Ficha 11 – Testing CRUD

GITHUB REPO: https://github.com/Diego-Alonso-05/Post_Office.git

(C)reate

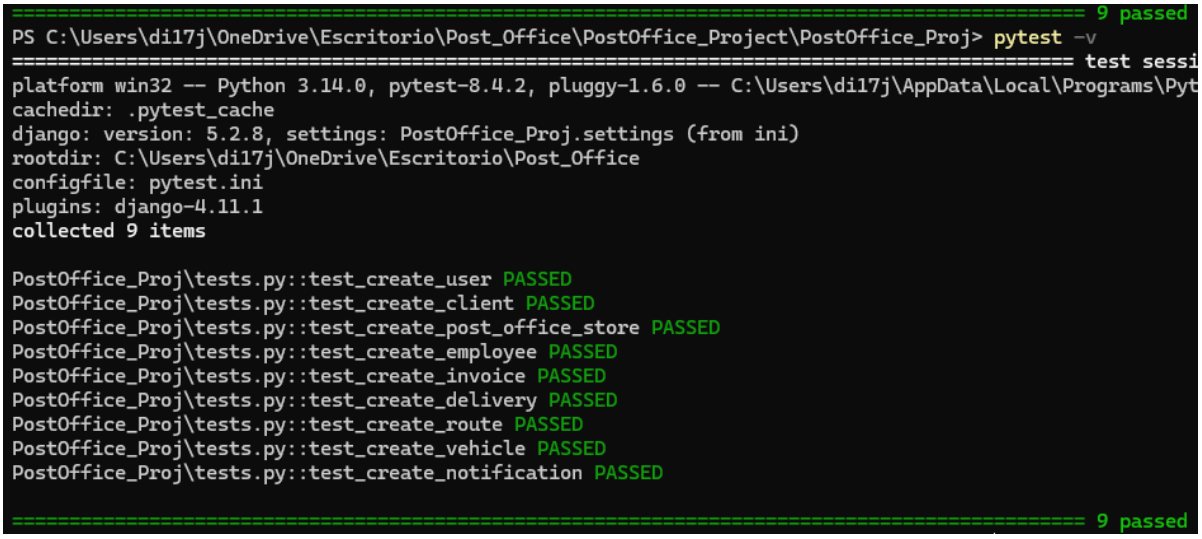
```
import pytest
from datetime import datetime, timezone
from pymongo import MongoClient

client = MongoClient("mongodb://localhost:27017/")
db = client["post_office_db"]

@pytest.fixture(autouse=True)
def clear_collections():
    for col in db.list_collection_names():
        db[col].delete_many({})
    yield

@pytest.mark.mongodb
def test_create_user():
    user = {
        "username": "johndoe",
        "psswd_hash": "hashed123",
        "name": "John Doe",
        "contact": "600111222",
        "address": "Rua Central 10",
        "email": "john@example.com",
        "created_at": datetime.now(timezone.utc),
        "updated_at": datetime.now(timezone.utc),
        "role": "CLIENT"
    }

    result = db.users.insert_one(user)
    assert result.inserted_id is not None
    assert db.users.count_documents({}) == 1

@pytest.mark.mongodb
def test_create_client():
    client_data = {
        "use_id_user": 1,
        "username": "client1",
```

```
        "psswd_hash": "hashcli",
        "name": "Client One",
        "contact": "600123456",
        "address": "Rua Verde 23",
        "email": "client@test.com",
        "tax_id": "123456789",
        "created_at": datetime.now(timezone.utc),
        "updated_at": datetime.now(timezone.utc),
        "role": "CLIENT"
    }

    result = db.clients.insert_one(client_data)
    assert result.inserted_id is not None
    assert db.clients.count_documents({}) == 1

@pytest.mark.mongodb
def test_create_post_office_store():
    store = {
        "use_id_user": 1,
        "id_user": 1,
        "name": "Central Store",
        "contact": "700888999",
        "address": "Rua Nova 12",
        "opening_time": "08:00",
        "closing_time": "18:00",
        "po_schedule": "Mon-Fri",
        "maximum_storage": 1000
    }

    result = db.post_office_stores.insert_one(store)
    assert result.inserted_id is not None
    assert db.post_office_stores.count_documents({}) == 1

@pytest.mark.mongodb
def test_create_employee():
    employee = {
        "use_id_user": 1,
        "id_user": 2,
        "id_postoffice_store": 1,
        "username": "emp1",
        "psswd_hash": "hash123",
        "name": "Maria Silva",
        "contact": "622222222",
        "address": "Rua da Luz 40",
        "email": "maria@office.com",
```

```
        "role": "EMPLOYEE",
        "position": "Driver",
        "schedule": "Mon-Fri",
        "wage": 1500.0,
        "is_active": True,
        "hire_date": datetime(2024, 5, 1,
tzinfo=timezone.utc)
    }

    result = db.employees.insert_one(employee)
    assert result.inserted_id is not None
    assert db.employees.count_documents({}) == 1

@pytest.mark.mongodb
def test_create_invoice():
    invoice = {
        "id_postoffice_store": 1,
        "use_id_user": 1,
        "emp_use_id_user": 1,
        "emp_id_user": 2,
        "id_user": 3,
        "cli_id_user": 4,
        "cost": 50.0,
        "invoice_datetime": datetime(2025, 11, 8,
tzinfo=timezone.utc),
        "invoice_status": "PAID"
    }

    result = db.invoices.insert_one(invoice)
    assert result.inserted_id is not None
    assert db.invoices.count_documents({}) == 1

@pytest.mark.mongodb
def test_create_delivery():
    delivery = {
        "use_id_user": 1,
        "id_user": 1,
        "id_invoice": 1,
        "status": "IN_TRANSIT",
        "recipient_address": "Rua Lisboa 45",
        "description": "Fragile package",
        "registered_at": datetime.now(timezone.utc)
    }
```

```
    result = db.deliveries.insert_one(delivery)
    assert result.inserted_id is not None
    assert db.deliveries.count_documents({}) == 1

@pytest.mark.mongodb
def test_create_route():
    route = {
        "use_id_user": 1,
        "emp_id_user": 2,
        "id_user": 1,
        "id_delivery": 10,
        "id_postoffice_store": 5,
        "description": "Daily route Lisbon-Porto",
        "delivery_status": "IN_PROGRESS",
        "delivery_date": datetime(2025, 11, 8,
tzinfo=timezone.utc),
        "kms_travelled": 312.5,
        "driver_notes": "No issues during trip"
    }

    result = db.routes.insert_one(route)
    assert result.inserted_id is not None
    assert db.routes.count_documents({}) == 1

@pytest.mark.mongodb
def test_create_vehicle():
    vehicle = {
        "id_route": 10,
        "vehicle_type": "Truck",
        "plate_number": "AA-11-BB",
        "capacity": 800.0,
        "brand": "Mercedes",
        "model": "Sprinter",
        "vehicle_status": "AVAILABLE",
        "year": 2022,
        "fuel_type": "Diesel",
        "last_maintenance_date": datetime(2025, 1, 15,
tzinfo=timezone.utc)
    }

    result = db.vehicles.insert_one(vehicle)
    assert result.inserted_id is not None
    assert db.vehicles.count_documents({}) == 1

@pytest.mark.mongodb
```

```python
def test_create_notification():
    notification = {
        "id_delivery": 5,
        "notification_type": "EMAIL",
        "recipient_contact": "client@test.com",
        "subject": "Delivery completed",
        "message": "Your delivery has been successfully
completed.",
        "status": "SENT",
        "created_at": datetime.now(timezone.utc)
    }

    result = db.notifications.insert_one(notification)
    assert result.inserted_id is not None
    assert db.notifications.count_documents({}) == 1
```



(R)ead

```
import pytest
from pymongo import MongoClient

client = MongoClient("mongodb://localhost:27017/")
db = client["post_office_db"]

@pytest.fixture(autouse=True)
def setup_sample_data():
    """Crea algunos datos básicos en Mongo antes de cada
test"""
    for col in db.list_collection_names():
        db[col].delete_many({})

    db.users.insert_many([
        {"username": "user1", "psswd_hash": "123", "role":
"CLIENT"},
        {"username": "user2", "psswd_hash": "456", "role":
"EMPLOYEE"}
    ])

    db.clients.insert_many([
        {"username": "client1", "email": "client1@test.com"},
        {"username": "client2", "email": "client2@test.com"}
    ])

    db.employees.insert_many([
        {"username": "emp1", "position": "Driver"},
        {"username": "emp2", "position": "Manager"}
    ])

    db.vehicles.insert_many([
        {"plate_number": "AA-11-BB", "vehicle_type":
"Truck"},
        {"plate_number": "CC-22-DD", "vehicle_type": "Van"}
    ])

    db.routes.insert_many([
        {"description": "North Route", "kms_travelled": 250},
        {"description": "South Route", "kms_travelled": 300}
    ])

    db.deliveries.insert_many([
        {"status": "IN_TRANSIT", "recipient_address": "Rua
```

```
Verde 23"},
        {"status": "DELIVERED", "recipient_address": "Rua
Azul 45"}
    ])

    db.invoices.insert_many([
        {"cost": 50.0, "invoice_status": "PAID"},
        {"cost": 30.0, "invoice_status": "PENDING"}
    ])

    db.stores.insert_many([
        {"name": "Central Store", "maximum_storage": 1000},
        {"name": "Mini Store", "maximum_storage": 200}
    ])

    yield

@pytest.mark.mongodb
def test_get_all_users():
    users = list(db.users.find({}))
    assert len(users) == 2
    assert users[0]["username"] == "user1"

@pytest.mark.mongodb
def test_get_all_clients():
    clients = list(db.clients.find({}))
    assert len(clients) == 2
    assert any(c["email"] == "client1@test.com" for c in
clients)

@pytest.mark.mongodb
def test_get_all_employees():
    employees = list(db.employees.find({}))
    assert len(employees) == 2
    assert employees[0]["position"] in ["Driver", "Manager"]

@pytest.mark.mongodb
def test_get_all_vehicles():
    vehicles = list(db.vehicles.find({}))
    assert len(vehicles) == 2
    assert any(v["vehicle_type"] == "Truck" for v in
vehicles)

@pytest.mark.mongodb
```

```python
def test_get_all_routes():
    routes = list(db.routes.find({}))
    assert len(routes) == 2
    assert routes[0]["description"] in ["North Route", "South Route"]

@pytest.mark.mongodb
def test_get_all_deliveries():
    deliveries = list(db.deliveries.find({}))
    assert len(deliveries) == 2
    assert any(d["status"] == "IN_TRANSIT" for d in deliveries)

@pytest.mark.mongodb
def test_get_all_invoices():
    invoices = list(db.invoices.find({}))
    assert len(invoices) == 2
    assert any(i["invoice_status"] == "PAID" for i in invoices)

@pytest.mark.mongodb
def test_get_all_stores():
    stores = list(db.stores.find({}))
    assert len(stores) == 2
    assert any(s["name"] == "Central Store" for s in stores)
```

```
C:\Users\qubit\Desktop\BD2\PostOffice\PostOffice_Project\PostOffice_Proj>pytest -s -v -rs ..\Tests\R_test_David.py
========================================================================================================
========= test session starts ==========================================================================
========================================
platform win32 -- Python 3.13.2, pytest-9.0.0, pluggy-1.6.0 -- C:\Python313\python.exe
cachedir: .pytest_cache
django: version: 5.2.7, settings: PostOffice_Proj.settings (from ini)
rootdir: C:\Users\qubit\Desktop\BD2\PostOffice
configfile: pytest.ini
plugins: django-4.11.1
collected 8 items



..\Tests\R_test_David.py::test_get_all_users PASSED
..\Tests\R_test_David.py::test_get_all_clients PASSED
..\Tests\R_test_David.py::test_get_all_clients PASSED
..\Tests\R_test_David.py::test_get_all_clients PASSED
..\Tests\R_test_David.py::test_get_all_employees PASSED
..\Tests\R_test_David.py::test_get_all_vehicles PASSED
..\Tests\R_test_David.py::test_get_all_routes PASSED
..\Tests\R_test_David.py::test_get_all_deliveries PASSED
..\Tests\R_test_David.py::test_get_all_invoices PASSED
..\Tests\R_test_David.py::test_get_all_stores PASSED

========================================================================================================
========== 8 passed in 0.59s ===========================================================================
========================================
```

(U)pdate

```
import pytest
from pymongo import MongoClient
from datetime import datetime

# ------------------------------
# Connect to your real MongoDB
# ------------------------------
client = MongoClient("mongodb://localhost:27017")
db = client["postoffice"]

deliveries = db["deliveries"]
notifications = db["notifications"]
postoffice_col = db["postoffice"]
routes = db["routes"]
users = db["users"]
vehicles = db["vehicles"]

# ------------------------------
# Deliveries UPDATE
# ------------------------------
def test_deliveries_update():
    result = deliveries.insert_one({
        "recipient": "Update User",
        "address": "123 Update St",
        "status": "Pending",
        "delivery_date": datetime.now()
    })
    doc_id = result.inserted_id
    print(f"Inserted delivery ID for update: {doc_id}")

    # READ (verify before update)
    doc = deliveries.find_one({"_id": doc_id})
    assert doc is not None
    assert doc["status"] == "Pending"

    # UPDATE
    deliveries.update_one({"_id": doc_id}, {"$set":
{"status": "Delivered"}})
    updated_doc = deliveries.find_one({"_id": doc_id})
    assert updated_doc["status"] == "Delivered"
    print("Delivery update verified.")
```

```
    # CLEANUP
    deliveries.delete_one({"_id": doc_id})
    assert deliveries.find_one({"_id": doc_id}) is None

# ------------------------------
# Notifications UPDATE
# ------------------------------
def test_notifications_update():
    result = notifications.insert_one({
        "title": "Old Notification",
        "message": "Initial message",
        "date": datetime.now()
    })
    doc_id = result.inserted_id
    print(f"Inserted notification ID for update: {doc_id}")

    doc = notifications.find_one({"_id": doc_id})
    assert doc["title"] == "Old Notification"

    # UPDATE
    notifications.update_one({"_id": doc_id}, {"$set":
{"title": "New Notification Title"}})
    updated_doc = notifications.find_one({"_id": doc_id})
    assert updated_doc["title"] == "New Notification Title"
    print("Notification update verified.")

    notifications.delete_one({"_id": doc_id})
    assert notifications.find_one({"_id": doc_id}) is None

# ------------------------------
# Postoffice UPDATE
# ------------------------------
def test_postoffice_update():
    result = postoffice_col.insert_one({
        "name": "Old Postoffice",
        "address": "456 Main St",
        "contact": "555-0000",
        "po_schedule_open": "08:00",
        "po_schedule_close": "17:00",
        "maximum_storage_capacity": 400
    })
    doc_id = result.inserted_id
    print(f"Inserted postoffice ID for update: {doc_id}")

    doc = postoffice_col.find_one({"_id": doc_id})
```

```
    assert doc["maximum_storage_capacity"] == 400

    # UPDATE
    postoffice_col.update_one({"_id": doc_id}, {"$set":
{"maximum_storage_capacity": 800}})
    updated_doc = postoffice_col.find_one({"_id": doc_id})
    assert updated_doc["maximum_storage_capacity"] == 800
    print("Postoffice update verified.")

    postoffice_col.delete_one({"_id": doc_id})
    assert postoffice_col.find_one({"_id": doc_id}) is None

# ------------------------------
# Routes UPDATE
# ------------------------------
def test_routes_update():
    result = routes.insert_one({
        "route_name": "Route A",
        "origin": "City X",
        "destination": "City Y",
        "distance_km": 100
    })
    doc_id = result.inserted_id
    print(f"Inserted route ID for update: {doc_id}")

    doc = routes.find_one({"_id": doc_id})
    assert doc["distance_km"] == 100

    # UPDATE
    routes.update_one({"_id": doc_id}, {"$set":
{"distance_km": 120}})
    updated_doc = routes.find_one({"_id": doc_id})
    assert updated_doc["distance_km"] == 120
    print("Route update verified.")

    routes.delete_one({"_id": doc_id})
    assert routes.find_one({"_id": doc_id}) is None

# ------------------------------
# Users UPDATE
# ------------------------------
def test_users_update():
    result = users.insert_one({
        "username": "user_update",
```

```
        "role": "client",
        "email": "updateuser@example.com",
        "password": "old_password"
    })
    doc_id = result.inserted_id
    print(f"Inserted user ID for update: {doc_id}")

    doc = users.find_one({"_id": doc_id})
    assert doc["role"] == "client"

    # UPDATE
    users.update_one({"_id": doc_id}, {"$set": {"role":
"admin"}})
    updated_doc = users.find_one({"_id": doc_id})
    assert updated_doc["role"] == "admin"
    print("User update verified.")

    users.delete_one({"_id": doc_id})
    assert users.find_one({"_id": doc_id}) is None

# ------------------------------
# Vehicles UPDATE
# ------------------------------
def test_vehicles_update():
    result = vehicles.insert_one({
        "vehicle_type": "Van",
        "plate_number": "UPD123",
        "capacity": 500,
        "brand": "BrandX",
        "model": "Model1",
        "vehicle_status": "Active",
        "year": 2024,
        "fuel_type": "Diesel",
        "last_maintenance_date": datetime.now()
    })
    doc_id = result.inserted_id
    print(f"Inserted vehicle ID for update: {doc_id}")

    doc = vehicles.find_one({"_id": doc_id})
    assert doc["vehicle_status"] == "Active"

    # UPDATE
    vehicles.update_one({"_id": doc_id}, {"$set":
{"vehicle_status": "Inactive"}})
    updated_doc = vehicles.find_one({"_id": doc_id})
```

```
    assert updated_doc["vehicle_status"] == "Inactive"
    print("Vehicle update verified.")

    vehicles.delete_one({"_id": doc_id})
    assert vehicles.find_one({"_id": doc_id}) is None
```

```
C:\Users\qubit\Desktop\BD2\PostOffice\PostOffice_Project\PostOffice_Proj>pytest -s -v -rs ..\Test
s\U_teste_mongo_Rodrigo.py
========================================= test session starts =========================================
platform win32 -- Python 3.13.2, pytest-9.0.0, pluggy-1.6.0 -- C:\Python313\python.exe
cachedir: .pytest_cache
django: version: 5.2.7, settings: PostOffice_Proj.settings (from ini)
rootdir: C:\Users\qubit\Desktop\BD2\PostOffice
configfile: pytest.ini
plugins: django-4.11.1
collected 6 items

..\Tests\U_teste_mongo_Rodrigo.py::test_deliveries_update Inserted delivery ID for update: 690fd4
e14671f2fb34f0197d
Delivery update verified.
PASSED
..\Tests\U_teste_mongo_Rodrigo.py::test_notifications_update Inserted notification ID for update:
 690fd4e14671f2fb34f0197e
Notification update verified.
PASSED
..\Tests\U_teste_mongo_Rodrigo.py::test_postoffice_update Inserted postoffice ID for update: 690f
d4e14671f2fb34f0197f
Postoffice update verified.
PASSED
..\Tests\U_teste_mongo_Rodrigo.py::test_routes_update Inserted route ID for update: 690fd4e14671f
2fb34f01980
Route update verified.
PASSED
..\Tests\U_teste_mongo_Rodrigo.py::test_users_update Inserted user ID for update: 690fd4e14671f2f
b34f01981
User update verified.
PASSED
..\Tests\U_teste_mongo_Rodrigo.py::test_vehicles_update Inserted vehicle ID for update: 690fd4e14
671f2fb34f01982
Vehicle update verified.
PASSED

========================================= 6 passed in 0.36s ==========================================
```

(D)elete – mongo

```
import pytest
from pymongo import MongoClient
from datetime import datetime

# ------------------------------
# Connect to your real MongoDB
# ------------------------------
client = MongoClient("mongodb://localhost:27017")
db = client["postoffice"]

deliveries = db["deliveries"]
notifications = db["notifications"]
postoffice_col = db["postoffice"]
routes = db["routes"]
users = db["users"]
vehicles = db["vehicles"]

# ------------------------------
# Deliveries CRUD
# ------------------------------
def test_deliveries_crud():
    result = deliveries.insert_one({
        "recipient": "Test User",
        "address": "123 Test St",
        "status": "Pending",
        "delivery_date": datetime.now()
    })
    doc_id = result.inserted_id
    print(f"Inserted delivery ID: {doc_id}")

    # READ
    doc = deliveries.find_one({"_id": doc_id})
    assert doc is not None
    assert doc["recipient"] == "Test User"

    # UPDATE
    deliveries.update_one({"_id": doc_id}, {"$set":
{"status": "Delivered"}})
    updated_doc = deliveries.find_one({"_id": doc_id})
    assert updated_doc["status"] == "Delivered"

    # DELETE
```

```
    deliveries.delete_one({"_id": doc_id})
    assert deliveries.find_one({"_id": doc_id}) is None
    print("Delivery CRUD verified.")

# ------------------------------
# Notifications CRUD
# ------------------------------
def test_notifications_crud():
    result = notifications.insert_one({
        "title": "Test Notification",
        "message": "This is a test notification.",
        "date": datetime.now()
    })
    doc_id = result.inserted_id
    print(f"Inserted notification ID: {doc_id}")

    # READ
    doc = notifications.find_one({"_id": doc_id})
    assert doc is not None
    assert doc["title"] == "Test Notification"

    # UPDATE
    notifications.update_one({"_id": doc_id}, {"$set":
{"title": "Updated Notification"}})
    updated_doc = notifications.find_one({"_id": doc_id})
    assert updated_doc["title"] == "Updated Notification"

    # DELETE
    notifications.delete_one({"_id": doc_id})
    assert notifications.find_one({"_id": doc_id}) is None
    print("Notifications CRUD verified.")

# ------------------------------
# Postoffice CRUD
# ------------------------------
def test_postoffice_crud():
    result = postoffice_col.insert_one({
        "name": "Test Postoffice",
        "address": "456 Main St",
        "contact": "555-5678",
        "po_schedule_open": "08:00",
        "po_schedule_close": "17:00",
        "maximum_storage_capacity": 500
    })
```

```
    doc_id = result.inserted_id
    print(f"Inserted postoffice ID: {doc_id}")

    # READ
    doc = postoffice_col.find_one({"_id": doc_id})
    assert doc is not None
    assert doc["name"] == "Test Postoffice"

    # UPDATE
    postoffice_col.update_one({"_id": doc_id}, {"$set":
{"maximum_storage_capacity": 600}})
    updated_doc = postoffice_col.find_one({"_id": doc_id})
    assert updated_doc["maximum_storage_capacity"] == 600

    # DELETE
    postoffice_col.delete_one({"_id": doc_id})
    assert postoffice_col.find_one({"_id": doc_id}) is None
    print("Postoffice CRUD verified.")

# ------------------------------
# Routes CRUD
# ------------------------------
def test_routes_crud():
    result = routes.insert_one({
        "route_name": "Test Route",
        "origin": "Origin A",
        "destination": "Destination B",
        "distance_km": 25
    })
    doc_id = result.inserted_id
    print(f"Inserted route ID: {doc_id}")

    # READ
    doc = routes.find_one({"_id": doc_id})
    assert doc is not None
    assert doc["route_name"] == "Test Route"

    # UPDATE
    routes.update_one({"_id": doc_id}, {"$set":
{"distance_km": 30}})
    updated_doc = routes.find_one({"_id": doc_id})
    assert updated_doc["distance_km"] == 30

    # DELETE
    routes.delete_one({"_id": doc_id})
```

```
    assert routes.find_one({"_id": doc_id}) is None
    print("Routes CRUD verified.")

# ------------------------------
# Users CRUD
# ------------------------------
def test_users_crud():
    result = users.insert_one({
        "username": "testuser",
        "role": "client",
        "email": "test@example.com",
        "password": "hashed_password"
    })
    doc_id = result.inserted_id
    print(f"Inserted user ID: {doc_id}")

    # READ
    doc = users.find_one({"_id": doc_id})
    assert doc is not None
    assert doc["username"] == "testuser"

    # UPDATE
    users.update_one({"_id": doc_id}, {"$set": {"role":
"admin"}})
    updated_doc = users.find_one({"_id": doc_id})
    assert updated_doc["role"] == "admin"

    # DELETE
    users.delete_one({"_id": doc_id})
    assert users.find_one({"_id": doc_id}) is None
    print("Users CRUD verified.")

# ------------------------------
# Vehicles CRUD
# ------------------------------
def test_vehicles_crud():
    result = vehicles.insert_one({
        "vehicle_type": "Van",
        "plate_number": "TEST123",
        "capacity": 1000,
        "brand": "TestBrand",
        "model": "TB1",
        "vehicle_status": "Active",
        "year": 2025,
```

```
        "fuel_type": "Electric",
        "last_maintenance_date": datetime.now()
    })
    doc_id = result.inserted_id
    print(f"Inserted vehicle ID: {doc_id}")

    # READ
    doc = vehicles.find_one({"_id": doc_id})
    assert doc is not None
    assert doc["plate_number"] == "TEST123"

    # UPDATE
    vehicles.update_one({"_id": doc_id}, {"$set":
{"vehicle_status": "Inactive"}})
    updated_doc = vehicles.find_one({"_id": doc_id})
    assert updated_doc["vehicle_status"] == "Inactive"

    # DELETE
    vehicles.delete_one({"_id": doc_id})
    assert vehicles.find_one({"_id": doc_id}) is None
    print("Vehicles CRUD verified.")
```

(D)elete – PgSql

```
import pytest
from django.db import connections

@pytest.mark.django_db
def testeprocedimentox():
    cur = connections['default'].cursor()

    # Start transaction
    cur.execute("BEGIN;")

    # Insert invoice
    cur.execute("""
        INSERT INTO invoices
        (invoice_status, invoice_type, quantity,
invoice_datetime, cost, paid, payment_method, name, address,
contact)
        VALUES
        ('Pending', 'Online', 3, CURRENT_TIMESTAMP, 150.00,
FALSE, 'Credit Card', 'John Doe', '123 Main St', '555-1234');
    """)

    # Get the ID of the invoice we just inserted
    cur.execute("""
        SELECT id_invoice
        FROM invoices
        WHERE name='John Doe'
        ORDER BY id_invoice DESC
        LIMIT 1;
    """)
    invoice_id = cur.fetchone()[0]

    # Delete the invoice
    cur.execute("DELETE FROM invoices WHERE id_invoice =
%s;", [invoice_id])

    # Check if deletion succeeded
    cur.execute("SELECT COUNT(*) FROM invoices WHERE
id_invoice = %s;", [invoice_id])
    count = cur.fetchone()[0]

    assert count == 0, "Invoice was not deleted successfully"
```

```
    # Rollback transaction so database remains unchanged
    cur.execute("ROLLBACK;")
```

```
pedro@PedroMonteiro MINGW64 ~/django_local_library/django_test/PostOffice_Proj (update_gitignore)
$ pytest -s -v -rs PostOffice_App/test_sql.py
============================= test session starts =============================
platform win32 -- Python 3.10.0, pytest-9.0.0, pluggy-1.6.0 -- C:\Users\pedro\AppData\Local\Programs\Python\Python310\python.exe
cachedir: .pytest_cache
django: version: 4.2.24, settings: PostOffice_Proj.settings (from env)
rootdir: C:\Users\pedro\django_local_library\django_test\PostOffice_Proj
plugins: django-4.11.1
collected 1 item

PostOffice_App/test_sql.py::testeprocedimentox PASSED

============================== 1 passed in 0.13s ==============================

pedro@PedroMonteiro MINGW64 ~/django_local_library/django_test/PostOffice_Proj (update_gitignore)
$ pytest -s -v -rs PostOffice_App/test_mongo.py
============================= test session starts =============================
platform win32 -- Python 3.10.0, pytest-9.0.0, pluggy-1.6.0 -- C:\Users\pedro\AppData\Local\Programs\Python\Python310\python.exe
cachedir: .pytest_cache
django: version: 4.2.24, settings: PostOffice_Proj.settings (from env)
rootdir: C:\Users\pedro\django_local_library\django_test\PostOffice_Proj
plugins: django-4.11.1
collected 6 items

PostOffice_App/test_mongo.py::test_deliveries_crud Inserted delivery ID: 690fcd2b11bf99b63a4387d4
Delivery CRUD verified.
PASSED
PostOffice_App/test_mongo.py::test_notifications_crud Inserted notification ID: 690fcd2b11bf99b63a4387d5
Notifications CRUD verified.
PASSED
PostOffice_App/test_mongo.py::test_postoffice_crud Inserted postoffice ID: 690fcd2b11bf99b63a4387d6
Postoffice CRUD verified.
PASSED
PostOffice_App/test_mongo.py::test_routes_crud Inserted route ID: 690fcd2b11bf99b63a4387d7
Routes CRUD verified.
PASSED
PostOffice_App/test_mongo.py::test_users_crud Inserted user ID: 690fcd2b11bf99b63a4387d8
Users CRUD verified.
PASSED
PostOffice_App/test_mongo.py::test_vehicles_crud Inserted vehicle ID: 690fcd2b11bf99b63a4387d9
Vehicles CRUD verified.
PASSED

============================== 6 passed in 0.23s ==============================
```

Essay developped by:

Pedro Monteiro - estgv14366;

Rodrigo Rolo - estgv18757;

Diego Alonso - pv33986;

David Gonzalez - pv33971;