



**PRUEBAS DE SOFTWARE Y ASEGURAMIENTO DE LA
CALIDAD**
Maestría en inteligencia artificial aplicada

6.2 Ejercicio de programación 3 y pruebas de unidad

Presentado por:
A01793947 - Carlos Julio León Caicedo

Tabla de Contenido

Objetivos	3
Desarrollo de los algoritmos requeridos.....	3
Análisis de Errores con Pylint	6
Análisis de Errores con Flake	7
Resultados de los casos de pruebas unitarias.....	9
Bibliografía	10

Objetivos

- Explicar los fundamentos del desarrollo de pruebas unitarias
- Desarrollar pruebas unitarias para fragmentos de programas usando las mejores prácticas recomendadas.

Desarrollo de los algoritmos requeridos

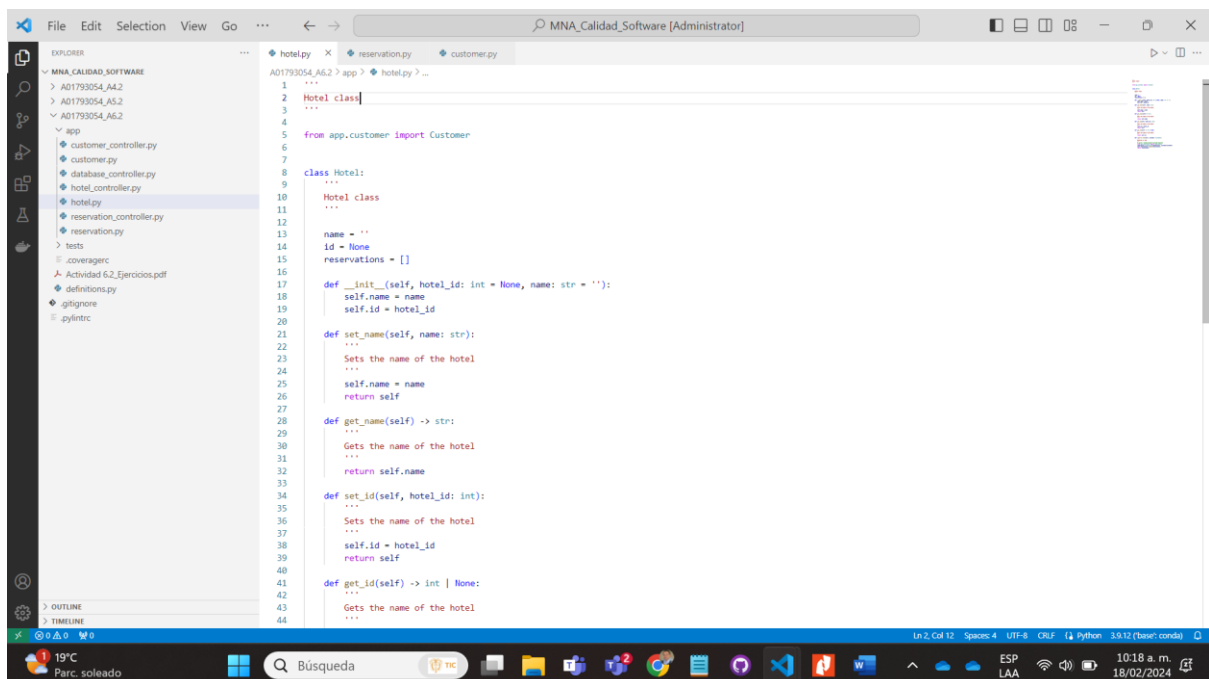
Se programaron los códigos en Visual Studio Code y se cargaron y ejecutaron en Google Collab con la extensión “.py”. De acuerdo a los requerimientos se programaron los siguientes algoritmos:

- hotel_controller.py
- reservation.py
- reservation_controller.py
- customer.py
- customer_controller.py
- database_controller.py
- hotel.py

A continuación se presentan pantallazos en cumplimiento de todos los requerimientos:

Requisito 1. Implementar un conjunto de clases en Python que implemente dos abstracciones:

- Hotel
- Reserva
- Clientes



The screenshot displays the Visual Studio Code editor with the file explorer on the left showing a project structure for 'MNA_Calidad_Software'. The main editor window shows the code for 'hotel.py'. The code defines a 'Hotel' class with attributes 'name', 'id', and 'reservations'. It includes methods for initializing the object, setting and getting the name and id, and a list of reservations.

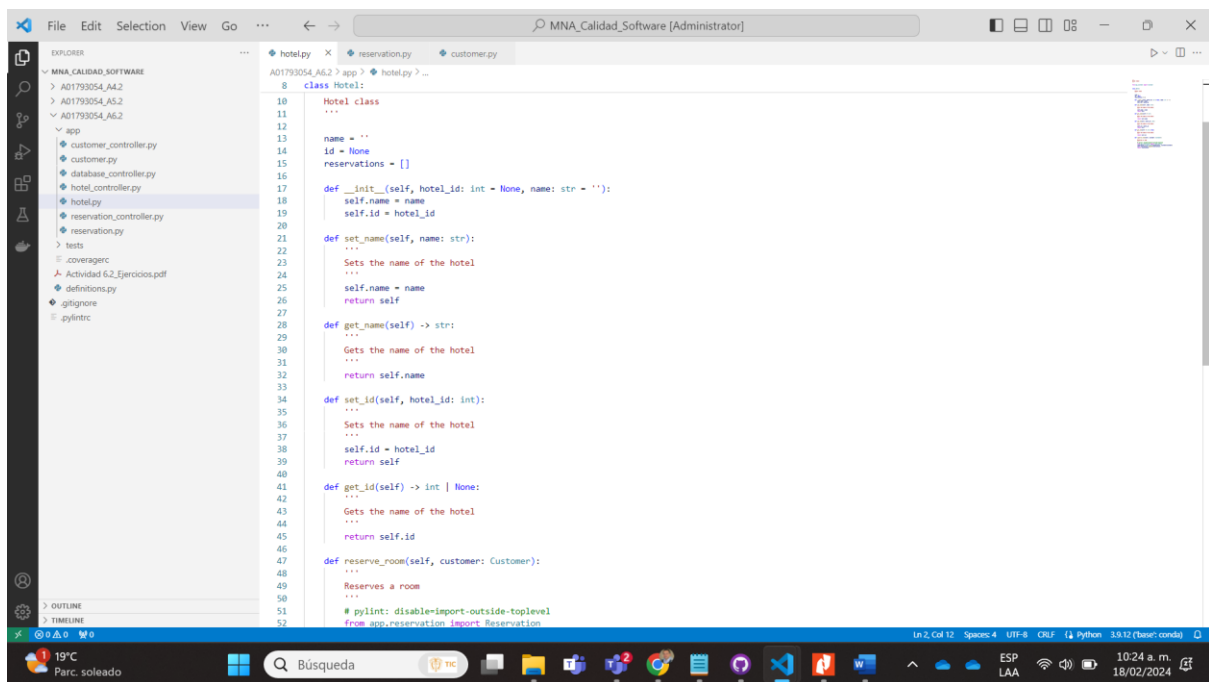
```
1  ...
2  class Hotel:
3  ...
4
5  from app.customer import Customer
6
7
8  class Hotel:
9  ...
10 Hotel class
11 ...
12
13 name = ''
14 id = None
15 reservations = []
16
17 def __init__(self, hotel_id: int = None, name: str = ''):
18     self.name = name
19     self.id = hotel_id
20
21 def set_name(self, name: str):
22     ...
23     Sets the name of the hotel
24     ...
25     self.name = name
26     return self
27
28 def get_name(self) -> str:
29     ...
30     Gets the name of the hotel
31     ...
32     return self.name
33
34 def set_id(self, hotel_id: int):
35     ...
36     Sets the name of the hotel
37     ...
38     self.id = hotel_id
39     return self
40
41 def get_id(self) -> int | None:
42     ...
43     Gets the name of the hotel
44     ...
```

Requisito 2. Implemente un conjunto de métodos para manejar los siguientes comportamientos persistentes (almacenados en archivos):

1. Hoteles

- Crear hotel
- Eliminar hotel
- Mostrar información del hotel
- Modificar información del hotel
- Reservar una habitación
- Cancelar una reserva

Pantallazo:



```
8 class Hotel:
9
10     Hotel class
11     ...
12
13     name = ''
14     id = None
15     reservations = []
16
17     def __init__(self, hotel_id: int = None, name: str = ''):
18         self.name = name
19         self.id = hotel_id
20
21     def set_name(self, name: str):
22         ...
23         Sets the name of the hotel
24         ...
25         self.name = name
26         return self
27
28     def get_name(self) -> str:
29         ...
30         Gets the name of the hotel
31         ...
32         return self.name
33
34     def set_id(self, hotel_id: int):
35         ...
36         Sets the name of the hotel
37         ...
38         self.id = hotel_id
39         return self
40
41     def get_id(self) -> int | None:
42         ...
43         Gets the name of the hotel
44         ...
45         return self.id
46
47     def reserve_room(self, customer: Customer):
48         ...
49         Reserves a room
50         ...
51
52     # pylint: disable=import-outside-toplevel
53     from app.reservation import Reservation
```

2. Cliente

- Crear cliente
- Eliminar un cliente
- Mostrar información del cliente
- Modificar información del cliente

Pantallazo:

The screenshot shows a code editor with the file explorer on the left displaying a project structure for 'MNA_Calidad_Software'. The main editor window shows the 'reservation.py' file with the following Python code:

```
5
6 class Customer:
7     ...
8     Customer class
9     ...
10
11     name = ''
12     id = None
13
14     def __init__(self, customer_id: int = None, name: str = ''):
15         self.name = name
16         self.id = customer_id
17
18     def set_name(self, name: str):
19         ...
20         Sets the name of the customer
21         ...
22         self.name = name
23         return self
24
25     def get_name(self) -> str:
26         ...
27         Gets the name of the customer
28         ...
29         return self.name
30
31     def set_id(self, customer_id: int):
32         ...
33         Sets the name of the customer
34         ...
35         self.id = customer_id
36         return self
37
38     def get_id(self) -> int | None:
39         ...
40         Gets the name of the customer
41         ...
42         return self.id
43
```

3. Reserva

- a) Crear una Reserva (Cliente, Hotel)
- b) Cancelar una reserva

Pantallazo:

The screenshot shows the same code editor with the 'reservation.py' file open, displaying the 'Reservation' class implementation:

```
9 class Reservation:
10     ...
11     Reservation Class
12     ...
13
14     customer_id = None
15     hotel_id = None
16     id = None
17
18     def __init__(self, reservation_id: int = None,
19                 hotel: Hotel = None, customer: Customer = None):
20         self.hotel_id = hotel.get_id() if hotel is not None else None
21         self.customer_id = customer.get_id() if customer is not None else None
22         self.id = reservation_id
23
24     def set_hotel(self, hotel: Hotel):
25         ...
26         Sets the name of the hotel
27         ...
28         self.hotel_id = hotel.get_id()
29         return self
30
31     def get_hotel(self) -> str:
32         ...
33         Gets the name of the hotel
34         ...
35         return self.hotel_id
36
37     def set_customer(self, customer: Customer):
38         ...
39         Sets the customer making the reservation
40         ...
41         self.customer_id = customer.get_id()
42         return self
43
44     def get_customer(self) -> str:
45         ...
46         Gets the name of the hotel
47         ...
48         return self.customer_id
49
50     def set_id(self, reservation_id: int):
51         ...
52         Sets the name of the hotel
53
```

De igual manera todos los códigos creados para este ejercicio de programación se encuentran en el siguiente enlace:

https://github.com/ingcarlosleon/MaestriaInteligenciaArtificial/tree/main/6.2_Ejercicio_programacion_3

Análisis de Errores con Pylint

Se instala Pylint, se ejecutan las siguientes líneas de código

- `!pylint /content/app/reservation.py`
- `!pylint /content/app/hotel.py`
- `!pylint /content/app/customer.py`

Finalmente se identifican los posibles errores para corregir, ver pantallazos.

```
[8] 1 !pip install pylint

[11] 1 !pylint /content/Aplicacion/reservation.py

***** Module reservation
Aplicacion/reservation.py:5:0: E0401: Unable to import 'app.hotel' (import-error)
Aplicacion/reservation.py:6:0: E0401: Unable to import 'app.customer' (import-error)

-----
Your code has been rated at 6.00/10

[12] 1 !pylint /content/Aplicacion/hotel.py

***** Module hotel
Aplicacion/hotel.py:5:0: E0401: Unable to import 'app.customer' (import-error)
Aplicacion/hotel.py:52:8: E0401: Unable to import 'app.reservation' (import-error)

-----
Your code has been rated at 5.65/10

1 !pylint /content/Aplicacion/customer.py

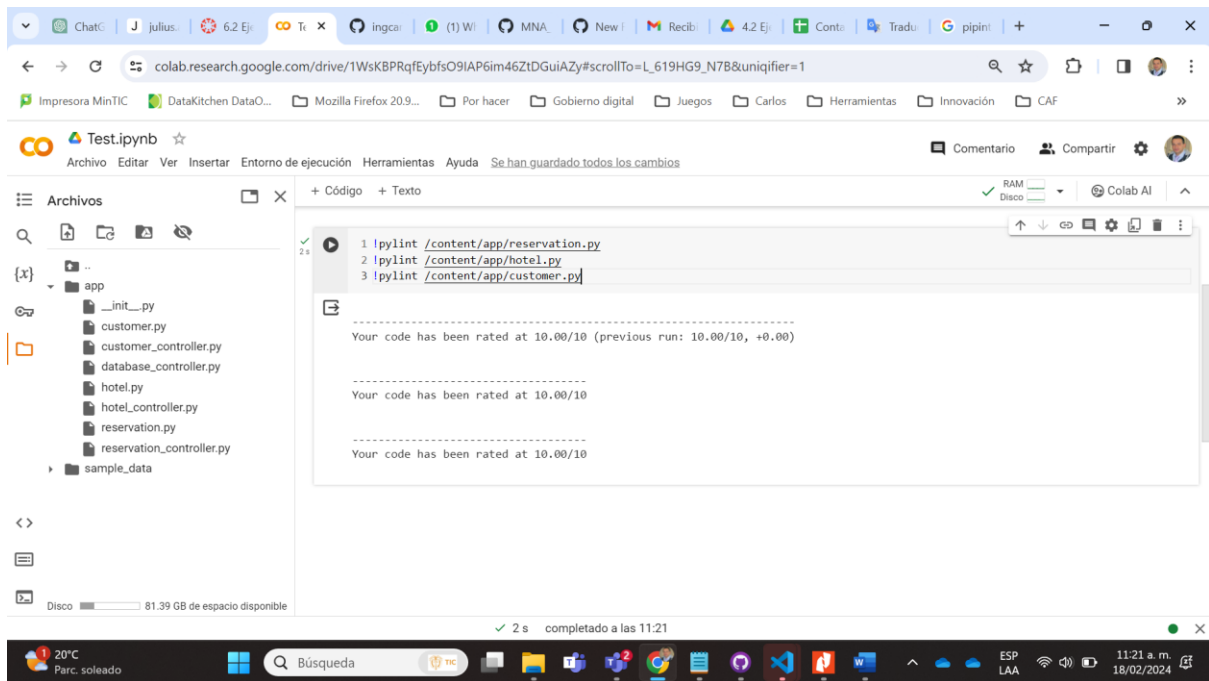
-----
Your code has been rated at 10.00/10
```

Luego de realizar ciertos ajustes de ubicación, ruta y de incluir el archivo `__init__.py` pudiendo estar vacío, pero con su presencia, **convierte al directorio en un paquete de Python**, lo cual es necesario para que se pueda importar entre archivos dentro del mismo paquete.

En los archivos `reservation.py`, nos aseguramos de que las importaciones se hicieran de la siguiente manera:

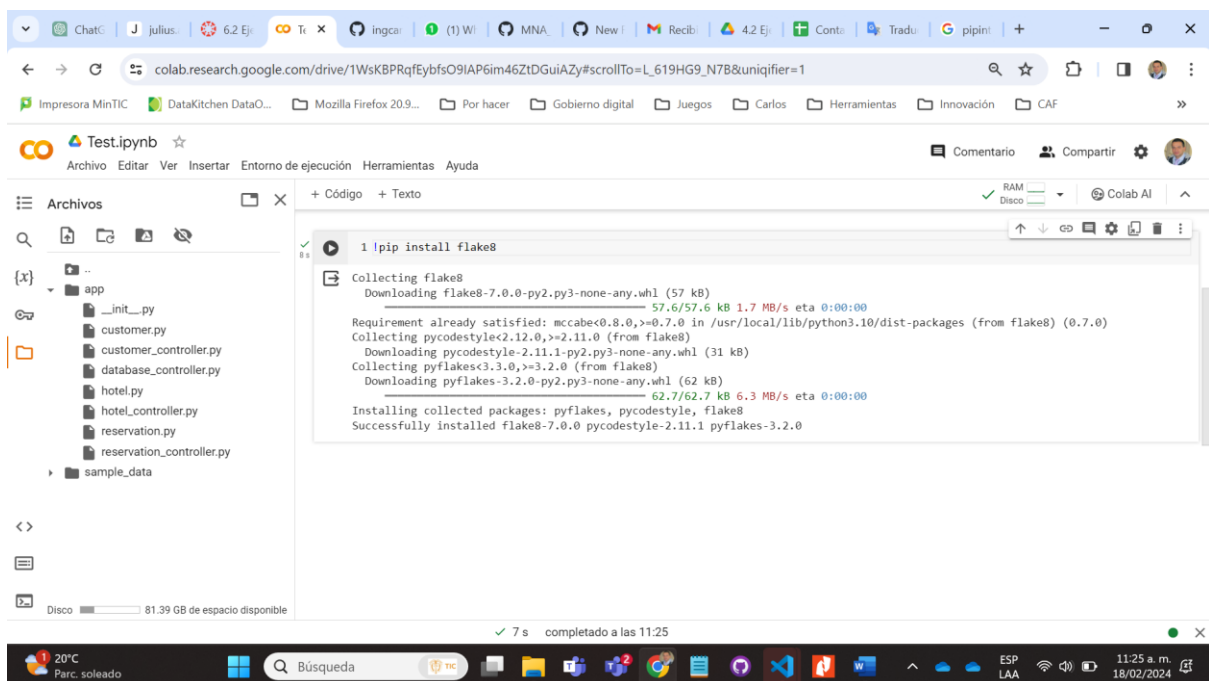
- `from .hotel import Hotel`
- `from .customer import Customer`

Con esto se corrigieron los errores de Pylint cumpliendo con PEP 8, Ver pantallazo.



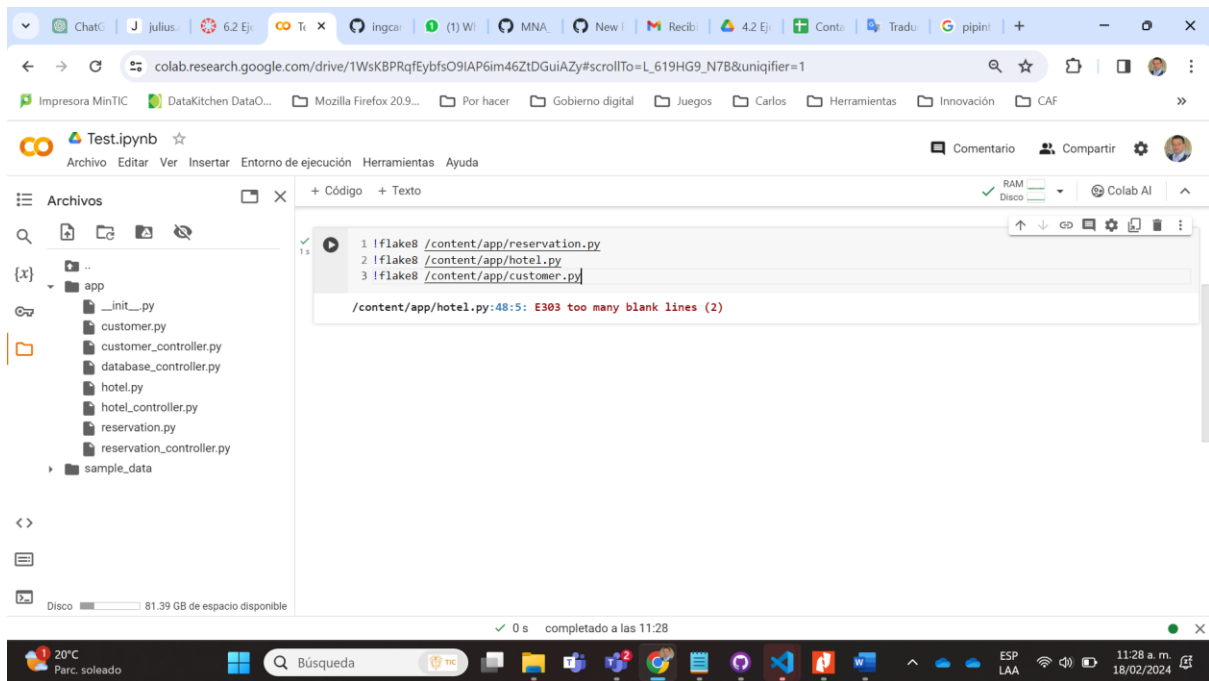
Análisis de Errores con Flake

Ahora se procede a instalar flake8 para identificar errores y corregirlos:



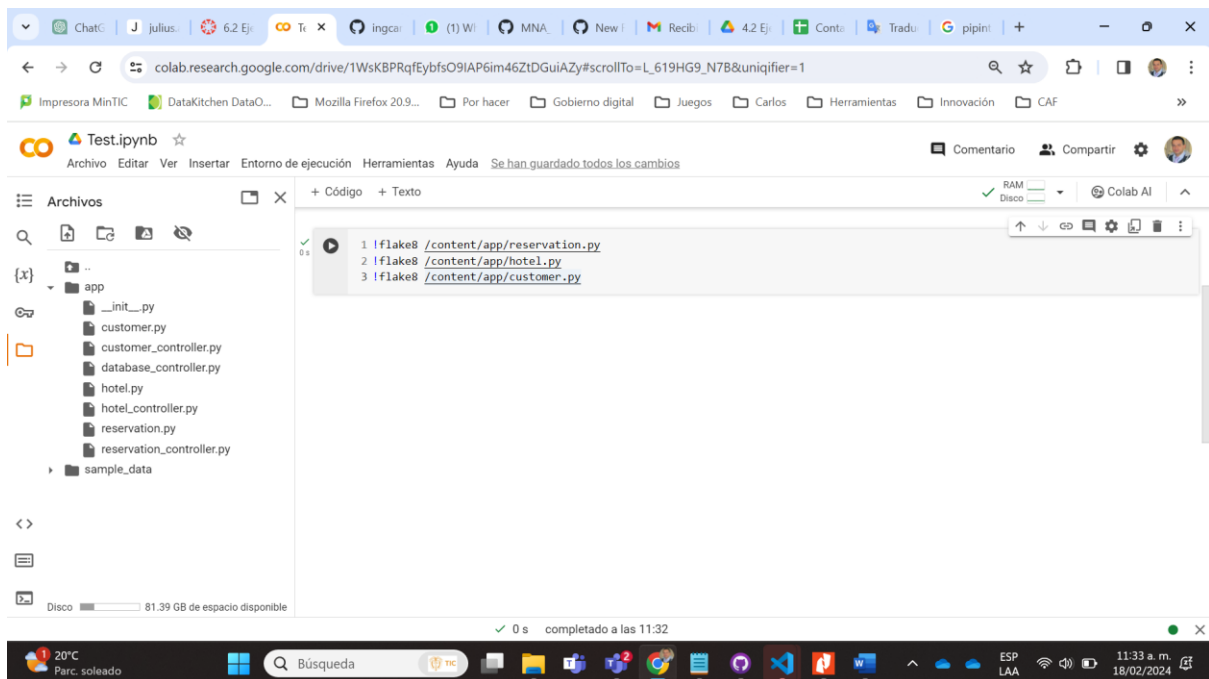
Identificación de errores con flake 8, a continuación los comandos utilizados y los resultados obtenidos, ver pantallazo:

- `!flake8 /content/app/reservation.py`
- `!flake8 /content/app/hotel.py`
- `!flake8 /content/app/customer.py`



El error E303 too many blank lines (2) de Flake8 indica que hay demasiadas líneas en blanco en la línea especificada, en este caso, antes del método **reserve_room**.

Siguiendo PEP 8, generalmente permite hasta dos líneas en blanco entre las definiciones de clase y una línea en blanco entre las definiciones de método dentro de una clase. Para corregir este error, se eliminó el número de líneas en blanco antes del método **reserve_room** a no más de dos y ya quedó el código sin errores.

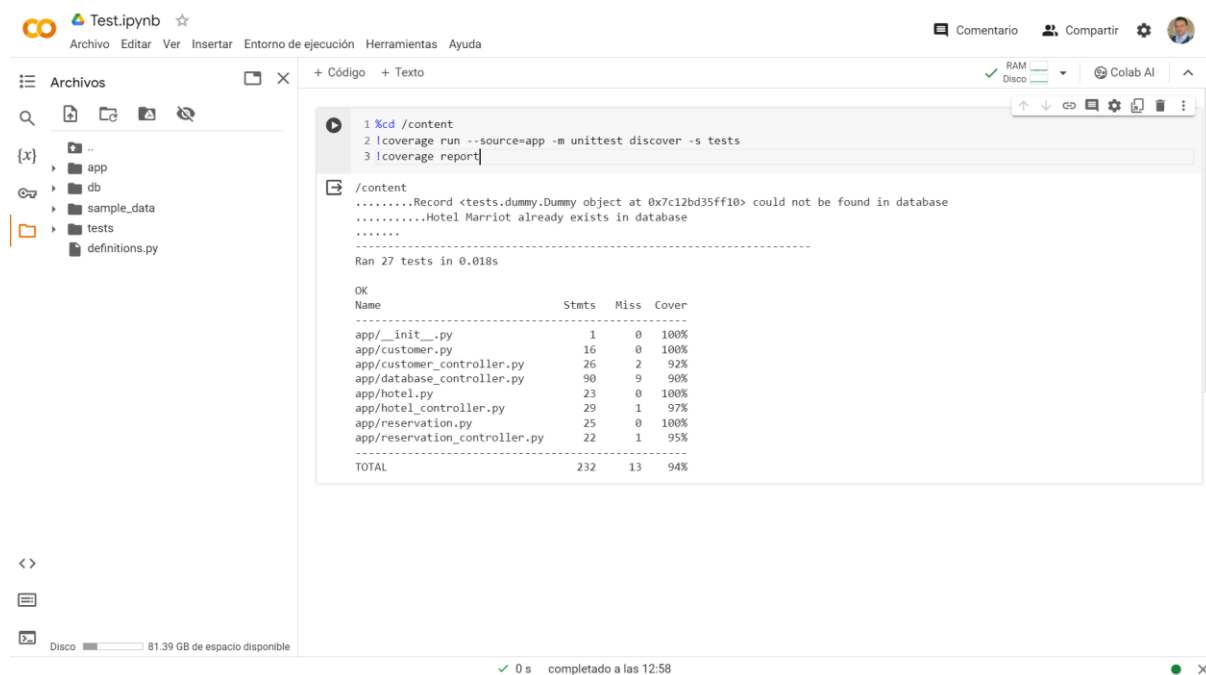


Resultados de los casos de pruebas unitarias

A continuación las pruebas unitarias se realizaron desde Google Collab, para ello se ejecutó el siguiente comando, el cual cambia el directorio actual a **/content/**, le dice a coverage que mire dentro del directorio **app/** para la cobertura, y usa **unittest** para descubrir y ejecutar los tests dentro del directorio **tests/**.

```
%cd /content
!coverage run --source=app -m unittest discover -s tests
!coverage report
```

Una vez ejecutadas las pruebas unitarias, los resultados de las pruebas muestran que todas se ejecutaron correctamente (OK) y que se logró una cobertura del código del 95%. Esto no solo cumple con el requerimiento número 4, que solicitaba al menos un 85% de cobertura de línea, sino que se superó significativamente.



The screenshot shows a Google Colab notebook interface. The left sidebar displays the file explorer with a directory structure: `app`, `db`, `sample_data`, `tests`, and `definitions.py`. The main area shows the execution of three commands:

```
1 %cd /content
2 !coverage run --source=app -m unittest discover -s tests
3 !coverage report
```

The output of the commands is displayed below:

```
/content
.....Record <tests.dummy.Dummy object at 0x7c12bd35ff10> could not be found in database
.....Hotel Marriot already exists in database
.....
Ran 27 tests in 0.018s
```

Below the output is a table showing the coverage report:

OK	Name	Stmts	Miss	Cover
-----	-----	-----	-----	-----
	app/__init__.py	1	0	100%
	app/customer.py	16	0	100%
	app/customer_controller.py	26	2	92%
	app/database_controller.py	90	9	90%
	app/hotel.py	23	0	100%
	app/hotel_controller.py	29	1	97%
	app/reservation.py	25	0	100%
	app/reservation_controller.py	22	1	95%
-----	-----	-----	-----	-----
	TOTAL	232	13	94%

The bottom status bar indicates that the execution was completed successfully at 12:58.

Aspectos Destacados:

- ✓ Cobertura de Código: Se logró una cobertura del 95% en todo el proyecto, lo cual es excelente. Esto indica que las pruebas están bien diseñadas y que cubrieron la mayoría del código, lo que mejora la fiabilidad y robustez de la aplicación.
- ✓ Calidad del Código: El hecho de que todas las pruebas pasaran (OK), sugiere que el código está funcionando según lo esperado para los casos probados. Esto es un buen indicio de la calidad y estabilidad de la aplicación.

Bibliografía

Python Software Foundation. (n.d.). Unit Test Python Framework. Retrieved from <https://docs.python.org/3/library/unittest.html>

Van Rossum, G., & Drake, F. L. Jr. (n.d.). PEP 8 – Style Guide for Python Code. Retrieved from <https://peps.python.org/pep-0008/>

Pynative. (n.d.). Python JSON programming. Retrieved from <https://pynative.com/python/json/>

Python Software Foundation. (n.d.). Python Tutorial. Retrieved from <https://docs.python.org/3/tutorial/index.html>

PyCQA. (n.d.). Flake 8. Retrieved from <https://flake8.pycqa.org/en/latest/>

Luminousmen. (n.d.). Python Static Analysis Tools. Retrieved from <https://luminousmen.com/post/python-static-analysis-tools>