FastAPI Interview Questions & Answers

■ Basic Level (Entry / Fresher)

Q: What is FastAPI? Why is it popular?
A: FastAPI is a modern, fast (high-performance) Python web framework for building APIs. It’s popular because it is built on Starlette (fast ASGI framework) and Pydantic (data validation). It provides automatic API documentation (Swagger UI, ReDoc), strong type hint support, and asynchronous programming using async/await.

Q: How do you create a simple FastAPI application?
A:
```
from fastapi import FastAPI
app = FastAPI()

@app.get('/')
def home():
return {'message': 'Hello, FastAPI!'}
```

Run using: `uvicorn main:app --reload`

Q: How does FastAPI handle request data validation?
A: FastAPI uses Pydantic models for validation. When a request is received, data is validated against the model schema. Invalid data results in a 422 Unprocessable Entity error.

Example:
```
from pydantic import BaseModel

class User(BaseModel):
name: str
age: int

@app.post('/users/')
def create_user(user: User):
return user
```

Q: What is the difference between @app.get() and @app.post()?
A: @app.get() defines an endpoint for retrieving data (safe, idempotent).
@app.post() defines an endpoint for creating resources (non-idempotent, modifies server state).

Q: What built-in documentation tools does FastAPI provide?
A: FastAPI automatically generates documentation at:
- http://127.0.0.1:8000/docs (Swagger UI)
- http://127.0.0.1:8000/redoc (ReDoc UI)

■ Intermediate Level (2+ Years Experience)

Q: How does FastAPI support asynchronous programming?
A: FastAPI supports async def functions for non-blocking I/O. This allows concurrent tasks such as DB queries or external API calls.

Example:

```
@app.get('/async')
async def async_endpoint():
return {'msg': 'This is async'}
```

Q: How do you handle authentication and authorization in FastAPI?
A: FastAPI provides OAuth2PasswordBearer, JWT, and API keys. Example using OAuth2:

```
from fastapi.security import OAuth2PasswordBearer
from fastapi import Depends

oauth2_scheme = OAuth2PasswordBearer(tokenUrl='token')

@app.get('/users/me')
def read_users_me(token: str = Depends(oauth2_scheme)):
return {'token': token}
```

Q: Explain dependency injection in FastAPI. Why is it useful?
A: Dependencies are reusable functions/classes injected into endpoints using Depends(). They are useful for authentication, database sessions, caching, or logging.

Example:

```
from fastapi import Depends

def get_db():
return 'DB Connection'

@app.get('/items/')
def get_items(db=Depends(get_db)):
return {'db': db}
```

Q: How does FastAPI integrate with databases?
A: FastAPI integrates with ORMs like SQLAlchemy, Tortoise ORM, or async libraries like databases. Typical pattern is using dependency injection for DB sessions.

Example:

```
from sqlalchemy.orm import Session

def get_db():
db = SessionLocal()
try:
yield db
finally:
db.close()
```

Q: What’s the difference between FastAPI and Flask/Django?
A: Flask: Lightweight but requires extra libraries for validation and docs.
Django: Full-stack framework, heavier for APIs.
FastAPI: Built-in validation, async support, automatic docs, faster performance.

Q: How do you handle background tasks in FastAPI?
A: FastAPI provides BackgroundTasks for async background jobs.

Example:

```
from fastapi import BackgroundTasks

def log_message(msg: str):
print(msg)

@app.post('/send/')
def send_msg(message: str, background_tasks: BackgroundTasks):
background_tasks.add_task(log_message, message)
return {'status': 'Message scheduled'}
```

Q: What are middlewares in FastAPI?
A: Middleware is a function that runs before and after every request. It can be used for logging, authentication, CORS, etc.

Example:

```
@app.middleware('http')
async def log_requests(request, call_next):
response = await call_next(request)
return response
```

Q: How does FastAPI handle file uploads?
A: FastAPI uses File and UploadFile for file uploads.

Example:

```
from fastapi import File, UploadFile

@app.post('/upload/')
async def upload_file(file: UploadFile = File(...)):
return {'filename': file.filename}
```

Q: How can you improve performance in a FastAPI application?
A: • Use async I/O for non-blocking tasks
• Use connection pooling for databases
• Add caching (Redis, in-memory)
• Optimize middlewares
• Deploy with Uvicorn + Gunicorn