Advanced FastAPI Interview Q&A;

1. What is FastAPI and why is it popular?

FastAPI is a modern, high-performance, web framework for Python that is based on ASGI. It is popular due to automatic request validation, type hints support, async performance, and auto-generated API documentation.

2. Difference between Flask and FastAPI?

Flask is synchronous and based on WSGI, while FastAPI is asynchronous and based on ASGI. FastAPI offers built-in data validation with Pydantic and automatic OpenAPI docs, unlike Flask.

3. How does FastAPI handle validation?

Validation in FastAPI is handled using Pydantic models. These models validate request bodies, query parameters, and responses automatically.

4. What is dependency injection in FastAPI?

Dependency injection allows you to declare shared resources (like DB sessions, services) using Depends(). It makes code reusable and testable.

5. How do you create background tasks in FastAPI?

FastAPI provides BackgroundTasks which can be injected into endpoints to execute tasks after sending the response.

6. Explain how async works in FastAPI.

Async endpoints use Python's asyncio library. It allows FastAPI to handle many requests concurrently without blocking.

7. How does FastAPI generate docs?

FastAPI automatically generates Swagger UI and ReDoc from OpenAPI specification.

8. How do you secure APIs with OAuth2 in FastAPI?

FastAPI provides OAuth2PasswordBearer for token authentication. Typically, JWT tokens are used for securing APIs.

9. How do you handle middleware in FastAPI?

You can use @app.middleware('http') to intercept requests and responses, useful for logging or authentication checks.

10. How do you implement caching in FastAPI?

Caching can be implemented using external tools like Redis, or in-memory caching with dependencies/middleware.

11. How does FastAPI handle WebSockets?

FastAPI supports WebSockets via @app.websocket routes using Starlette’s WebSocket support.

12. How to integrate FastAPI with databases?

FastAPI supports SQLAlchemy, Tortoise ORM, or async drivers (like asyncpg) for database integration.

13. How do you do testing in FastAPI?

FastAPI provides TestClient (built on requests) for testing endpoints, often with pytest.

14. How to configure logging in FastAPI?

Python logging module or third-party libraries like loguru can be configured to capture logs.

15. How do you serve static files?

Static files can be served using StaticFiles from starlette: app.mount('/static', StaticFiles(directory='static'), name='static').

16. How do you handle file uploads?

File uploads are handled using File and UploadFile, which provides async file operations.

17. How do you implement custom exception handling?

You can define custom exception classes and register them with @app.exception_handler.

18. What is Starlette's role in FastAPI?

FastAPI is built on top of Starlette, which provides ASGI support, routing, and WebSockets.

19. How do you structure large FastAPI projects?

Use modular design: routers for endpoints, schemas for validation, services for logic, db for models, and core for configs.

20. How do you deploy FastAPI?

Deploy using Uvicorn, or Gunicorn + Uvicorn workers, often behind Nginx or in Docker/Kubernetes.

21. What is ASGI and how does it relate to FastAPI?

ASGI (Asynchronous Server Gateway Interface) is the async standard that FastAPI is built on.

22. Difference between WSGI and ASGI?

WSGI supports only synchronous requests, while ASGI supports async requests and WebSockets.

23. How do you handle database transactions?

With SQLAlchemy sessions or async sessions injected as dependencies.

24. How do you handle business validation rules?

You can use Pydantic validators or custom dependency functions to enforce business rules.

25. How do you implement rate limiting?

Rate limiting can be implemented using middleware or libraries like slowapi with Redis backend.

26. How do you configure CORS?

Use CORSMiddleware to allow/deny origins, methods, and headers.

27. What are response classes in FastAPI?

FastAPI provides JSONResponse, HTMLResponse, FileResponse, and StreamingResponse for custom responses.

28. How do you stream large files?

Use StreamingResponse with a file iterator to send large files efficiently.

29. How do you integrate Celery with FastAPI?

Celery workers can be used for distributed background tasks, triggered from FastAPI endpoints.

30. How do you implement logging?

Attach middleware for request/response logging, use logging/loguru for app logs.

31. How do you implement API versioning?

Use router prefixes like /api/v1 and /api/v2 for versioning.

32. How do you document APIs?

FastAPI auto-docs using Swagger UI and ReDoc; tags and descriptions can be added for clarity.

33. How do you handle environment configs?

Use Pydantic's BaseSettings with .env files for managing environment variables.

34. How to improve FastAPI performance?

Use async DB drivers, caching (Redis), gzip compression, and scale with multiple workers.

35. How do you implement social login (Google/Facebook)?

Use libraries like Authlib or fastapi-users for OAuth2 integrations.

36. How do you serve static files in production?

Static files should be served via CDN or reverse proxy like Nginx.

37. How do you test async endpoints?

Use pytest with pytest-asyncio, and httpx for async test clients.

38. How do you ensure security best practices in FastAPI?

Use HTTPS, JWT tokens, OAuth2, CORS, input validation, and dependency injection for auth checks.

39. What are FastAPI limitations?

Smaller ecosystem than Django/Flask, async ORM maturity is still growing, steep async learning curve.

40. When would you not use FastAPI?

Not ideal for simple scripts, very static websites, or projects where async adds unnecessary complexity.