ATDF102

Avance N°2

Semana: 6

Sumativa 2

Integrantes: Pedro Garrido, Daniel Beth, Pablo Morales, Benjamín Aranda y Byron Contreras

src/main.py

```
from menu.menu import main

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

src/mock_data.py

```
from modules.region import Region
from modules.commune import Commune
from modules.client import Client
from modules.condo import Condo
from modules.house_type import HouseType
from modules.house import House
from modules.payments import Payments
from modules.client_type import ClientType
from modules.payment_type import PaymentType
from modules.payment_month import PaymentMonth
from modules.payment_year import PaymentYear
import random

def generate_rut():
    rut_num = random.randint(5000000, 25000000)
    verif_digit = random.randint(0, 9)
    return f"{rut_num}-{verif_digit}"

def load_mock_data():
    # Mock Regions
    regions = [
        Region(id=1, name="Región de Arica y Parinacota"),
        Region(id=2, name="Región de Tarapacá"),
        Region(id=3, name="Región de Antofagasta"),
        Region(id=4, name="Región de Atacama"),
        Region(id=5, name="Región de Coquimbo"),
        Region(id=6, name="Región de Valparaíso"),
        Region(id=7, name="Región Metropolitana de Santiago"),
        Region(id=8, name="Región del Libertador General Bernardo O'Higgins"),
        Region(id=9, name="Región del Maule"),
        Region(id=10, name="Región de Ñuble"),
        Region(id=11, name="Región del Biobío"),
        Region(id=12, name="Región de La Araucanía"),
        Region(id=13, name="Región de Los Ríos"),
        Region(id=14, name="Región de Los Lagos"),
        Region(id=15, name="Región de Aysén del General Carlos Ibáñez del Campo"),
        Region(id=16, name="Región de Magallanes y de la Antártica Chilena"),
    ]

    # Mock Communes
    communes = [
        Commune(id=1, name="Arica", region_id=1),
        Commune(id=2, name="Camarones", region_id=1),
        Commune(id=3, name="Iquique", region_id=2),
        Commune(id=4, name="Alto Hospicio", region_id=2),
        Commune(id=5, name="Antofagasta", region_id=3),
        Commune(id=6, name="Mejillones", region_id=3),
        Commune(id=7, name="Copiapó", region_id=4),
        Commune(id=8, name="Caldera", region_id=4),
        Commune(id=9, name="La Serena", region_id=5),
        Commune(id=10, name="Coquimbo", region_id=5),
        Commune(id=11, name="Valparaíso", region_id=6),
        Commune(id=12, name="Viña del Mar", region_id=6),
        Commune(id=13, name="Santiago", region_id=7),
        Commune(id=14, name="Las Condes", region_id=7),
        Commune(id=15, name="Rancagua", region_id=8),
        Commune(id=16, name="Machalí", region_id=8),
        Commune(id=17, name="Talca", region_id=9),
        Commune(id=18, name="Curicó", region_id=9),
        Commune(id=19, name="Chillán", region_id=10),
        Commune(id=20, name="Bulnes", region_id=10),
        Commune(id=21, name="Concepción", region_id=11),
        Commune(id=22, name="Talcahuano", region_id=11),
        Commune(id=23, name="Temuco", region_id=12),
```

```
        Commune(id=24, name="Padre Las Casas", region_id=12),
        Commune(id=25, name="Valdivia", region_id=13),
        Commune(id=26, name="La Unión", region_id=13),
        Commune(id=27, name="Puerto Montt", region_id=14),
        Commune(id=28, name="Osorno", region_id=14),
        Commune(id=29, name="Coyhaique", region_id=15),
        Commune(id=30, name="Aysén", region_id=15),
        Commune(id=31, name="Punta Arenas", region_id=16),
        Commune(id=32, name="Puerto Natales", region_id=16),
    ]

    # Mock Client Types
    client_types = [
        ClientType(id=1, name="Arrendatario"),
        ClientType(id=2, name="Propietario"),
    ]

    # Mock Clients
    first_names = ["Juan", "María", "Pedro", "Ana", "Luis", "Carolina", "José", "Daniela", "Fernando",
"Valentina"]
    last_names = ["González", "Muñoz", "Rojas", "Díaz", "Pérez", "Soto", "Contreras", "Silva",
"Martínez", "Sepúlveda"]
    clients = []
    for i in range(1, 21):
        full_name = f"{random.choice(first_names)} {random.choice(last_names)}"
        email = f"{full_name.lower().replace(' ', '.')}@example.com"
        phone = f"+569{random.randint(10000000, 99999999)}"
        clients.append(Client(id=i, rut=generate_rut(), full_name=full_name, email=email, phone=phone))

    # Mock Condos
    condo_names = ["Los Álamos", "El Roble", "Las Palmas", "Los Jardines", "El Bosque"]
    street_names = ["Avenida Siempre Viva", "Calle Falsa", "Pasaje Los Aromos", "Avenida del Mar",
"Calle Principal"]
    condos = []
    for i in range(1, 6):
        condos.append(Condo(id=i, name=f"Condominio {random.choice(condo_names)}",
street=random.choice(street_names), number=str(random.randint(100, 999)), commune_id=random.randint(1,
32)))

    # Mock House Types
    house_types = [
        HouseType(id=1, name="Casa"),
        HouseType(id=2, name="Departamento"),
    ]

    # Mock Houses
    houses = []
    for i in range(1, 21):
        houses.append(House(id=i, street=random.choice(street_names), number=f"{random.randint(1,
100)}{random.choice(['A', 'B', 'C', ''])}", type_id=random.randint(1, 2), condo_id=random.randint(1, 5),
client_id=i))

    # Mock Payment Types
    payment_types = [
        PaymentType(id=1, name="Efectivo"),
        PaymentType(id=2, name="Transferencia"),
        PaymentType(id=3, name="Tarjeta de Crédito"),
        PaymentType(id=4, name="Tarjeta de Débito"),
    ]

    # Mock Payment Months
    payment_months = [PaymentMonth(id=i, name=m, month_number=i) for i, m in enumerate([
        "Enero", "Febrero", "Marzo", "Abril", "Mayo", "Junio",
        "Julio", "Agosto", "Septiembre", "Octubre", "Noviembre", "Diciembre"
    ], 1)]
```

```
    # Mock Payment Years
    payment_years = [PaymentYear(id=i, year=y) for i, y in enumerate(range(2020, 2026), 1)]

    # Mock Payments
    payments = []
    for i in range(1, 51):
        client_id = random.randint(1, 20)
        house_id = client_id # Assuming one house per client for simplicity
        payments.append(Payments(id=i, id_client=client_id, id_house=house_id,
payment_year_id=random.randint(1, 6), payment_month_id=random.randint(1, 12),
payment_type=random.randint(1, 4), amount=random.randint(30000, 150000), description=f"Pago de gastos
comunes"))

    return {
        "regions": regions,
        "communes": communes,
        "client_types": client_types,
        "clients": clients,
        "condos": condos,
        "house_types": house_types,
        "houses": houses,
        "payment_types": payment_types,
        "payment_months": payment_months,
        "payment_years": payment_years,
        "payments": payments,
    }
```

src/maintainers/client_maintainer.py

```
def list_clients(clients):
    """Prints a list of all clients."""
    if not clients:
        print("No hay clientes registrados.")
        return
    for client in clients:
        print(f"ID: {client.id}, RUT: {client.rut}, Nombre: {client.full_name}, Email: {client.email},
Teléfono: {client.phone}")

def add_client(clients):
    """Prompts the user for client information and creates a new client."""
    rut = input("Ingrese RUT: ")
    full_name = input("Ingrese nombre completo: ")
    email = input("Ingrese email: ")
    phone = input("Ingrese teléfono: ")
    new_client_id = max([client.id for client in clients]) + 1
    new_client = {"id": new_client_id, "rut": rut, "full_name": full_name, "email": email, "phone":
phone}
    clients.append(new_client)
    print(f"Cliente '{new_client['full_name']}' creado con éxito.")

def edit_client(clients):
    """Prompts the user for a client ID and new information to update a client."""
    list_clients(clients)
    client_id = int(input("Ingrese el ID del cliente a editar: ") )
    rut = input("Ingrese nuevo RUT: ")
    full_name = input("Ingrese nuevo nombre completo: ")
    email = input("Ingrese nuevo email: ")
    phone = input("Ingrese nuevo teléfono: ")
    for client in clients:
        if client.id == client_id:
            client.rut = rut
            client.full_name = full_name
            client.email = email
            client.phone = phone
            print(f"Cliente '{client.full_name}' actualizado con éxito.")
            return
    print("Cliente no encontrado.")

def remove_client(clients):
    """Prompts the user for a client ID to delete a client."""
    list_clients(clients)
    client_id = int(input("Ingrese el ID del cliente a eliminar: "))
    initial_len = len(clients)
    clients[:] = [client for client in clients if client.id != client_id]
    if len(clients) < initial_len:
        print("Cliente eliminado con éxito.")
    else:
        print("Cliente no encontrado.")

def client_maintainer(data):
    """Shows the client maintainer menu and handles user input."""
    while True:
        print("\n--- Mantenedor de Clientes ---")
        print("1. Listar clientes")
        print("2. Agregar cliente")
        print("3. Editar cliente")
        print("4. Eliminar cliente")
        print("5. Volver al menú principal")
        choice = input("Seleccione una opción: ")

        if choice == "1":
```

```
    list_clients(data["clients"])
elif choice == "2":
    add_client(data["clients"])
elif choice == "3":
    edit_client(data["clients"])
elif choice == "4":
    remove_client(data["clients"])
elif choice == "5":
    break
else:
    print("Opción no válida. Intente de nuevo.")
```

src/maintainers/client_type_maintainer.py

```
def list_client_types(client_types):
    """Prints a list of all client types."""
    if not client_types:
        print("No hay tipos de clientes registrados.")
        return
    for client_type in client_types:
        print(f"ID: {client_type.id}, Nombre: {client_type.name}")

def add_client_type(client_types):
    """Prompts the user for client type information and creates a new client type."""
    name = input("Ingrese nombre del tipo de cliente: ")
    new_client_type_id = max([ct.id for ct in client_types]) + 1
    new_client_type = {"id": new_client_type_id, "name": name}
    client_types.append(new_client_type)
    print(f"Tipo de cliente '{new_client_type['name']}' creado con éxito.")

def edit_client_type(client_types):
    """Prompts the user for a client type ID and new information to update a client type."""
    list_client_types(client_types)
    client_type_id = int(input("Ingrese el ID del tipo de cliente a editar: "))
    name = input("Ingrese nuevo nombre: ")
    for client_type in client_types:
        if client_type.id == client_type_id:
            client_type.name = name
            print(f"Tipo de cliente '{client_type.name}' actualizado con éxito.")
            return
    print("Tipo de cliente no encontrado.")

def remove_client_type(client_types):
    """Prompts the user for a client type ID to delete a client type."""
    list_client_types(client_types)
    client_type_id = int(input("Ingrese el ID del tipo de cliente a eliminar: "))
    initial_len = len(client_types)
    client_types[:] = [client_type for client_type in client_types if client_type.id != client_type_id]
    if len(client_types) < initial_len:
        print("Tipo de cliente eliminado con éxito.")
    else:
        print("Tipo de cliente no encontrado.")

def client_type_maintainer(data):
    """Shows the client type maintainer menu and handles user input."""
    while True:
        print("\n--- Mantenedor de Tipos de Cliente ---")
        print("1. Listar tipos de cliente")
        print("2. Agregar tipo de cliente")
        print("3. Editar tipo de cliente")
        print("4. Eliminar tipo de cliente")
        print("5. Volver al menú principal")
        choice = input("Seleccione una opción: ")

        if choice == "1":
            list_client_types(data["client_types"])
        elif choice == "2":
            add_client_type(data["client_types"])
        elif choice == "3":
            edit_client_type(data["client_types"])
        elif choice == "4":
            remove_client_type(data["client_types"])
        elif choice == "5":
            break
        else:
            print("Opción no válida. Intente de nuevo.")
```

src/maintainers/commune_maintainer.py

```
def list_communes(communes):
    """Prints a list of all communes."""
    if not communes:
        print("No hay comunas registradas.")
        return
    for commune in communes:
        print(f"ID: {commune.id}, Nombre: {commune.name}, ID Región: {commune.region_id}")

def add_commune(communes):
    """Prompts the user for commune information and creates a new commune."""
    name = input("Ingrese nombre de la comuna: ")
    region_id = int(input("Ingrese ID de la región: "))
    new_commune_id = max([c.id for c in communes]) + 1
    new_commune = {"id": new_commune_id, "name": name, "region_id": region_id}
    communes.append(new_commune)
    print(f"Comuna '{new_commune['name']}' creada con éxito.")

def edit_commune(communes):
    """Prompts the user for a commune ID and new information to update a commune."""
    list_communes(communes)
    commune_id = int(input("Ingrese el ID de la comuna a editar: "))
    name = input("Ingrese nuevo nombre: ")
    region_id = int(input("Ingrese nuevo ID de la región: "))
    for commune in communes:
        if commune.id == commune_id:
            commune.name = name
            commune.region_id = region_id
            print(f"Comuna '{commune.name}' actualizada con éxito.")
            return
    print("Comuna no encontrada.")

def remove_commune(communes):
    """Prompts the user for a commune ID to delete a commune."""
    list_communes(communes)
    commune_id = int(input("Ingrese el ID de la comuna a eliminar: "))
    initial_len = len(communes)
    communes[:] = [commune for commune in communes if commune.id != commune_id]
    if len(communes) < initial_len:
        print("Comuna eliminada con éxito.")
    else:
        print("Comuna no encontrada.")

def commune_maintainer(data):
    """Shows the commune maintainer menu and handles user input."""
    while True:
        print("\n--- Mantenedor de Comunas ---")
        print("1. Listar comunas")
        print("2. Agregar comuna")
        print("3. Editar comuna")
        print("4. Eliminar comuna")
        print("5. Volver al menú principal")
        choice = input("Seleccione una opción: ")

        if choice == "1":
            list_communes(data["communes"])
        elif choice == "2":
            add_commune(data["communes"])
        elif choice == "3":
            edit_commune(data["communes"])
        elif choice == "4":
            remove_commune(data["communes"])
        elif choice == "5":
```

```
        break
    else:
        print("Opción no válida. Intente de nuevo.")
```

src/maintainers/condo_maintainer.py

```
def list_condos(condos):
    """Prints a list of all condos."""
    if not condos:
        print("No hay condominios registrados.")
        return
    for condo in condos:
        print(f"ID: {condo.id}, Calle: {condo.street}, Número: {condo.number}, Nombre: {condo.name}, ID
Comuna: {condo.commune_id}")

def add_condo(condos):
    """Prompts the user for condo information and creates a new condo."""
    street = input("Ingrese calle: ")
    number = input("Ingrese número: ")
    name = input("Ingrese nombre: ")
    commune_id = int(input("Ingrese ID de la comuna: "))
    new_condo_id = max([c.id for c in condos]) + 1
    new_condo = {"id": new_condo_id, "street": street, "number": number, "name": name, "commune_id":
commune_id}
    condos.append(new_condo)
    print(f"Condominio '{new_condo['name']}' creado con éxito.")

def edit_condo(condos):
    """Prompts the user for a condo ID and new information to update a condo."""
    list_condos(condos)
    condo_id = int(input("Ingrese el ID del condominio a editar: "))
    street = input("Ingrese nueva calle: ")
    number = input("Ingrese nuevo número: ")
    name = input("Ingrese nuevo nombre: ")
    commune_id = int(input("Ingrese nuevo ID de la comuna: "))
    for condo in condos:
        if condo.id == condo_id:
            condo.street = street
            condo.number = number
            condo.name = name
            condo.commune_id = commune_id
            print(f"Condominio '{condo.name}' actualizado con éxito.")
            return
    print("Condominio no encontrado.")

def remove_condo(condos):
    """Prompts the user for a condo ID to delete a condo."""
    list_condos(condos)
    condo_id = int(input("Ingrese el ID del condominio a eliminar: "))
    initial_len = len(condos)
    condos[:] = [condo for condo in condos if condo.id != condo_id]
    if len(condos) < initial_len:
        print("Condominio eliminado con éxito.")
    else:
        print("Condominio no encontrado.")

def condo_maintainer(data):
    """Shows the condo maintainer menu and handles user input."""
    while True:
        print("\n--- Mantenedor de Condominios ---")
        print("1. Listar condominios")
        print("2. Agregar condominio")
        print("3. Editar condominio")
        print("4. Eliminar condominio")
        print("5. Volver al menú principal")
        choice = input("Seleccione una opción: ")

        if choice == "1":
```

```
    list_condos(data["condos"])
elif choice == "2":
    add_condo(data["condos"])
elif choice == "3":
    edit_condo(data["condos"])
elif choice == "4":
    remove_condo(data["condos"])
elif choice == "5":
    break
else:
    print("Opción no válida. Intente de nuevo.")
```

src/maintainers/house_maintainer.py

```
def list_houses(houses):
    """Prints a list of all houses."""
    if not houses:
        print("No hay casas registradas.")
        return
    for house in houses:
        print(f"ID: {house.id}, Calle: {house.street}, Número: {house.number}, ID Tipo: {house.type_id},
ID Condominio: {house.condo_id}")

def add_house(houses):
    """Prompts the user for house information and creates a new house."""
    street = input("Ingrese calle: ")
    number = input("Ingrese número: ")
    type_id = int(input("Ingrese ID del tipo de casa: "))
    condo_id = int(input("Ingrese ID del condominio: "))
    client_id = int(input("Ingrese ID del cliente: "))
    new_house_id = max([h.id for h in houses]) + 1
    new_house = {"id": new_house_id, "street": street, "number": number, "type_id": type_id, "condo_id":
condo_id, "client_id": client_id}
    houses.append(new_house)
    print(f"Casa en '{new_house['street']} {new_house['number']}' creada con éxito.")

def edit_house(houses):
    """Prompts the user for a house ID and new information to update a house."""
    list_houses(houses)
    house_id = int(input("Ingrese el ID de la casa a editar: "))
    street = input("Ingrese nueva calle: ")
    number = input("Ingrese nuevo número: ")
    type_id = int(input("Ingrese nuevo ID del tipo de casa: "))
    condo_id = int(input("Ingrese nuevo ID del condominio: "))
    client_id = int(input("Ingrese nuevo ID del cliente: "))
    for house in houses:
        if house.id == house_id:
            house.street = street
            house.number = number
            house.type_id = type_id
            house.condo_id = condo_id
            house.client_id = client_id
            print(f"Casa en '{house.street} {house.number}' actualizada con éxito.")
            return
    print("Casa no encontrada.")

def remove_house(houses):
    """Prompts the user for a house ID to delete a house."""
    list_houses(houses)
    house_id = int(input("Ingrese el ID de la casa a eliminar: "))
    initial_len = len(houses)
    houses[:] = [house for house in houses if house.id != house_id]
    if len(houses) < initial_len:
        print("Casa eliminada con éxito.")
    else:
        print("Casa no encontrada.")

def house_maintainer(data):
    """Shows the house maintainer menu and handles user input."""
    while True:
        print("\n--- Mantenedor de Casas ---")
        print("1. Listar casas")
        print("2. Agregar casa")
        print("3. Editar casa")
        print("4. Eliminar casa")
        print("5. Volver al menú principal")
```

```
choice = input("Seleccione una opción: ")

if choice == "1":
    list_houses(data["houses"])
elif choice == "2":
    add_house(data["houses"])
elif choice == "3":
    edit_house(data["houses"])
elif choice == "4":
    remove_house(data["houses"])
elif choice == "5":
    break
else:
    print("Opción no válida. Intente de nuevo.")
```

src/maintainers/house_type_maintainer.py

```
def list_house_types(house_types):
    """Prints a list of all house types."""
    if not house_types:
        print("No hay tipos de casas registrados.")
        return
    for house_type in house_types:
        print(f"ID: {house_type.id}, Nombre: {house_type.name}")

def add_house_type(house_types):
    """Prompts the user for house type information and creates a new house type."""
    name = input("Ingrese nombre del tipo de casa: ")
    new_house_type_id = max([ht.id for ht in house_types]) + 1
    new_house_type = {"id": new_house_type_id, "name": name}
    house_types.append(new_house_type)
    print(f"Tipo de casa '{new_house_type['name']}' creado con éxito.")

def edit_house_type(house_types):
    """Prompts the user for a house type ID and new information to update a house type."""
    list_house_types(house_types)
    house_type_id = int(input("Ingrese el ID del tipo de casa a editar: "))
    name = input("Ingrese nuevo nombre: ")
    for house_type in house_types:
        if house_type.id == house_type_id:
            house_type.name = name
            print(f"Tipo de casa '{house_type.name}' actualizado con éxito.")
            return
    print("Tipo de casa no encontrado.")

def remove_house_type(house_types):
    """Prompts the user for a house type ID to delete a house type."""
    list_house_types(house_types)
    house_type_id = int(input("Ingrese el ID del tipo de casa a eliminar: "))
    initial_len = len(house_types)
    house_types[:] = [house_type for house_type in house_types if house_type.id != house_type_id]
    if len(house_types) < initial_len:
        print("Tipo de casa eliminado con éxito.")
    else:
        print("Tipo de casa no encontrado.")

def house_type_maintainer(data):
    """Shows the house type maintainer menu and handles user input."""
    while True:
        print("\n--- Mantenedor de Tipos de Casa ---")
        print("1. Listar tipos de casa")
        print("2. Agregar tipo de casa")
        print("3. Editar tipo de casa")
        print("4. Eliminar tipo de casa")
        print("5. Volver al menú principal")
        choice = input("Seleccione una opción: ")

        if choice == "1":
            list_house_types(data["house_types"])
        elif choice == "2":
            add_house_type(data["house_types"])
        elif choice == "3":
            edit_house_type(data["house_types"])
        elif choice == "4":
            remove_house_type(data["house_types"])
        elif choice == "5":
            break
        else:
            print("Opción no válida. Intente de nuevo.")
```

src/maintainers/payments_maintainer.py

```
def list_payments(payments):
    """Prints a list of all payments."""
    if not payments:
        print("No hay pagos registrados.")
        return
    for payment in payments:
        print(f"ID: {payment.id}, ID Cliente: {payment.id_client}, ID Casa: {payment.id_house}, ID Año:
{payment.payment_year_id}, ID Mes: {payment.payment_month_id}, ID Tipo Pago: {payment.payment_type},
Monto: {payment.amount}, Descripción: {payment.description}")

def add_payment(payments):
    """Prompts the user for payment information and creates a new payment."""
    id_client = int(input("Ingrese ID del cliente: "))
    id_house = int(input("Ingrese ID de la casa: "))
    payment_year_id = int(input("Ingrese ID del año de pago: "))
    payment_month_id = int(input("Ingrese ID del mes de pago: "))
    payment_type = int(input("Ingrese ID del tipo de pago: "))
    amount = float(input("Ingrese monto: "))
    description = input("Ingrese descripción: ")
    new_payment_id = max([p.id for p in payments]) + 1
    new_payment = {"id": new_payment_id, "id_client": id_client, "id_house": id_house,
"payment_year_id": payment_year_id, "payment_month_id": payment_month_id, "payment_type": payment_type,
"amount": amount, "description": description}
    payments.append(new_payment)
    print(f"Pago de '{new_payment['amount']}' creado con éxito.")

def edit_payment(payments):
    """Prompts the user for a payment ID and new information to update a payment."""
    list_payments(payments)
    payment_id = int(input("Ingrese el ID del pago a editar: "))
    id_client = int(input("Ingrese nuevo ID del cliente: "))
    id_house = int(input("Ingrese nuevo ID de la casa: "))
    payment_year_id = int(input("Ingrese nuevo ID del año de pago: "))
    payment_month_id = int(input("Ingrese nuevo ID del mes de pago: "))
    payment_type = int(input("Ingrese nuevo ID del tipo de pago: "))
    amount = float(input("Ingrese nuevo monto: "))
    description = input("Ingrese nueva descripción: ")
    for payment in payments:
        if payment.id == payment_id:
            payment.id_client = id_client
            payment.id_house = id_house
            payment.payment_year_id = payment_year_id
            payment.payment_month_id = payment_month_id
            payment.payment_type = payment_type
            payment.amount = amount
            payment.description = description
            print(f"Pago de '{payment.amount}' actualizado con éxito.")
            return
    print("Pago no encontrado.")

def remove_payment(payments):
    """Prompts the user for a payment ID to delete a payment."""
    list_payments(payments)
    payment_id = int(input("Ingrese el ID del pago a eliminar: "))
    initial_len = len(payments)
    payments[:] = [payment for payment in payments if payment.id != payment_id]
    if len(payments) < initial_len:
        print("Pago eliminado con éxito.")
    else:
        print("Pago no encontrado.")

def payment_maintainer(data):
```

```
"""Shows the payment maintainer menu and handles user input."""
while True:
    print("\n--- Mantenedor de Pagos ---")
    print("1. Listar pagos")
    print("2. Agregar pago")
    print("3. Editar pago")
    print("4. Eliminar pago")
    print("5. Volver al menú principal")
    choice = input("Seleccione una opción: ")

    if choice == "1":
        list_payments(data["payments"])
    elif choice == "2":
        add_payment(data["payments"])
    elif choice == "3":
        edit_payment(data["payments"])
    elif choice == "4":
        remove_payment(data["payments"])
    elif choice == "5":
        break
    else:
        print("Opción no válida. Intente de nuevo.")
```

src/maintainers/payment_month_maintainer.py

```
def list_payment_months(payment_months):
    """Prints a list of all payment months."""
    if not payment_months:
        print("No hay meses de pago registrados.")
        return
    for payment_month in payment_months:
        print(f"ID: {payment_month.id}, Nombre: {payment_month.name}, Número:
{payment_month.month_number}")

def add_payment_month(payment_months):
    """Prompts the user for payment month information and creates a new payment month."""
    name = input("Ingrese nombre del mes: ")
    month_number = int(input("Ingrese número del mes: "))
    new_payment_month_id = max([pm.id for pm in payment_months]) + 1
    new_payment_month = {"id": new_payment_month_id, "name": name, "month_number": month_number}
    payment_months.append(new_payment_month)
    print(f"Mes de pago '{new_payment_month['name']}' creado con éxito.")

def edit_payment_month(payment_months):
    """Prompts the user for a payment month ID and new information to update a payment month."""
    list_payment_months(payment_months)
    payment_month_id = int(input("Ingrese el ID del mes de pago a editar: "))
    name = input("Ingrese nuevo nombre: ")
    month_number = int(input("Ingrese nuevo número del mes: "))
    for payment_month in payment_months:
        if payment_month.id == payment_month_id:
            payment_month.name = name
            payment_month.month_number = month_number
            print(f"Mes de pago '{payment_month.name}' actualizado con éxito.")
            return
    print("Mes de pago no encontrado.")

def remove_payment_month(payment_months):
    """Prompts the user for a payment month ID to delete a payment month."""
    list_payment_months(payment_months)
    payment_month_id = int(input("Ingrese el ID del mes de pago a eliminar: "))
    initial_len = len(payment_months)
    payment_months[:] = [payment_month for payment_month in payment_months if payment_month.id !=
payment_month_id]
    if len(payment_months) < initial_len:
        print("Mes de pago eliminado con éxito.")
    else:
        print("Mes de pago no encontrado.")

def payment_month_maintainer(data):
    """Shows the payment month maintainer menu and handles user input."""
    while True:
        print("\n--- Mantenedor de Meses de Pago ---")
        print("1. Listar meses de pago")
        print("2. Agregar mes de pago")
        print("3. Editar mes de pago")
        print("4. Eliminar mes de pago")
        print("5. Volver al menú principal")
        choice = input("Seleccione una opción: ")

        if choice == "1":
            list_payment_months(data["payment_months"])
        elif choice == "2":
            add_payment_month(data["payment_months"])
        elif choice == "3":
            edit_payment_month(data["payment_months"])
        elif choice == "4":
```

```
    remove_payment_month(data["payment_months"])
elif choice == "5":
    break
else:
    print("Opción no válida. Intente de nuevo.")
```

src/maintainers/payment_type_maintainer.py

```
def list_payment_types(payment_types):
    """Prints a list of all payment types."""
    if not payment_types:
        print("No hay tipos de pago registrados.")
        return
    for payment_type in payment_types:
        print(f"ID: {payment_type.id}, Nombre: {payment_type.name}")

def add_payment_type(payment_types):
    """Prompts the user for payment type information and creates a new payment type."""
    name = input("Ingrese nombre del tipo de pago: ")
    new_payment_type_id = max([pt.id for pt in payment_types]) + 1
    new_payment_type = {"id": new_payment_type_id, "name": name}
    payment_types.append(new_payment_type)
    print(f"Tipo de pago '{new_payment_type['name']}' creado con éxito.")

def edit_payment_type(payment_types):
    """Prompts the user for a payment type ID and new information to update a payment type."""
    list_payment_types(payment_types)
    payment_type_id = int(input("Ingrese el ID del tipo de pago a editar: "))
    name = input("Ingrese nuevo nombre: ")
    for payment_type in payment_types:
        if payment_type.id == payment_type_id:
            payment_type.name = name
            print(f"Tipo de pago '{payment_type.name}' actualizado con éxito.")
            return
    print("Tipo de pago no encontrado.")

def remove_payment_type(payment_types):
    """Prompts the user for a payment type ID to delete a payment type."""
    list_payment_types(payment_types)
    payment_type_id = int(input("Ingrese el ID del tipo de pago a eliminar: "))
    initial_len = len(payment_types)
    payment_types[:] = [payment_type for payment_type in payment_types if payment_type.id !=
payment_type_id]
    if len(payment_types) < initial_len:
        print("Tipo de pago eliminado con éxito.")
    else:
        print("Tipo de pago no encontrado.")

def payment_type_maintainer(data):
    """Shows the payment type maintainer menu and handles user input."""
    while True:
        print("\n--- Mantenedor de Tipos de Pago ---")
        print("1. Listar tipos de pago")
        print("2. Agregar tipo de pago")
        print("3. Editar tipo de pago")
        print("4. Eliminar tipo de pago")
        print("5. Volver al menú principal")
        choice = input("Seleccione una opción: ")

        if choice == "1":
            list_payment_types(data["payment_types"])
        elif choice == "2":
            add_payment_type(data["payment_types"])
        elif choice == "3":
            edit_payment_type(data["payment_types"])
        elif choice == "4":
            remove_payment_type(data["payment_types"])
        elif choice == "5":
            break
        else:
```

```
        print("Opción no válida. Intente de nuevo.")
```

src/maintainers/payment_year_maintainer.py

```
def list_payment_years(payment_years):
    """Prints a list of all payment years."""
    if not payment_years:
        print("No hay años de pago registrados.")
        return
    for payment_year in payment_years:
        print(f"ID: {payment_year.id}, Año: {payment_year.year}")

def add_payment_year(payment_years):
    """Prompts the user for payment year information and creates a new payment year."""
    year = int(input("Ingrese año: "))
    new_payment_year_id = max([py.id for py in payment_years]) + 1
    new_payment_year = {"id": new_payment_year_id, "year": year}
    payment_years.append(new_payment_year)
    print(f"Año de pago '{new_payment_year['year']}' creado con éxito.")

def edit_payment_year(payment_years):
    """Prompts the user for a payment year ID and new information to update a payment year."""
    list_payment_years(payment_years)
    payment_year_id = int(input("Ingrese el ID del año de pago a editar: "))
    year = int(input("Ingrese nuevo año: "))
    for payment_year in payment_years:
        if payment_year.id == payment_year_id:
            payment_year.year = year
            print(f"Año de pago '{payment_year.year}' actualizado con éxito.")
            return
    print("Año de pago no encontrado.")

def remove_payment_year(payment_years):
    """Prompts the user for a payment year ID to delete a payment year."""
    list_payment_years(payment_years)
    payment_year_id = int(input("Ingrese el ID del año de pago a eliminar: "))
    initial_len = len(payment_years)
    payment_years[:] = [payment_year for payment_year in payment_years if payment_year.id !=
payment_year_id]
    if len(payment_years) < initial_len:
        print("Año de pago eliminado con éxito.")
    else:
        print("Año de pago no encontrado.")

def payment_year_maintainer(data):
    """Shows the payment year maintainer menu and handles user input."""
    while True:
        print("\n--- Mantenedor de Años de Pago ---")
        print("1. Listar años de pago")
        print("2. Agregar año de pago")
        print("3. Editar año de pago")
        print("4. Eliminar año de pago")
        print("5. Volver al menú principal")
        choice = input("Seleccione una opción: ")

        if choice == "1":
            list_payment_years(data["payment_years"])
        elif choice == "2":
            add_payment_year(data["payment_years"])
        elif choice == "3":
            edit_payment_year(data["payment_years"])
        elif choice == "4":
            remove_payment_year(data["payment_years"])
        elif choice == "5":
            break
        else:
```

```
        print("Opción no válida. Intente de nuevo.")
```

src/maintainers/region_maintainer.py

```
def list_regions(regions):
    """Prints a list of all regions."""
    if not regions:
        print("No hay regiones registradas.")
        return
    for region in regions:
        print(f"ID: {region.id}, Nombre: {region.name}")

def add_region(regions):
    """Prompts the user for region information and creates a new region."""
    name = input("Ingrese nombre de la región: ")
    new_region_id = max([r.id for r in regions]) + 1
    new_region = {"id": new_region_id, "name": name}
    regions.append(new_region)
    print(f"Región '{new_region['name']}' creada con éxito.")

def edit_region(regions):
    """Prompts the user for a region ID and new information to update a region."""
    list_regions(regions)
    region_id = int(input("Ingrese el ID de la región a editar: "))
    name = input("Ingrese nuevo nombre: ")
    for region in regions:
        if region.id == region_id:
            region.name = name
            print(f"Región '{region.name}' actualizada con éxito.")
            return
    print("Región no encontrada.")

def remove_region(regions):
    """Prompts the user for a region ID to delete a region."""
    list_regions(regions)
    region_id = int(input("Ingrese el ID de la región a eliminar: "))
    initial_len = len(regions)
    regions[:] = [region for region in regions if region.id != region_id]
    if len(regions) < initial_len:
        print("Región eliminada con éxito.")
    else:
        print("Región no encontrada.")

def region_maintainer(data):
    """Shows the region maintainer menu and handles user input."""
    while True:
        print("\n--- Mantenedor de Regiones ---")
        print("1. Listar regiones")
        print("2. Agregar región")
        print("3. Editar región")
        print("4. Eliminar región")
        print("5. Volver al menú principal")
        choice = input("Seleccione una opción: ")

        if choice == "1":
            list_regions(data["regions"])
        elif choice == "2":
            add_region(data["regions"])
        elif choice == "3":
            edit_region(data["regions"])
        elif choice == "4":
            remove_region(data["regions"])
        elif choice == "5":
            break
        else:
            print("Opción no válida. Intente de nuevo.")
```

src/menu/menu.py

```
RED = "\033[91m"
GREEN = "\033[92m"
YELLOW = "\033[93m"
BLUE = "\033[94m"
MAGENTA = "\033[95m"
CYAN = "\033[96m"
RESET = "\033[0m"  # Reset to default color

from mock_data import load_mock_data
from reports.payment_by_code import query_payment_by_code
from reports.payments_by_condo_month_year import list_payments_by_condo_month_year
from reports.total_collected_by_condo import total_collected_by_condo
from reports.total_paid_by_client_house import total_paid_by_client_house
from reports.total_paid_by_client_department import total_paid_by_client_department
from reports.total_collected_by_condo_month_year import total_collected_by_condo_month_year

def show_main_menu():
    print("Menú Principal")
    print("1. Mantenedores")
    print("2. Reportes")
    print("3. Salir")

def show_maintainers_menu():
    print("Submenú Mantenedores")
    print("1. Clientes")
    print("2. Tipos de Clientes")
    print("3. Comunas")
    print("4. Condominios")
    print("5. Tipos de Casas")
    print("6. Casas")
    print("7. Meses de Pago")
    print("8. Tipos de Pago")
    print("9. Años de Pago")
    print("10. Pagos")
    print("11. Regiones")
    print("12. Volver al Menú Principal")

def show_reports_menu():
    print("Submenú Reportes")
    print("1. Consultar pago de gastos comunes por código")
    print("2. Listar los pagos de gastos comunes por condominio, mes, año.")
    print("3. El monto total de dinero recaudado por los condominios (individual y global)")
    print("4. El monto total de dinero pagados por los clientes con casa (individual y global)")
    print("5. El monto total de dinero pagados por los clientes con departamento (individual y global)")
    print("6. El monto total de dinero recaudado por los condominios por mes y año (casa y
departamento)")
    print("7. Volver al Menú Principal")

def main():
    data = load_mock_data()
    while True:
        show_main_menu()
        choice = input("Seleccione una opción: ")

        if choice == "1":
            while True:
                show_maintainers_menu()
                sub_choice = input("Seleccione una opción: ")
                if sub_choice == "1":
                    from maintainers.client_maintainer import client_maintainer
                    client_maintainer(data)
                elif sub_choice == "2":
```

```
                from maintainers.client_type_maintainer import client_type_maintainer
                client_type_maintainer(data)
            elif sub_choice == "3":
                from maintainers.commune_maintainer import commune_maintainer
                commune_maintainer(data)
            elif sub_choice == "4":
                from maintainers.condo_maintainer import condo_maintainer
                condo_maintainer(data)
            elif sub_choice == "5":
                from maintainers.house_type_maintainer import house_type_maintainer
                house_type_maintainer(data)
            elif sub_choice == "6":
                from maintainers.house_maintainer import house_maintainer
                house_maintainer(data)
            elif sub_choice == "7":
                from maintainers.payment_month_maintainer import payment_month_maintainer
                payment_month_maintainer(data)
            elif sub_choice == "8":
                from maintainers.payment_type_maintainer import payment_type_maintainer
                payment_type_maintainer(data)
            elif sub_choice == "9":
                from maintainers.payment_year_maintainer import payment_year_maintainer
                payment_year_maintainer(data)
            elif sub_choice == "10":
                from maintainers.payments_maintainer import payment_maintainer
                payment_maintainer(data)
            elif sub_choice == "11":
                from maintainers.region_maintainer import region_maintainer
                region_maintainer(data)
            elif sub_choice == "12":
                break
            print("="*25)
            print("seleccione opcion valida\n")
            print("="*25)
    elif choice == "2":
        while True:
            show_reports_menu()
            sub_choice = input("Seleccione una opción: ")
            if sub_choice == "1":
                query_payment_by_code(data)
            elif sub_choice == "2":
                list_payments_by_condo_month_year(data)
            elif sub_choice == "3":
                total_collected_by_condo(data)
            elif sub_choice == "4":
                total_paid_by_client_house(data)
            elif sub_choice == "5":
                total_paid_by_client_department(data)
            elif sub_choice == "6":
                total_collected_by_condo_month_year(data)
            elif sub_choice == "7":
                break
    elif choice == "3":
        break
    else:
        print("Opción no válida. Intente de nuevo.")

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

src/modules/client.py

```
from dataclasses import dataclass, field
from typing import List

@dataclass
class Client:
    """Represents a client that can own or rent properties."""

    id: int
    rut: str
    full_name: str
    email: str
    phone: str

clients: List[Client] = []
next_client_id = 1

def create_client(rut: str, full_name: str, email: str, phone: str) -> Client:
    """Creates a new client and adds it to the in-memory list."""
    global next_client_id
    new_client = Client(id=next_client_id, rut=rut, full_name=full_name, email=email, phone=phone)
    clients.append(new_client)
    next_client_id += 1
    return new_client

def read_clients() -> List[Client]:
    """Returns the list of all clients."""
    return clients

def update_client(client_id: int, rut: str, full_name: str, email: str, phone: str) -> Client | None:
    """Updates a client's information."""
    for client in clients:
        if client.id == client_id:
            client.rut = rut
            client.full_name = full_name
            client.email = email
            client.phone = phone
            return client
    return None

def delete_client(client_id: int) -> bool:
    """Deletes a client from the in-memory list."""
    global clients
    initial_len = len(clients)
    clients = [client for client in clients if client.id != client_id]
    return len(clients) < initial_len
```

src/modules/client_type.py

```
from dataclasses import dataclass
from typing import List

@dataclass
class ClientType:
    """Represents a type of client."""

    id: int
    name: str

client_types: List[ClientType] = []
next_client_type_id = 1

def create_client_type(name: str) -> ClientType:
    """Creates a new client type and adds it to the in-memory list."""
    global next_client_type_id
    new_client_type = ClientType(id=next_client_type_id, name=name)
    client_types.append(new_client_type)
    next_client_type_id += 1
    return new_client_type

def read_client_types() -> List[ClientType]:
    """Returns the list of all client types."""
    return client_types

def update_client_type(client_type_id: int, name: str) -> ClientType | None:
    """Updates a client type's information."""
    for client_type in client_types:
        if client_type.id == client_type_id:
            client_type.name = name
            return client_type
    return None

def delete_client_type(client_type_id: int) -> bool:
    """Deletes a client type from the in-memory list."""
    global client_types
    initial_len = len(client_types)
    client_types = [client_type for client_type in client_types if client_type.id != client_type_id]
    return len(client_types) < initial_len
```

src/modules/commune.py

```
from dataclasses import dataclass
from typing import List

@dataclass
class Commune:
    """Represents a local administrative division that groups condos."""

    id: int
    name: str
    region_id: int

communes: List[Commune] = []
next_commune_id = 1

def create_commune(name: str, region_id: int) -> Commune:
    """Creates a new commune and adds it to the in-memory list."""
    global next_commune_id
    new_commune = Commune(id=next_commune_id, name=name, region_id=region_id)
    communes.append(new_commune)
    next_commune_id += 1
    return new_commune

def read_communes() -> List[Commune]:
    """Returns the list of all communes."""
    return communes

def update_commune(commune_id: int, name: str, region_id: int) -> Commune | None:
    """Updates a commune's information."""
    for commune in communes:
        if commune.id == commune_id:
            commune.name = name
            commune.region_id = region_id
            return commune
    return None

def delete_commune(commune_id: int) -> bool:
    """Deletes a commune from the in-memory list."""
    global communes
    initial_len = len(communes)
    communes = [commune for commune in communes if commune.id != commune_id]
    return len(communes) < initial_len
```

src/modules/condo.py

```
from dataclasses import dataclass
from typing import List

@dataclass
class Condo:
    """Represents a condo building that contains multiple houses."""

    id: int
    street: str
    number: str
    name: str
    commune_id: int

condos: List[Condo] = []
next_condo_id = 1

def create_condo(street: str, number: str, name: str, commune_id: int) -> Condo:
    """Creates a new condo and adds it to the in-memory list."""
    global next_condo_id
    new_condo = Condo(id=next_condo_id, street=street, number=number, name=name, commune_id=commune_id)
    condos.append(new_condo)
    next_condo_id += 1
    return new_condo

def read_condos() -> List[Condo]:
    """Returns the list of all condos."""
    return condos

def update_condo(condo_id: int, street: str, number: str, name: str, commune_id: int) -> Condo | None:
    """Updates a condo's information."""
    for condo in condos:
        if condo.id == condo_id:
            condo.street = street
            condo.number = number
            condo.name = name
            condo.commune_id = commune_id
            return condo
    return None

def delete_condo(condo_id: int) -> bool:
    """Deletes a condo from the in-memory list."""
    global condos
    initial_len = len(condos)
    condos = [condo for condo in condos if condo.id != condo_id]
    return len(condos) < initial_len
```

src/modules/house.py

```
from dataclasses import dataclass
from typing import List

@dataclass
class House:
    """Represents an individual housing unit within a condo complex."""

    id: int
    street: str
    number: str
    type_id: int
    condo_id: int
    client_id: int

houses: List[House] = []
next_house_id = 1

def create_house(street: str, number: str, type_id: int, condo_id: int, client_id: int) -> House:
    """Creates a new house and adds it to the in-memory list."""
    global next_house_id
    new_house = House(id=next_house_id, street=street, number=number, type_id=type_id,
condo_id=condo_id, client_id=client_id)
    houses.append(new_house)
    next_house_id += 1
    return new_house

def read_houses() -> List[House]:
    """Returns the list of all houses."""
    return houses

def update_house(house_id: int, street: str, number: str, type_id: int, condo_id: int, client_id: int) -
> House | None:
    """Updates a house's information."""
    for house in houses:
        if house.id == house_id:
            house.street = street
            house.number = number
            house.type_id = type_id
            house.condo_id = condo_id
            house.client_id = client_id
            return house
    return None

def delete_house(house_id: int) -> bool:
    """Deletes a house from the in-memory list."""
    global houses
    initial_len = len(houses)
    houses = [house for house in houses if house.id != house_id]
    return len(houses) < initial_len
```

src/modules/house_type.py

```
from dataclasses import dataclass
from typing import List

@dataclass
class HouseType:
    """Classifies a house according to its type (e.g., apartment, duplex)."""

    id: int
    name: str

house_types: List[HouseType] = []
next_house_type_id = 1

def create_house_type(name: str) -> HouseType:
    """Creates a new house type and adds it to the in-memory list."""
    global next_house_type_id
    new_house_type = HouseType(id=next_house_type_id, name=name)
    house_types.append(new_house_type)
    next_house_type_id += 1
    return new_house_type

def read_house_types() -> List[HouseType]:
    """Returns the list of all house types."""
    return house_types

def update_house_type(house_type_id: int, name: str) -> HouseType | None:
    """Updates a house type's information."""
    for house_type in house_types:
        if house_type.id == house_type_id:
            house_type.name = name
            return house_type
    return None

def delete_house_type(house_type_id: int) -> bool:
    """Deletes a house type from the in-memory list."""
    global house_types
    initial_len = len(house_types)
    house_types = [house_type for house_type in house_types if house_type.id != house_type_id]
    return len(house_types) < initial_len
```

src/modules/payments.py

```
from dataclasses import dataclass
from typing import List

@dataclass
class Payments:
    """Stores billing information for a specific property and client."""

    id: int
    id_client: int
    id_house: int
    payment_year_id: int
    payment_month_id: int
    payment_type: int
    amount: float
    description: str

payments: List[Payments] = []
next_payment_id = 1

def create_payment(id_client: int, id_house: int, payment_year_id: int, payment_month_id: int,
payment_type: int, amount: float, description: str) -> Payments:
    """Creates a new payment and adds it to the in-memory list."""
    global next_payment_id
    new_payment = Payments(id=next_payment_id, id_client=id_client, id_house=id_house,
payment_year_id=payment_year_id, payment_month_id=payment_month_id, payment_type=payment_type,
amount=amount, description=description)
    payments.append(new_payment)
    next_payment_id += 1
    return new_payment

def read_payments() -> List[Payments]:
    """Returns the list of all payments."""
    return payments

def update_payment(payment_id: int, id_client: int, id_house: int, payment_year_id: int,
payment_month_id: int, payment_type: int, amount: float, description: str) -> Payments | None:
    """Updates a payment's information."""
    for payment in payments:
        if payment.id == payment_id:
            payment.id_client = id_client
            payment.id_house = id_house
            payment.payment_year_id = payment_year_id
            payment.payment_month_id = payment_month_id
            payment.payment_type = payment_type
            payment.amount = amount
            payment.description = description
            return payment
    return None

def delete_payment(payment_id: int) -> bool:
    """Deletes a payment from the in-memory list."""
    global payments
    initial_len = len(payments)
    payments = [payment for payment in payments if payment.id != payment_id]
    return len(payments) < initial_len
```

src/modules/payment_month.py

```
from dataclasses import dataclass
from typing import List

@dataclass
class PaymentMonth:
    """Represents the calendar month associated with a payment."""

    id: int
    name: str
    month_number: int

payment_months: List[PaymentMonth] = []
next_payment_month_id = 1

def create_payment_month(name: str, month_number: int) -> PaymentMonth:
    """Creates a new payment month and adds it to the in-memory list."""
    global next_payment_month_id
    new_payment_month = PaymentMonth(id=next_payment_month_id, name=name, month_number=month_number)
    payment_months.append(new_payment_month)
    next_payment_month_id += 1
    return new_payment_month

def read_payment_months() -> List[PaymentMonth]:
    """Returns the list of all payment months."""
    return payment_months

def update_payment_month(payment_month_id: int, name: str, month_number: int) -> PaymentMonth | None:
    """Updates a payment month's information."""
    for payment_month in payment_months:
        if payment_month.id == payment_month_id:
            payment_month.name = name
            payment_month.month_number = month_number
            return payment_month
    return None

def delete_payment_month(payment_month_id: int) -> bool:
    """Deletes a payment month from the in-memory list."""
    global payment_months
    initial_len = len(payment_months)
    payment_months = [payment_month for payment_month in payment_months if payment_month.id !=
payment_month_id]
    return len(payment_months) < initial_len
```

src/modules/payment_type.py

```
from dataclasses import dataclass
from typing import List

@dataclass
class PaymentType:
    """Defines the payment method used for a transaction."""

    id: int
    name: str

payment_types: List[PaymentType] = []
next_payment_type_id = 1

def create_payment_type(name: str) -> PaymentType:
    """Creates a new payment type and adds it to the in-memory list."""
    global next_payment_type_id
    new_payment_type = PaymentType(id=next_payment_type_id, name=name)
    payment_types.append(new_payment_type)
    next_payment_type_id += 1
    return new_payment_type

def read_payment_types() -> List[PaymentType]:
    """Returns the list of all payment types."""
    return payment_types

def update_payment_type(payment_type_id: int, name: str) -> PaymentType | None:
    """Updates a payment type's information."""
    for payment_type in payment_types:
        if payment_type.id == payment_type_id:
            payment_type.name = name
            return payment_type
    return None

def delete_payment_type(payment_type_id: int) -> bool:
    """Deletes a payment type from the in-memory list."""
    global payment_types
    initial_len = len(payment_types)
    payment_types = [payment_type for payment_type in payment_types if payment_type.id !=
payment_type_id]
    return len(payment_types) < initial_len
```

src/modules/payment_year.py

```
from dataclasses import dataclass
from typing import List

@dataclass
class PaymentYear:
    """Represents the calendar year associated with a payment."""

    id: int
    year: int

payment_years: List[PaymentYear] = []
next_payment_year_id = 1

def create_payment_year(year: int) -> PaymentYear:
    """Creates a new payment year and adds it to the in-memory list."""
    global next_payment_year_id
    new_payment_year = PaymentYear(id=next_payment_year_id, year=year)
    payment_years.append(new_payment_year)
    next_payment_year_id += 1
    return new_payment_year

def read_payment_years() -> List[PaymentYear]:
    """Returns the list of all payment years."""
    return payment_years

def update_payment_year(payment_year_id: int, year: int) -> PaymentYear | None:
    """Updates a payment year's information."""
    for payment_year in payment_years:
        if payment_year.id == payment_year_id:
            payment_year.year = year
            return payment_year
    return None

def delete_payment_year(payment_year_id: int) -> bool:
    """Deletes a payment year from the in-memory list."""
    global payment_years
    initial_len = len(payment_years)
    payment_years = [payment_year for payment_year in payment_years if payment_year.id !=
payment_year_id]
    return len(payment_years) < initial_len
```

src/modules/region.py

```
from dataclasses import dataclass
from typing import List

@dataclass
class Region:
    """Represents a geographic region that contains multiple communes."""

    id: int
    name: str

regions: List[Region] = []
next_region_id = 1

def create_region(name: str) -> Region:
    """Creates a new region and adds it to the in-memory list."""
    global next_region_id
    new_region = Region(id=next_region_id, name=name)
    regions.append(new_region)
    next_region_id += 1
    return new_region

def read_regions() -> List[Region]:
    """Returns the list of all regions."""
    return regions

def update_region(region_id: int, name: str) -> Region | None:
    """Updates a region's information."""
    for region in regions:
        if region.id == region_id:
            region.name = name
            return region
    return None

def delete_region(region_id: int) -> bool:
    """Deletes a region from the in-memory list."""
    global regions
    initial_len = len(regions)
    regions = [region for region in regions if region.id != region_id]
    return len(regions) < initial_len
```

src/reports/payments_by_condo_month_year.py

```
def list_payments_by_condo_month_year(data):
    condo_id = int(input("Ingrese el ID del condominio: "))
    month_id = int(input("Ingrese el ID del mes: "))
    year_id = int(input("Ingrese el ID del año: "))

    payments = [p for p in data["payments"]
                if data["houses"][p.id_house - 1].condo_id == condo_id and \
                   p.payment_month_id == month_id and \
                   p.payment_year_id == year_id]

    if payments:
        print("\nPagos encontrados:")
        for p in payments:
            print(f"  - Código: {p.id}, Monto: ${p.amount}")
    else:
        print("\nNo se encontraron pagos para los criterios seleccionados.")
```

src/reports/payment_by_code.py

```
def query_payment_by_code(data):
    code = int(input("Ingrese el ID del pago a consultar: "))
    payment = next((p for p in data["payments"] if p.id == code), None)

    if payment:
        print("\nPago encontrado:")
        print(f"  ID: {payment.id}")
        print(f"  Monto: ${payment.amount}")
        print(f"  Descripción: {payment.description}")
    else:
        print(f"\nNo se encontró ningún pago con el ID '{code}'.")
```

src/reports/total_collected_by_condo.py

```
def total_collected_by_condo(data):
    # Individual
    print("Monto total recaudado por condominio (individual):")
    for condo in data["condos"]:
        total = sum(p.amount for p in data["payments"] if data["houses"][p.id_house - 1].condo_id ==
condo.id)
        print(f"  - {condo.name}: ${total}")

    # Global
    global_total = sum(p.amount for p in data["payments"])
    print(f"\nMonto total recaudado (global): ${global_total}")
```

src/reports/total_collected_by_condo_month_year.py

```
def total_collected_by_condo_month_year(data):
    month_id = int(input("Ingrese el ID del mes: "))
    year_id = int(input("Ingrese el ID del año: "))

    print(f"\nMonto total recaudado por condominio para el mes {month_id} del año {year_id}:")
    for condo in data["condos"]:
        total_casa = sum(p.amount for p in data["payments"]
                         if data["houses"][p.id_house - 1].condo_id == condo.id and \
                            p.payment_month_id == month_id and \
                            p.payment_year_id == year_id and \
                            data["houses"][p.id_house - 1].type_id == 1) # Casa

        total_depto = sum(p.amount for p in data["payments"]
                          if data["houses"][p.id_house - 1].condo_id == condo.id and \
                             p.payment_month_id == month_id and \
                             p.payment_year_id == year_id and \
                             data["houses"][p.id_house - 1].type_id == 2) # Departamento

        print(f"  - {condo.name}:")
        print(f"    - Casas: ${total_casa}")
        print(f"    - Departamentos: ${total_depto}")
```

src/reports/total_paid_by_client_department.py

```
def total_paid_by_client_department(data):
    # Individual
    print("\nMonto total pagado por clientes con departamento (individual):")
    for client in data["clients"]:
        total = sum(p.amount for p in data["payments"]
                    if data["houses"][p.id_house - 1].client_id == client.id and \
                       data["houses"][p.id_house - 1].type_id == 2) # HouseType 'Departamento'
        if total > 0:
            print(f"  - {client.full_name}: ${total}")

    # Global
    global_total = sum(p.amount for p in data["payments"] if data["houses"][p.id_house - 1].type_id ==
2)
    print(f"\nMonto total pagado por clientes con departamento (global): ${global_total}")
```

src/reports/total_paid_by_client_house.py

```
def total_paid_by_client_house(data):
    # Individual
    print("\nMonto total pagado por clientes con casa (individual):")
    for client in data["clients"]:
        total = sum(p.amount for p in data["payments"]
                    if data["houses"][p.id_house - 1].client_id == client.id and \
                       data["houses"][p.id_house - 1].type_id == 1) # HouseType 'Casa'
        if total > 0:
            print(f"  - {client.full_name}: ${total}")

    # Global
    global_total = sum(p.amount for p in data["payments"] if data["houses"][p.id_house - 1].type_id ==
1)
    print(f"\nMonto total pagado por clientes con casa (global): ${global_total}")
```