Table of Contents

Bookstore API — Deliverables

Combined source, docs, tests, and report

API Source — app.py

```
"""
Bookstore Management API (Flask + flask-restx)
Endpoints:
- GET /api/books/<isbn>
- POST /api/orders
- POST /api/delivery
Simple API key auth: header 'X-API-Key'.
Uses the inventory_mock, sales_mock, and delivery_mock modules.
Swagger UI available at /docs
"""
import os
from functools import wraps
from decimal import Decimal
from flask import Flask, request, jsonify
import werkzeug
try:
    # Python 3.8+ provides importlib.metadata
    from importlib import metadata as importlib_metadata
except Exception:
    import importlib_metadata

# Some newer werkzeug releases removed the __version__ attribute which
# older Flask testing utilities expect. Provide a compatibility shim so
# the test client can construct a default User-Agent string.
try:
    if not hasattr(werkzeug, '__version__'):
        try:
            werkzeug.__version__ = importlib_metadata.version('werkzeug')
        except Exception:
            werkzeug.__version__ = '0'
except Exception:
    # Best effort; do not fail import if metadata isn't available
    pass

from inventory_mock import get_book, reserve_stock, list_inventory
from sales_mock import process_payment, create_order, get_order
from delivery_mock import create_delivery

API_KEY = os.environ.get('BOOKSTORE_API_KEY', 'secret123')


def create_app():
    app = Flask(__name__)

    def require_api_key(f):
        @wraps(f)
        def wrapper(*args, **kwargs):
            key = request.headers.get('X-API-Key')
            if not key or key != API_KEY:
                return jsonify({'message': 'Unauthorized'}), 401
            return f(*args, **kwargs)
        return wrapper

    @app.route('/api/books/', methods=['GET'])
    @require_api_key
    def list_books():
        return jsonify(list_inventory())

    @app.route('/api/books/<string:isbn>', methods=['GET'])
    @require_api_key
    def get_book_route(isbn):
        book = get_book(isbn)
        if not book:
            return jsonify({'message': 'Book not found'}), 404
        return jsonify(book)
```

```
    @app.route('/api/orders/', methods=['POST'])
    @require_api_key
    def create_order_route():
        data = request.get_json(silent=True) or {}
        customer_id = data.get('customer_id')
        items = data.get('items', [])
        payment = data.get('payment')

        # Validate items and compute total
        total = Decimal('0.0')
        for it in items:
            isbn = it.get('isbn')
            qty = int(it.get('quantity', 0))
            if qty <= 0:
                return jsonify({'message': 'Quantity must be positive'}), 400
            book = get_book(isbn)
            if not book:
                return jsonify({'message': f'Book {isbn} not found'}), 400
            total += Decimal(str(book['price'])) * qty

        # Process payment
        ok, result = process_payment(float(total), payment)
        if not ok:
            return jsonify({'message': f'Payment failed: {result}'}), 402
        tx_id = result

        # Reserve stock
        for it in items:
            isbn = it['isbn']
            qty = int(it['quantity'])
            success, msg = reserve_stock(isbn, qty)
            if not success:
                return jsonify({'message': f'Inventory issue for {isbn}: {msg}'}), 409

        # Create order record in sales mock
        order = create_order(customer_id, items, float(total), tx_id)
        return jsonify({'order_id': order['order_id'], 'status': order['status'], 'total': order['total']}), 201

    @app.route('/api/orders/<string:order_id>', methods=['GET'])
    @require_api_key
    def get_order_route(order_id):
        order = get_order(order_id)
        if not order:
            return jsonify({'message': 'Order not found'}), 404
        return jsonify(order)

    @app.route('/api/delivery/', methods=['POST'])
    @require_api_key
    def create_delivery_route():
        data = request.get_json(silent=True) or {}
        order_id = data.get('order_id')
        address = data.get('address')
        courier = data.get('courier')
        if not order_id or not address:
            return jsonify({'message': 'order_id and address required'}), 400
        d = create_delivery(order_id, address, courier)
        return jsonify(d), 201

    # Serve OpenAPI spec
    @app.route('/openapi.yaml', methods=['GET'])
    def serve_openapi():
        from flask import send_from_directory
        return send_from_directory('.', 'openapi.yaml')

    return app


if __name__ == '__main__':
    app = create_app()
```

```
app.run(host='0.0.0.0', port=5001, debug=True)
```

Inventory mock — inventory_mock.py

```
"""
Simple in-memory Inventory mock service.
Provides functions to get book details and reserve stock.
"""
from copy import deepcopy

# Sample inventory keyed by ISBN
_inventory = {
    '9780143127550': {
        'isbn': '9780143127550',
        'title': 'The Martian',
        'author': 'Andy Weir',
        'price': 12.99,
        'stock': 10
    },
    '9780262033848': {
        'isbn': '9780262033848',
        'title': 'Introduction to Algorithms',
        'author': 'Cormen et al.',
        'price': 89.99,
        'stock': 3
    },
    '9780140449136': {
        'isbn': '9780140449136',
        'title': 'The Odyssey',
        'author': 'Homer',
        'price': 9.5,
        'stock': 25
    }
}


def get_book(isbn):
    book = _inventory.get(isbn)
    return deepcopy(book) if book else None


def reserve_stock(isbn, qty):
    """Attempt to reserve qty units of isbn. Returns True if successful, False otherwise."""
    book = _inventory.get(isbn)
    if not book:
        return False, 'Book not found'
    if book['stock'] < qty:
        return False, 'Insufficient stock'
    book['stock'] -= qty
    return True, None


def replenish_stock(isbn, qty):
    book = _inventory.get(isbn)
    if not book:
        return False
    book['stock'] += qty
    return True


def list_inventory():
    return deepcopy(list(_inventory.values()))
```

Sales mock — sales_mock.py

```
"""
Simple Sales mock service: processes payments and creates orders in-memory.
"""
from uuid import uuid4
from copy import deepcopy

_orders = {}


def process_payment(amount, payment_info):
    """Simulate a payment gateway call. Returns (success, transaction_id or error)."""
    # Very simple simulation: accept if card_number present and amount > 0
    if not payment_info or not payment_info.get('card_number'):
        return False, 'Missing card details'
    if amount <= 0:
        return False, 'Invalid amount'
    # simulate approval
    tx_id = str(uuid4())
    return True, tx_id


def create_order(customer_id, items, total, payment_tx_id):
    order_id = str(uuid4())
    order = {
        'order_id': order_id,
        'customer_id': customer_id,
        'items': deepcopy(items),
        'total': total,
        'payment_tx_id': payment_tx_id,
        'status': 'created'
    }
    _orders[order_id] = order
    return deepcopy(order)


def get_order(order_id):
    order = _orders.get(order_id)
    return deepcopy(order) if order else None


def list_orders():
    return deepcopy(list(_orders.values()))
```

Delivery mock — delivery_mock.py

```
"""
Simple Delivery mock service: stores delivery tasks in-memory.
"""
from uuid import uuid4
from copy import deepcopy

_deliveries = {}


def create_delivery(order_id, address, courier=None):
    delivery_id = str(uuid4())
    d = {
        'delivery_id': delivery_id,
        'order_id': order_id,
        'address': address,
        'courier': courier or 'local',
        'status': 'scheduled'
    }
    _deliveries[delivery_id] = d
    return deepcopy(d)


def update_delivery_status(delivery_id, status):
    d = _deliveries.get(delivery_id)
    if not d:
        return None
    d['status'] = status
    return deepcopy(d)


def get_delivery(delivery_id):
    d = _deliveries.get(delivery_id)
    return deepcopy(d) if d else None


def list_deliveries():
    return deepcopy(list(_deliveries.values()))
```

OpenAPI spec — openapi.yaml

```
openapi: 3.0.3
info:
  title: Bookstore Management API
  version: 1.0.0
  description: API for integrating Inventory, Sales and Delivery subsystems (demo)
servers:
  - url: http://localhost:5001
paths:
  /api/books/{isbn}:
    get:
      summary: Retrieve book details
      parameters:
        - name: isbn
          in: path
          required: true
          schema:
            type: string
      responses:
        '200':
          description: Book found
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Book'
        '404':
          description: Book not found
  /api/books:
    get:
      summary: List inventory
      responses:
        '200':
          description: Inventory list
          content:
            application/json:
              schema:
                type: array
                items:
                  $ref: '#/components/schemas/Book'
  /api/orders:
    post:
      summary: Place a new order
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/OrderRequest'
      responses:
        '201':
          description: Order created
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/OrderResponse'
        '400':
          description: Bad request
        '402':
          description: Payment required / failed
        '409':
          description: Inventory conflict
  /api/orders/{order_id}:
    get:
      summary: Get order details
      parameters:
        - name: order_id
          in: path
```

```
          required: true
          schema:
            type: string
      responses:
        '200':
          description: Order found
  /api/delivery:
    post:
      summary: Create delivery task for an order
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/DeliveryRequest'
      responses:
        '201':
          description: Delivery scheduled
components:
  schemas:
    Book:
      type: object
      properties:
        isbn:
          type: string
        title:
          type: string
        author:
          type: string
        price:
          type: number
          format: float
        stock:
          type: integer
    OrderItem:
      type: object
      properties:
        isbn:
          type: string
        quantity:
          type: integer
    OrderRequest:
      type: object
      required: [customer_id, items, payment]
      properties:
        customer_id:
          type: string
        items:
          type: array
          items:
            $ref: '#/components/schemas/OrderItem'
        payment:
          type: object
          description: Payment details (demo only)
    OrderResponse:
      type: object
      properties:
        order_id:
          type: string
        status:
          type: string
        total:
          type: number
    DeliveryRequest:
      type: object
      required: [order_id, address]
      properties:
        order_id:
          type: string
```

```
        address:
          type: string
        courier:
          type: string
security:
  - ApiKeyAuth: []
components:
  securitySchemes:
    ApiKeyAuth:
      type: apiKey
      in: header
      name: X-API-Key
```

Tests — tests\test_api.py

```
import os
import json
import pytest

from app import create_app

API_KEY = 'secret123'

@pytest.fixture
def client():
    app = create_app()
    app.config['TESTING'] = True
    client = app.test_client()
    yield client

def test_get_book_success(client):
    r = client.get('/api/books/9780143127550', headers={'X-API-Key': API_KEY})
    assert r.status_code == 200
    data = r.get_json()
    assert data['isbn'] == '9780143127550'

def test_get_book_not_found(client):
    r = client.get('/api/books/0000000000', headers={'X-API-Key': API_KEY})
    assert r.status_code == 404

def test_place_order_and_delivery_flow(client):
    # place an order
    payload = {
        'customer_id': 'cust-1',
        'items': [{'isbn': '9780143127550', 'quantity': 1}],
        'payment': {'card_number': '4111111111111111', 'expiry': '12/26'}
    }
    r = client.post('/api/orders/', json=payload, headers={'X-API-Key': API_KEY})
    assert r.status_code == 201
    body = r.get_json()
    assert 'order_id' in body
    order_id = body['order_id']

    # create delivery
    d_payload = {'order_id': order_id, 'address': '123 Main St'}
    rd = client.post('/api/delivery/', json=d_payload, headers={'X-API-Key': API_KEY})
    assert rd.status_code == 201
    delivery = rd.get_json()
    assert delivery['order_id'] == order_id

def test_auth_required(client):
    r = client.get('/api/books/9780143127550')
    assert r.status_code == 401
```

Test results — test_results.txt

ÿþ.■.■.■.■ ■
■
■4■ ■p■a■s■s■e■d■ ■i■n■ ■0■.■2■0■s■
■
■

Report — report.md

```
# Bookstore API — Design & Implementation Report

## Overview
This project implements a small RESTful API that integrates three mock subsystems common to a bookstore:

## Architecture and Design
- Framework: Flask with `flask-restx` to provide REST endpoints and automatic Swagger UI.
- Subsystems: implemented as separate in-memory modules:
  - `inventory_mock.py` — manages book data and stock levels.
  - `sales_mock.py` — simulates payment processing and order persistence.
  - `delivery_mock.py` — creates delivery tasks and updates status.
- API endpoints:
  - `GET /api/books/{isbn}` — fetch book details.
  - `GET /api/books` — list inventory.
  - `POST /api/orders` — place an order (validates items, processes payment, reserves stock, creates ord
  - `GET /api/orders/{order_id}` — fetch order details.
  - `POST /api/delivery` — schedule a delivery for an order.
- Authentication: API key required via `X-API-Key` header. For demo purposes the default key is `secret1
- Data exchange: JSON request/response.
- Documentation: `openapi.yaml` (OpenAPI 3.0 spec) and Swagger UI served at `/docs` when running the app

## Implementation Notes
- Inventory reservation is performed after payment succeeds; in a production system you'd likely reserve
- The sales mock returns a UUID transaction id when payment is simulated as successful.
- The order payload expects `customer_id`, a list of `{ isbn, quantity }`, and a `payment` object that c

## Testing
- Unit tests are located in `tests/test_api.py` and use Flask's test client to exercise endpoints.
- Because local pytest environments can have plugin conflicts, the `run_tests.ps1` helper ensures plugin
- Test highlights:
  - GET book success and not-found cases.
  - End-to-end order placement flow including payment and inventory reservation.
  - Delivery scheduling follow-up for created orders.

## Security and Limitations
- The demo uses an API key for simplicity. For production use OAuth2/JWT and proper secrets management.
- Payment processing is mocked. Replace `sales_mock.process_payment` with a PCI-compliant gateway integr
- State is in-memory: orders, inventory changes, and delivery tasks are not persisted beyond process lif

## How to run
1. Create a virtual environment and install dependencies from `requirements.txt`.
2. Start the API: `python app.py` (ensure `BOOKSTORE_API_KEY` is set to match client requests).
3. Use Swagger UI at `http://localhost:5001/docs` or curl/Postman to exercise endpoints.

## Next steps and improvements
- Persist data to a database (Postgres/SQLite) and add migrations.
- Implement idempotency keys for order creation to handle retries safely.
- Add asynchronous background tasks for delivery processing and payment webhooks.
- Harden authentication and add role-based access control.
```