Examen Unidad 1 - Enunciado

Consigna

Dejar funcionando una API pública que proporcione una serie de endpoints para gestionar pagos en línea. Los usuarios pueden realizar pagos utilizando diferentes métodos, y cada método de pago tiene una lógica de validación y ejecución específica. Además, la API soporta la funcionalidad para cancelar pagos antes de su confirmación.

EndPoints de la API

- GET /payments
    - Description: Obtiene todos los pagos del sistema.
    - Arguments: Ninguno.

- POST /payments/{payment_id}
    - Description: Registra un pago con su información.
    - Arguments:
        - payment_id (path, string)
        - amount (query, number)
        - payment_method (query, string)

- POST /payments/{payment_id}/update
    - Description: Actualiza la información de un pago existente.
    - Method: POST
    - Arguments: payment_id (path, string), amount (query, number), payment_method (query, string).

- POST /payments/{payment_id}/pay
    - Description: Marca un pago como pagado.
    - Arguments: payment_id (path, string).

- POST /payments/{payment_id}/revert
    - Description: Revierte un pago al estado registrado.
    - Arguments: payment_id (path, string).

Aclaración respecto a los argumentos

Ej: https://localhost/api/**{arg_1}**/endpoint?**arg_2={arg_2}**

- **arg_1** es un argumento que se pasa por path
- **arg_2** en cambio se pasa por query

Más abajo en el código de referencia pueden ver cómo se llevaría esto a la práctica.

Flujo de Pago

1. **Registrar**: El usuario registra un pago con un id, monto y un método de pago. Este pago se crea en estado ”REGISTRADO”.
2. **Pagar**: El sistema valida el pago en base al método seleccionado. Si la validación es exitosa, el pago pasa al estado “PAGADO” sino al estado “FALLIDO”.
3. **Revertir**: Cambia un pago en estado FALLIDO a estado REGISTRADO.
4. **Updatear**: Cambia los atributos de un pago si y solo si está en estado REGISTRADO.

Lógica de Validación

Método de Pago 1: Tarjeta de Crédito

- **Condiciones**:
    - Verifica que el pago sea menor a $ 10.000.
    - Valida que no haya más de 1 pago con este medio de pago en estado “REGISTRADO”.

Método de Pago 2: PayPal

- **Condiciones**:
    - Verifica que el pago sea menor de $ 5000

Testeo de Integración

Para probar el sistema end to end se recomienda utilizar la UI de debugging accesible por http://127.0.0.1:8000/docs cuando corran el desarrollo utilizando “`fastapi dev main.py`”. Notar que esto es distinto de los unit tests que se piden más adelante.

Algunas pruebas a realizar:

- Probar el flujo normal de un pago válido.
- Probar que las validaciones de paypal y tarjeta de crédito sean correctas.
- Verificar si es posible revertir un pago fallido, alterarla mediante la API y reintentar exitosamente su pago.

Requerimientos para aprobar:

- Ambos miembros del equipo trabajarán en el mismo repositorio y deberán colaborar al mismo tiempo mediante Pull Requests como se vio en clase.
- Cada miembro debe enviar al menos 1 PR sobre el cual el otro miembro del equipo tendrá que revisar e incluir comentarios y observaciones que crea relevante.
- Una vez finalizado el desarrollo se compartirá el repositorio para ser revisado por el docente.
- En el repositorio se deberá incluir un archivo README.md en el cual se debe incluir:
    - Instrucciones para correr el desarrollo y los tests por línea de comando.
    - URL donde quedó funcionando la API.
    - Una sección explicando:
        - Decisiones de diseño de la solución, incluir trade-offs entre distintas opciones consideradas. (ej: ¿Por qué se eligió escribir los tests de tal o cual manera y qué dejó afuera y por qué?)
        - Suposiciones asumidas (ej: ¿Cuál es el flujo de validación correcto?)
        - Diagramas que les parezcan razonables para visualizar, estructura, flujo de información, transiciones de estados o cualquier otro que les parezca razonable.
        - Demás puntos que crean relevantes que demuestre comprensión y aplicación de los tópicos vistos en la materia
- Usando github actions implementar:
    - Al menos 3 tests automáticos en cada PR hacia el branch main. Los mismos deberán estar visibles en cada PR. (Notar que no necesitan ser tests de integración como los provistos).
    - Deployment automático al host cuando se haga un merge en el branch “production”.

- Si bien **está permitido utilizar herramientas de IA** tengan en cuenta que forma parte de la evaluación que demuestren comprensión de los temas no solo a nivel de implementación de la consigna sino también en la medida que puedan dejas asentadas observaciones significativas en forma de la documentación del proyecto.
- Para la entrega hay que usar este formulario: https://docs.google.com/forms/d/e/1FAIpQLSc-SxnPk62u2TFwCtC1bfTA6tSUyHeCnQ7HVNCZtStKnC8qzQ/viewform?usp=sharing&ouid=115698548409582664084

Código de Referencia

Con el fin de que puedan enfocarse en desarrollar en este examen los temas vistos en la materia les dejamos un punto de partida:

```python
import json

from fastapi import FastAPI

STATUS = "status"
AMOUNT = "amount"
PAYMENT_METHOD = "payment_method"

STATUS_REGISTRADO = "REGISTRADO"
STATUS_PAGADO = "PAGADO"
STATUS_FALLIDO = "FALLIDO"

DATA_PATH = "data.json"

app = FastAPI()


def load_all_payments():
    with open(DATA_PATH, "r") as f:
        data = json.load(f)
    return data


def save_all_payments(data):
    with open(DATA_PATH, "w") as f:
        json.dump(data, f, indent=4)


def load_payment(payment_id):
    data = load_all_payments()[payment_id]
    return data


def save_payment_data(payment_id, data):
    all_data = load_all_payments()
    all_data[str(payment_id)] = data
    save_all_payments(all_data)


def save_payment(payment_id, amount, payment_method, status):
    data = {
        AMOUNT: amount,
        PAYMENT_METHOD: payment_method,
```

```
        STATUS: status,
    }
    save_payment_data(payment_id, data)


"""
# Ejemplo de uso:
# Actualizando el status de un pago:
data = load_payment(payment_id)
data[STATUS] = STATUS_PAGADO
save_payment_data(payment_id, data)
"""



# Endpoints a implementar:
# * GET en el path /payments que retorne todos los pagos.
# * POST en el path /payments/{payment_id} que registre un nuevo pago.
# * POST en el path /payments/{payment_id}/update que cambie los parametros de una pago (amount,
payment_method)
# * POST en el path /payments/{payment_id}/pay que intente.
# * POST en el path /payments/{payment_id}/revert que revertir el pago.


"""
# Ejemplos:

@app.get("/path/{arg_1}")
async def endpoint_a(arg_1: str, arg_2: float):
    # Este es un endpoint GET que recibe un argumento (arg_1) por path y otro por query (arg_2).
    return {}

@app.post("/path/{arg_1}/some_action")
async def endpoint_b(arg_1: str, arg_2: float, arg_3: str):
    # Este es un endpoint POST que recibe un argumento (arg_1) por path y otros dos por query (arg_2 y
arg_3).
    return {}
"""
```