API実習　2025　課題レポート（第5回）

API 実習課題:　**第4回課題にて策定したAPI要件定義に基づき、API提供者としてAPI設計・実装する**

レポート04（要件定義）に対応した、最終課題05：API設計と実装 の精緻なレポート課題設定、**要件定義の充足度に連動した採点基準（100点）**である。

出題範囲：　API実習2025　第1回～第15回まで

提出期限：　2026/2/10(火) 12:00

最終課題05：自作APIの設計と実装（レポート04要件の充足）

1. 背景

- 最終課題05では、レポート04**API提供者視点で設計・実装可能なレベルの要件定義**を**実際のAPI設計と実装**に発展させ、**要件→設計→実装→検証**の一貫性を示すことを目的とする

2. 問題設定（課題の中身）

レポート04で定義した業務・ユースケースに対応する**APIを実装**し、**要件定義の充足度を評価・検証・説明**すること

必須要件（MAXライン）

1. **API契約（仕様）**を整備（/openapi.yaml または /openapi.json）　**OpenAPI 3.0+**　FastAPIの出力相当
2. **機能要件の実装**
   - 主要リソース（最低2種類）
   - CRUD（合計4操作以上）＋**検索**（フィルタ/ページングのいずれか）
   - **エラーハンドリング**（標準化されたエラー応答、コード/メッセージ/詳細）
   - **バージョニング**（例： `/v1` の明示）
3. **非機能要件の実装・測定**
   - **認証/認可**（例：BASIC認証,OAuthJWT、トークン検証、ロール別アクセス）
   - **レート制限**（例：IP/トークン単位）
   - **性能テスト**の結果（目標と実測を提示）
4. **運用要件**
   - **ロギング**方針（構造化ログ）と**監査ログ**の要点
   - **バックアップ**/復旧手順（データを持つ場合）

5. **再現手順**
    - コンテナ（Docker）またはローカルで**起動手順**を README に記載
    - **テストスイート**（ユニット＋API契約テスト）
    - **デモ用クライアント**（curl スクリプト・Postman コレクション・簡易UIのいずれか）

> ＊レポート04で明記した**業務フロー／ユースケース／関連技術／非機能**を、最終課題05で**整合性をもって充足**する

提出物（推奨ディレクトリ構成）

```
.
├─ README.md                  # 起動方法・環境・要件→設計→実装の紐付け
├─ openapi.yaml               # API契約（OpenAPI）
├─ src/                       # 実装コード
├─ tests/                     # ユニット/契約テスト（最低10ケース）
├─ scripts/                   # デモ用curl・負荷試験スクリプト
├─ figs/                      # 図版（アーキテクチャ/シーケンス/ER/ユースケース）
├─ reports-05.md              # 最終課題レポート（下記構成）
└─ postman_collection.json    # 任意（デモ・検証用）
```

最終課題05：レポート（reports-05.md）の精緻な構成

> レポート04の構成項目と評価観点を維持しつつ、「**要件→設計→実装→検証**」の**トレーサビリティ**を明確化

1. **背景・業務領域の具体化**
    - レポート04の背景を要約し、**変更点**（あれば）を明示。理由と影響を記述
2. **課題・目的・APIが必要な理由**
    - **KPI/成功指標**を再掲（レポート04）→**今回の実測**と比較
3. **ユーザ／ステークホルダ**
    - **権限モデル**（管理者/一般/外部アプリ）と**認可ルール**を表で提示
4. **業務フロー・ユースケース**
    - Mermaid図で**2件以上の主要ユースケース**を実装に対応づけ（ID付与）
5. **関連技術・先行事例**
    - 採用技術を**実装バージョン**付きで列挙（例：言語、FW、DB）
6. **技術比較／選定理由**
    - 候補の利点/欠点/トレードオフ＋**採用理由**（ユースケース適合性）

7. **機能要件（エンドポイントの明確さ・一貫性）**
    - OpenAPIへ**双方向リンク**（エンドポイント→要件ID）
    - **ステータスコード/エラー形式**の規約を明記
8. **非機能要件（測定可能な指標・現実性）**
    - **性能（スループット、同時接続）の目標値**
    - **セキュリティ**（認証/認可/暗号化/脆弱性対策/監査ログ）
    - **可用性**（SLA目標、冗長化構成/障害対応手順）
    - **拡張性**（スケール戦略）／**保守性**（テスト/デプロイ/ドキュメント運用）
9. **まとめ**

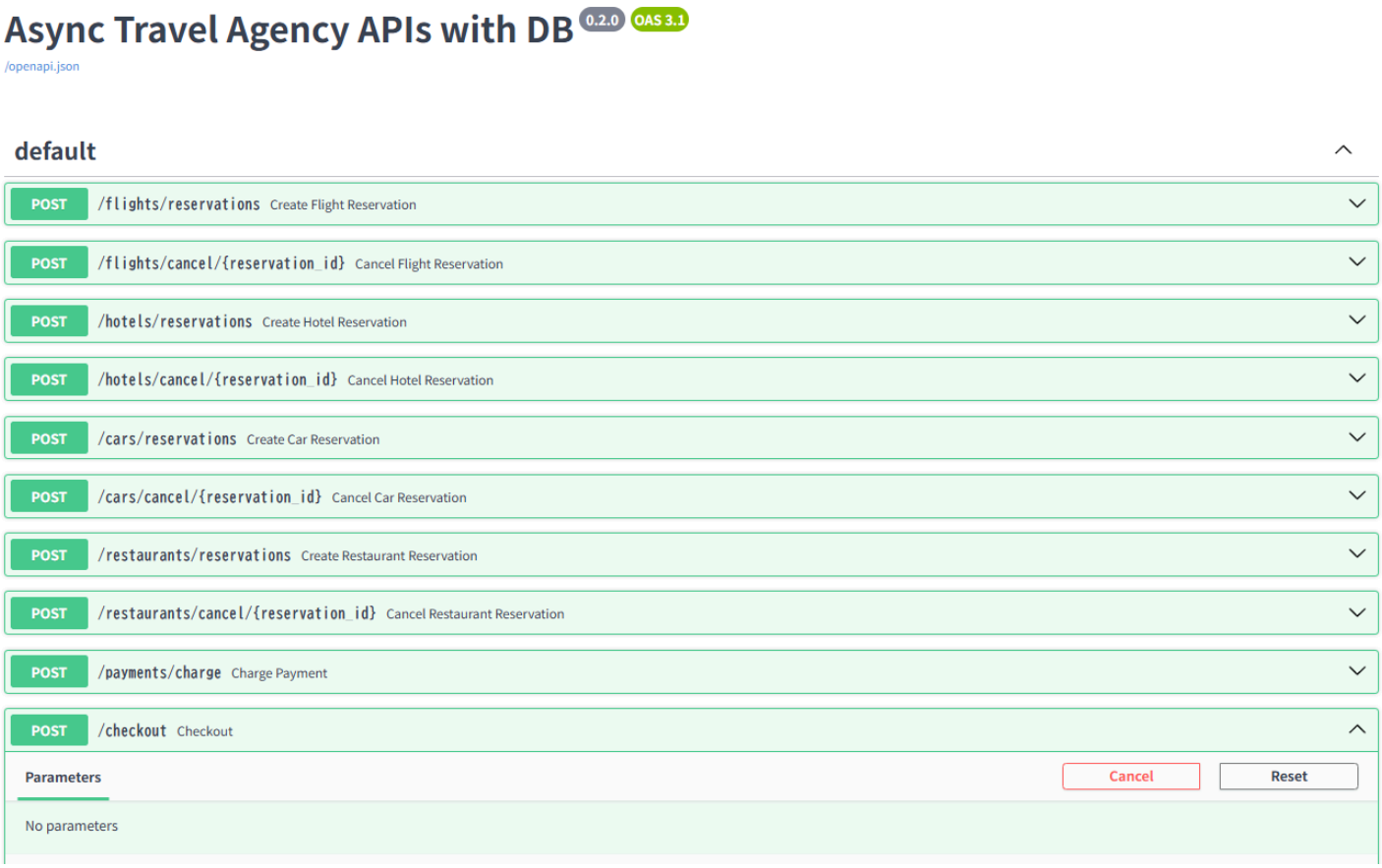
最終課題05：チェックリスト

- ▢ **OpenAPI**（例・エラー・バージョン・セキュリティスキーム）
- ▢ **ユースケース2件以上**を**実装で再現**（成功/失敗/例外パターン）
- ▢ **性能指標**（スループット・エラー率）**測定**と**目標比較**
- ▢ **認証/認可**と**レート制限**の動作確認
- ▢ **監視/ログ/監査**の方針と実装の証跡
- ▢ **運用要件**（バックアップ/復旧、秘密情報管理）を明記
- ▢ **テスト**（契約/ユニット/認可/例外）
- ▢ **参考文献**の明記（標準仕様/事例）

README テンプレート

```
# 最終課題05：自作API（v1）
## 目的
レポート04で定義した業務課題をAPIで解決し、KPIを測定・検証する。

## 要件→設計→実装→検証（トレーサビリティ）
| 要件ID | ユースケース | OpenAPIパス | 実装（src/...） | テスト（tests/...） | KPI目標 | KPI実測 |
|--------|--------------|-------------|-----------------|---------------------|---------|------
| RQ-01  | UC-01        | /v1/items   | items.py        | test_items.py::...  | p95<200ms| 178

## 環境/起動
- Docker: `docker compose up -d`
- ローカル: `python -m venv .venv && ...`

## 検証手順
- 契約テスト: `pytest -m contract`
- デモ: `scripts/demo.sh`（curl）または Postman

## 参考
- openapi.yaml / figs/architecture.png
```

参考サンプルコード例：　旅行代理店（フライト予約⇒ホテル予約⇒レンタカー予約のAPI連携）




```
$ python3 -m venv venv
$ source venv/bin/activate
$ pip3 install update pip
$ pip3 install --upgrade pip
$ pip3 install fastapi uvicorn sqlalchemy aiosqlite
$ uvicorn flight-hotel-car-main:app --reload

`http://localhost:8000/docs`
```

```
""" Usage:
$ python3 -m venv venv
$ source venv/bin/activate
$ pip3 install update pip
$ pip3 install --upgrade pip
$ pip3 install fastapi uvicorn sqlalchemy aiosqlite
$ uvicorn flight-hotel-car-main:app --reload

`http://localhost:8000/docs`
"""

import asyncio
from fastapi import FastAPI, HTTPException, Depends
from pydantic import BaseModel, Field
from typing import Optional, List
from uuid import uuid4
from datetime import date, datetime
from contextlib import asynccontextmanager

from sqlalchemy.ext.asyncio import create_async_engine, AsyncSession, async_sessionmaker
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column
from sqlalchemy import select, update, delete
from sqlalchemy.dialects.sqlite import JSON

# ====== Database Setup ======
DATABASE_URL = "sqlite+aiosqlite:///./flight_booking.db"

engine = create_async_engine(DATABASE_URL, echo=True)
AsyncSessionLocal = async_sessionmaker(engine, expire_on_commit=False)

class Base(DeclarativeBase):
    pass

# ====== ORM Models ======

class Flight(Base):
    __tablename__ = "flights"
    reservation_id: Mapped[str] = mapped_column(primary_key=True)
    status: Mapped[str]
    price: Mapped[int]
    currency: Mapped[str]
    req: Mapped[dict] = mapped_column(JSON) # Store request data as JSON

class Hotel(Base):
    __tablename__ = "hotels"
    reservation_id: Mapped[str] = mapped_column(primary_key=True)
```

```
    status: Mapped[str]
    price: Mapped[int]
    currency: Mapped[str]
    nights: Mapped[int]
    req: Mapped[dict] = mapped_column(JSON)

class Car(Base):
    __tablename__ = "cars"
    reservation_id: Mapped[str] = mapped_column(primary_key=True)
    status: Mapped[str]
    price: Mapped[int]
    currency: Mapped[str]
    days: Mapped[int]
    req: Mapped[dict] = mapped_column(JSON)

class Restaurant(Base):
    __tablename__ = "restaurants"
    reservation_id: Mapped[str] = mapped_column(primary_key=True)
    status: Mapped[str]
    price: Mapped[int]
    currency: Mapped[str]
    party_size: Mapped[int]
    req: Mapped[dict] = mapped_column(JSON)

class Payment(Base):
    __tablename__ = "payments"
    payment_id: Mapped[str] = mapped_column(primary_key=True)
    status: Mapped[str]
    amount: Mapped[int]
    currency: Mapped[str]
    method: Mapped[str]
    idempotency_key: Mapped[str]

class Idempotency(Base):
    __tablename__ = "idempotency"
    key: Mapped[str] = mapped_column(primary_key=True)
    response_json: Mapped[dict] = mapped_column(JSON)


# ====== Lifespan & Dependencies ======

@asynccontextmanager
async def lifespan(app: FastAPI):
    # Create tables on startup
    async with engine.begin() as conn:
        await conn.run_sync(Base.metadata.create_all)
```

```
    yield
    # Cleanup if needed

app = FastAPI(title="Async Travel Agency APIs with DB", version="0.2.0", lifespan=lifespan)

async def get_db():
    async with AsyncSessionLocal() as session:
        yield session

# ====== Pydantic Models (Unchanged mostly) ======
# フライト予約リクエスト
class FlightReservationRequest(BaseModel):
    origin: str
    destination: str
    depart_date: date
    return_date: Optional[date] = None
    passenger_name: str
    seat_class: str = Field(default="economy", pattern="^(economy|premium|business)$")

class FlightReservationResponse(BaseModel):
    reservation_id: str
    status: str
    price: int
    currency: str = "JPY"

# ホテル予約リクエスト
class HotelReservationRequest(BaseModel):
    city: str
    check_in: date
    check_out: date
    guest_name: str
    room_type: str = Field(default="standard", pattern="^(standard|deluxe|suite)$")

class HotelReservationResponse(BaseModel):
    reservation_id: str
    status: str
    price: int
    currency: str = "JPY"
    nights: int

# レンタカー予約リクエスト
class CarReservationRequest(BaseModel):
    pickup_location: str
    pickup_datetime: datetime
    dropoff_datetime: datetime
    driver_name: str
```

```
    car_class: str = Field(default="compact", pattern="^(compact|standard|van|premium)$")

class CarReservationResponse(BaseModel):
    reservation_id: str
    status: str
    price: int
    currency: str = "JPY"
    days: int

# レストラン予約リクエスト
class RestaurantReservationRequest(BaseModel):
    city: str
    restaurant_name: Optional[str] = None
    date_time: datetime
    party_size: int = Field(gt=0, le=20)
    contact_name: str
    cuisine: Optional[str] = None

class RestaurantReservationResponse(BaseModel):
    reservation_id: str
    status: str
    price: int
    currency: str = "JPY"
    party_size: int

# 支払いリクエスト
class PaymentChargeRequest(BaseModel):
    amount: int = Field(gt=0)
    currency: str = Field(default="JPY", pattern="^(JPY)$")
    method: str = Field(default="credit_card", pattern="^(credit_card|wallet)$")
    idempotency_key: str
    simulate_failure: bool = False

class PaymentChargeResponse(BaseModel):
    payment_id: str
    status: str
    amount: int
    currency: str
    method: str
    idempotency_key: str

# CheckOut
class CheckoutRequest(BaseModel):
    flight: FlightReservationRequest
    hotel: HotelReservationRequest
    car: CarReservationRequest
```

```
    restaurant: RestaurantReservationRequest
    payment_method: str = Field(default="credit_card", pattern="^(credit_card|wallet)$")
    idempotency_key: str
    simulate_failure: bool = False

class CheckoutResponse(BaseModel):
    flight: FlightReservationResponse
    hotel: HotelReservationResponse
    car: CarReservationResponse
    restaurant: RestaurantReservationResponse
    payment: PaymentChargeResponse
    total_amount: int
    currency: str = "JPY"

# ====== Pricing helpers (Pure functions) ======
def price_flight(req: FlightReservationRequest) -> int:
    base = 25000
    leg_count = 2 if req.return_date else 1
    multipliers = {"economy": 1.0, "premium": 1.3, "business": 1.8}
    return int(base * leg_count * multipliers.get(req.seat_class, 1.0))

def nights_between(check_in: date, check_out: date) -> int:
    return max((check_out - check_in).days, 0)

def price_hotel(req: HotelReservationRequest) -> (int, int):
    nights = nights_between(req.check_in, req.check_out)
    base_per_night = {"standard": 10000, "deluxe": 15000, "suite": 25000}
    price = base_per_night.get(req.room_type, 10000) * max(nights, 1)
    return price, max(nights, 1)

def days_between_dt(start: datetime, end: datetime) -> int:
    delta = end.date() - start.date()
    return max(delta.days, 1)

def price_car(req: CarReservationRequest) -> (int, int):
    days = days_between_dt(req.pickup_datetime, req.dropoff_datetime)
    base_per_day = {"compact": 5000, "standard": 7000, "van": 9000, "premium": 12000}
    price = base_per_day.get(req.car_class, 5000) * days
    return price, days

def price_restaurant(req: RestaurantReservationRequest) -> int:
    return 1000 * req.party_size

# ====== Flight endpoints ======
@app.post("/flights/reservations", response_model=FlightReservationResponse)
async def create_flight_reservation(req: FlightReservationRequest, db: AsyncSession = Depends(g
```

```
    rid = f"f-{uuid4().hex[:8]}"
    price = price_flight(req)

    # DB Model
    new_flight = Flight(
        reservation_id=rid,
        status="hold",
        price=price,
        currency="JPY",
        req=req.model_dump(mode='json') # Save JSON req
    )
    db.add(new_flight)
    await db.commit()
    await db.refresh(new_flight)

    return FlightReservationResponse(
        reservation_id=new_flight.reservation_id,
        status=new_flight.status,
        price=new_flight.price,
        currency=new_flight.currency
    )

@app.post("/flights/cancel/{reservation_id}", response_model=FlightReservationResponse)
async def cancel_flight_reservation(reservation_id: str, db: AsyncSession = Depends(get_db)):
    result = await db.execute(select(Flight).where(Flight.reservation_id == reservation_id))
    flight = result.scalar_one_or_none()

    if not flight:
        raise HTTPException(status_code=404, detail="flight reservation not found")

    flight.status = "canceled"
    await db.commit()
    await db.refresh(flight)

    return FlightReservationResponse(
        reservation_id=flight.reservation_id,
        status=flight.status,
        price=flight.price,
        currency=flight.currency
    )

# ====== Hotel endpoints ======
@app.post("/hotels/reservations", response_model=HotelReservationResponse)
async def create_hotel_reservation(req: HotelReservationRequest, db: AsyncSession = Depends(ge
    rid = f"h-{uuid4().hex[:8]}"
    price, nights = price_hotel(req)
```

```
    new_hotel = Hotel(
        reservation_id=rid,
        status="hold",
        price=price,
        currency="JPY",
        nights=nights,
        req=req.model_dump(mode='json')
    )
    db.add(new_hotel)
    await db.commit()
    await db.refresh(new_hotel)

    return HotelReservationResponse(
        reservation_id=new_hotel.reservation_id,
        status=new_hotel.status,
        price=new_hotel.price,
        currency=new_hotel.currency,
        nights=new_hotel.nights
    )

@app.post("/hotels/cancel/{reservation_id}", response_model=HotelReservationResponse)
async def cancel_hotel_reservation(reservation_id: str, db: AsyncSession = Depends(get_db)):
    result = await db.execute(select(Hotel).where(Hotel.reservation_id == reservation_id))
    hotel = result.scalar_one_or_none()

    if not hotel:
        raise HTTPException(status_code=404, detail="hotel reservation not found")

    hotel.status = "canceled"
    await db.commit()
    await db.refresh(hotel)

    return HotelReservationResponse(
        reservation_id=hotel.reservation_id,
        status=hotel.status,
        price=hotel.price,
        currency=hotel.currency,
        nights=hotel.nights
    )

# ====== Car endpoints ======
@app.post("/cars/reservations", response_model=CarReservationResponse)
async def create_car_reservation(req: CarReservationRequest, db: AsyncSession = Depends(get_db)
    rid = f"c-{uuid4().hex[:8]}"
    price, days = price_car(req)
```

```
    new_car = Car(
        reservation_id=rid,
        status="hold",
        price=price,
        currency="JPY",
        days=days,
        req=req.model_dump(mode='json')
    )
    db.add(new_car)
    await db.commit()
    await db.refresh(new_car)

    return CarReservationResponse(
        reservation_id=new_car.reservation_id,
        status=new_car.status,
        price=new_car.price,
        currency=new_car.currency,
        days=new_car.days
    )

@app.post("/cars/cancel/{reservation_id}", response_model=CarReservationResponse)
async def cancel_car_reservation(reservation_id: str, db: AsyncSession = Depends(get_db)):
    result = await db.execute(select(Car).where(Car.reservation_id == reservation_id))
    car = result.scalar_one_or_none()

    if not car:
        raise HTTPException(status_code=404, detail="car reservation not found")

    car.status = "canceled"
    await db.commit()
    await db.refresh(car)

    return CarReservationResponse(
        reservation_id=car.reservation_id,
        status=car.status,
        price=car.price,
        currency=car.currency,
        days=car.days
    )

# ====== Restaurant endpoints ======
@app.post("/restaurants/reservations", response_model=RestaurantReservationResponse)
async def create_restaurant_reservation(req: RestaurantReservationRequest, db: AsyncSession = D
    rid = f"r-{uuid4().hex[:8]}"
    price = price_restaurant(req)
```

```
    new_resto = Restaurant(
        reservation_id=rid,
        status="hold",
        price=price,
        currency="JPY",
        party_size=req.party_size,
        req=req.model_dump(mode='json')
    )
    db.add(new_resto)
    await db.commit()
    await db.refresh(new_resto)

    return RestaurantReservationResponse(
        reservation_id=new_resto.reservation_id,
        status=new_resto.status,
        price=new_resto.price,
        currency=new_resto.currency,
        party_size=new_resto.party_size
    )

@app.post("/restaurants/cancel/{reservation_id}", response_model=RestaurantReservationResponse)
async def cancel_restaurant_reservation(reservation_id: str, db: AsyncSession = Depends(get_db)):
    result = await db.execute(select(Restaurant).where(Restaurant.reservation_id == reservation_id))
    resto = result.scalar_one_or_none()

    if not resto:
        raise HTTPException(status_code=404, detail="restaurant reservation not found")

    resto.status = "canceled"
    await db.commit()
    await db.refresh(resto)

    return RestaurantReservationResponse(
        reservation_id=resto.reservation_id,
        status=resto.status,
        price=resto.price,
        currency=resto.currency,
        party_size=resto.party_size
    )

# ====== Payment endpoints ======
@app.post("/payments/charge", response_model=PaymentChargeResponse)
async def charge_payment(req: PaymentChargeRequest, db: AsyncSession = Depends(get_db)):
    # Idempotency check with DB
    result = await db.execute(select(Idempotency).where(Idempotency.key == req.idempotency_key))
```

```
    existing_idempotency = result.scalar_one_or_none()

    if existing_idempotency:
        return PaymentChargeResponse(**existing_idempotency.response_json)

    payment_id = f"p-{uuid4().hex[:8]}"
    status = "failed" if req.simulate_failure else "succeeded"

    # Create Response Object
    resp = PaymentChargeResponse(
        payment_id=payment_id, status=status, amount=req.amount,
        currency=req.currency, method=req.method, idempotency_key=req.idempotency_key
    )

    # Store in DB
    new_payment = Payment(
        payment_id=payment_id,
        status=status,
        amount=req.amount,
        currency=req.currency,
        method=req.method,
        idempotency_key=req.idempotency_key
    )

    # Store Idempotency
    new_idempotency = Idempotency(
        key=req.idempotency_key,
        response_json=resp.model_dump(mode='json')
    )

    db.add(new_payment)
    db.add(new_idempotency)
    await db.commit()

    return resp

# ====== Orchestrator (Checkout) ======
@app.post("/checkout", response_model=CheckoutResponse)
async def checkout(req: CheckoutRequest, db: AsyncSession = Depends(get_db)):
    # 1) Hold reservations
    # Note: We are calling the internal logic of endpoints basically, but in FastAPI it's bette
    # For simplicity in this refactor, we are calling the internal logic which happens to be ac
    # However, since the endpoint functions expect `db` dependency, we can just call them direc
    # A cleaner structure would extract "Service" layer functions, but we will call the endpoi
    # (which is technically okay in FastAPI if deps are handled, but here we manually pass db)
```

```
flight = await create_flight_reservation(req.flight, db)
hotel  = await create_hotel_reservation(req.hotel, db)
car    = await create_car_reservation(req.car, db)
resto  = await create_restaurant_reservation(req.restaurant, db)

total = flight.price + hotel.price + car.price + resto.price

# 2) Charge payment
payment_req = PaymentChargeRequest(
    amount=total, currency="JPY", method=req.payment_method,
    idempotency_key=req.idempotency_key, simulate_failure=req.simulate_failure
)
payment = await charge_payment(payment_req, db)

# 3) Confirm or rollback
if payment.status == "succeeded":
    # Confirm all - Update DB
    if flight_db := await db.get(Flight, flight.reservation_id):
        flight_db.status = "confirmed"
        flight.status = "confirmed"
    if hotel_db := await db.get(Hotel, hotel.reservation_id):
        hotel_db.status = "confirmed"
        hotel.status = "confirmed"
    if car_db := await db.get(Car, car.reservation_id):
        car_db.status = "confirmed"
        car.status = "confirmed"
    if resto_db := await db.get(Restaurant, resto.reservation_id):
        resto_db.status = "confirmed"
        resto.status = "confirmed"

    await db.commit()

else:
    # Rollback (cancel all) - utilizing the cancel endpoints (or logic)
    await cancel_flight_reservation(flight.reservation_id, db)
    await cancel_hotel_reservation(hotel.reservation_id, db)
    await cancel_car_reservation(car.reservation_id, db)
    await cancel_restaurant_reservation(resto.reservation_id, db)

    flight.status = "canceled"
    hotel.status = "canceled"
    car.status = "canceled"
    resto.status = "canceled"

return CheckoutResponse(
    flight=flight, hotel=hotel, car=car, restaurant=resto,
```

```
    payment=payment, total_amount=total, currency="JPY"
)
```