FastAPI - Practice Sheet

10 Q&A on type hints, Pydantic, async, DI, and OpenAPI

Q1. How do Python type hints drive FastAPI's validation and docs?

A: FastAPI reads function parameter annotations at import time and builds JSON Schema from them using Pydantic. It validates incoming requests against the schema, coerces types, and populates the OpenAPI spec all from the same source of truth.

Q2. What is a Pydantic model and how does it validate data?

A: A Pydantic BaseModel subclass defines fields with Python type annotations. On instantiation Pydantic validates each field, coerces compatible types (e.g., '3' to int), and raises ValidationError with detailed messages if validation fails.

Q3. How does FastAPI's dependency injection work with Depends()?

A: A dependency is any callable annotated in a parameter as Depends(get_db). FastAPI calls it before the route handler, resolves nested Depends(), and injects the result. yield-based dependencies act as context managers for setup/teardown (e.g., DB sessions).

Q4. How does FastAPI handle path operation ordering to avoid shadowing?

A: FastAPI registers routes in declaration order. A fixed path /users/me must be declared before the parameterized /users/{user_id}; otherwise the router matches 'me' as a user_id value. Declare more specific routes first.

Q5. What is the difference between async def and def in a FastAPI route?

A: async def routes run on the event loop never block them with synchronous I/O. def routes are automatically run in a threadpool by FastAPI so they don't block the event loop. Use async def with await for async libraries; def for synchronous blocking code.

Q6. How do you implement OAuth2 password flow with JWT in FastAPI?

A: Use OAuth2PasswordBearer to extract tokens from Authorization headers. On login, verify credentials, create a JWT with python-jose encoding claims and expiry, and return it. Protect routes with Depends(get_current_user) which decodes and validates the token.

Q7. How does FastAPI auto-generate OpenAPI documentation?

A: FastAPI builds an OpenAPI 3 schema from route decorators, type annotations, and Pydantic models at startup. It serves interactive Swagger UI at /docs and ReDoc at /redoc. Use response_model, tags, and summary arguments to customize the spec.

Q8. How do you add CORS support to a FastAPI application?

A: Add CORSMiddleware: app.add_middleware(CORSMiddleware, allow_origins=['https://myfrontend.com'], allow_methods=['*'], allow_headers=['*']). FastAPI handles preflight OPTIONS requests automatically. Use allow_origins=['*'] only in development.

Q9. How do you use SQLAlchemy async sessions with FastAPI?

A: Use create_async_engine with an async driver (asyncpg for PostgreSQL). Define a get_db dependency that yields an AsyncSession. Annotate all repository methods with async and use await session.execute(stmt). Return results with scalars().all().

Q10. How do you test FastAPI endpoints with TestClient?

A: Instantiate TestClient(app). Call client.get('/path') or client.post('/path', json={...}) synchronously. Override dependencies for tests using app.dependency_overrides[get_db] = override_get_db. Use pytest fixtures to manage setup and teardown.