Library Management Hands-On Exercise

This exercise builds on the inventory API class example, advancing to a **library system** with two MongoDB collections (`books`, `transactions`). Focus on schema design without joins—use denormalization (duplicate key data across docs) for efficient queries. Follow the numbered steps for a guided flow.

Step 1: Project Setup

Create the folder structure:

```
library_mongo/
├── app/
│   ├── __init__.py
│   ├── main.py
│   ├── models.py       # Pydantic schemas
│   ├── database.py     # Mongo connection
│   ├── crud.py         # DB operations
│   └── routes.py       # API endpoints
├── requirements.txt    # fastapi, uvicorn, pymongo
└── README.md
```

Install deps: `pip install fastapi uvicorn pymongo`.

Run: `uvicorn app.main:app --reload`.

Why two collections? Books store static info + availability status; transactions log history. Denormalize: Add `current_status` and `last_transaction_id` to books for fast availability checks (no scan of all transactions).

Step 2: Design Schemas

Think: How to query “available books” or “overdue checkouts” without joins?

books collection (sample doc):

```
{
  "_id": ObjectId("..."),
  "title": "Python Crash Course",
  "author": "Eric Matthes",
  "isbn": "1593279280",
  "total_copies": 5,
  "available_copies": 3,
  "current_status": "available",  // "available", "checked_out",
        "maintenance"
  "last_transaction_id": ObjectId("...")  // For quick link
}
```

- `available_copies > 0` implies available.

transactions collection (sample doc):

```
{
  "_id": ObjectId("..."),
  "book_id": ObjectId("..."),  // Reference book's _id
  "borrower_name": "Alice Student",
  "borrower_id": "STU001",
  "checkout_date": ISODate("2026-02-03T12:00:00Z"),
  "due_date": ISODate("2026-02-10T12:00:00Z"),
  "return_date": null,
  "status": "checked_out",  // "checked_out", "returned",
        "overdue"
  "fine_amount": 0.0
}
```

- Denormalize borrower_name for easy lists; compute overdue as `due_date < now() AND status="checked_out"`.

Task: In `models.py`, define Pydantic models:

- `BookCreate: title (str), author (str), isbn (str, min_len=13), total_copies (int >0)`
- `BookResponse: includes _id (str), available_copies, current_status`
- `CheckoutRequest: borrower_name (str), borrower_id (str), due_days (int 1-30)`
- `ReturnRequest: borrower_id (str)`
- Add validators (e.g., ISBN digits only).

In `database.py`:

```
from pymongo import MongoClient
client = MongoClient("mongodb://localhost:27017")
db = client.library_db
books_col = db.books
transactions_col = db.transactions
```

Insert data: Use the given dataInsert.py file to instert the sample data.

Step 3: Implement CRUD in crud.py

Write functions (reuse patterns from class inventory):

- `create_book(payload: dict) -> dict`: Insert, check ISBN unique (`find_one({"isbn": payload["isbn"]})`).
- `list_books(status: str=None, author: str=None) -> list`: Filter `{"current_status": status}` or `{"author": {"$regex": author}}`.
- `get_book(book_id: str) -> dict`: `find_one({"_id": ObjectId(book_id)})`; serialize _id to str.
- `update_book(book_id: str, payload: dict) -> dict`: `update_one({"_id": ObjectId(book_id)}, {"$set": payload})`.

Transaction logic (harder part):

- `checkout_book(book_id: str, req: CheckoutRequest) -> dict`:

1. Find book: `books_col.find_one({"_id": ObjectId(book_id), "available_copies": {"$gt": 0}})`
2. Atomic update: `books_col.update_one(..., {"$inc": {"available_copies": -1}, "$set": {"current_status": "checked_out", "last_transaction_id": new_id}})`
3. Insert transaction: `transactions_col.insert_one({...})`

- `return_book(book_id: str, req: ReturnRequest) -> dict`:

1. Find last tx: `transactions_col.find_one({"book_id": ObjectId(book_id), "borrower_id": req.borrower_id, "status": "checked_out"}, sort=[("checkout_date", -1)])`
2. If overdue: fine = days_late * 10
3. Update tx: `{"$set": {"status": "returned", "return_date": now, "fine_amount": fine}}`

4. Update book: `{"$inc": {"available_copies": 1}, "$set": {"current_status": "available"}}`

- `list_transactions(book_id=None, status=None, borrower_id=None)`: Filter e.g., `{"status": status, "due_date": {"$lt": datetime.utcnow()}}` for overdue.
- Handle errors: Raise HTTPException(400/404).

Tip: Use `find_one_and_update(..., return_document=True)` for atomicity. Import `datetime` for dates.[^1]

```python
# Sample logic
from datetime import datetime

# Helper: Get current UTC time for overdue checks
def get_current_time():
    """Returns datetime.utcnow() for consistent date
        comparisons."""
    return datetime.utcnow()

# Usage in queries (copy-paste these examples)
def is_overdue(transaction):
    """Simple function to check if a transaction is overdue."""
    return transaction.get('status') == 'checked_out' and \
           transaction.get('due_date') and \
           transaction['due_date'] < get_current_time()

# Example in list_transactions (add this logic)
def list_overdue_transactions():
    now = get_current_time()
    overdue = list(transactions_col.find({
        "status": "checked_out",
        "due_date": {"$lt": now}  # Mongo filter equivalent of
        due_date < now
    }).sort("due_date"))
    return [{"title": books_col.find_one({"_id": t["book_id"]})
        ["title"],
             **t} for t in overdue]  # Denormalize title for
        display

# Calculate fine (days late)
def calculate_fine(return_date, due_date):
    if not return_date or return_date > due_date:
        days_late = (get_current_time() - due_date).days
        return max(0, days_late * 10)  # 10rs/day
    return 0.0
```

Step 4: Build Routes in routes.py

```
from fastapi import APIRouter, HTTPException, Depends
from bson import ObjectId
router = APIRouter(prefix="/library", tags=["library"])

@router.post("/books/")
async def create_book(payload: BookCreate):
    book = create_book(payload.dict())
    return book

# Add all others similarly; use path params for ids
@router.post("/transactions/checkout/{book_id}")
async def checkout(book_id: str, req: CheckoutRequest):
    # Call crud.checkout_book(book_id, req.dict())
```

Include query params: `@router.get("/books/") async def list_books(status: str=None, author: str=None)`

Step 5: Wire Up main.py and Test

As in class:

```
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from .routes import router

app = FastAPI(title="Library Mongo API")
app.add_middleware(CORSMiddleware, allow_origins=["*"], ...)
app.include_router(router)
```

Test sequence (curl or Swagger):

1. POST /library/books/ – Create books.
2. GET /library/books/?status=available
3. POST /library/transactions/checkout/{book_id} – Checkout.
4. GET /library/transactions/?status=checked_out

5. POST /library/transactions/return/{book_id} – Return (test overdue by manual due_date).
6. Edge: Try checkout unavailable book → 400.