# Sprint Completion Status Report

**Student Name:** Ziheng Huang

**CUID:** zh2701

**Sprint Number:** [Sprint 0]

**Duration:** 9/13 – 9/13

**Report Date:** 9/14

## 1. Sprint Goal

**Defined Goal:**

1. Clone Professor Ferguson’s *Simple Microservices Repository.*
2. Create a project that is my version using two different resources.
   1. Copy the structure of Professor Ferguson’s repository
   2. Define two models.
   3. Implement “API first” definition by implementing placeholder routes for each resource:
      1. GET /<resource>
      2. POST /<resource>
      3. GET /<resource>/{id}
      4. PUT /<resource>/{id}
      5. DELETE /<resource>/{id}
   4. Annotate models and paths to autogenerate OpenAPI document.
   5. Tested OpenAPI document dispatching to methods.

**Outcome:** Achieved

**Notes:** N/A

## 2. Completed Work ✅

### Resource 1

from \_\_future\_\_ import annotations  
  
from typing import Optional  
from uuid import UUID, uuid4  
from datetime import datetime  
from pydantic import BaseModel, Field  
  
  
class BookBase(BaseModel):  
 id: UUID = Field(  
 ...,  
 description="Book ID.",  
 json\_schema\_extra={"example": "550e8400-e29b-41d4-a716-446655440000"},  
 )  
 title: str = Field(  
 ...,  
 description="Book title",  
 json\_schema\_extra={"example": "Trumpbook"},  
 )  
 author: Optional[str] = Field(  
 None,  
 description="Book author",  
 json\_schema\_extra={"example": "Trump"},  
 )  
 price: float = Field(  
 ...,  
 description="Non-negative price",  
 ge=0,  
 json\_schema\_extra={"example": 33.33},  
 )  
  
 model\_config = {  
 "json\_schema\_extra": {  
 "examples": [  
 {  
 "id": "550e8400-e29b-41d4-a716-446655440000",  
 "title": "Trumpbook",  
 "author": "Trump",  
 "price": 33.33,  
 }  
 ]  
 }  
 }  
  
  
class BookCreate(BookBase):  
 model\_config = {  
 "json\_schema\_extra": {  
 "examples": [  
 {  
 "id": "5b526ffa-7b06-41fc-a24e-e87520970da2",  
 "title": "Bidenbook",  
 "author": "Biden",  
 "price": 20.00,  
 }  
 ]  
 }  
 }  
  
  
class BookReplace(BaseModel):  
 title: str = Field(  
 ...,  
 description="Book title",  
 json\_schema\_extra={"example": "Trumpbook"},  
 )  
 author: Optional[str] = Field(  
 None,  
 description="Book author",  
 json\_schema\_extra={"example": "Trump"},  
 )  
 price: float = Field(  
 ...,  
 description="Non-negative price",  
 ge=0,  
 json\_schema\_extra={"example": 33.33},  
 )  
  
 model\_config = {  
 "json\_schema\_extra": {  
 "examples": [  
 {  
 "title": "Bidenbook",  
 "author": "Biden",  
 "price": 67.44,  
 }  
 ]  
 }  
 }  
  
  
class BookUpdate(BaseModel):  
 title: Optional[str] = Field(  
 None,  
 description="Book title",  
 json\_schema\_extra={"example": "Trumpbook"},  
 )  
 author: Optional[str] = Field(  
 None,  
 description="Book author",  
 json\_schema\_extra={"example": "Trump"},  
 )  
 price: Optional[float] = Field(  
 None,  
 description="Non-negative price",  
 ge=0,  
 json\_schema\_extra={"example": 33.33},  
 )  
  
 model\_config = {  
 "json\_schema\_extra": {  
 "examples": [  
 {"title": "Trumpbook: Revised Edition"},  
 {"price": 14.99},  
 {"author": "Trump"},  
 ]  
 }  
 }  
  
  
class BookRead(BookBase):  
 created\_at: datetime = Field(  
 default\_factory=datetime.utcnow,  
 description="Creation timestamp (UTC).",  
 json\_schema\_extra={"example": "2025-01-15T10:20:30Z"},  
 )  
 updated\_at: datetime = Field(  
 default\_factory=datetime.utcnow,  
 description="Last update timestamp (UTC).",  
 json\_schema\_extra={"example": "2025-01-16T12:00:00Z"},  
 )  
  
 model\_config = {  
 "json\_schema\_extra": {  
 "examples": [  
 {  
 "id": "550e8400-e29b-41d4-a716-446655440000",  
 "title": "Trumpbook",  
 "author": "Trump",  
 "price": 33.33,  
 "created\_at": "2025-01-15T10:20:30Z",  
 "updated\_at": "2025-01-16T12:00:00Z",  
 }  
 ]  
 }  
 }

### Resource 2

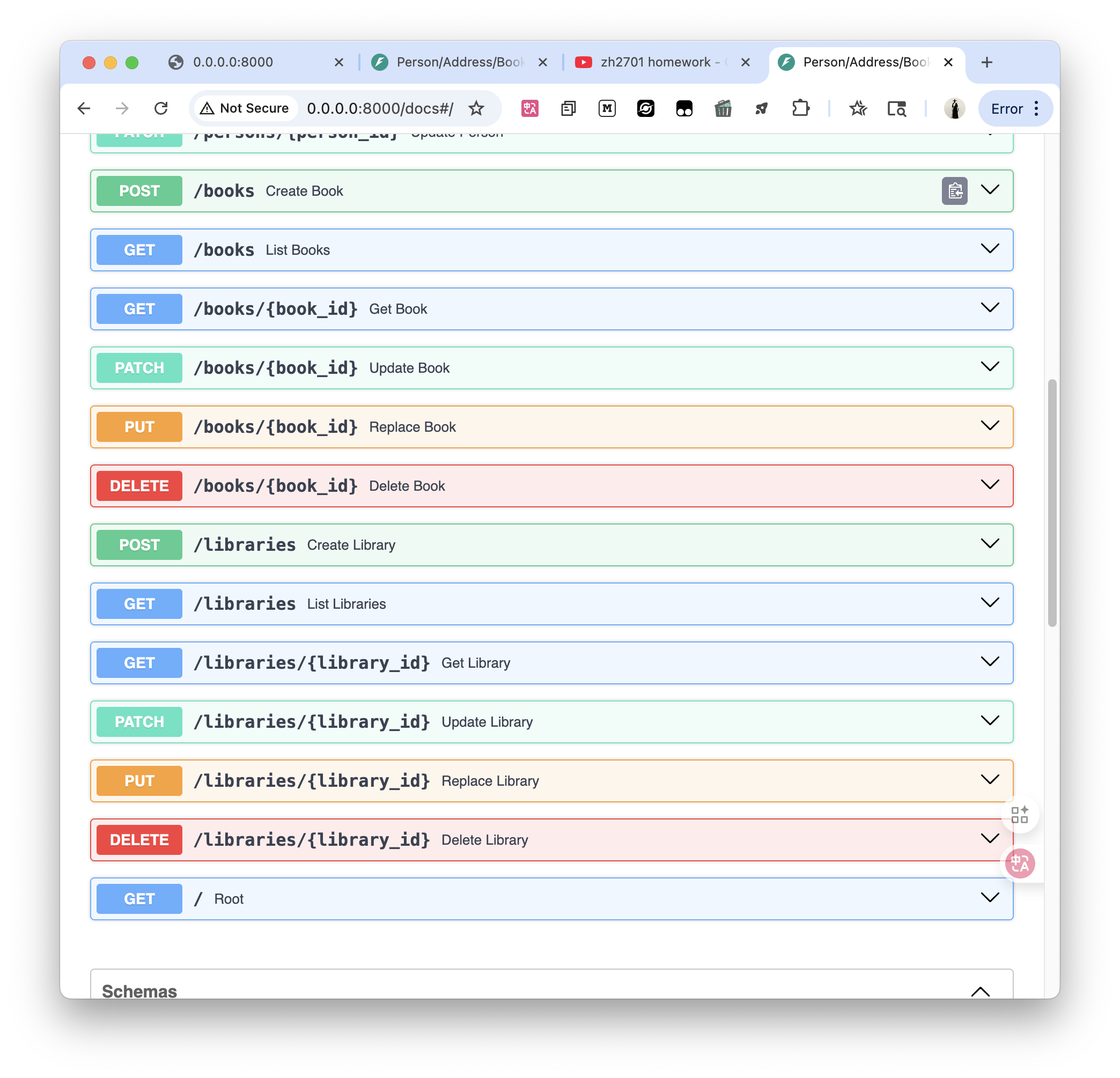
from \_\_future\_\_ import annotations  
  
from typing import Optional  
from uuid import UUID, uuid4  
from datetime import datetime  
from pydantic import BaseModel, Field  
  
  
class LibraryBase(BaseModel):  
 id: UUID = Field(  
 ...,  
 description="Library ID.",  
 json\_schema\_extra={"example": "550e8400-e29b-41d4-a716-446655440000"},  
 )  
 code: str = Field(  
 ...,  
 description="Library code",  
 json\_schema\_extra={"example": "BUT"},  
 )  
 name: str = Field(  
 ...,  
 description="Library name",  
 json\_schema\_extra={"example": "Butler Library"},  
 )  
  
 model\_config = {  
 "json\_schema\_extra": {  
 "examples": [  
 {  
 "id": "550e8400-e29b-41d4-a716-446655440000",  
 "code": "BUT",  
 "name": "Butler Library",  
 }  
 ]  
 }  
 }  
  
  
   
  
class LibraryCreate(LibraryBase):  
 model\_config = {  
 "json\_schema\_extra": {  
 "examples": [  
 {  
 "id": "11111111-1111-4111-8111-111111111111",  
 "code": "SEL",  
 "name": "Science & Engineering Library",  
 }  
 ]  
 }  
 }  
  
  
class LibraryReplace(BaseModel):  
 code: str = Field(  
 ...,  
 description="Library code",  
 json\_schema\_extra={"example": "BUT"},  
 )  
 name: str = Field(  
 ...,  
 description="Library name",  
 json\_schema\_extra={"example": "Butler Library"},  
 )  
  
 model\_config = {  
 "json\_schema\_extra": {  
 "examples": [  
 {  
 "code": "SEL",  
 "name": "Science & Engineering Library",  
 }  
 ]  
 }  
 }  
  
  
class LibraryUpdate(BaseModel):  
 code: Optional[str] = Field(  
 None,  
 description="Library code",  
 json\_schema\_extra={"example": "BUT"},  
 )  
 name: Optional[str] = Field(  
 None,  
 description="Library name",  
 json\_schema\_extra={"example": "Butler Library"},  
 )  
  
 model\_config = {  
 "json\_schema\_extra": {  
 "examples": [  
 {"code": "AVY"},  
 {"name": "Avery Architectural & Fine Arts Library"},  
 ]  
 }  
 }  
  
  
  
class LibraryRead(LibraryBase):  
 id: UUID = Field(  
 ...,  
 description="Library ID.",  
 json\_schema\_extra={"example": "550e8400-e29b-41d4-a716-446655440000"},  
 )  
 created\_at: datetime = Field(  
 default\_factory=datetime.utcnow,  
 description="Creation timestamp (UTC).",  
 json\_schema\_extra={"example": "2025-01-15T10:20:30Z"},  
 )  
 updated\_at: datetime = Field(  
 default\_factory=datetime.utcnow,  
 description="Last update timestamp (UTC).",  
 json\_schema\_extra={"example": "2025-01-16T12:00:00Z"},  
 )  
  
 model\_config = {  
 "json\_schema\_extra": {  
 "examples": [  
 {  
 "id": "550e8400-e29b-41d4-a716-446655440000",  
 "code": "BUT",  
 "name": "Butler Library",  
 "created\_at": "2025-01-15T10:20:30Z",  
 "updated\_at": "2025-01-16T12:00:00Z",  
 }  
 ]  
 }  
 }

### main.py Routes

Note: Cut and paste your routes here.

from models.book import BookCreate, BookRead, BookUpdate, BookReplace  
from models.library import LibraryCreate, LibraryRead, LibraryUpdate, LibraryReplace  
from uuid import uuid4  
  
books: Dict[UUID, BookRead] = {}  
libraries: Dict[UUID, LibraryRead] = {}  
  
def add\_data():  
 book1 = BookRead(  
 # id=uuid4(),  
 id=UUID("90143569-66f2-493d-a4a6-b519bb75d10a"),  
 title="Tgggg",  
 author="Fgggg",  
 price=33.33  
 )  
 book2 = BookRead(  
 # id=uuid4(),  
 id=UUID("de1288cc-5eb7-42b1-9e66-5a4b9a29a261"),  
 title="Bidenbook",  
 author="Ggggg",  
 price=1333.99  
 )  
 book3 = BookRead(  
 # id=uuid4(),  
 id=UUID("634af327-9d9d-49fd-a671-2cef810de932"),  
 title="Ddddddd",  
 author="Hhhhhh",  
 price=14.00  
 )  
 books[book1.id] = book1  
 books[book2.id] = book2  
 books[book3.id] = book3  
  
 lib1 = LibraryRead(  
 # id=uuid4(),  
 id=UUID("7c8f1060-db19-4e6f-b087-2f944c4aede5"),  
 code="BUT",  
 name="Butler Library"  
 )  
 lib2 = LibraryRead(  
 # id=uuid4(),  
 id=UUID("cc8b6202-2568-411a-aa47-e138c7bd0f4e"),  
 code="AVY",  
 name="Avery Architectural & Fine Arts Library"  
 )  
 lib3 = LibraryRead(  
 # id=uuid4(),  
 id=UUID("b8a518f6-c4ff-459d-b5c1-973d1b8b3c7d"),  
 code="SEL",  
 name="Science & Engineering Library"  
 )  
 libraries[lib1.id] = lib1  
 libraries[lib2.id] = lib2  
 libraries[lib3.id] = lib3  
  
  
 print(f"Book IDs: {list(books.keys())}")  
 print(f"Library IDs: {list(libraries.keys())}")  
  
add\_data()  
  
  
  
  
# -----------------------------------------------------------------------------  
# Book endpoints  
# -----------------------------------------------------------------------------  
@app.post("/books", response\_model=BookRead, status\_code=201)  
def create\_book(book: BookCreate):  
 book\_read = BookRead(\*\*book.model\_dump())  
 books[book\_read.id] = book\_read  
 return book\_read  
  
@app.get("/books", response\_model=List[BookRead])  
def list\_books(  
 author: Optional[str] = Query(None, description="Filter by author (exact match)"),  
 title\_contains: Optional[str] = Query(None, description="Filter by title containing substring"),  
 min\_price: Optional[float] = Query(None, description="Minimum price filter", ge=0),  
 max\_price: Optional[float] = Query(None, description="Maximum price filter", ge=0),  
 limit: int = Query(10, description="Number of results to return", ge=1, le=20),  
 offset: int = Query(0, description="Number of results to skip", ge=0),  
):  
 results = list(books.values())  
  
 if author is not None:  
 results = [b for b in results if b.author is not None and b.author == author]  
 if title\_contains is not None:  
 results = [b for b in results if title\_contains.lower() in b.title.lower()]  
 if min\_price is not None:  
 results = [b for b in results if b.price >= min\_price]  
 if max\_price is not None:  
 results = [b for b in results if b.price <= max\_price]  
  
 return results[offset:offset + limit]  
  
@app.get("/books/{book\_id}", response\_model=BookRead)  
def get\_book(  
 book\_id: UUID = Path(..., description="Book ID"),  
 fields: Optional[str] = Query(None, description=" fields to return separated by comma(e.g., 'title,price')"),  
):  
 if book\_id not in books:  
 raise HTTPException(status\_code=404, detail="Book not found")  
  
 book = books[book\_id]  
  
 if fields is not None:  
 field\_list = [f.strip() for f in fields.split(',')]  
 valid\_fields = ['id', 'title', 'author', 'price', 'created\_at', 'updated\_at']  
 result = {}  
 for field in field\_list:  
 if field in valid\_fields and hasattr(book, field):  
 result[field] = getattr(book, field)  
 return result  
  
 return book  
  
@app.patch("/books/{book\_id}", response\_model=BookRead)  
def update\_book(book\_id: UUID, update: BookUpdate):  
 if book\_id not in books:  
 raise HTTPException(status\_code=404, detail="Book not found")  
 stored = books[book\_id].model\_dump()  
 stored.update(update.model\_dump(exclude\_unset=True))  
 books[book\_id] = BookRead(\*\*stored)  
 return books[book\_id]  
  
@app.put("/books/{book\_id}", response\_model=BookRead)  
def replace\_book(book\_id: UUID, book: BookReplace):  
 if book\_id not in books:  
 raise HTTPException(status\_code=404, detail="Book not found")  
 book\_read = BookRead(id=book\_id, \*\*book.model\_dump())  
 books[book\_id] = book\_read  
 return book\_read  
  
@app.delete("/books/{book\_id}")  
def delete\_book(book\_id: UUID):  
 if book\_id not in books:  
 raise HTTPException(status\_code=404, detail="Book not found")  
 del books[book\_id]  
 return {"message": "Book deleted successfully"}  
  
# -----------------------------------------------------------------------------  
# Library endpoints  
# -----------------------------------------------------------------------------  
@app.post("/libraries", response\_model=LibraryRead, status\_code=201)  
def create\_library(library: LibraryCreate):  
 if library.id in libraries:  
 raise HTTPException(status\_code=400, detail="Library with this ID already exists")  
  
 for existing\_library in libraries.values():  
 if existing\_library.code.lower() == library.code.lower():  
 raise HTTPException(status\_code=400, detail="A library with this code already exists")  
 if existing\_library.name.lower() == library.name.lower():  
 raise HTTPException(status\_code=400, detail="A library with this name already exists")  
  
 library\_read = LibraryRead(\*\*library.model\_dump())  
 libraries[library\_read.id] = library\_read  
 return library\_read  
  
@app.get("/libraries", response\_model=List[LibraryRead])  
def list\_libraries(  
 code: Optional[str] = Query(None, description="Filter by code"),  
 name: Optional[str] = Query(None, description="Filter by name"),  
 name\_contains: Optional[str] = Query(None, description="Filter by name containing substring"),  
 limit: int = Query(50, description="Number of results to return", ge=1, le=20),  
 offset: int = Query(0, description="Number of results to skip", ge=0),  
):  
 results = list(libraries.values())  
  
 if code is not None:  
 results = [l for l in results if l.code.lower() == code.lower()]  
 if name is not None:  
 results = [l for l in results if l.name.lower() == name.lower()]  
 if name\_contains is not None:  
 results = [l for l in results if name\_contains.lower() in l.name.lower()]  
  
 return results[offset:offset + limit]  
  
@app.get("/libraries/{library\_id}", response\_model=LibraryRead)  
def get\_library(library\_id: UUID):  
 if library\_id not in libraries:  
 raise HTTPException(status\_code=404, detail="Library not found")  
 return libraries[library\_id]  
  
@app.patch("/libraries/{library\_id}", response\_model=LibraryRead)  
def update\_library(library\_id: UUID, update: LibraryUpdate):  
 if library\_id not in libraries:  
 raise HTTPException(status\_code=404, detail="Library not found")  
 if update.code is not None or update.name is not None:  
 for existing\_library in libraries.values():  
 if existing\_library.id != library\_id:  
 if update.code is not None and existing\_library.code.lower() == update.code.lower():  
 raise HTTPException(status\_code=400, detail="A library with this code already exists")  
 if update.name is not None and existing\_library.name.lower() == update.name.lower():  
 raise HTTPException(status\_code=400, detail="A library with this name already exists")  
  
 stored = libraries[library\_id].model\_dump()  
 stored.update(update.model\_dump(exclude\_unset=True))  
 libraries[library\_id] = LibraryRead(\*\*stored)  
 return libraries[library\_id]  
  
@app.put("/libraries/{library\_id}", response\_model=LibraryRead)  
def replace\_library(library\_id: UUID, library: LibraryReplace):  
 if library\_id not in libraries:  
 raise HTTPException(status\_code=404, detail="Library not found")  
  
 for existing\_library in libraries.values():  
 if existing\_library.id != library\_id:  
 if existing\_library.code.lower() == library.code.lower():  
 raise HTTPException(status\_code=400, detail="A library with this code already exists")  
 if existing\_library.name.lower() == library.name.lower():  
 raise HTTPException(status\_code=400, detail="A library with this name already exists")  
  
 library\_read = LibraryRead(id=library\_id, \*\*library.model\_dump())  
 libraries[library\_id] = library\_read  
 return library\_read  
  
@app.delete("/libraries/{library\_id}")  
def delete\_library(library\_id: UUID):  
 if library\_id not in libraries:  
 raise HTTPException(status\_code=404, detail="Library not found")  
 del libraries[library\_id]  
 return {"message": "Library deleted successfully"}

### OpenAPI Document (Partial)



### Link to Recording of Demo

<https://youtu.be/YqGC-Zg4Y4o>

or search "zh2701 homework - COMSW4153 - V0 sprint"

### Link to GitHub Repository

<https://github.com/HuangZiheng-o-O/zh2701-SimpleMicroservices>