Título	Tarea 2 - API de Gestión de Biblioteca
Curso	Bases de Datos II - Ingeniería en Computación e Informática
Docente	Diego Álvarez S.
Fecha	20/11/2025

UMAG
Universidad de Magallanes

Objetivo

Aplicar conocimientos de desarrollo de APIs con Litestar y SQLAlchemy, implementando migraciones de base de datos con Alembic, configuración de DTOs, operaciones CRUD y lógica de negocio usando el patrón repositorio - controlador.

Descripción

Se debe extender el proyecto base de una API REST para gestión de biblioteca, disponible el repositorio dialvarezs/learning-litestar-bd2-2025. El proyecto actual incluye modelos básicos (`User` , `Book` , `Loan`) con operaciones CRUD simples. El estudiante deberán crear nuevos modelos, actualizar los existentes, configurar migraciones, DTOs apropiados e implementar controladores y repositorios.

Requerimientos

1. Crear el modelo `Category` con relación many-to-many a `Book` :
   - Atributos: `id` , `name: str` (único), `description: str | None`
   - Crear tabla intermedia `book_categories` para la relación many-to-many
   - Configurar adecuadamente las relaciones en ambos modelos
   - Crear migración con Alembic que incluya ambas tablas y sus índices
   - Configurar DTOs (`CategoryReadDTO` , `CategoryCreateDTO` , `CategoryUpdateDTO`) excluyendo correctamente los campos de auditoría y las relaciones
   - Implementar controlador `CategoryController` con operaciones CRUD completas (listar, obtener por ID, crear, actualizar, eliminar)

2. Crear el modelo `Review` para reseñas de libros:
   - Atributos: `id` , `rating: int` , `comment: str` , `review_date: date` , `user_id` , `book_id`
   - Establecer relaciones con `User` y `Book`
   - Crear migración con Alembic
   - Configurar DTOs apropiados, el `ReviewReadDTO` debe incluir las relaciones `user` y `book`
   - Implementar controlador con CRUD básico
   - En el endpoint de creación, validar que el `rating` esté en el rango 1 a 5 y retornar error HTTP 400 si no cumple esta condición

3. Actualizar el modelo `Book` agregando campos de inventario y descripción:
   - Nuevos campos: `stock: int` , `description: str | None` , `language: str` , `publisher: str | None`
   - El campo `stock` debe tener valor por defecto de 1
   - El campo `language` debe usar códigos ISO 639-1 de 2 letras (ej: “es”, “en”, “fr”)
   - Crear migración con Alembic para añadir estos campos a la tabla existente
   - Modificar el endpoint de creación de libros para validar que `stock` sea mayor a 0 al crear
   - Agregar validación en el endpoint de actualización para que `stock` no pueda ser negativo

4. Actualizar el modelo `User`:
   - Nuevos campos: `email: str` (único), `phone: str | None`, `address: str | None`, `is_active: bool` (default `True`)
   - Crear migración con Alembic
   - Actualizar el `UserReadDTO` para excluir correctamente `password` y `loans`
   - En el endpoint de creación y actualización, validar que el email tenga formato válido (usar expresión regular o librería de validación)
   - El campo `is_active` no debe ser modificable directamente por el usuario (excluirlo de los DTOs de creación y actualización)

5. Actualizar el modelo `Loan`:
   - Nuevos campos: `due_date: date`, `fine_amount: Decimal | None` (con precisión de 2 decimales), `status: LoanStatus`
   - Crear enum `LoanStatus` con valores: `ACTIVE`, `RETURNED`, `OVERDUE`
   - Crear migración con Alembic que incluya los nuevos campos y el enum
   - El `status` debe tener valor por defecto `ACTIVE` en el modelo
   - El campo `due_date` debe calcularse en el endpoint de creación como 14 días después de `loan_dt`
   - Actualizar `LoanCreateDTO` para excluir `due_date`, `fine_amount` y `status`, pero incluirlos en el `LoanReadDTO`, y en el `LoanUpdateDTO` solo permitir actualizar `status`

6. Implementar los siguientes métodos en `BookRepository`:
   - `.get_available_books() -> Sequence[Book]`: retornar libros con `stock > 0`
   - `.find_by_category(category_id: int) -> Sequence[Book]`: buscar libros de una categoría
   - `.get_most_reviewed_books(limit: int = 10) -> Sequence[Book]`: libros ordenados por cantidad de reseñas (usar `func.count()` de SQLAlchemy)
   - `.update_stock(book_id: int, quantity: int) -> Book`: actualizar stock de un libro, validar que no quede negativo
   - `.search_by_author(author_name: str) -> Sequence[Book]`: buscar libros por nombre de autor (búsqueda parcial usando `ilike`)

   Crear endpoints en `BookController` que utilicen estos métodos del repositorio

7. Implementar los siguientes métodos en `LoanRepository`:
   - `.get_active_loans() -> Sequence[Loan]`: préstamos con `status == ACTIVE`
   - `.get_overdue_loans() -> Sequence[Loan]`: préstamos con `due_date` pasada y `status == ACTIVE`, actualizar su `status` a `OVERDUE`
   - `.calculate_fine(loan_id: int) -> Decimal`: calcular multa ($500 por día de retraso)
   - `.return_book(loan_id: int) -> Loan`: procesar devolución (actualizar `status` a `RETURNED`, establecer `return_dt` a la fecha actual, calcular y guardar `fine_amount` si corresponde, incrementar `stock`)
   - `.get_user_loan_history(user_id: int) -> Sequence[Loan]`: historial completo de préstamos de un usuario ordenado por fecha

   Crear endpoints en `LoanController` para estas operaciones (por ejemplo: `POST /loans/{id}/return`, `GET /loans/overdue`, `GET /loans/user/{user_id}`)

8. Crear una base de datos inicial con datos de ejemplo e incluir en el proyecto el respaldo de esta base datos con el nombre `initial_data.sql`. Debe contener al menos:
   - 5 categorías (Ficción, No Ficción, Ciencia, Historia, Fantasía)
   - 10 libros distribuidos en las categorías, cuyos ISBNs deben seguir el formato `ISBN-BD2-2025-XXXX` donde XXXX es un número correlativo de 4 dígitos (ej: ISBN-BD2-2025-1234)
   - 5 usuarios con información completa (con contraseñas hasheadas usando argon2)

- 8 préstamos variados (algunos activos, algunos devueltos, algunos vencidos)
- 15 reseñas distribuidas entre los libros y usuarios

TIP: Puedes exportar una base de datos de PostgreSQL usando `pg_dump -Ox <base_de_datos>`.

Formato de Entrega

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 entrega en Pregrado Virtual debe indicarse la URL del repositorio **o no será considerado**.

El proyecto debe estar basado en el repositorio proporcionado como base. Todas las modificaciones deben realizarse sobre la estructura existente. Las migraciones de Alembic deben estar correctamente versionadas y aplicarse sin errores sobre una base de datos PostgreSQL limpia. De no cumplir con estas condiciones básicas de estructura y funcionamiento, se penalizará con un máximo de 10 puntos.

En el repositorio debe actualizarse el `README.md` para que contenga:
- Descripción breve de los cambios realizados y decisiones de diseño
- Cumplimiento de los requerimientos de la tarea mediante una tabla que liste cada requerimiento y su estado (cumplido / no cumplido / parcial), y opcionalmente una observación breve.

Evaluación de la Tarea

Cada uno de los requerimientos entregará **9 puntos** al ser desarrollado por completo. En caso de ser desarrollado parcialmente, se evaluará según el grado de avance.

La correcta aplicación de migraciones, configuración apropiada de DTOs, manejo correcto de relaciones SQLAlchemy, uso de buenas prácticas, tipado correcto y legibilidad del código se evaluará con un máximo de **10 puntos**. Las herramientas `ruff` pueden ser útiles para mejorar este punto.

Para obtener nota máxima se requieren **70 puntos**.

Consideraciones

El código entregado debe ser original y reflejar el entendimiento del estudiante. Serán motivo de evaluación con **nota mínima (1.0)** los siguientes casos:
- Una alta similitud entre dos o más tareas.
- Tareas en las cuales se detecte entrega de código generado íntegramente por una herramienta de IA sin evidencias de adaptación. El uso de herramientas de inteligencia artificial (IA) está permitido en el apoyo del desarrollo de la tarea y en la resolución de errores, pero no se permite la entrega de tareas generadas con IA de forma autónoma.

En ambos casos, el estudiante podrá apelar esta evaluación mediante una interrogación oral, en la cual deberá demostrar dominio sobre el trabajo entregado.

Los retrasos en la entrega serán penalizados con **1 punto de la nota cada 6 horas**, es decir, si la tarea se entrega entre 0 y 6 horas después de la fecha límite la nota máxima será 6.0, entre 6 y 12 horas la nota máxima será 5.0, y así sucesivamente.