Bases de Datos II (2024) - Tarea 2

Diego Alvarez S -- 29/10/2024

Objetivo

Aplicar conocimientos de desarrollo de APIs REST utilizando Litestar y SQLAlchemy.

Descripción

La construcción de esta API aborda una aplicación de gastos compartidos. Es decir, permite el registro de compras realizadas en un grupo de personas, donde una paga el monto total y el resto le paga a él.

En el repositorio github.com/dialvarezs/hw2-bd2-2024 encontrarás el código base para esta tarea. Ya hay varias funciones implementadas, como la gestión de usuarios y creación básica de gastos. El esquema de datos consiste en:

- `users`: contiene la información de los usuarios: nombre, contraseña, etc.
- `expenses`: contiene los gastos generados. Cada gasto registra quien lo creó (`created_by`), el monto total, la fecha y una descripción.
- `debts`: contiene la información de las deudas asociadas a cada gasto. Cada gasto registra el usuario y gastos asociado, el monto de la deuda y si esta fue pagada.

Al ingresar un nuevo gasto, el usuario que lo crea selecciona qué usuarios forman parte del gasto. El sistema registra el gasto y genera deudas para cada uno de los usuarios por el total dividido el número de usuarios (contando al que creó el gasto).

Por ejemplo, si Juan crea un gasto de $20.000 en el que participaron Pedro, Catalina y Constanza, el sistema ingresará una deuda de $5.000 a Pedro, Catalina y Constanza. La petición en formato JSON para esta petición debería parecido a lo siguiente (asumiendo que Pedro Catalina y Constanza tienen IDs 2, 3 y 4):

```
{
  "title": "Pizzas",
  "debts": [
    { "user_id": 2 },
    { "user_id": 3 },
    { "user_id": 4 },
  ],
  "datetime": "2024-10-29T19:07:46",
  "amount": 20000
}
```

Requerimientos

1. **Inicialización de login**

   Crea un usuario inicial para el sistema y intenta iniciar sesión. ¿No funciona el inicio de sesión? Descubre por qué.

   Adicionalmente, añade un nuevo campo a la tabla de usuarios, que registre el último login del usuario (esto campo debe rellenarse automáticamente y no debe ser editable por el usuario).

2. **Cambio de contraseña**

   Implementa un endpoint que permita a un usuario cambiar su propia contraseña. El endpoint debe tener la ruta `/accounts/users/me/change-password`. Este endpoint debe requerir la contraseña actual y la nueva contraseña, y sólo cambiar la contraseña si es que la nueva contraseña no es igual a cualquiera de las últimas 3 contraseñas utilizadas por el usuario.

3. **Actualización de gastos**

   El endpoint para modificar los gastos implementado sólo permite editar detalles básicos (título, descripción, etc), pero no las deudas asociadas. Implementa esta opción.

4. **Pago de deuda**

   Actualmente no existe una forma de pagar deudas. Implementa un endpoint que permita esto. La ruta de esta debe ser `/expenses/expenses/{id}/pay`. El sistema debe reconocer automáticamente al usuario que tiene iniciada sesión, verificar si tiene deuda asociada a ese gasto, y pagarla en caso de que exista tal deuda.

5. **Estado de gastos**

   Agregar el campo `status` a la tabla de gastos. Este campo debe reflejar el estado del gasto, es decir, si tiene deudas pendientes asociadas o no. El estado debe cambiar automáticamente cuando se paguen todas las deudas asociadas al gasto. Queda a elección libre la implementación del campo, pero una buena alternativa puede ser usar Enum.

6. **Información asociada a usuario**

   En la implementación actual, al consultar un usuario mediante `/accounts/users/{id}` se listan todos los gastos creados por el usuario y deudas asociadas a este, independiente de su estado. Modifica la respuesta para que sólo se incluyan las deudas impagas y los gastos con estado pendiente (para esto deberás completar el punto 4 antes). Adicionalmente, implementa los endpoints `/accounts/users/{id}/expenses` y `/accounts/users/{id}/debts`, que deben entregar la lista completa de gastos y deudas asociadas al usuario, respectivamente.

7. **Gastos con montos diferenciados**

   Por defecto, cada gasto es repartido equitativamente entre todos los usuarios asociados. Implementa una opción para permitir que las deudas puedan ser configuradas por porcentaje, por ejemplo que se permita lo siguiente:

   ```
   {
     "title": "Pizzas",
     "debts": [
       { "user_id": 2, "percentage": 50 },
       { "user_id": 3 },
       { "user_id": 4 },
     ],
     "datetime": "2024-10-29T19:07:46",
     "amount": 20000
   }
   ```

   En este caso, el usuario con ID dos pagará el 50% y el resto se repartirá entre lo demás.

8. **Eliminación de gastos**

   Al eliminar un gasto, este se elimina definitivamente. Implementa una opción para que exista una "papelera" de gastos, desde donde puedan ser recuperados.

9. **Actualización de duración de token JWT**

   Modifica las propiedades de la Token JWT para que su duración sea de 3 horas.

10. **Verificación de estado de usuarios**

    Al crear un nuevo gasto, no debe ser posible agregar usuarios que se encuentren desactivados

Evaluación

- Cada uno de los requerimientos entregará 10 puntos al ser desarrollado por completo. En caso de ser desarrollado parcialmente, se evaluará según el grado de avance.

- El orden en el código, uso de buenas prácticas y legibilidad del código se evaluará con un máximo de 5 puntos. Recuerda que usar ruff puede ser útil para esto.
- El uso correcto de tipos se puntuará con un máximo de 5 puntos. Para verificar esto se utilizará la herramienta mypy de la siguiente manera:
  - Ejecución de `mypy` sin errores: 4 puntos.
  - Ejecución de `mypy --strict` sin errores: 5 puntos.
- El código deberá entregarse en un repositorio de GitHub, en el que se pueda ver el historial de commits. En caso de no querer dejar el repositorio público, se puede dar acceso al usuario dialvarezs. Se evaluará el último commit antes de la fecha de entrega. En la plataforma Pregrado Virtual debe indicarse la URL del repositorio. Debe entregarse todo el código necesario para ejecutar el proyecto, incluyendo migraciones de Alembic.
- Si se detecta un alto nivel de similitud entre tareas entregadas, se interrogará a los involucrados sobre su dominio del contenido. En caso de confirmarse plagio, se evaluará con nota mínima.
- Por cada 3 horas de retraso se restará un punto completo de la nota final, es decir:
  - Retraso de 3 horas o menos: nota máxima 6.0
  - Retraso de menos de 6 horas: nota máxima 5.0
  - Etc...