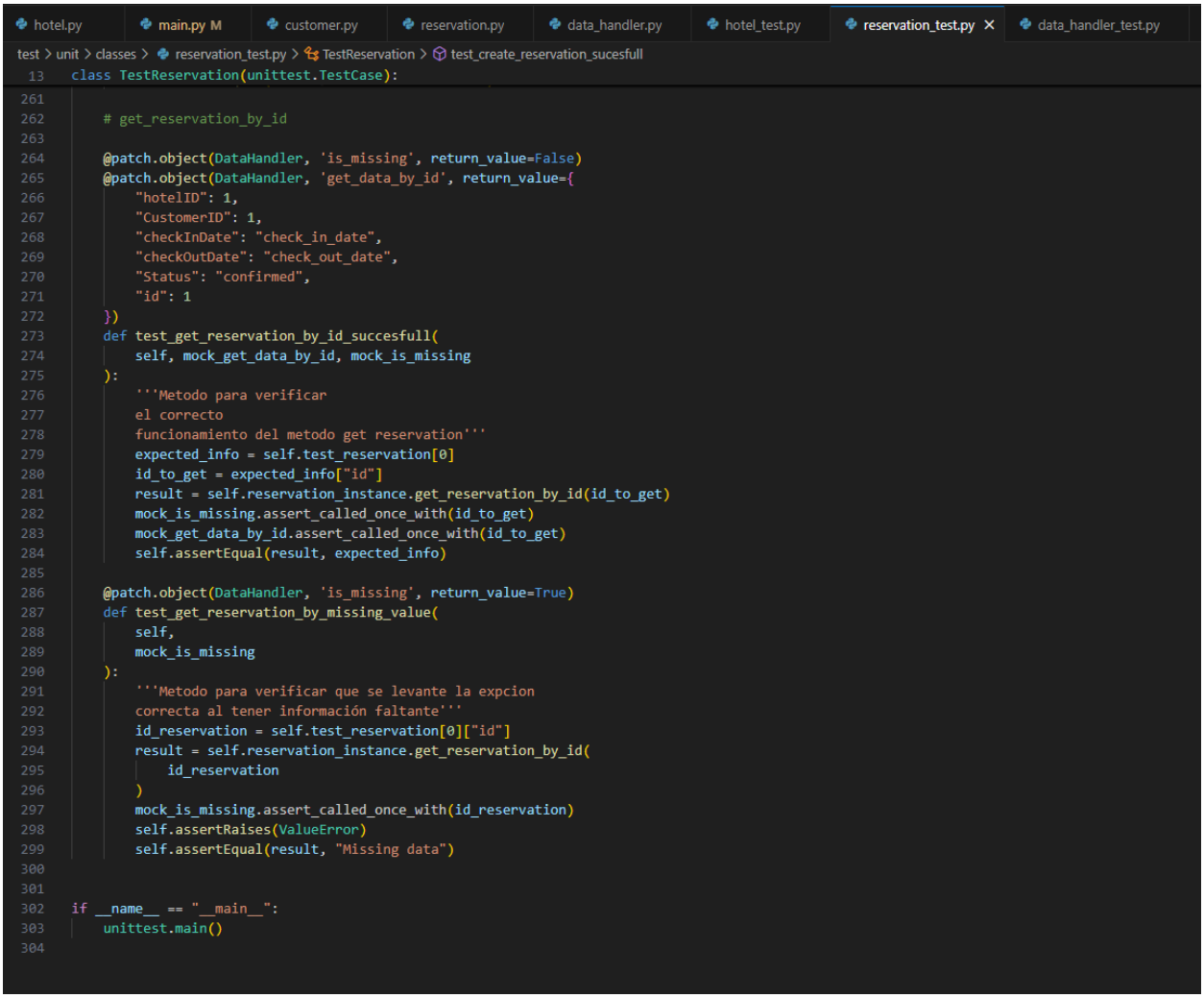



hotel.py
main.py M
customer.py
reservation.py
data_handler.py
hotel_test.py
reservation_test.py
data_handler_test.py
test > unit > classes > reservation_test.py > TestReservation > test_create_reservation_sucesfull
class TestReservation(unittest.TestCase):
def setUp(self):
}]
test create_reservation
@patch.object(DataHandler, 'is_missing', return_value=False)
@patch.object(Customer, 'get_customer_data', return_value={
"name": "name",
"lastName": "lastName",
"id": 1
})
@patch.object(Hotel, 'get_hotel_info', return_value=[{
"name": "name",
"country": "country",
"state": "state",
"address": "address",
"id": 1
}])
@patch.object(DataHandler, 'create_registry', return_value=[{
"hotelID": 1,
"CustomerID": 1,
"checkInDate": "check_in_date",
"checkOutDate": "check_out_date",
"Status": "confirmed",
"id": 1
}])
def test_create_reservation_sucesfull(
self,
mock_create_registry,
mock_get_hotel_info,
mock_get_customer_data,
mock_is_missing
):
'''Metodo para verificar
el correcto
funcionamiento de crear reservación'''
hotel_reservation = {
"hotel_id": 1,
"customer_id": 1,
"check_in_date": "check_in_date",
"check_out_date": "check_out_date",
"Status": "confirmed"
}
expected_reservation = self.test_reservation[0]
reservation = expected_reservation.copy()
del reservation["id"]
result = self.reservation_instance.create_reservation(
hotel_reservation,
self.hotel_isntance
)
mock_is_missing.assert_has_calls([
call(hotel_reservation["hotel_id"]),
call(hotel_reservation["customer_id"]),
call(hotel_reservation["check_in_date"]),
call(hotel_reservation["check_out_date"]),
])
mock_get_customer_data.assert_called_once_with(
hotel_reservation["customer_id"]
)
mock_get_hotel_info.assert_called_once_with(
hotel_reservation["hotel_id"]
)
print(expected_reservation)
mock_create_registry.assert_called_once_with(reservation)
self.assertEqual(result, [expected_reservation])

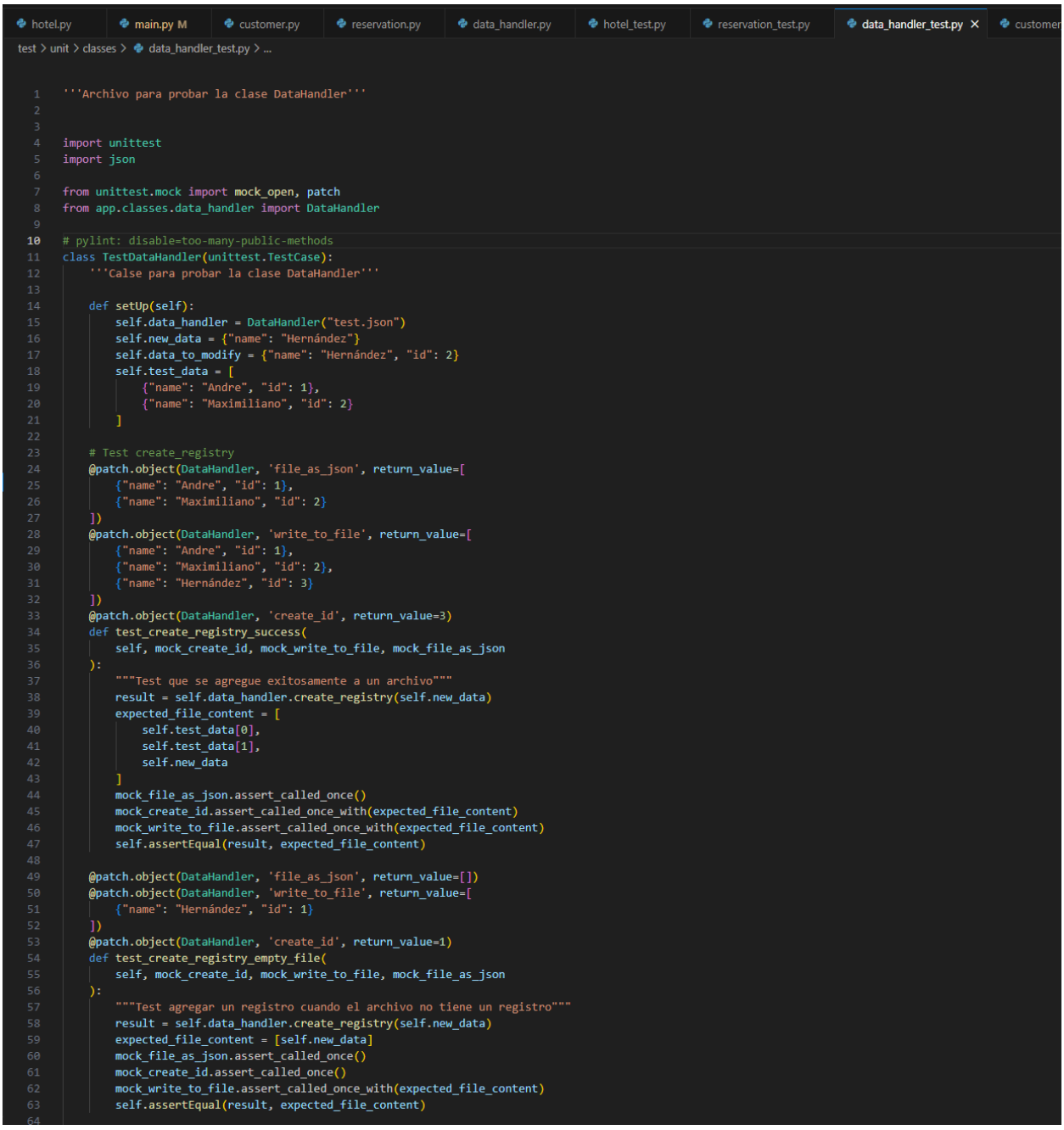

hotel.py
main.py M
customer.py
reservation.py
data_handler.py
hotel_test.py
reservation_test.py
data_handler_test.py
customer_test.py
test > unit > classes > reservation_test.py > TestReservation > test_create_reservation_sucesfull
class TestReservation(unittest.TestCase):
self.assertEqual(result, [expected_reservation])
@patch.object(DataHandler, 'is_missing', return_value=True)
def test_create_reservation_missing_value(self, mock_is_missing):
'''Metodo para verificar que se levante la expcion
correcta al tener información faltante'''
hotel_reservation = {
"hotel_id": 1,
"customer_id": 1,
"check_in_date": "",
"check_out_date": "check_out_date",
"Status": "confirmed"
}
result = self.reservation_instance.create_reservation(
hotel_reservation,
self.hotel_isntance
)
mock_is_missing.assert_called_once()
self.assertRaises(ValueError)
self.assertEqual(result, "Missing data")
@patch.object(DataHandler, 'is_missing', return_value=False)
@patch.object(Customer, 'get_customer_data', return_value="No data found")
@patch.object(Hotel, 'get_hotel_info', return_value=[{
"name": "name",
"country": "country",
"state": "state",
"address": "address",
"id": 1
}])
def test_create_reservation_customer_not_found(
self,
mock_get_hotel_info,
mock_get_customer_data,
mock_is_missing
):
'''Metodo para validar
que arroja el error correcto
cuando no encuentra el customer'''
hotel_reservation = {
"hotel_id": 2,
"customer_id": 2,
"check_in_date": "check_in_date",
"check_out_date": "check_out_date",
"Status": "confirmed"
}
expected_reservation = self.test_reservation[0]
reservation = expected_reservation.copy()
del reservation["id"]
result = self.reservation_instance.create_reservation(
hotel_reservation,
self.hotel_isntance
)
mock_is_missing.assert_has_calls([
call(hotel_reservation["hotel_id"]),
call(hotel_reservation["customer_id"]),
call(hotel_reservation["check_in_date"]),
call(hotel_reservation["check_out_date"]),
])
mock_get_customer_data.assert_called_once_with(
hotel_reservation["customer_id"]
)
mock_get_hotel_info.assert_called_once_with(
hotel_reservation["hotel_id"]
)
self.assertEqual(

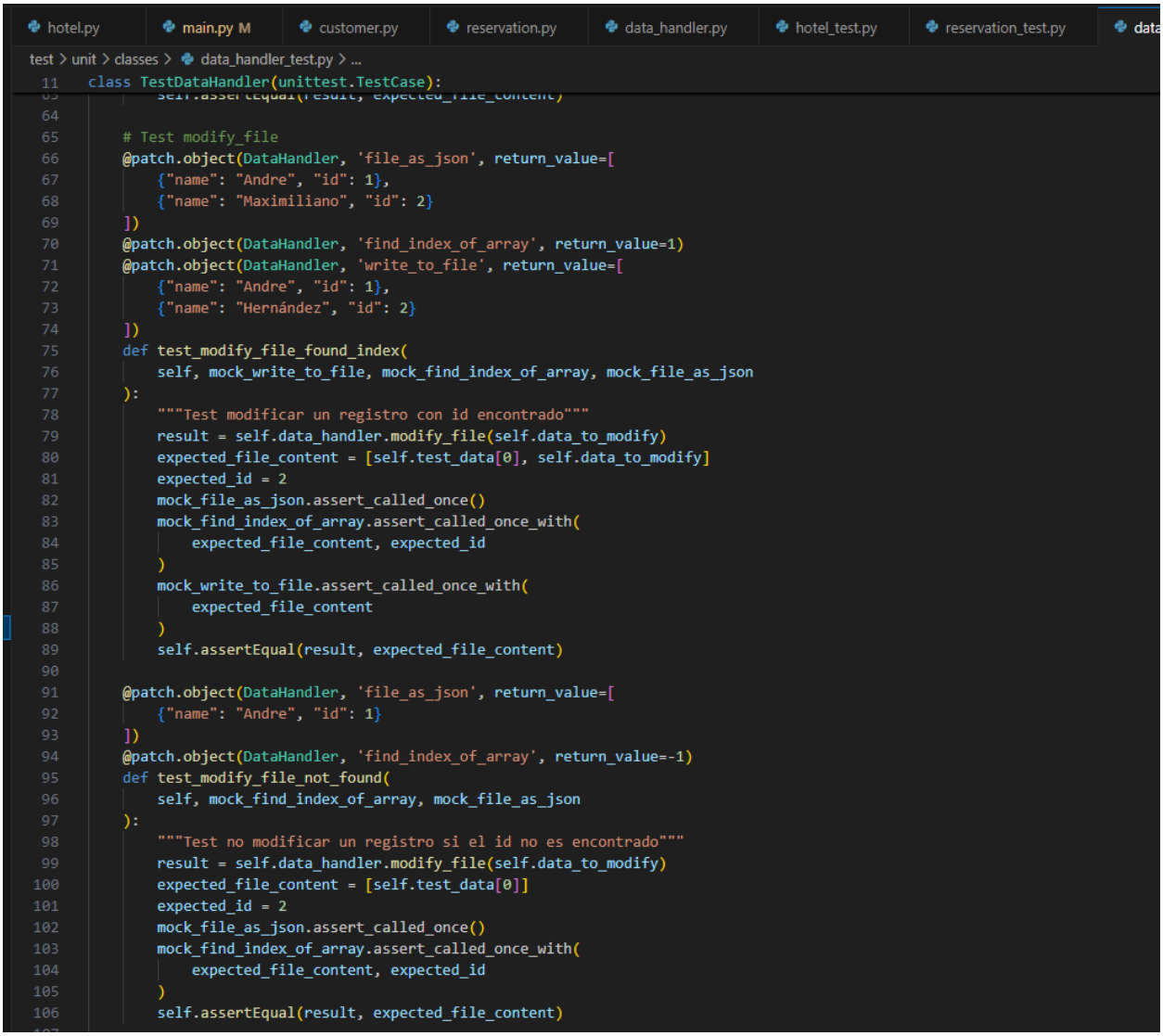

hotel.py
main.py M
customer.py
reservation.py
data_handler.py
hotel_test.py
reservation_test.py
data_handler_test.py
test > unit > classes > reservation_test.py > TestReservation > test_create_reservation_sucesfull
class TestReservation(unittest.TestCase):
def test_create_reservation_customer_not_found(
)
self.assertEqual(
result,
"Hotel or Customer with specified ID does not exist"
)
@patch.object(DataHandler, 'is_missing', return_value=False)
@patch.object(Customer, 'get_customer_data', return_value={
"name": "name",
"lastName": "lastName",
"id": 1
})
@patch.object(Hotel, 'get_hotel_info', return_value="No data found")
def test_create_reservation_hotel_not_found(
self,
mock_get_hotel_info,
mock_get_customer_data,
mock_is_missing
):
'''Metodo para validar
que arroja el error correcto
cuando no encuentra el hotel'''
hotel_reservation = {
"hotel_id": 2,
"customer_id": 2,
"check_in_date": "check_in_date",
"check_out_date": "check_out_date",
"Status": "confirmed"
}
expected_reservation = self.test_reservation[0]
reservation = expected_reservation.copy()
del reservation["id"]
result = self.reservation_instance.create_reservation(
hotel_reservation,
self.hotel_isntance
)
mock_is_missing.assert_has_calls([
call(hotel_reservation["hotel_id"]),
call(hotel_reservation["customer_id"]),
call(hotel_reservation["check_in_date"]),
call(hotel_reservation["check_out_date"]),
])
mock_get_customer_data.assert_called_once_with(
hotel_reservation["customer_id"]
)
mock_get_hotel_info.assert_called_once_with(
hotel_reservation["hotel_id"]
)
self.assertEqual(
result,
"Hotel or Customer with specified ID does not exist"
)


hotel.py
main.py M
customer.py
reservation.py
data_handler.py
hotel_test.py
test > unit > classes > reservation_test.py > TestReservation > test_create_reservation_sucesfull
13 class TestReservation(unittest.TestCase):
206
207 # test cancel_reservation
208
209 @patch.object(Reservation, 'get_reservation_by_id', return_value={
210 "hotelID": 1,
211 "CustomerID": 1,
212 "checkInDate": "check_in_date",
213 "checkOutDate": "check_out_date",
214 "Status": "confirmed",
215 "id": 1
216 })
217 @patch.object(DataHandler, 'modify_file', return_value=[{
218 "hotelID": 1,
219 "CustomerID": 1,
220 "checkInDate": "check_in_date",
221 "checkOutDate": "check_out_date",
222 "Status": "canceled",
223 "id": 1
224 }])
225 def test_cancel_reservation_succesfull(
226 self,
227 mock_modify_file,
228 mock_get_reservation_by_id
229):
230 '''Metodo para verificar el correcto
231 funcionamiento del metodo
232 cancel reservation'''
233 expected_result = {
234 "hotelID": 1,
235 "CustomerID": 1,
236 "checkInDate": "check_in_date",
237 "checkOutDate": "check_out_date",
238 "Status": "canceled",
239 "id": 1
240 }
241 id_to_modify = self.test_reservation[0]["id"]
242 result = self.reservation_instance.cancel_reservation(id_to_modify)
243 mock_get_reservation_by_id.assert_called_once_with(id_to_modify)
244 mock_modify_file.assert_called_once_with(expected_result)
245 self.assertEqual(result, [expected_result])
246
247 @patch.object(
248 Reservation,
249 'get_reservation_by_id',
250 return_value="No data found"
251)
252 def test_cancel_reservation_not_found(self, mock_get_reservation_by_id):
253 '''Metodo para verificar el correcto
254 funcionamiento del metodo
255 cancel reservation cuando no
256 encunetra la reservación'''
257 id_to_modify = self.test_reservation[0]["id"]
258 result = self.reservation_instance.cancel_reservation(id_to_modify)
259 mock_get_reservation_by_id.assert_called_once_with(id_to_modify)
260 self.assertEqual(result, "Id does not exist")

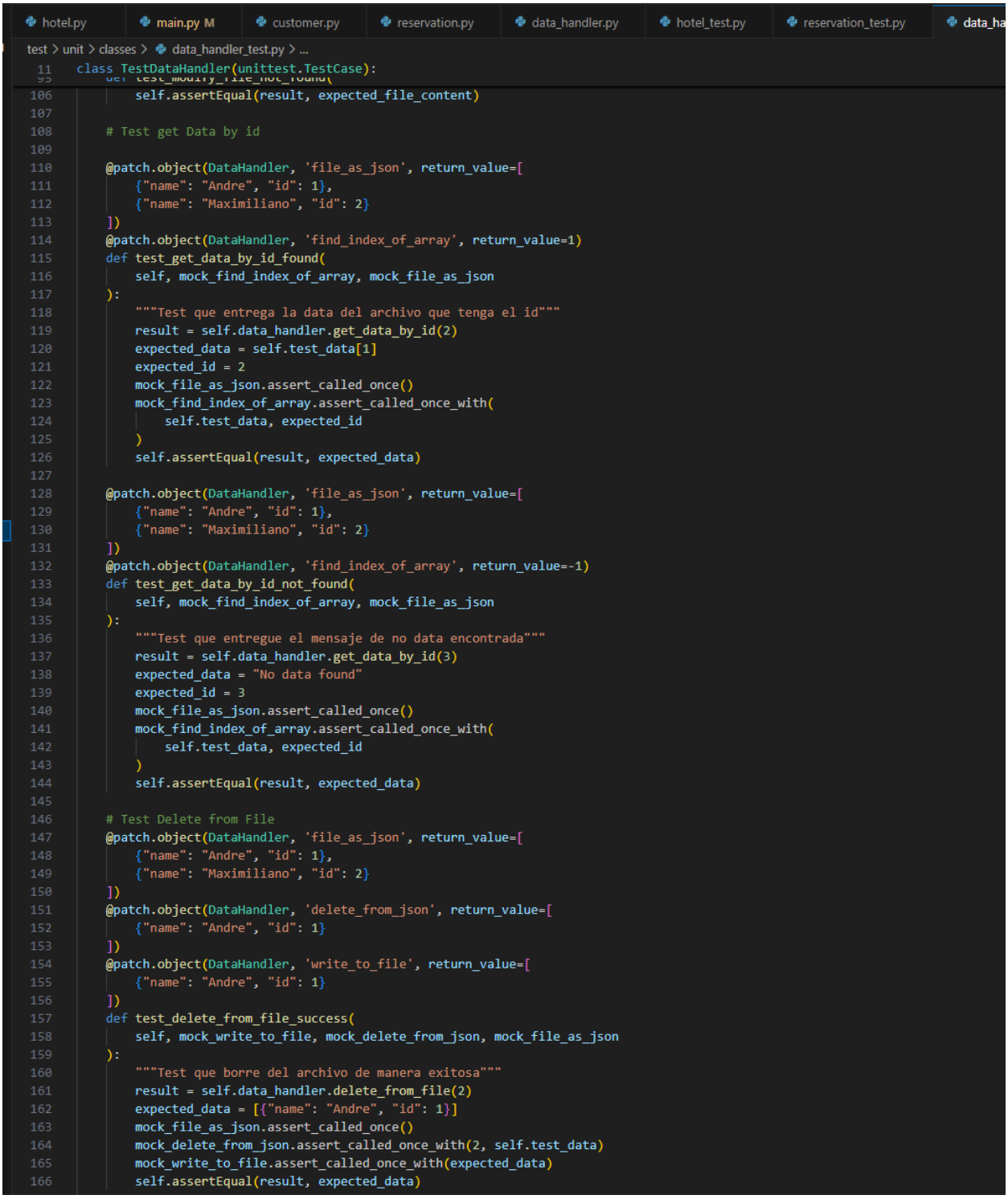


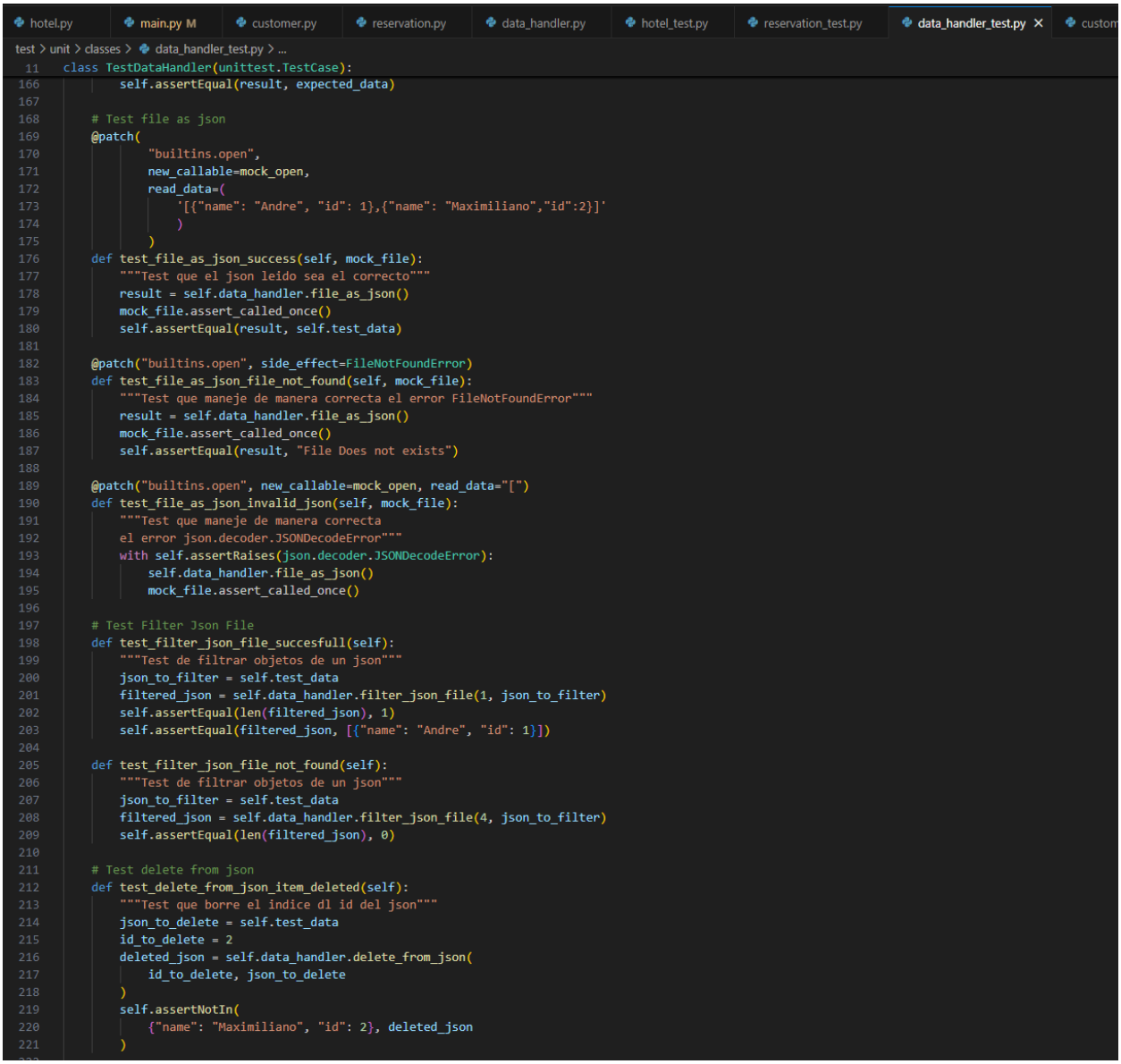
Pruebas unitarias clase datahandler


hotel.py
main.py M
customer.py
reservation.py
data_handler.py
hotel_test.py
reservation_test.py
data_handler_test.py
test > unit > classes > data_handler_test.py > ...
'''Archivo para probar la clase DataHandler'''
import unittest
import json
from unittest.mock import mock_open, patch
from app.classes.data_handler import DataHandler
# pylint: disable=too-many-public-methods
class TestDataHandler(unittest.TestCase):
'''Calse para probar la clase DataHandler'''
def setUp(self):
self.data_handler = DataHandler("test.json")
self.new_data = {"name": "Hernández"}
self.data_to_modify = {"name": "Hernández", "id": 2}
self.test_data = [
{"name": "Andre", "id": 1},
{"name": "Maximiliano", "id": 2}
]
Test create_registry
@patch.object(DataHandler, 'file_as_json', return_value=[
{"name": "Andre", "id": 1},
{"name": "Maximiliano", "id": 2}
])
@patch.object(DataHandler, 'write_to_file', return_value=[
{"name": "Andre", "id": 1},
{"name": "Maximiliano", "id": 2},
{"name": "Hernández", "id": 3}
])
@patch.object(DataHandler, 'create_id', return_value=3)
def test_create_registry_success(
self, mock_create_id, mock_write_to_file, mock_file_as_json
):
"""Test que se agregue exitosamente a un archivo"""
result = self.data_handler.create_registry(self.new_data)
expected_file_content = [
self.test_data[0],
self.test_data[1],
self.new_data
]
mock_file_as_json.assert_called_once()
mock_create_id.assert_called_once_with(expected_file_content)
mock_write_to_file.assert_called_once_with(expected_file_content)
self.assertEqual(result, expected_file_content)
@patch.object(DataHandler, 'file_as_json', return_value=[])
@patch.object(DataHandler, 'write_to_file', return_value=[
{"name": "Hernández", "id": 1}
])
@patch.object(DataHandler, 'create_id', return_value=1)
def test_create_registry_empty_file(
self, mock_create_id, mock_write_to_file, mock_file_as_json
):
"""Test agregar un registro cuando el archivo no tiene un registro"""
result = self.data_handler.create_registry(self.new_data)
expected_file_content = [self.new_data]
mock_file_as_json.assert_called_once()
mock_create_id.assert_called_once()
mock_write_to_file.assert_called_once_with(expected_file_content)
self.assertEqual(result, expected_file_content)




```python
class TestDataHandler(unittest.TestCase):
        self.assertEqual(result, expected_file_content)

    # Test modify_file
    @patch.object(DataHandler, 'file_as_json', return_value=[
        {"name": "Andre", "id": 1},
        {"name": "Maximiliano", "id": 2}
    ])
    @patch.object(DataHandler, 'find_index_of_array', return_value=1)
    @patch.object(DataHandler, 'write_to_file', return_value=[
        {"name": "Andre", "id": 1},
        {"name": "Hernández", "id": 2}
    ])
    def test_modify_file_found_index(
        self, mock_write_to_file, mock_find_index_of_array, mock_file_as_json
    ):
        """Test modificar un registro con id encontrado"""
        result = self.data_handler.modify_file(self.data_to_modify)
        expected_file_content = [self.test_data[0], self.data_to_modify]
        expected_id = 2
        mock_file_as_json.assert_called_once()
        mock_find_index_of_array.assert_called_once_with(
            expected_file_content, expected_id
        )
        mock_write_to_file.assert_called_once_with(
            expected_file_content
        )
        self.assertEqual(result, expected_file_content)

    @patch.object(DataHandler, 'file_as_json', return_value=[
        {"name": "Andre", "id": 1}
    ])
    @patch.object(DataHandler, 'find_index_of_array', return_value=-1)
    def test_modify_file_not_found(
        self, mock_find_index_of_array, mock_file_as_json
    ):
        """Test no modificar un registro si el id no es encontrado"""
        result = self.data_handler.modify_file(self.data_to_modify)
        expected_file_content = [self.test_data[0]]
        expected_id = 2
        mock_file_as_json.assert_called_once()
        mock_find_index_of_array.assert_called_once_with(
            expected_file_content, expected_id
        )
        self.assertEqual(result, expected_file_content)
```


hotel.py
main.py M
customer.py
reservation.py
data_handler.py
hotel_test.py
reservation_test.py
data_ha
test > unit > classes > data_handler_test.py > ...
class TestDataHandler(unittest.TestCase):
self.assertEqual(result, expected_file_content)
# Test get Data by id
@patch.object(DataHandler, 'file_as_json', return_value=[
{"name": "Andre", "id": 1},
{"name": "Maximiliano", "id": 2}
])
@patch.object(DataHandler, 'find_index_of_array', return_value=1)
def test_get_data_by_id_found(
self, mock_find_index_of_array, mock_file_as_json
):
"""Test que entrega la data del archivo que tenga el id"""
result = self.data_handler.get_data_by_id(2)
expected_data = self.test_data[1]
expected_id = 2
mock_file_as_json.assert_called_once()
mock_find_index_of_array.assert_called_once_with(
self.test_data, expected_id
)
self.assertEqual(result, expected_data)
@patch.object(DataHandler, 'file_as_json', return_value=[
{"name": "Andre", "id": 1},
{"name": "Maximiliano", "id": 2}
])
@patch.object(DataHandler, 'find_index_of_array', return_value=-1)
def test_get_data_by_id_not_found(
self, mock_find_index_of_array, mock_file_as_json
):
"""Test que entregue el mensaje de no data encontrada"""
result = self.data_handler.get_data_by_id(3)
expected_data = "No data found"
expected_id = 3
mock_file_as_json.assert_called_once()
mock_find_index_of_array.assert_called_once_with(
self.test_data, expected_id
)
self.assertEqual(result, expected_data)
# Test Delete from File
@patch.object(DataHandler, 'file_as_json', return_value=[
{"name": "Andre", "id": 1},
{"name": "Maximiliano", "id": 2}
])
@patch.object(DataHandler, 'delete_from_json', return_value=[
{"name": "Andre", "id": 1}
])
@patch.object(DataHandler, 'write_to_file', return_value=[
{"name": "Andre", "id": 1}
])
def test_delete_from_file_success(
self, mock_write_to_file, mock_delete_from_json, mock_file_as_json
):
"""Test que borre del archivo de manera exitosa"""
result = self.data_handler.delete_from_file(2)
expected_data = [{"name": "Andre", "id": 1}]
mock_file_as_json.assert_called_once()
mock_delete_from_json.assert_called_once_with(2, self.test_data)
mock_write_to_file.assert_called_once_with(expected_data)
self.assertEqual(result, expected_data)


hotel.py
main.py M
customer.py
reservation.py
data_handler.py
hotel_test.py
reservation_test.py
data_handler_test.py
test > unit > classes > data_handler_test.py > ...
class TestDataHandler(unittest.TestCase):
self.assertEqual(result, expected_data)
# Test file as json
@patch(
"builtins.open",
new_callable=mock_open,
read_data=(
'[{"name": "Andre", "id": 1},{"name": "Maximiliano","id":2}]'
)
)
def test_file_as_json_success(self, mock_file):
"""Test que el json leido sea el correcto"""
result = self.data_handler.file_as_json()
mock_file.assert_called_once()
self.assertEqual(result, self.test_data)
@patch("builtins.open", side_effect=FileNotFoundError)
def test_file_as_json_file_not_found(self, mock_file):
"""Test que maneje de manera correcta el error FileNotFoundError"""
result = self.data_handler.file_as_json()
mock_file.assert_called_once()
self.assertEqual(result, "File Does not exists")
@patch("builtins.open", new_callable=mock_open, read_data="[")
def test_file_as_json_invalid_json(self, mock_file):
"""Test que maneje de manera correcta
el error json.decoder.JSONDecodeError"""
with self.assertRaises(json.decoder.JSONDecodeError):
self.data_handler.file_as_json()
mock_file.assert_called_once()
# Test Filter Json File
def test_filter_json_file_succesfull(self):
"""Test de filtrar objetos de un json"""
json_to_filter = self.test_data
filtered_json = self.data_handler.filter_json_file(1, json_to_filter)
self.assertEqual(len(filtered_json), 1)
self.assertEqual(filtered_json, [{"name": "Andre", "id": 1}])
def test_filter_json_file_not_found(self):
"""Test de filtrar objetos de un json"""
json_to_filter = self.test_data
filtered_json = self.data_handler.filter_json_file(4, json_to_filter)
self.assertEqual(len(filtered_json), 0)
# Test delete from json
def test_delete_from_json_item_deleted(self):
"""Test que borre el indice dl id del json"""
json_to_delete = self.test_data
id_to_delete = 2
deleted_json = self.data_handler.delete_from_json(
id_to_delete, json_to_delete
)
self.assertNotIn(
{"name": "Maximiliano", "id": 2}, deleted_json
)

hotel.py | main.py M | customer.py | reservation.py | data_handler.py | hotel_test.py | reservation_test.py | data_handler_test.py

test > unit > classes > data_handler_test.py > ...

```python
class TestDataHandler(unittest.TestCase):
        )

    # Test find index of Array
    def test_find_index_of_array_found_index(self):
        """Test que encuentre el indice del id proporcionado"""
        json_to_filter = self.test_data
        index = self.data_handler.find_index_of_array(json_to_filter, 1)
        self.assertEqual(index, 0)

    def test_find_index_of_array_not_found_index(self):
        """Test que eregrese -1 cuando no encuentra el id"""
        json_to_filter = self.test_data
        index = self.data_handler.find_index_of_array(json_to_filter, 4)
        self.assertEqual(index, -1)

    # Test write to file
    @patch(
            "builtins.open",
            new_callable=mock_open,
            read_data=(
                '[{"name": "Andre", "id": 1},{"name": "Maximiliano","id":2}]'
            )
        )
    def test_write_to_file_success(self, mock_file):
        """Test de que se escribe de manera correcta al archivo"""
        result = self.data_handler.write_to_file(self.test_data)
        mock_file.assert_any_call("test.json", "w", encoding="utf-8")
        self.assertEqual(result, self.test_data)

    @patch("builtins.open", side_effect=TypeError)
    def test_write_to_file_failure(self, mock_file):
        """Test de write_to_file maneja errores"""
        result = self.data_handler.write_to_file(self.test_data)
        mock_file.assert_called_once()
        self.assertRaises(TypeError)
        self.assertEqual(result, "Error writing to file")

    # Test create ID
    def test_create_id_succesfull(self):
        '''Metod to test creation of id'''
        result = self.data_handler.create_id(self.test_data)
        self.assertEqual(result, 3)

    def test_create_id_empty(self):
        '''Metod to test creation of id when file is empty'''
        result = self.data_handler.create_id([])
        self.assertEqual(result, 1)
```

hotel.py | main.py M | customer.py | reservation.py | data_handler.py | hotel_test.py | reservation_test.py | data_handler_test.py

test > unit > classes > data_handler_test.py > ...

```python
class TestDataHandler(unittest.TestCase):
    def test_create_id_empty(self):
        self.assertEqual(result, 1)

    # Test is missing
    def test_is_missing_no_missing(self):
        '''Metodo para verificar que regresa
        false si el valor no es nulo o vacio'''
        result = self.data_handler.is_missing("test")
        self.assertEqual(result, False)

    def test_is_missing_none(self):
        '''Metodo para verificar que regresa
        true si el valor es nulo'''
        result = self.data_handler.is_missing(None)
        self.assertEqual(result, True)

    def test_is_missing_empty(self):
        '''Metodo para verificar que regresa
        true si el valor es vacio'''
        result = self.data_handler.is_missing("")
        self.assertEqual(result, True)


if __name__ == "__main__":
    unittest.main()
```

Resultado flake 8

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\soyel\Documents\Maestria\Pruebas de Software y aseguramiento de calidad\Semana 6> flake8 ./
PS C:\Users\soyel\Documents\Maestria\Pruebas de Software y aseguramiento de calidad\Semana 6>
```

Resultado Pylint

```
PS C:\Users\soyel\Documents\Maestria\Pruebas de Software y aseguramiento de calidad\Semana 6> pylint ./
************* Module app.__init__
app\__init__ .py:1:0: C0103: Module name "__init__ " doesn't conform to snake_case naming style (invalid-name)

-------------------------------------------------------------------
Your code has been rated at 9.98/10 (previous run: 9.98/10, +0.00)
```

El error se dejó, ya que ese archivo fue para agregar el correcto formato del modulo y ese debe ser su nombre, no se puede cambiar.

Pruebas Unitarias Sin errores Clase Hotel

```
PS C:\Users\soyel\Documents\Maestria\Pruebas de Software y aseguramiento de calidad\Semana 6> $env:PYTHONPATH = $PWD
PS C:\Users\soyel\Documents\Maestria\Pruebas de Software y aseguramiento de calidad\Semana 6> python test/unit/classes/hotel_test.py
...........
----------------------------------------------------------------------
Ran 11 tests in 0.004s

OK
PS C:\Users\soyel\Documents\Maestria\Pruebas de Software y aseguramiento de calidad\Semana 6>
```

Pruebas unitarias Sin errores Clase Customer

```
PS C:\Users\soyel\Documents\Maestria\Pruebas de Software y aseguramiento de calidad\Semana 6> python test/unit/classes/customer_test.py
........
----------------------------------------------------------------------
Ran 8 tests in 0.003s

OK
PS C:\Users\soyel\Documents\Maestria\Pruebas de Software y aseguramiento de calidad\Semana 6>
```

Pruebas unitarias Sin errores Clase Reservation

```
PS C:\Users\soyel\Documents\Maestria\Pruebas de Software y aseguramiento de calidad\Semana 6> python test/unit/classes/reservation_test.py
.....{'hotelID': 1, 'CustomerID': 1, 'checkInDate': 'check_in_date', 'checkOutDate': 'check_out_date', 'Status': 'confirmed', 'id': 1}
...
----------------------------------------------------------------------
Ran 8 tests in 0.004s

OK
PS C:\Users\soyel\Documents\Maestria\Pruebas de Software y aseguramiento de calidad\Semana 6>
```

Pruebas unitarias Sin errores Clase DataHandler

```
PS C:\Users\soyel\Documents\Maestria\Pruebas de Software y aseguramiento de calidad\Semana 6> python test/unit/classes/data_handler_test.py
......File does not exists
.Empty Json, give propper format to file
.............Error writing to file
..
----------------------------------------------------------------------
Ran 22 tests in 0.013s

OK
```

Pruebas Unitarias Forzando un error Clase Hotel

Cambio realizado

```
# Test create Hotel
@patch.object(DataHandler, 'is_missing', return_value=False)
@patch.object(DataHandler, 'create_registry', return_value=[{
        "name": "hotel",
        "country": "country",
        "state": "state",
        "address": "address",
        "id": 1
}])
def test_create_hotel_succesfull(
    self, mock_create_registry, mock_is_missing
):
    '''Metodo para verificar
    el correcto
    funcionamiento de crear hotel'''
    hotel = self.test_hotel[0].copy()
    del hotel["id"]
    result = self.hotel_instance.create_hotel(
        hotel["name"],
        hotel["country"],
        hotel["state"],
        hotel["address"]
    )
    mock_is_missing.assert_has_calls([
        call(hotel["name"]),
        call(hotel["country"]),
        call(hotel["state"]),
        call(hotel["address"])
    ], any_order=True)
    mock_create_registry.assert_called_once_with(hotel)
    self.assertEqual(result, {})
```

Resultado

```
PS C:\Users\soyel\Documents\Maestria\Pruebas de Software y aseguramiento de calidad\Semana 6> python test/unit/classes/hotel_test.py
..F.......
======================================================================
FAIL: test_create_hotel_succesfull (__main__.TestHotel.test_create_hotel_succesfull)
Metodo para verificar
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\soyel\AppData\Local\Programs\Python\Python311\Lib\unittest\mock.py", line 1375, in patched
    return func(*newargs, **newkeywargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\soyel\Documents\Maestria\Pruebas de Software y aseguramiento de calidad\Semana 6\test\unit\classes\hotel_test.py", line 64, in test_create_hotel_succesfull
    self.assertEqual(result, {})
AssertionError: [{'name': 'hotel', 'country': 'country', [44 chars]: 1}] != {}

----------------------------------------------------------------------
Ran 11 tests in 0.008s

FAILED (failures=1)
```

Pruebas unitarias Sin errores Clase Customer

Cambio realizado

```python
# Test create customer
@patch.object(DataHandler, 'is_missing', return_value=False)
@patch.object(DataHandler, 'create_registry', return_value=[{
        "name": "customer",
        "lastName": "lastName",
        "id": 1
}])
def test_create_customer_succesfull(
    self,
    mock_create_registry,
    mock_is_missing
):
    '''Metodo para verificar
    el correcto
    funcionamiento de crear Customer'''
    customer = self.test_customer[0].copy()
    del customer["id"]
    result = self.customer_instance.create_customer(
        customer["name"],
        customer["lastName"]
    )
    mock_is_missing.assert_has_calls([
        call(customer["name"]),
        call(customer["la
    ], any_order=True)
    mock_create_registry.assert_called_once_with(customer)
    self.assertEqual(result, {})
```

(function) assert_called_once_with: Any

Resultado

```
PS C:\Users\soyel\Documents\Maestria\Pruebas de Software y aseguramiento de calidad\Semana 6> python test/unit/classes/customer_test.py
.F.....
======================================================================
FAIL: test_create_customer_succesfull (__main__.TestCustomer.test_create_customer_succesfull)
Metodo para verificar
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\soyel\AppData\Local\Programs\Python\Python311\Lib\unittest\mock.py", line 1375, in patched
    return func(*newargs, **newkeywargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\soyel\Documents\Maestria\Pruebas de Software y aseguramiento de calidad\Semana 6\test\unit\classes\customer_test.py", line 49, in test_create_customer_succesfull
    self.assertEqual(result, {})
AssertionError: [{'name': 'customer', 'lastName': 'lastName', 'id': 1}] != {}

----------------------------------------------------------------------
Ran 8 tests in 0.009s

FAILED (failures=1)
```

Pruebas unitarias Sin errores Clase Reservation

Cambio realizado

```python
def test_get_reservation_by_id_succesfull(
    self, mock_get_data_by_id, mock_is_missing
):
    '''Metodo para verificar
    el correcto
    funcionamiento del metodo get reservation'''
    expected_info = self.test_reservation[0]
    id_to_get = expected_info["id"]
    result = self.reservation_instance.get_reservation_by_id(id_to_get)
    mock_is_missing.assert_called_once_with(id_to_get)
    mock_get_data_by_id.assert_called_once_with(id_to_get)
    self.assertEqual(result, expected_info)

@patch.object(DataHandler, 'is_missing', return_value=True)
def test_get_reservation_by_missing_value(
    self,
    mock_is_missing
):
    '''Metodo para verificar que se levante la expcion
    correcta al tener información faltante'''
    id_reservation = self.test_reservation[0]["id"]
    result = self.reservation_instance.get_reservation_by_id(
        id_reservation
    )
    mock_is_missing.assert_called_once_with(id_reservation)
    self.assertRaises(ValueError)
    self.assertEqual(result, "M")
```

Resultado

```
PS C:\Users\soyel\Documents\Maestria\Pruebas de Software y aseguramiento de calidad\Semana 6> python test/unit/classes/reservation_test.py
.....{'hotelID': 1, 'CustomerID': 1, 'checkInDate': 'check_in_date', 'checkOutDate': 'check_out_date', 'Status': 'confirmed', 'id': 1}
..F
======================================================================
FAIL: test_get_reservation_by_missing_value (__main__.TestReservation.test_get_reservation_by_missing_value)
Metodo para verificar que se levante la expcion
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\soyel\AppData\Local\Programs\Python\Python311\Lib\unittest\mock.py", line 1375, in patched
    return func(*newargs, **newkeywargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\soyel\Documents\Maestria\Pruebas de Software y aseguramiento de calidad\Semana 6\test\unit\classes\reservation_test.py", line 299, in test_get_reservation_by_missing_value
    self.assertEqual(result, "M")
AssertionError: 'Missing data' != 'M'
- Missing data
+ M


----------------------------------------------------------------------
Ran 8 tests in 0.007s

FAILED (failures=1)
```

Pruebas unitarias Sin errores Clase DataHandler

Cambio realizado

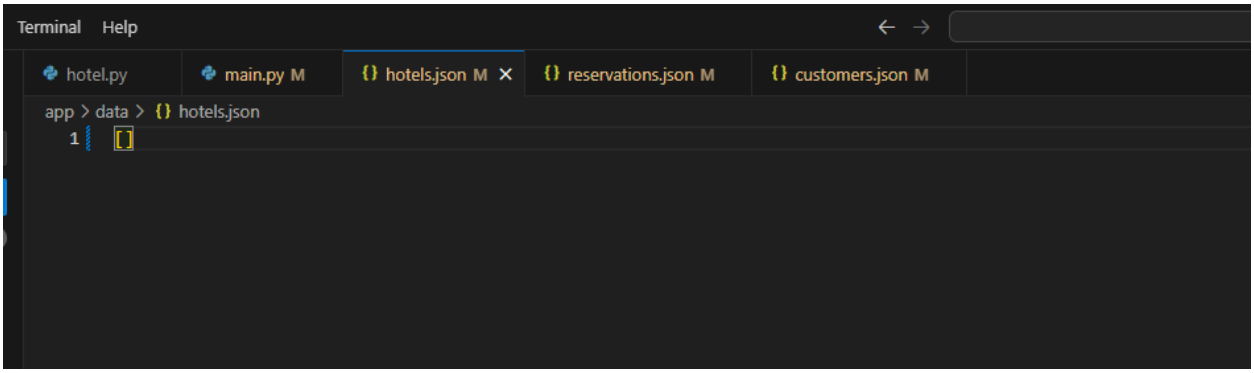
```python
# Test create_registry
@patch.object(DataHandler, 'file_as_json', return_value=[
    {"name": "Andre", "id": 1},
    {"name": "Maximiliano", "id": 2}
])
@patch.object(DataHandler, 'write_to_file', return_value=[
    {"name": "Andre", "id": 1},
    {"name": "Maximiliano", "id": 2},
    {"name": "Hernández", "id": 3}
])
@patch.object(DataHandler, 'create_id', return_value=3)
def test_create_registry_success(
    self, mock_create_id, mock_write_to_file, mock_file_as_json
):
    """Test que se agregue exitosamente a un archivo"""
    result = self.data_handler.create_registry(self.new_data)
    expected_file_content = [
        self.test_data[0],
        self.test_data[1],
        self.new_data
    ]
    mock_file_as_json.assert_called_once()
    mock_create_id.assert_called_once_with(expected_file_content)
    mock_write_to_file.assert_called_once_with(expected_file_content)
    self.assertEqual(result, 1)
```

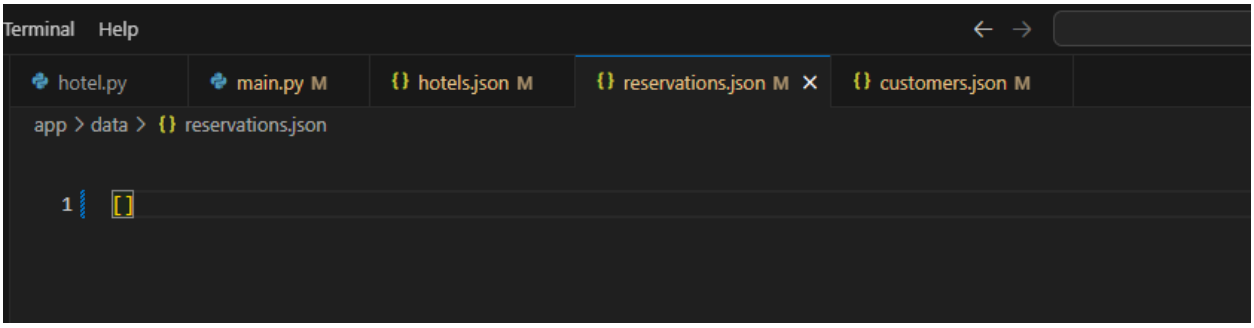
Resultado

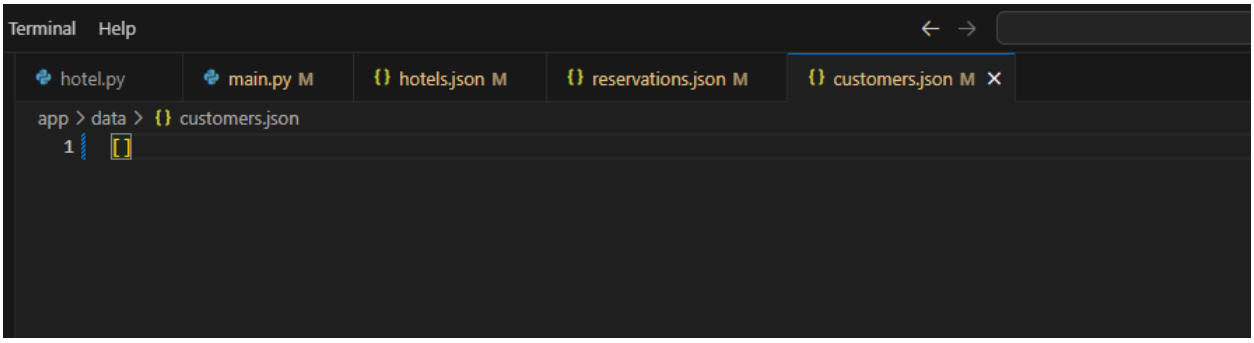
```
PS C:\Users\soyel\Documents\Maestria\Pruebas de Software y aseguramiento de calidad\Semana 6> python test/unit/classes/data_handler_test.py
...F..File does not exists
.Empty Json, give propper format to file
.............Error writing to file
..
======================================================================
FAIL: test_create_registry_success (__main__.TestDataHandler.test_create_registry_success)
Test que se agregue exitosamente a un archivo
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\soyel\AppData\Local\Programs\Python\Python311\Lib\unittest\mock.py", line 1375, in patched
    return func(*newargs, **newkeywargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\soyel\Documents\Maestria\Pruebas de Software y aseguramiento de calidad\Semana 6\test\unit\classes\data_handler_test.py", line 48, in test_create_registry_success
    self.assertEqual(result, 1)
AssertionError: [{'name': 'Andre', 'id': 1}, {'name': 'Ma[48 chars]: 3}] != 1

----------------------------------------------------------------------
Ran 22 tests in 0.015s

FAILED (failures=1)
```

Corrida programa de control

Archivos de datos antes de la corrida







Ejecución programa de control

```
PS C:\Users\soyel\Documents\Maestria\Pruebas de Software y aseguramiento de calidad\Semana 6> python .\app\main.py
{'name': 'modifiedName', 'country': 'country', 'state': 'state', 'address': 'address', 'id': 1}
No data found
[{'name': 'name', 'lastName': 'lastName', 'id': 1}, {'name': 'name2', 'lastName': 'lastName2', 'id': 2}]
{'name': 'modifiedName', 'lastName': 'lastName', 'id': 1}
No data found
Failed creating reservation
[{'hotelID': 1, 'CustomerID': 1, 'checkInDate': '23/12/2025', 'checkOutDate': '28/12/2025', 'Status': 'canceled', 'id': 1}]
Id does not exist
PS C:\Users\soyel\Documents\Maestria\Pruebas de Software y aseguramiento de calidad\Semana 6>
```

Archivos de datos después de la corrida

hotel.py | main.py M | {} hotels.json | {} reservations.json M | {} customers.json

app > data > {} hotels.json > {} 0

```
1  [{"name": "modifiedName", "country": "country", "state": "state", "address": "address", "id": 1}]
```

hotel.py | main.py M | {} hotels.json | {} reservations.json M | {} customers.json

app > data > {} reservations.json > {} 0

```
1  [{"hotelID": 1, "CustomerID": 1, "checkInDate": "23/12/2025", "checkOutDate": "28/12/2025", "Status": "canceled", "id": 1}]
```

hotel.py | main.py M | {} hotels.json | {} reservations.json M | {} customers.json

app > data > {} customers.json > ...

```
1  [{"name": "modifiedName", "lastName": "lastName", "id": 1}]
```

Code Coverage

```
PS C:\Users\soyel\Documents\Maestria\Pruebas de Software y aseguramiento de calidad\Semana 6> coverage report -m
Name                                          Stmts   Miss  Cover   Missing
---------------------------------------------------------------------------
app\classes\__init__.py                           0      0   100%
app\classes\customer.py                          36      0   100%
app\classes\data_handler.py                      72      0   100%
app\classes\hotel.py                             55      0   100%
app\classes\reservation.py                       38      0   100%
tests\unit\classes\customer_test.py              78      1    99%   165
tests\unit\classes\data_handler_test.py         143      2    99%   196, 291
tests\unit\classes\hotel_test.py                105      1    99%   258
tests\unit\classes\reservation_test.py           95      1    99%   303
---------------------------------------------------------------------------
TOTAL                                           622      5    99%
```