Grupo 801:

Desarrollo de Software

EJERCICIO GUIADO 3

Integrantes del grupo

Pablo Garaulet 100475228
Miguel González 100475158
Álvaro Moreno Martín 100472226

Índice

1. Objetivos

El principal objetivo ha sido aplicar técnicas avanzadas de refactoring y principios de diseño simple. Este proyecto no solo se ha centrado en mejorar la calidad del código, sino también en reforzar nuestra comprensión de las prácticas sostenibles y eficientes en el desarrollo de software.

2. Proceso de Refactoring

El proceso de refactoring ha sido complejo y se ha desarrollado en varias etapas para asegurar mejoras continuas en el código:

- Identificación de problemas: Inicialmente, hemos dedicado tiempo a identificar problemas específicos en el código, como nombres de variables poco claros, duplicación de código, funciones excesivamente largas, cambios divergentes, clases sobrecargadas y un uso excesivo de comentarios.

- Resolución de problemas identificados:
  - Renombrado de elementos: Hemos ajustado los nombres de variables, funciones y clases para reflejar de manera más precisa su función y utilidad.
  - Eliminación de duplicaciones: Hemos extraído funciones comunes a diferentes partes del código para reducir la redundancia y facilitar futuras modificaciones.
  - Refactorización de funciones largas: Hemos extraído las funciones largas y las hemos convertido en subfunciones más pequeñas mediante el uso de Extract Method. Estos métodos que extraíamos, los implementábamos en las clases hijas (como ReservationJsonStore) de la clase padre JsonStore si eran métodos específicos de esas clases, o en la clase padre JsonStore si eran métodos generales. Cuando llamamos a estos métodos creando objetos de la clases hijas de JsonStore desde clases de la carpeta uc3m_travel, como la clase hotel_manager, si la clase hija en cuestión contenía este método lo ejecutaba desde sí misma, y si no lo contenía y lo contenía la padre, entonces pasaba la instrucción automáticamente a su clase padre JsonStore, que ejecutaba este método. De igual forma lo hemos implementado al hacer las validaciones de los distintos atributos(phone_number, id_card…) . En la clase padre validamos todos los atributos, y las clases hijas contienen métodos específicos necesarios solo para esos atributos.
  - División de cambios divergentes: Hemos segmentado las funcionalidades que causaban modificaciones frecuentes en múltiples lugares del código, asegurando una mejor cohesión y menor acoplamiento.
  - Descomposición de clases grandes: Hemos aplicado técnicas como Extraer Clase y Extraer Superclase para dividir responsabilidades de forma más efectiva.

- Actualización del código según normativa PEP8: Tras cada cambio, hemos revisado el código para asegurar su alineación con la normativa de estilo PEP8, utilizando herramientas como pyLint para validar las modificaciones.

- Ejecución de pruebas: Hemos ejecutado casos de prueba para verificar que las modificaciones no introdujeran errores y que el código refactorizado mantuviera su funcionalidad deseada.

- Registro en GitHub: Hemos utilizado comandos de Commit y Push para mantener un registro detallado de los cambios y garantizar que nuestro trabajo en equipo fuese coherente y fácilmente accesible.

3. Diseño Simple

En la fase de diseño simple, hemos aplicado principios de diseño para simplificar la arquitectura del software:

➢ Implementación del patrón Singleton: Hemos asegurado que clases críticas como la gestión de hoteles solo pudieran instanciarse una vez, mejorando el control sobre recursos compartidos y la integridad de la operación.

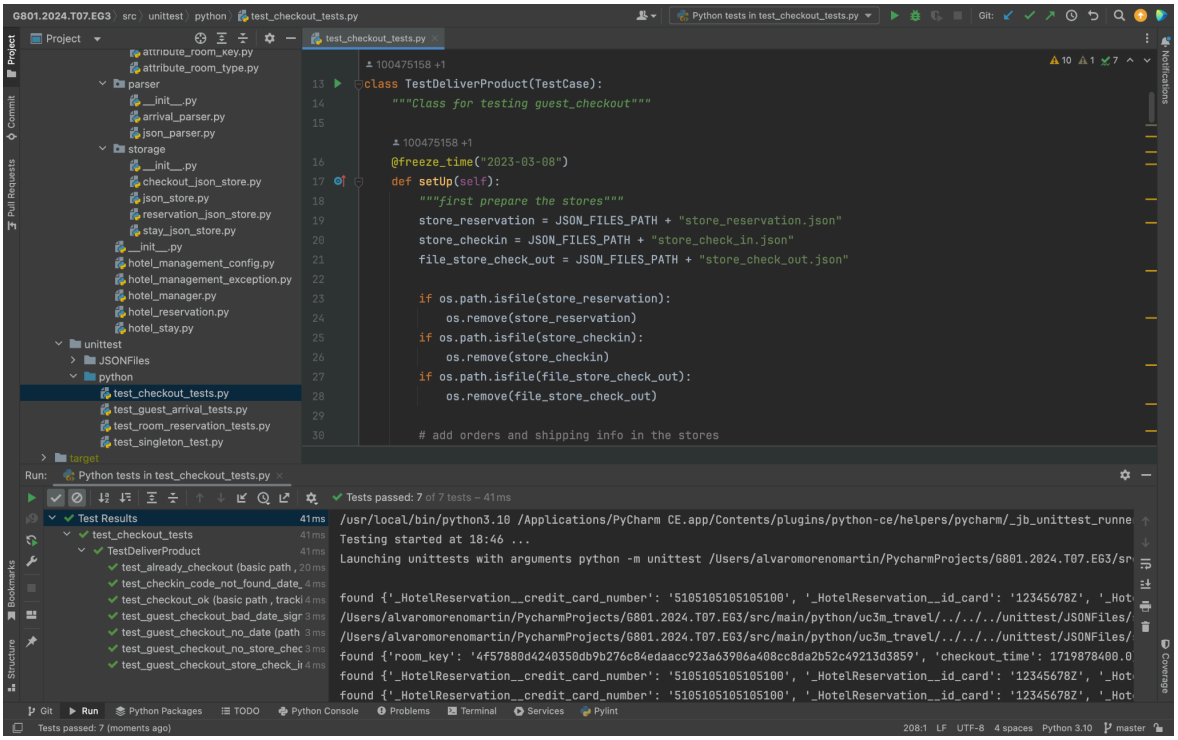
➢ Consolidación y documentación de cambios: Similar a la fase de refactoring, todos los cambios y mejoras se han documentado y registrado adecuadamente en GitHub, incluyendo detalles de implementación y pruebas.

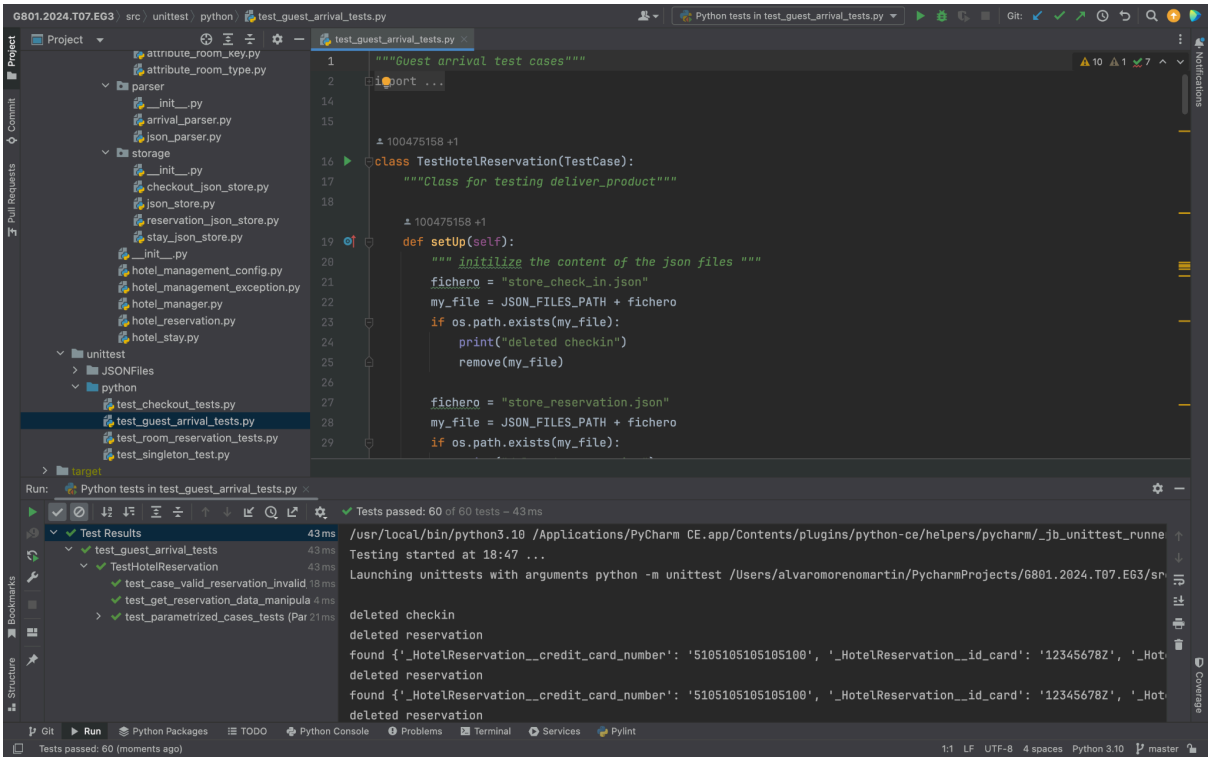
4. Conclusión

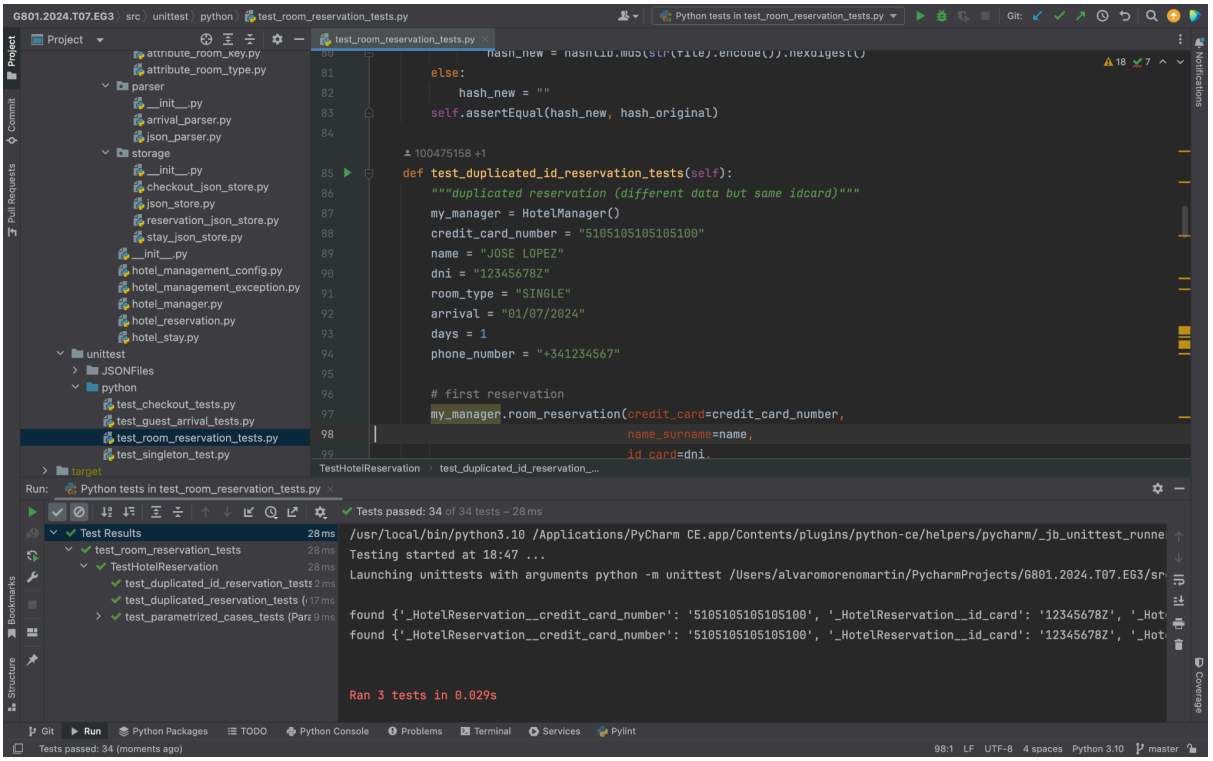
Este proyecto ha mejorado significativamente la calidad del código fuente y nos ha proporcionado habilidades en refactoring y diseño de software, fomentando un código más limpio, mantenible y escalable. Es cierto que el proceso no ha sido fácil y además ha sido muy laborioso, no obstante el equipo está contento con el resultado final después de las muchas horas dedicadas a que todo salga adelante.

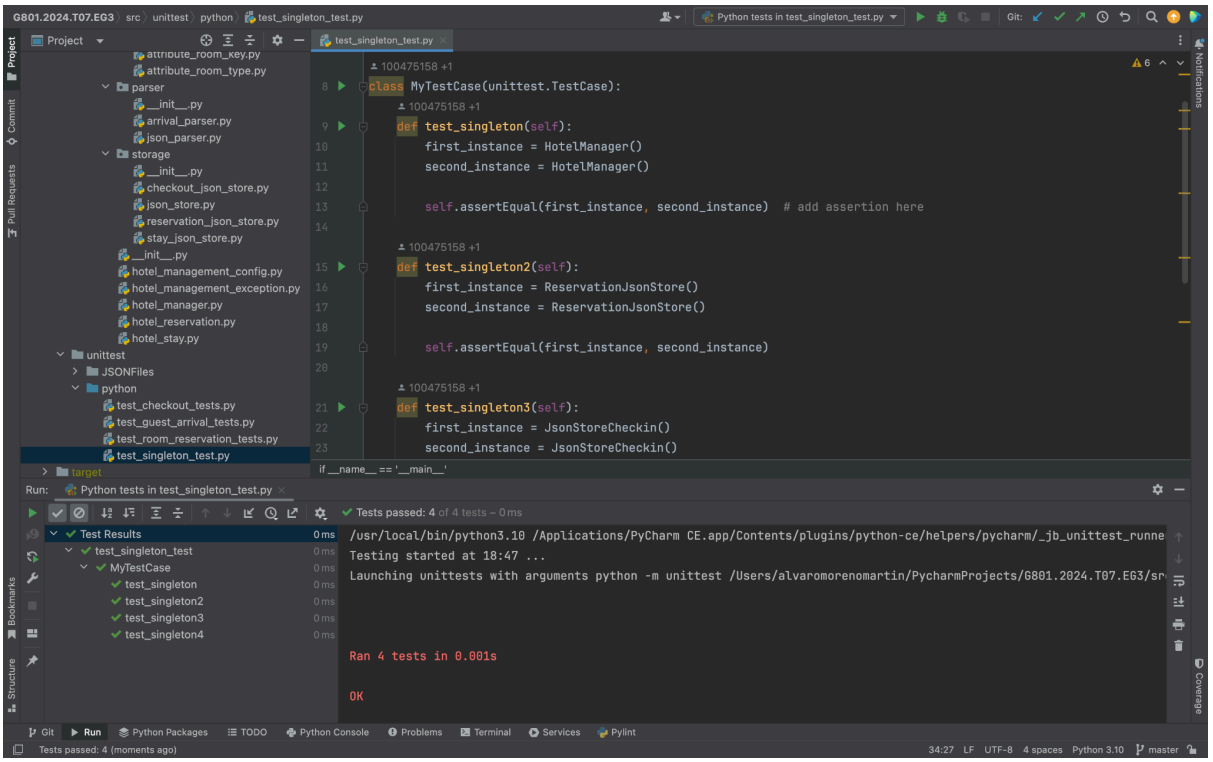
ANEXO I:

Tests ejecutados y superados:

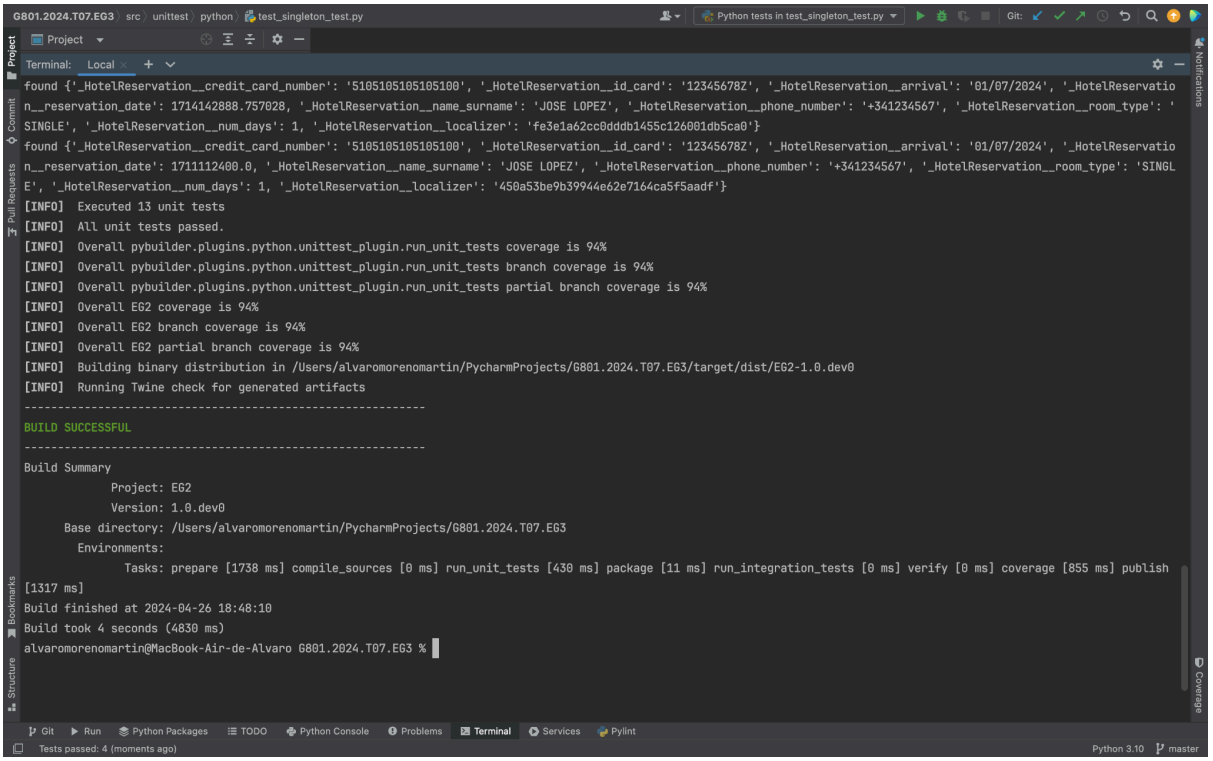





else:
hash_new = ""
self.assertEqual(hash_new, hash_original)
def test_duplicated_id_reservation_tests(self):
"""duplicated reservation (different data but same idcard)"""
my_manager = HotelManager()
credit_card_number = "5105105105105100"
name = "JOSE LOPEZ"
dni = "12345678Z"
room_type = "SINGLE"
arrival = "01/07/2024"
days = 1
phone_number = "+341234567"
# first reservation
my_manager.room_reservation(credit_card=credit_card_number,
name_surname=name,
id_card=dni,
Tests passed: 34 of 34 tests – 28 ms
/usr/local/bin/python3.10 /Applications/PyCharm CE.app/Contents/plugins/python-ce/helpers/pycharm/_jb_unittest_runne
Testing started at 18:47 ...
Launching unittests with arguments python -m unittest /Users/alvaromorenomartin/PycharmProjects/G801.2024.T07.EG3/sr
found {'_HotelReservation__credit_card_number': '5105105105105100', '_HotelReservation__id_card': '12345678Z', '_Hot
found {'_HotelReservation__credit_card_number': '5105105105105100', '_HotelReservation__id_card': '12345678Z', '_Hot
Ran 3 tests in 0.029s


class MyTestCase(unittest.TestCase):
def test_singleton(self):
first_instance = HotelManager()
second_instance = HotelManager()
self.assertEqual(first_instance, second_instance) # add assertion here
def test_singleton2(self):
first_instance = ReservationJsonStore()
second_instance = ReservationJsonStore()
self.assertEqual(first_instance, second_instance)
def test_singleton3(self):
first_instance = JsonStoreCheckin()
second_instance = JsonStoreCheckin()
if __name__ == '__main__'
Tests passed: 4 of 4 tests – 0 ms
/usr/local/bin/python3.10 /Applications/PyCharm CE.app/Contents/plugins/python-ce/helpers/pycharm/_jb_unittest_runne
Testing started at 18:47 ...
Launching unittests with arguments python -m unittest /Users/alvaromorenomartin/PycharmProjects/G801.2024.T07.EG3/sr
Ran 4 tests in 0.001s
OK

Pybuilder:



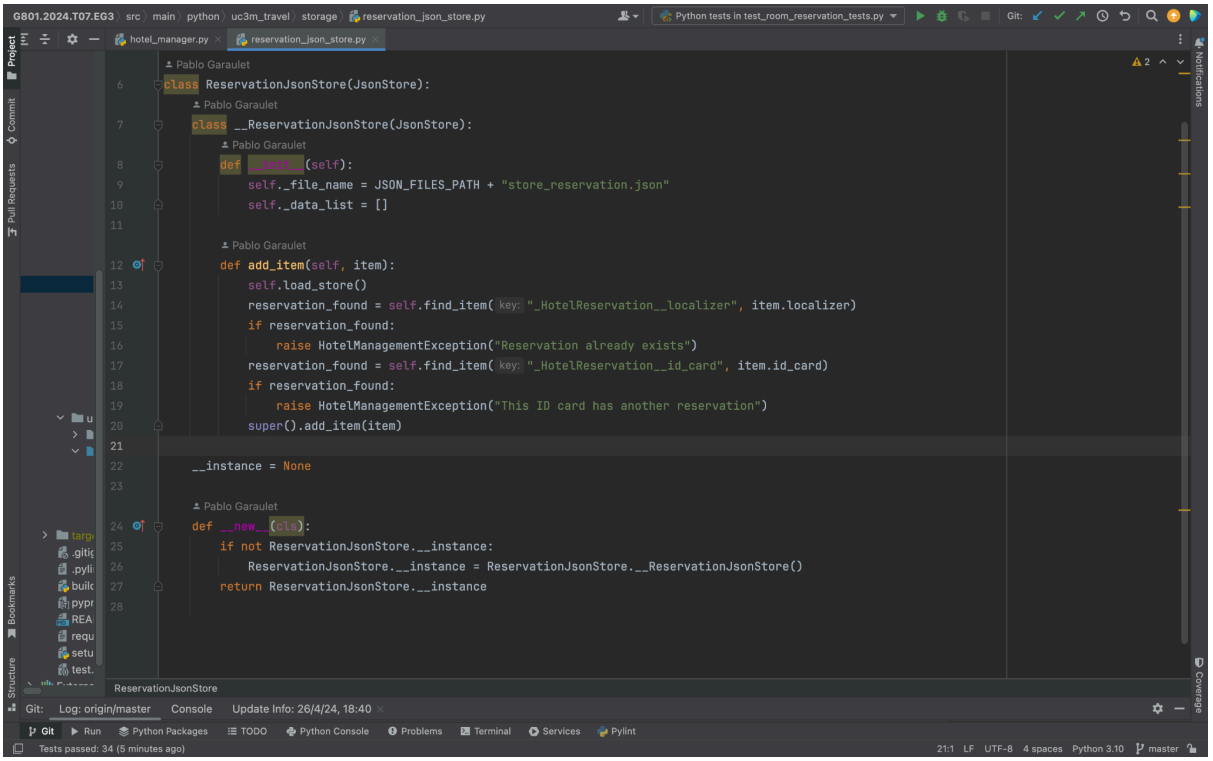
Ejemplos de usos de Extract Method:

En room_reservation:

```
100475158 +1
@staticmethod
def room_reservation(credit_card: str,
                     name_surname: str,
                     id_card: str,
                     phone_number: str,
                     room_type: str,
                     arrival_date: str,
                     num_days: int) -> str:
    """manges the hotel reservation: creates a reservation and saves it into a json file"""

    my_reservation = HotelReservation(id_card=id_card,
                                      credit_card_number=credit_card,
                                      name_surname=name_surname,
                                      phone_number=phone_number,
                                      room_type=room_type,
                                      arrival=arrival_date,
                                      num_days=num_days)
    reservation_store = ReservationJsonStore()
    reservation_store.add_item(my_reservation)
    reservation_store.save_store()

    return my_reservation.localizer
```



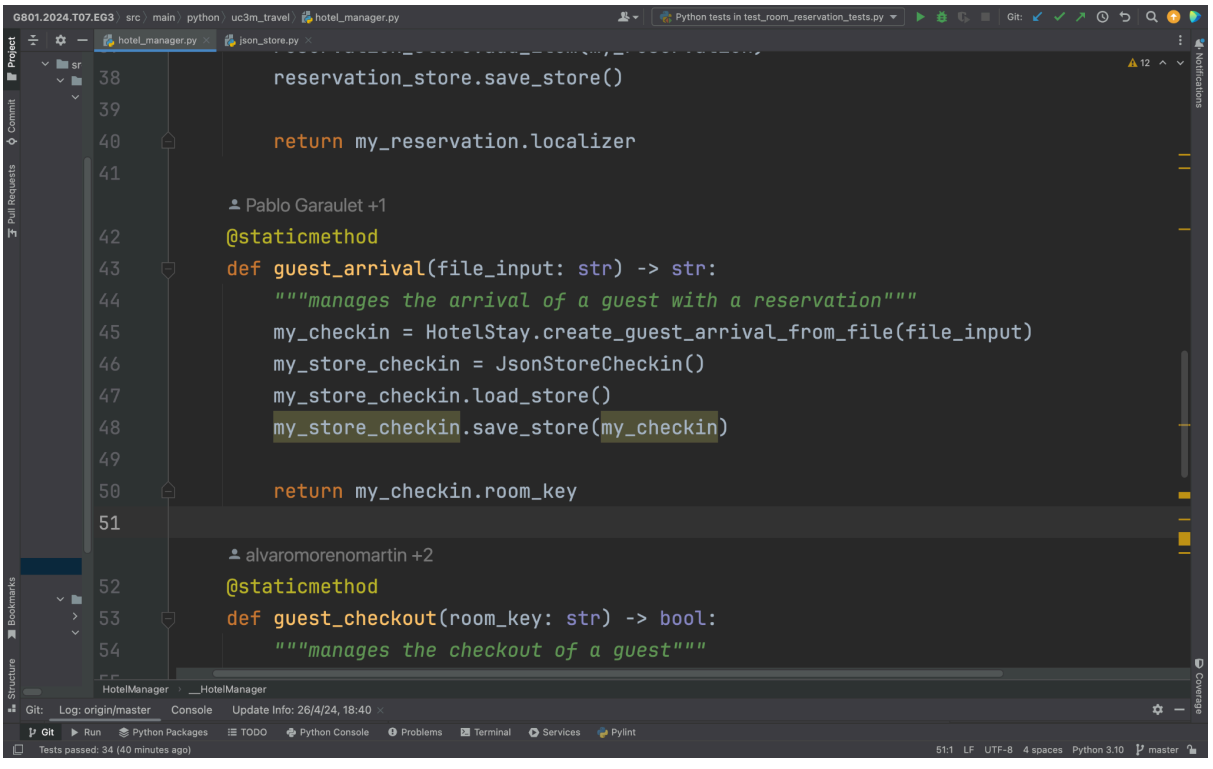
```
class ReservationJsonStore(JsonStore):
    class __ReservationJsonStore(JsonStore):
        def __init__(self):
            self._file_name = JSON_FILES_PATH + "store_reservation.json"
            self._data_list = []

        def add_item(self, item):
            self.load_store()
            reservation_found = self.find_item( key: "_HotelReservation__localizer", item.localizer)
            if reservation_found:
                raise HotelManagementException("Reservation already exists")
            reservation_found = self.find_item( key: "_HotelReservation__id_card", item.id_card)
            if reservation_found:
                raise HotelManagementException("This ID card has another reservation")
            super().add_item(item)

    __instance = None

    def __new__(cls):
        if not ReservationJsonStore.__instance:
            ReservationJsonStore.__instance = ReservationJsonStore.__ReservationJsonStore()
        return ReservationJsonStore.__instance
```

En guest_arrival:



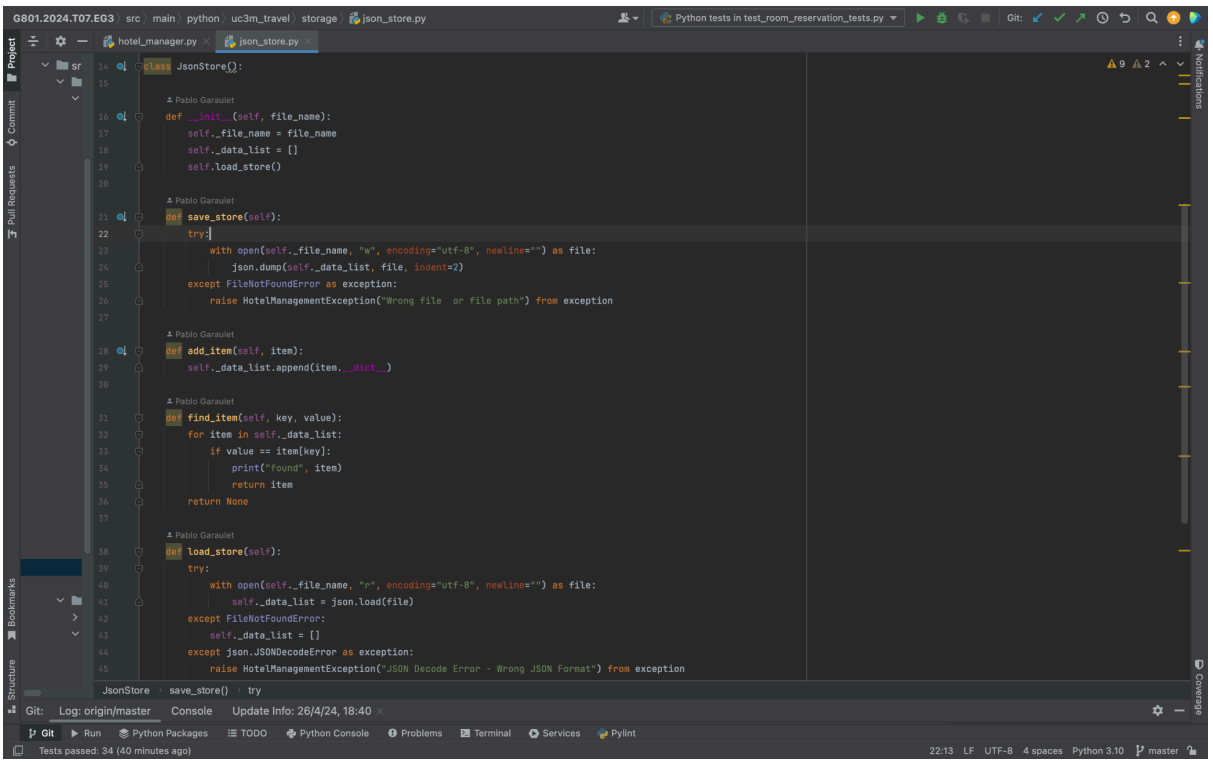
```
        reservation_store.save_store()

        return my_reservation.localizer

    @staticmethod
    def guest_arrival(file_input: str) -> str:
        """manages the arrival of a guest with a reservation"""
        my_checkin = HotelStay.create_guest_arrival_from_file(file_input)
        my_store_checkin = JsonStoreCheckin()
        my_store_checkin.load_store()
        my_store_checkin.save_store(my_checkin)

        return my_checkin.room_key

    @staticmethod
    def guest_checkout(room_key: str) -> bool:
        """manages the checkout of a guest"""
```


G801.2024.T07.EG3 src main python uc3m_travel storage json_store.py
Python tests in test_room_reservation_tests.py
hotel_manager.py
json_store.py
class JsonStore():
Pablo Garaulet
def __init__(self, file_name):
self._file_name = file_name
self._data_list = []
self.load_store()
Pablo Garaulet
def save_store(self):
try:
with open(self._file_name, "w", encoding="utf-8", newline="") as file:
json.dump(self._data_list, file, indent=2)
except FileNotFoundError as exception:
raise HotelManagementException("Wrong file or file path") from exception
Pablo Garaulet
def add_item(self, item):
self._data_list.append(item.__dict__)
Pablo Garaulet
def find_item(self, key, value):
for item in self._data_list:
if value == item[key]:
print("found", item)
return item
return None
Pablo Garaulet
def load_store(self):
try:
with open(self._file_name, "r", encoding="utf-8", newline="") as file:
self._data_list = json.load(file)
except FileNotFoundError:
self._data_list = []
except json.JSONDecodeError as exception:
raise HotelManagementException("JSON Decode Error - Wrong JSON Format") from exception
JsonStore save_store() try
Git: Log: origin/master Console Update Info: 26/4/24, 18:40
Git Run Python Packages TODO Python Console Problems Terminal Services Pylint
Tests passed: 34 (40 minutes ago)
22:13 LF UTF-8 4 spaces Python 3.10 master