Python Technical Questions & Answers

Top Python Interview Questions & Answers for 10+ Years Experienced Candidates

If you’re a **senior-level Python developer (10+ years of experience)**, expect **deep technical, architectural, and problem-solving questions** covering **advanced Python concepts, performance optimization, security, scalability, and best practices**.

1️⃣ What are Python’s key features?

✅ **Answer:**

- **High-Level & Easy to Read**
- **Interpreted & Dynamically Typed**
- **Memory Management & Garbage Collection**
- **Multi-Paradigm (OOP, Functional, Procedural)**
- **Extensive Libraries & Frameworks**

2️⃣ How is memory managed in Python?

✅ **Answer:**
Python uses **Automatic Memory Management** with:

- **Reference Counting** – Keeps track of object references.
- **Garbage Collector** – Removes cyclic references (circular dependencies).
- **Memory Pooling (PyMalloc)** – Allocates memory blocks efficiently.

✔ **Example (Reference Counting):**

python
CopyEdit

```
import sys
a = "Python"
b = a  # Reference count increases
print(sys.getrefcount(a))  # Outputs reference count
del a  # Reference count decreases
```

✔ **Example (Garbage Collection):**

python
CopyEdit

```
import gc
gc.collect()  # Forces garbage collection
```

3️⃣ What are Python's built-in data types?

✅ **Answer:**

Type	Example
Numeric	`int`, `float`, `complex`
Sequence	`list`, `tuple`, `range`
Text	`str`
Set	`set`, `frozenset`
Mapping	`dict`
Boolean	`True`, `False`
Binary	`bytes`, `bytearray`, `memoryview`

✔ **Example (List vs Tuple Performance):**

python
CopyEdit

```
import timeit
print(timeit.timeit("x=(1,2,3,4,5)", number=1000000))  # Faster
print(timeit.timeit("x=[1,2,3,4,5]", number=1000000))  # Slower
```

🔹 **Why?** Tuples are **immutable**, making them faster than lists.

4️⃣ What is the difference between deep copy and shallow copy?

✅ **Answer:**

Type	Behavior
Shallow Copy	Copies **references** (changes reflect in both).
Deep Copy	Copies **actual objects** (independent copies).

✔ **Example:**

python
CopyEdit
```
import copy
list1 = [[1, 2], [3, 4]]
shallow = copy.copy(list1)
deep = copy.deepcopy(list1)

list1[0][0] = 99
print(shallow[0][0])  # 99 (Affected)
print(deep[0][0])  # 1 (Unaffected)
```

5️⃣ What are Python decorators?

✅ **Answer:**
Decorators modify function behavior **without modifying their structure**.

✔ **Example – Logging Decorator:**

python
CopyEdit
```
def logger(func):
    def wrapper(*args, **kwargs):
        print(f"Executing {func.__name__} with {args}")
        return func(*args, **kwargs)
    return wrapper

@logger
def add(a, b):
    return a + b

print(add(3, 5))
```

6️⃣ How do Python generators work?

✅ **Answer:**
Generators **yield values** instead of returning them, making them memory efficient.

✔ **Example:**

python
CopyEdit
```
def count():
    yield 1
    yield 2
    yield 3

gen = count()
print(next(gen))  # 1
print(next(gen))  # 2
```

🔹 **Why use generators?**

- Saves memory by **yielding values lazily**.

- Efficient for **large datasets & infinite sequences**.

7️⃣ What are Python’s built-in functions for functional programming?

✅ **Answer:**

Function	**Use Case**
map()	Applies a function to an iterable
filter()	Filters elements based on a condition
reduce()	Reduces an iterable to a single value
lambda	Anonymous functions

✔ **Example:**

python
CopyEdit
```python
from functools import reduce

nums = [1, 2, 3, 4, 5]
print(list(map(lambda x: x * 2, nums)))  # [2, 4, 6, 8, 10]
print(list(filter(lambda x: x % 2 == 0, nums)))  # [2, 4]
print(reduce(lambda x, y: x + y, nums))  # 15
```

8️⃣ What are metaclasses in Python?

✅ **Answer:**
Metaclasses **control the creation of classes**.

✔ **Example – Custom Metaclass:**

python

CopyEdit
```
class Meta(type):
    def __new__(cls, name, bases, dct):
        dct['custom_attr'] = 42
        return super().__new__(cls, name, bases, dct)

class MyClass(metaclass=Meta):
    pass

print(MyClass.custom_attr)  # 42
```

🔹 **Why use metaclasses?**

- Enforce class-level validation.
- Dynamically modify class behavior.

9️⃣ How do you handle concurrency in Python?

✅ **Answer:**

✔ **Threads (Multithreading)** – Suitable for I/O-bound tasks.

python
CopyEdit
```
import threading

def task():
    print("Thread running")

t = threading.Thread(target=task)
t.start()
t.join()
```

✔ **Multiprocessing (Parallel Execution)** – Suitable for CPU-bound tasks.

python

CopyEdit
```
import multiprocessing

def task():
    print("Process running")

p = multiprocessing.Process(target=task)
p.start()
p.join()
```

✔ **Asyncio (Event Loop for Asynchronous Execution)** – Ideal for **high-performance network apps**.

python
CopyEdit
```
import asyncio

async def main():
    print("Async Task")
    await asyncio.sleep(1)

asyncio.run(main())
```

🔟 What are the key differences between Python 2 and Python 3?

✅ **Answer:**

Feature	Python 2	Python 3
Print Statement	`print "Hello"`	`print("Hello")`
Integer Division	`5/2 = 2`	`5/2 = 2.5`
Unicode	ASCII default	Unicode default
Error Handling	`except Exception, e:`	`except Exception as e:`

Iterators	`range()` returns list	`range()` returns generator

🔹 **Python 2 is outdated, always use Python 3!**

Final Thoughts

As a **10+ years experienced developer**, you need to focus on:

- **Advanced Python concepts (decorators, metaclasses, async programming, memory optimization).**
- **Architecture and Performance tuning.**
- **Real-world problems and best practices.**

Top Python Django Interview Questions & Answers for 10+ Years Experienced Candidates

For senior-level Django developers, expect deep technical questions on architecture, ORM optimizations, security, scalability, and best practices.

1️⃣What are Django's key features?

✅ **Answer:**

- **MTV Architecture – Model, Template, View.**
- **ORM (Object-Relational Mapper) – No SQL required for DB operations.**
- **Built-in Admin Panel – For managing database records.**
- **Middleware Support – Modify requests and responses.**
- **Scalability & Security – Inbuilt CSRF, XSS protection.**

2️⃣ Explain Django’s MTV architecture.

✅ **Answer:**

Componen t	Description
Model	Defines database structure using Python classes.
Template	Renders HTML UI dynamically.
View	Handles request-processing logic.

✔ **Example:**

python

CopyEdit

```
# models.py

class Book(models.Model):

    title = models.CharField(max_length=100)
```

python

CopyEdit

```
# views.py

from django.shortcuts import render
from .models import Book


def book_list(request):

    books = Book.objects.all()

    return render(request, "books.html", {"books": books})
```

html

CopyEdit

```
<!-- templates/books.html -->

{% for book in books %}

  <p>{{ book.title }}</p>

{% endfor %}
```

3️⃣ What are Django middlewares and how do they work?

✅ **Answer:**
 Middleware is a function that processes requests and responses globally before they reach the view or after leaving the view.

✔ **Example – Custom Middleware (Logging Requests):**

python

CopyEdit

```
class LogMiddleware:

    def __init__(self, get_response):

        self.get_response = get_response
    def __call__(self, request):

        print(f"Request Path: {request.path}")

        response = self.get_response(request)

        return response
```

✔ **Register Middleware in `settings.py`:**

python

CopyEdit

```
MIDDLEWARE = [

    'django.middleware.security.SecurityMiddleware',

    'myapp.middleware.LogMiddleware',  # Custom middleware

]
```

4️⃣ What are Django signals?

✅ **Answer:**
Django Signals allow decoupled components to communicate when certain actions occur (e.g., saving a model).

✔ **Example – Send Email on User Signup:**

python

CopyEdit

```
from django.db.models.signals import post_save

from django.contrib.auth.models import User
from django.dispatch import receiver


@receiver(post_save, sender=User)
def send_welcome_email(sender, instance, created, **kwargs):
    if created:
        print(f"Welcome email sent to {instance.email}")
```

5️⃣ What is Django's ORM, and how does it optimize queries?

✅ **Answer:**
Django ORM allows database interactions using Python without writing raw SQL.

✔ **Performance Optimization Techniques:**
1️⃣ **Use `select_related()` and `prefetch_related()` to reduce queries.**
2️⃣ **Avoid `for` loops for database queries.**
3️⃣ **Use `only()` and `defer()` to limit fields fetched.**

✔ **Example – Query Optimization:**

python

CopyEdit

```
# Bad (Multiple Queries)
books = Book.objects.all()
for book in books:
    print(book.author.name)  # Causes multiple queries


# Optimized (Single Query)
```

```
books = Book.objects.select_related("author").all()
```

6️⃣ How do you implement caching in Django?

✅ **Answer:**
Django provides multiple caching backends (Memcached, Redis, Database, File-based).

✔ Example – Using Redis Cache:
1️⃣ Install Redis:

bash

CopyEdit

```
pip install django-redis
```

2️⃣ Configure in `settings.py`:

python

CopyEdit

```
CACHES = {

    'default': {

        'BACKEND': 'django_redis.cache.RedisCache',

        'LOCATION': 'redis://127.0.0.1:6379/1',

        'OPTIONS': {'CLIENT_CLASS':
'django_redis.client.DefaultClient'},

    }

}
```

3️⃣ **Use in Views:**

python

CopyEdit

```python
from django.core.cache import cache


def expensive_view(request):
    data = cache.get("my_data")
    if not data:
        data = "Heavy computation result"
        cache.set("my_data", data, timeout=60)
    return HttpResponse(data)
```

7️⃣ How do you secure a Django application?

✅ **Answer:**
✔ **Best Security Practices:**

- **Use `ALLOWED_HOSTS` – Prevent HTTP Host Header attacks.**
- **Enable CSRF Protection – Use `@csrf_protect` or `{% csrf_token %}`.**
- **Use `SECURE_SSL_REDIRECT` and `SECURE_HSTS_SECONDS`.**
- **Validate user input to prevent SQL injection.**

✔ **Example – CSRF Protection in Forms:**

html

CopyEdit

```
<form method="POST">

    {% csrf_token %}

    <input type="text" name="username">

    <button type="submit">Submit</button>

</form>
```

8️⃣ How does Django handle file uploads?

✅ **Answer:**
Django provides `FileField` and `ImageField` for file handling.

✔ **Example – Handling Image Uploads:**

python

CopyEdit

```python
# models.py

class Profile(models.Model):

    picture = models.ImageField(upload_to="uploads/")
```

✔ **Settings for Media Files:**

python

CopyEdit

```python
MEDIA_URL = "/media/"

MEDIA_ROOT = os.path.join(BASE_DIR, "media")
```

✔ **URL Configuration:**

python

CopyEdit

```
from django.conf import settings

from django.conf.urls.static import static


urlpatterns += static(settings.MEDIA_URL,
document_root=settings.MEDIA_ROOT)
```

9️⃣ What are class-based views (CBVs) in Django?

✅ **Answer:**
CBVs provide reusable views with built-in functionality, replacing function-based views (FBVs).

✔ **Example – ListView for Displaying Books:**

python

CopyEdit

```
from django.views.generic import ListView

from .models import Book


class BookListView(ListView):

    model = Book

    template_name = "books.html"
```

✔ **URL Configuration:**

python

CopyEdit

```
from django.urls import path

from .views import BookListView


urlpatterns = [

    path('books/', BookListView.as_view(), name='book-list'),

]
```

🔟 What are Django REST Framework (DRF) serializers?

✅ **Answer:**
Serializers convert complex data (DB objects) into JSON and vice versa.

✔ **Example – Serializing Django Model:**

python

CopyEdit

```
from rest_framework import serializers

from .models import Book


class BookSerializer(serializers.ModelSerializer):

    class Meta:

        model = Book
        fields = "__all__"
```

✔ **Use in Django View:**

python

CopyEdit

```
from rest_framework.response import Response
from rest_framework.views import APIView
from .models import Book
from .serializers import BookSerializer


class BookAPIView(APIView):
    def get(self, request):
        books = Book.objects.all()
        serializer = BookSerializer(books, many=True)
        return Response(serializer.data)
```

Final Thoughts

As a 10+ years experienced Django developer, you should focus on:

- **Advanced ORM Optimizations & Query Performance.**
- **Security Best Practices (CSRF, SQL Injection, Authentication).**
- **High-Performance Caching & Scaling (Redis, Celery, Async Views).**
- **Django REST Framework (DRF) & API Security.**

Top Python Flask Interview Questions & Answers for 10+ Years Experienced Candidates

For senior-level Flask developers, expect deep technical questions on architecture, scalability, security, API best practices, and performance optimizations.

1️⃣ What are the key features of Flask?

✅ **Answer:**
Flask is a lightweight, micro-framework for web development in Python. Key features include:

- **Minimalistic & Extensible – No built-in ORM, uses third-party extensions.**
- **Jinja2 Templating Engine – For rendering dynamic HTML.**
- **Built-in Development Server & Debugger.**
- **URL Routing & REST API Support.**
- **WSGI Compliant & Asynchronous Support (via `gevent` or `asyncio`).**

✔ **Example – Basic Flask App:**

python

CopyEdit

```python
from flask import Flask


app = Flask(__name__)


@app.route("/")
def home():
    return "Hello, Flask!"
```

```
if __name__ == "__main__":

    app.run(debug=True)
```

2️⃣ How does Flask differ from Django?

✅ **Answer:**

Feature	Flask	Django
Architecture	Micro-framework	Full-stack framework
ORM	Not built-in, uses SQLAlchemy	Built-in Django ORM
Flexibility	Highly customizable	More opinionated
Use Case	Lightweight apps, APIs, microservices	Large, monolithic apps

---

3️⃣ How does Flask handle database operations?

✅ **Answer:**
Flask does not have a built-in ORM, but SQLAlchemy is commonly used.

✔ Example – Using Flask-SQLAlchemy:

python

CopyEdit

```
from flask import Flask

from flask_sqlalchemy import SQLAlchemy


app = Flask(__name__)

app.config["SQLALCHEMY_DATABASE_URI"] = "sqlite:///test.db"

db = SQLAlchemy(app)


class User(db.Model):

    id = db.Column(db.Integer, primary_key=True)

    name = db.Column(db.String(100))


db.create_all()
```

4️⃣ How do you structure a large Flask application?

✅ **Answer:**
For scalable Flask apps, use the Blueprint pattern.

✔ **Folder Structure:**

bash

CopyEdit

```
/my_flask_app

    /app
```

```
        /routes
            user.py
            product.py
        /models
            user.py
            product.py
        __init__.py
    run.py
```

✔ **Registering Blueprints (`__init__.py`):**

python

CopyEdit

```
from flask import Flask
from app.routes.user import user_bp

app = Flask(__name__)
app.register_blueprint(user_bp, url_prefix="/users")
```

✔ **Creating a Blueprint (`routes/user.py`):**

python

CopyEdit

```
from flask import Blueprint
```

```
user_bp = Blueprint("user", __name__)


@user_bp.route("/")

def user_home():

    return "User Home"
```

5️⃣ What is Flask Middleware?

✅ **Answer:**
Middleware processes requests before reaching the view or responses before sending to the client.

✔ Example – Custom Middleware:

python

CopyEdit

```
from flask import request


@app.before_request

def before_request():

    print(f"Incoming request: {request.url}")


@app.after_request

def after_request(response):

    response.headers["X-Frame-Options"] = "DENY"

    return response
```

6️⃣ How do you implement authentication in Flask?

✅ **Answer:**
Use Flask-JWT-Extended for JWT-based authentication.

✔ **Example – Implementing JWT Authentication:**

python

CopyEdit

```
from flask import Flask, jsonify

from flask_jwt_extended import JWTManager, create_access_token,
jwt_required, get_jwt_identity


app = Flask(__name__)

app.config["JWT_SECRET_KEY"] = "supersecret"

jwt = JWTManager(app)


@app.route("/login", methods=["POST"])

def login():

    access_token = create_access_token(identity="user1")

    return jsonify(access_token=access_token)


@app.route("/protected", methods=["GET"])

@jwt_required()

def protected():
    return jsonify(message=f"Hello {get_jwt_identity()}!")


if __name__ == "__main__":

    app.run()
```

7️⃣ How do you implement Flask caching for performance optimization?

✅ **Answer:**
Use Flask-Caching with Redis or Memcached.

✔ **Example – Setting Up Redis Cache:**

python

CopyEdit

```python
from flask import Flask

from flask_caching import Cache


app = Flask(__name__)

app.config["CACHE_TYPE"] = "RedisCache"

app.config["CACHE_REDIS_URL"] = "redis://localhost:6379/0"

cache = Cache(app)


@app.route("/slow")

@cache.cached(timeout=60)
```

```
def slow_function():
    return "Cached Response"

if __name__ == "__main__":
    app.run()
```

8️⃣ How do you handle background tasks in Flask?

✅ **Answer:**
Use Celery for background tasks.

✔ Steps to Integrate Celery:
1️⃣ Install Celery:

bash

CopyEdit

```
pip install celery
```

2️⃣ Configure Celery (`celery.py`):

python

CopyEdit

```
from celery import Celery

celery = Celery(
    "tasks",
    broker="redis://localhost:6379/0",
    backend="redis://localhost:6379/0",
)
```

3️⃣ **Define Background Task (`tasks.py`):**

python

CopyEdit

```python
from celery import Celery


@celery.task
def add(x, y):
    return x + y
```

4️⃣ **Run Celery Worker:**

bash

CopyEdit

```bash
celery -A tasks worker --loglevel=info
```

9️⃣ How do you secure a Flask API?

✅ **Answer:**

- **Use HTTPS (TLS/SSL).**
- **Implement JWT-based authentication.**

- **Sanitize user input to prevent SQL injection.**
- **Use Rate Limiting (Flask-Limiter).**

✔ Example – Enforcing Rate Limits:

python

CopyEdit

```
from flask_limiter import Limiter


limiter = Limiter(app, key_func=lambda: request.remote_addr)


@app.route("/limited")

@limiter.limit("10 per minute")

def limited_route():

    return "Limited API"
```

🔟 How do you handle error handling in Flask?

✅ Answer:
Use error handlers for structured responses.

✔ Example – Custom 404 Error Handling:

python

CopyEdit

```
@app.errorhandler(404)

def not_found(error):
    return jsonify({"error": "Not Found"}), 404
```

✔ **Example – Handling Uncaught Exceptions:**

python

CopyEdit

```
@app.errorhandler(Exception)

def handle_exception(e):

    return jsonify({"error": str(e)}), 500
```

Final Thoughts

For a 10+ years experienced Flask developer, focus on:
✔ **Building Large-Scale Flask Applications (Blueprints, Modular Structure).**
✔ **Optimizing Performance (Caching, SQLAlchemy Optimizations).**
✔ **Secure REST API Development (JWT, OAuth, Rate Limiting).**
✔ **Background Jobs & Asynchronous Processing (Celery, Async Views).**
✔ **Deployment Strategies (Docker, Nginx, Gunicorn, Kubernetes).**

Top Python FastAPI Interview Questions & Answers for 10+ Years Experienced Candidates

For senior-level FastAPI developers, expect deep technical questions on architecture, performance, async handling, security, database integrations, and API best practices.

1️⃣ What is FastAPI, and how does it differ from Flask?

✅ **Answer:**
FastAPI is a modern, high-performance web framework for Python, designed for building APIs with automatic validation, async support, and type hints.

Feature	FastAPI	Flask
Performance	Faster (ASGI-based, async support)	Slower (WSGI-based, synchronous)
Type Checking	Built-in using Pydantic	No built-in validation
Async Support	First-class support for async/await	Limited async support
API Documentation	Auto-generated (Swagger, ReDoc)	Requires Flask-RESTPlus or Flask-Swagger
WebSocket Support	Yes	Needs extra setup

2️⃣ What are the key features of FastAPI?

✅ **Answer:**

- **ASGI-based framework (supports async processing).**
- **Automatic API documentation (Swagger UI & ReDoc).**
- **Pydantic for request validation & serialization.**
- **Dependency Injection System for modular services.**
- **WebSockets, GraphQL, Background Tasks Support.**

✔ **Example – Simple FastAPI Application:**

python

CopyEdit

```python
from fastapi import FastAPI


app = FastAPI()


@app.get("/")

def read_root():

    return {"message": "Hello, FastAPI!"}
```

3️⃣ How does FastAPI handle request validation?

✅ **Answer:**
FastAPI uses Pydantic models for automatic request validation.

✔ Example – Request Body Validation using Pydantic:

python

CopyEdit

```python
from fastapi import FastAPI

from pydantic import BaseModel


app = FastAPI()


class User(BaseModel):
    name: str

    age: int

    email: str


@app.post("/create_user/")

def create_user(user: User):

    return {"message": f"User {user.name} created!"}
```

🔹 **FastAPI automatically validates the request body and returns error messages if the input is incorrect.**

4️⃣ How do you make FastAPI applications asynchronous?

✅ **Answer:**
FastAPI natively supports async/await for non-blocking I/O operations.

✔ **Example – Async Function in FastAPI:**

python

CopyEdit

```
import asyncio

from fastapi import FastAPI



app = FastAPI()



@app.get("/async_task")

async def async_task():
```

```python
    await asyncio.sleep(2)

    return {"message": "Async task completed"}
```

🚀 **Advantages:**

- **Non-blocking API requests.**
- **Better performance with high-concurrency applications.**

5️⃣ How do you handle database operations in FastAPI?

✅ **Answer:**
Use SQLAlchemy with Async Support for database interactions.

✔ **Example – FastAPI with SQLAlchemy & Async Support:**

python

CopyEdit

```python
from sqlalchemy.ext.asyncio import AsyncSession, create_async_engine

from sqlalchemy.orm import sessionmaker



DATABASE_URL = "sqlite+aiosqlite:///./test.db"




engine = create_async_engine(DATABASE_URL, echo=True)

SessionLocal = sessionmaker(autocommit=False, autoflush=False,
bind=engine, class_=AsyncSession)



async def get_db():
    async with SessionLocal() as session:
        yield session
```

6️⃣ How does FastAPI automatically generate API documentation?

✅ **Answer:**
FastAPI automatically generates interactive documentation using Swagger UI & ReDoc.

✔ **Access API Docs:**

- **Swagger UI:** `http://127.0.0.1:8000/docs`
- **ReDoc UI:** `http://127.0.0.1:8000/redoc`

✔ **Example – Custom API Metadata:**

python

CopyEdit

```
app = FastAPI(
    title="My API",
    description="This is a sample FastAPI application",
    version="1.0.0"
)
```

7️⃣ How do you implement authentication in FastAPI?

✅ **Answer:**
 Use OAuth2 with JWT tokens for secure authentication.

✔ Example – FastAPI Authentication with JWT:

python

CopyEdit

```
from fastapi import Depends, FastAPI, HTTPException

from fastapi.security import OAuth2PasswordBearer

from jose import JWTError, jwt


app = FastAPI()


oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")


SECRET_KEY = "your-secret-key"


def verify_token(token: str = Depends(oauth2_scheme)):

    try:

        payload = jwt.decode(token, SECRET_KEY, algorithms=["HS256"])

        return payload

    except JWTError:

        raise HTTPException(status_code=401, detail="Invalid token")


@app.get("/protected/")
```

```
def protected_route(user: dict = Depends(verify_token)):
    return {"message": "Authenticated", "user": user}
```

8️⃣ How do you handle dependency injection in FastAPI?

✅ **Answer:**
FastAPI provides built-in Dependency Injection to manage reusable components.

✔ **Example – Using Dependency Injection:**

python

CopyEdit

```
from fastapi import Depends, FastAPI


app = FastAPI()


def get_api_key():
    return "my-secret-api-key"


@app.get("/secure-data/")
def secure_data(api_key: str = Depends(get_api_key)):
    return {"api_key": api_key}
```

9️⃣ How do you handle background tasks in FastAPI?

✅ **Answer:**
Use FastAPI BackgroundTasks to execute non-blocking tasks.

✔ Example – Sending an Email in the Background:

python

CopyEdit

```python
from fastapi import BackgroundTasks, FastAPI


app = FastAPI()


def send_email(email: str):
    print(f"Sending email to {email}")


@app.post("/send_email/")
def send_email_request(email: str, background_tasks: BackgroundTasks):
    background_tasks.add_task(send_email, email)
    return {"message": "Email sent in the background"}
```

🔟 How do you secure a FastAPI application?

✅ **Answer:**

- **Use HTTPS with TLS.**
- **Implement JWT-based authentication.**
- **Enable CORS protection (`fastapi.middleware.cors`).**

- Use API rate limiting (`slowapi`).

✔ **Example – CORS Protection in FastAPI:**

python

CopyEdit

```python
from fastapi.middleware.cors import CORSMiddleware


app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],  # Change to specific domains in production
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

1️⃣1️⃣ How do you deploy a FastAPI application in production?

✅ **Answer:**
Use **Gunicorn + Uvicorn + Nginx** for deployment.

✔ **Example – Running FastAPI with Uvicorn & Gunicorn:**

bash

CopyEdit

```
gunicorn -k uvicorn.workers.UvicornWorker main:app --workers 4 --bind
0.0.0.0:8000
```

✔ Example – Dockerfile for FastAPI:

dockerfile

CopyEdit

```dockerfile
FROM python:3.10


WORKDIR /app

COPY . /app


RUN pip install fastapi uvicorn


CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

Final Thoughts

For a 10+ years experienced FastAPI developer, focus on:
✔ Building High-Performance Async APIs (WebSockets, Streaming).
✔ Optimizing Database Queries (Async SQLAlchemy, Redis).
✔ Security Best Practices (OAuth2, JWT, Rate Limiting).
✔ Production Deployment Strategies (Gunicorn, Docker, Kubernetes).
✔ API Performance Tuning (Caching, Compression, Profiling).