Actividad ABP: Desarrollo de una Aplicación Multiplataforma con Flet y MongoDB

Contexto y Justificación

La creciente demanda de aplicaciones multiplataforma requiere que los desarrolladores sean capaces de crear soluciones eficientes que puedan ejecutarse en diversos dispositivos y sistemas operativos. Para cumplir este objetivo, tecnologías como Flet, una biblioteca para el desarrollo de interfaces de usuario con Python, y MongoDB, una base de datos documental flexible y escalable, se están convirtiendo en herramientas clave. Este proyecto permitirá a los estudiantes aplicar conocimientos de diseño, programación y bases de datos en un entorno realista y colaborativo.

Enunciado

En grupos de tres estudiantes, deben desarrollar una aplicación multiplataforma utilizando **Flet para Python** como framework para la interfaz de usuario y **MongoDB** como base de datos documental. La aplicación debe incluir las funcionalidades básicas de un sistema CRUD (Crear, Leer, Actualizar, Eliminar) y estará orientada a resolver una necesidad práctica en un contexto específico, que podréis elegir de una lista o proponer ustedes mismos.

Objetivo Principal

Diseñar, desarrollar e implementar una aplicación multiplataforma funcional que permita gestionar datos utilizando una interfaz intuitiva y una base de datos documental.

Requisitos del Proyecto

1. Diseño de la Aplicación

- La aplicación debe tener un diseño de interfaz de usuario intuitivo y atractivo desarrollado con **Flet**.
- Debe incluir:

  - **Pantalla principal** que permita visualizar la lista de elementos almacenados.
  - **Pantallas secundarias** para las operaciones de creación, edición y eliminación.
  - Un sistema de **navegación** claro entre las distintas pantallas.
- Adaptabilidad para su uso en dispositivos de escritorio y móviles: Flet permite la conversión de la aplicación Python a Android de forma relativamente sencilla.

2. Modelo de Datos

- Diseñar un modelo de datos claro y bien estructurado que se almacenará en **MongoDB**.
- Los datos a gestionar deben incluir una validación de datos básica (ej. correos válidos, restricciones de formato para fechas, etc.).
- En el Anexo I de esta práctica podrás encontrar un listado de modelos de entre los que puedes elegir. Queda a la elección del grupo plantear un modelo distinto a los reflejados en el anexo. En cualquier caso, el modelo debe tener al menos tres colecciones distintas y alguna relación entre ellas.
- También es posible elegir alguno de los modelos del Anexo I y adaptarlo.

3. Funcionalidades CRUD

- **Crear**: Un formulario para añadir nuevos registros a la base de datos.
- **Leer**: Una lista que muestre todos los registros almacenados, con opciones de búsqueda o filtrado.
- **Actualizar**: Una interfaz para editar registros existentes.
- **Eliminar**: Confirmación previa para eliminar registros de manera segura.

4. Conexión con la Base de Datos

- Configurar e integrar **MongoDB** como base de datos documental.
- Incluir la lógica necesaria para:
  - Conectarse a la base de datos.
  - Realizar operaciones CRUD mediante consultas eficientes.
  - Manejar posibles errores en la comunicación con la base de datos o en la ejecución de las operaciones CRUD (ej, pérdida de conexión).

5. Código Limpio y Documentado

- Seguir las buenas prácticas de programación:
  - Uso de funciones/métodos claros y reutilizables.
  - Organización del código en módulos. Ver Anexo II.
- Comentar las secciones clave del código para facilitar su comprensión.

- Incluir un archivo README.md que explique:
  - Objetivo de la aplicación.
  - Cómo instalar y ejecutar la aplicación.
  - Dependencias necesarias.

6. Presentación del Proyecto

- Preparar una presentación de 10 minutos que incluya:
  - Descripción del problema que resuelve la aplicación.
  - Demostración de las principales funcionalidades.
  - Dificultades encontradas y soluciones implementadas.
  - Reparto de tareas dentro del equipo.

Criterios de Evaluación

El proyecto será evaluado sobre un total de **100 puntos**, distribuidos de la siguiente manera:

1. **Diseño de la interfaz de usuario (20 puntos)**:
   - Facilidad de uso.
   - Estética visual y organización de la información.
2. **Modelo de datos y validación (15 puntos)**:
   - Claridad y coherencia del modelo de datos.
   - Implementación de validaciones.
3. **Funcionalidades CRUD (25 puntos)**:
   - Correcta implementación de todas las operaciones CRUD.
   - Manejo de errores en las operaciones.
4. **Conexión con MongoDB (20 puntos)**:
   - Eficiencia en las consultas.
   - Manejo de la conexión y seguridad de los datos.
5. **Calidad del código (10 puntos)**:
   - Limpieza, organización y uso de buenas prácticas.
   - Calidad de los comentarios y documentación.
6. **Presentación del proyecto (10 puntos)**:
   - Claridad y profesionalismo en la exposición.
   - Calidad de la demostración.

Dinámica del Trabajo

1. **Formación de Grupos**:
   - Los grupos serán de tres integrantes.
   - Cada miembro asumirá un rol principal: Diseñador de la interfaz, Desarrollador backend, y Gestor de integración (aunque todos deben colaborar en todas las áreas).
   - Cualquier otra disposición que los miembros del grupo consideren.
2. **Fases del Proyecto**:
   - **Fase 1: Análisis y planificación** (1 semana):
     - Selección del contexto y definición de requisitos específicos.
     - Diseño preliminar del modelo de datos y la interfaz.
   - **Fase 2: Desarrollo** (2 semanas):
     - Implementación del backend (CRUD y conexión con MongoDB).
     - Creación de la interfaz con Flet.
   - **Fase 3: Pruebas e integración** (1 semana):
     - Pruebas de funcionalidad y resolución de errores.
     - Finalización y preparación para la presentación.
   - **Fase 4: Presentación** (última semana):
     - Presentación del proyecto al resto del grupo y evaluación.
3. **Sesiones de Seguimiento**:
   - La profesora llevará el seguimiento de los avances de cada uno de los grupos durante las sesiones de clase. Dejará constancia mediante registro de los avances del grupo en cada una de las etapas del proyecto.
   - Durante estas sesiones el grupo la puede consultar para aclarar dudas, comentar incidencias, proponer modificaciones, etc.
   - Se llevará un control exhaustivo de las faltas de asistencia para determinar, en caso necesario, la intervención efectiva de cada uno de los miembros de los grupos en el proyecto.

Material de Apoyo

- **Documentación de Flet**: https://flet.dev
- **Documentación de MongoDB**: https://www.mongodb.com/docs
- **Apuntes de BAE - 1º DAM sobre MongoDB (en el Campus).**
- **Curso completo Flet** Curso Completo de Flet

Biblioteca pymongo:

```
pip install pymongo dnspython
```

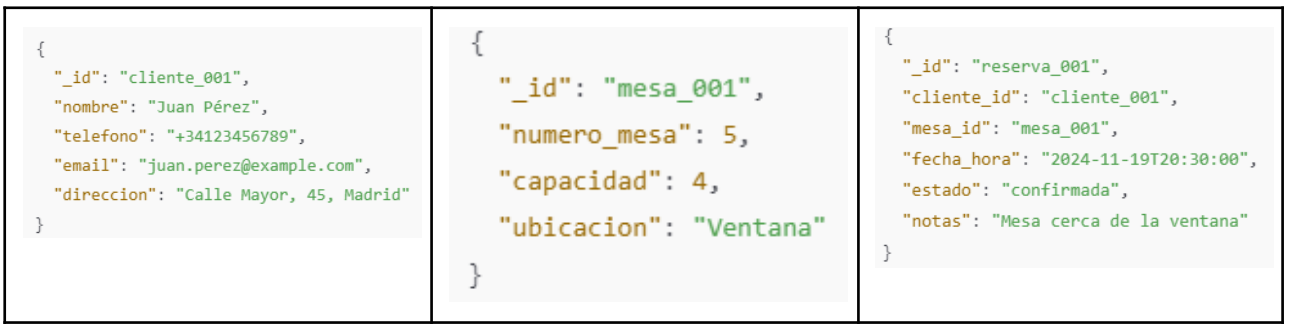
Productos finales a entregar:

- Código del proyecto (subido a un repositorio GitHub privado).
- Archivo README.md detallado.
- Presentación en formato PDF o PowerPoint.

ANEXO I - MODELOS DE DATOS

1. Gestión de Reservas en un Restaurante

Contexto: Gestión de clientes, mesas y reservas.



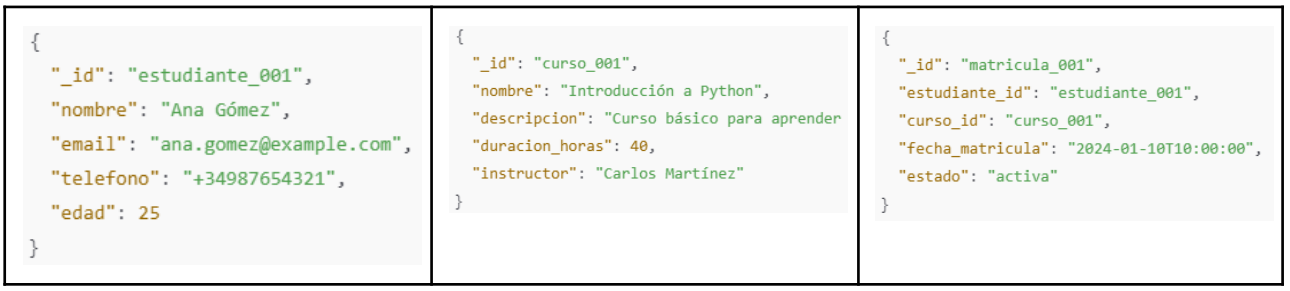
```
{
  "_id": "cliente_001",
  "nombre": "Juan Pérez",
  "telefono": "+34123456789",
  "email": "juan.perez@example.com",
  "direccion": "Calle Mayor, 45, Madrid"
}
```

```
{
  "_id": "mesa_001",
  "numero_mesa": 5,
  "capacidad": 4,
  "ubicacion": "Ventana"
}
```

```
{
  "_id": "reserva_001",
  "cliente_id": "cliente_001",
  "mesa_id": "mesa_001",
  "fecha_hora": "2024-11-19T20:30:00",
  "estado": "confirmada",
  "notas": "Mesa cerca de la ventana"
}
```

2. Sistema de Gestión de Cursos Online

Contexto: Gestión de estudiantes, cursos y matrículas.



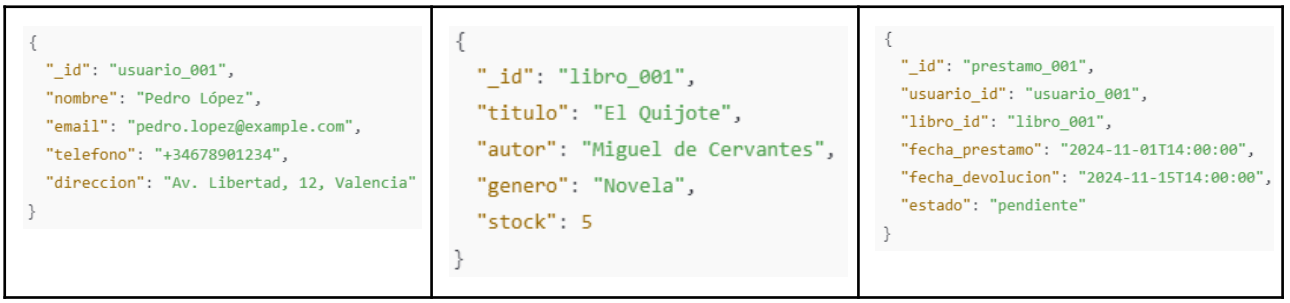
```
{
  "_id": "estudiante_001",
  "nombre": "Ana Gómez",
  "email": "ana.gomez@example.com",
  "telefono": "+34987654321",
  "edad": 25
}
```

```
{
  "_id": "curso_001",
  "nombre": "Introducción a Python",
  "descripcion": "Curso básico para aprender
  "duracion_horas": 40,
  "instructor": "Carlos Martínez"
}
```

```
{
  "_id": "matricula_001",
  "estudiante_id": "estudiante_001",
  "curso_id": "curso_001",
  "fecha_matricula": "2024-01-10T10:00:00",
  "estado": "activa"
}
```

3. Sistema de Gestión de Biblioteca

Contexto: Gestión de usuarios, libros y préstamos.



```
{
  "_id": "usuario_001",
  "nombre": "Pedro López",
  "email": "pedro.lopez@example.com",
  "telefono": "+34678901234",
  "direccion": "Av. Libertad, 12, Valencia"
}
```

```
{
  "_id": "libro_001",
  "titulo": "El Quijote",
  "autor": "Miguel de Cervantes",
  "genero": "Novela",
  "stock": 5
}
```

```
{
  "_id": "prestamo_001",
  "usuario_id": "usuario_001",
  "libro_id": "libro_001",
  "fecha_prestamo": "2024-11-01T14:00:00",
  "fecha_devolucion": "2024-11-15T14:00:00",
  "estado": "pendiente"
}
```

4. Plataforma de Comercio Electrónico

Contexto: Gestión de usuarios, productos y pedidos.

<table>
<tr>
<td>

```
{
  "_id": "usuario_001",
  "nombre": "Laura Fernández",
  "email": "laura.fernandez@example.com",
  "direccion": "Calle Flores, 23, Sevilla",
  "telefono": "+34567890123"
}
```

</td>
<td>

```
{
  "_id": "producto_001",
  "nombre": "Portátil Lenovo",
  "descripcion": "Portátil de 15 pulgadas
  "precio": 599.99,
  "stock": 10
}
```

</td>
<td>

```
{
  "_id": "pedido_001",
  "usuario_id": "usuario_001",
  "productos": [
    { "producto_id": "producto_001", "cantidad": 1 }
  ],
  "total": 599.99,
  "fecha_pedido": "2024-11-18T12:00:00",
  "estado": "enviado"
}
```

</td>
</tr>
</table>

5. Sistema de Gestión de Clínicas

Contexto: Gestión de pacientes, médicos y citas.

<table>
<tr>
<td>

```
{
  "_id": "paciente_001",
  "nombre": "Carlos Ruiz",
  "email": "carlos.ruiz@example.com",
  "telefono": "+34123456789",
  "direccion": "Calle Salud, 8, Granada"
}
```

</td>
<td>

```
{
  "_id": "medico_001",
  "nombre": "Dra. María Sánchez",
  "especialidad": "Dermatología",
  "telefono": "+34987654321",
  "email": "maria.sanchez@example.com"
}
```

</td>
<td>

```
{
  "_id": "cita_001",
  "paciente_id": "paciente_001",
  "medico_id": "medico_001",
  "fecha_hora": "2024-12-01T10:30:00",
  "motivo": "Revisión de lunares",
  "estado": "confirmada"
}
```

</td>
</tr>
</table>

ANEXO II. ORGANIZACIÓN EN MÓDULOS

Cuando hablo de la organización del código en módulos, me refiero a una práctica común en desarrollo de software que consiste en dividir el código de una aplicación en **archivos y paquetes más pequeños y específicos**. Esto facilita el mantenimiento, mejora la legibilidad, permite la reutilización de componentes y hace que la colaboración entre varios desarrolladores sea más sencilla.

¿Qué es un módulo en Python?

Un **módulo** en Python es simplemente un archivo con extensión `.py` que contiene definiciones de funciones, clases, variables, y código relacionado. Puedes importar estos módulos en otros archivos para usar sus funcionalidades.

Ventajas de Usar Módulos

- **Código más organizado**:
  - Evitas tener todo el código en un solo archivo enorme (ej., `app.py`).
  - Cada archivo tiene una función o rol específico.
- **Reutilización**:
  - Puedes importar módulos en otros proyectos o partes de la aplicación.
- **Facilidad de mantenimiento**:
  - Si necesitas cambiar una funcionalidad, solo modificas el archivo correspondiente.
- **Colaboración**:
  - En proyectos en equipo, diferentes desarrolladores pueden trabajar en módulos separados.

En prácticas anteriores ya se ha ido trabajando al menos con archivos separados para determinadas funcionalidades,por ejemplo para la conexión de la base de datos, la creación de los modelos y las operaciones CRUD por separado; no obstante, no podría considerarse trabajo en módulos. Les dejo un ejemplo o propuesta que pueden implementar en este proyecto:

```
project/
│
├── app.py                  # Archivo principal para iniciar la aplicación
├── models/                 # Directorio para modelos de datos
│   ├── __init__.py         # Hace que `models` sea un paquete
│   ├── user_model.py       # Modelo para usuarios
│   ├── reservation_model.py # Modelo para reservas
│
├── services/               # Directorio para lógica de negocio y acceso a la BD
│   ├── __init__.py         # Hace que `services` sea un paquete
│   ├── mongo_service.py    # Módulo para conexión a MongoDB
│   ├── crud_operations.py  # Módulo para operaciones CRUD
│
```

```
├── views/                  # Directorio para componentes de interfaz de usuario (UI)
│   ├── __init__.py         # Hace que `views` sea un paquete
│   ├── main_view.py        # Vista principal de la aplicación
│   ├── form_view.py        # Vista del formulario (crear/editar)
│
└── utils/                  # Directorio para utilidades y configuraciones
    ├── __init__.py         # Hace que `utils` sea un paquete
    ├── config.py           # Archivo de configuración (MongoDB URI, claves, etc.)
    ├── validators.py       # Funciones para validar datos
```

Los frameworks para APIS, como por ejemplo Django, también trabajan bajo esta estructura, por tanto, ya estarás familiarizado con ella cuando abordemos ese concepto.

models/:

- Contiene los modelos de datos que definen la estructura de los documentos de MongoDB.
- Ejemplo: `user_model.py`:

```python
from pydantic import BaseModel
class UserModel(BaseModel):
    nombre: str
    email: str
    telefono: str
```

services/:

- Contiene la lógica para interactuar con la base de datos.
- Ejemplo: `mongo_service.py`:

```python
from pymongo import MongoClient
def get_db():
    client = MongoClient("mongodb://localhost:27017/")
    db = client['mi_base_de_datos']
    return db
```

- Ejemplo: `crud_operations.py`:

```python
from services.mongo_service import get_db
def create_user(user_data):
    db = get_db()
    db.usuarios.insert_one(user_data)
```

views/:

- Contiene las vistas de la aplicación creadas con **Flet**. Cada vista se encarga de una parte específica de la UI.
- Ejemplo: `main_view.py`

```
import flet as ft
def main_view(page: ft.Page):
    page.title = "Aplicación Multiplataforma"
    page.add(ft.Text("Bienvenido a la aplicación"))
```

utils/:

- Contiene utilidades como configuraciones, validaciones o funciones auxiliares.
- Ejemplo: `config.py`

```
MONGO_URI = "mongodb://localhost:27017/"
```

- Ejemplo: `validators.py`

```
def validate_email(email: str):
    if "@" not in email:
        raise ValueError("Email no válido")
```

Archivo principal `app.py`:

- Es el punto de entrada de la aplicación. Se encarga de iniciar la UI y coordinar la lógica entre los módulos.

```
import flet as ft
from views.main_view import main_view

def main(page: ft.Page):
    main_view(page)

if __name__ == "__main__":
    ft.app(target=main)
```