**Round 1 – Basic FastAPI & Backend Concepts**

**Q1:**  
What is FastAPI and why is it popular compared to Flask or Django?

(Think and answer. Then I'll guide you with the best response.)

**Q2:**  
How does FastAPI handle request validation and serialization?

* Can you explain the role of **Pydantic models** here?

**"FastAPI uses Python type hints along with Pydantic models to handle request validation and response serialization automatically.**

* **Request validation:** When you define an endpoint and annotate the request body with a Pydantic model, FastAPI automatically checks the incoming JSON against that model. It validates data types, required fields, and even performs custom constraints. If validation fails, FastAPI returns a 422 Unprocessable Entity response with details — without any extra code.
* **Serialization (response handling):** Pydantic also helps convert Python objects (like database models) into JSON-compatible responses automatically, ensuring data is serialized properly and safely.

**Role of Pydantic models:**

1. Define schemas for request and response bodies using Python classes.
2. Automatically perform type checking and validation.
3. Support advanced validation with validators and custom field constraints.
4. Convert ORM objects to JSON with orm\_mode.

**Example:**

from pydantic import BaseModel, Field

class User(BaseModel):

name: str

age: int = Field(gt=18) # must be > 18

@app.post("/users")

async def create\_user(user: User):

return user

**Q3:**  
Suppose you have an API /users that accepts JSON input { "name": "John", "age": 15 }.  
You need to **validate** that age is always greater than 18 and return a **400 Bad Request** otherwise.

* How would you do this in FastAPI?

**Round 2 – Real World Scenarios**

**Q4 (Scenario):**  
A user logs into your system from multiple devices.  
How will you design the **login and authentication** flow securely?

* What token strategy will you use (access & refresh tokens)?
* How will you **prevent token theft**?

**Q5 (Scenario):**  
You have an API that fetches data from the database, but sometimes the response takes too long.

* How do you **debug** and **fix** performance bottlenecks?
* Which tools or techniques would you use?

**Round 3 – Database & Scalability**

**Q6:**  
How do you handle **database connections** in FastAPI?

* What is connection pooling and why is it important?

from sqlalchemy import create\_engine

from sqlalchemy.orm import sessionmaker, declarative\_base

# Example PostgreSQL URL format:

# "postgresql+psycopg2://username:password@localhost/db\_name"

DATABASE\_URL = "postgresql+psycopg2://user:password@localhost/mydb"

# Create engine with connection pooling

engine = create\_engine(

DATABASE\_URL,

pool\_size=10, # Number of connections in the pool

max\_overflow=20, # Extra connections beyond pool\_size when needed

pool\_timeout=30, # Wait time before giving up on a connection

pool\_recycle=1800 # Recycle connections after 30 min to avoid stale ones

)

# Session factory

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

# Base class for models

Base = declarative\_base()

# Dependency: open/close DB session for each request

def get\_db():

db = SessionLocal()

try:

yield db

finally:

db.close()

**Q7 (Scenario):**  
Your backend is running fine locally, but when you deploy it to production with 10,000 users, the server crashes.

* How do you **scale** your FastAPI application to handle high traffic?
* Would you use multiple workers, load balancers, caching, etc.?

**Round 4 – Security & Cookies**

**Q8:**  
What is a cookie and how do you set and access cookies in FastAPI?

* Can someone **steal cookies**? How do you prevent it?

**Q9 (Scenario):**  
You are building an API like Netflix where one user account can only stream on 5 devices at a time.

* How do you **track devices** and enforce this rule?

**Round 5 – Advanced Real-world Problems**

**Q10 (Scenario):**  
Imagine Amazon backend: When a user adds an item to the cart, you need to **save the cart data**, **update stock inventory**, and **send an email confirmation**.

* How will you design this flow in FastAPI?
* Will you make all requests synchronous or use **background tasks**?

**1) FastAPI & Python Basics**

1. What is FastAPI and why is it faster compared to Django/Flask?
2. Explain what @app.get() or @app.post() does internally.
3. What is a Pydantic model? Why is it used in FastAPI?
4. Difference between BaseModel vs dataclass.
5. How does FastAPI handle validation and serialization automatically?
6. What is dependency injection in FastAPI? Give an example.
7. What is middleware? Can you write a middleware to log all incoming requests?
8. How do you handle exceptions globally in FastAPI?
9. What is Uvicorn and why do we use it with FastAPI?
10. What is async vs sync in Python? Why does FastAPI prefer async?

**2) Authentication & Security (Relevant to your project)**

1. What is JWT? How does it work internally?

“In FastAPI, we typically implement JWT authentication by generating an access token when the user logs in. I would use a library like python-jose for signing and verifying JWTs.  
When a user logs in with their credentials, I validate them against the database, then create a JWT with user info (like user\_id) and an expiry. This token is returned to the client, usually in the Authorization header as Bearer <token>.  
For protected routes, I use Depends() with a function that verifies the JWT signature and expiry. If it’s valid, the request proceeds; otherwise, I return 401 Unauthorized.”

1. Difference between **access token** and **refresh token**.
2. How do you store JWT securely? Which is safer: cookie vs localStorage?
3. How will you design authentication for multiple devices? (You already practiced this.)
4. How do you invalidate a JWT before its expiry? (Token blacklist / DB approach.)
5. How do you prevent **CSRF attacks** when using cookies?
6. How do you prevent **XSS attacks**?
7. What is CORS and how do you enable it in FastAPI?

"CORS stands for Cross-Origin Resource Sharing. It's a browser security feature that blocks requests from a different origin (domain, port, or protocol) unless the server explicitly allows it.  
For example, if my frontend is running on http://localhost:3000 and my backend API is on http://localhost:8000, the browser blocks requests by default. CORS lets the backend specify which origins are allowed."

1. What is the difference between **session-based** and **token-based** authentication?

**1) Session-Based Authentication**

* **How it works:**
  + User logs in → Server validates credentials → Server creates a session and stores it **in memory or database**.
  + Server sends a **session ID** (usually stored in a cookie) to the client.
  + For each request, the browser sends the session ID → Server looks up the session in its storage and validates it.
* **Pros:**
  + Simple to implement.
  + Works well for traditional web apps.
* **Cons:**
  + Server must store sessions (can become heavy for scaling).
  + Harder for distributed systems (needs sticky sessions or shared storage).
  + Vulnerable to CSRF if not protected properly.

**2) Token-Based Authentication (e.g., JWT)**

* **How it works:**
  + User logs in → Server generates a signed token (like JWT) that contains user info & expiry.
  + Token is sent to the client (stored in localStorage, cookie, etc.).
  + Client sends token in each request (e.g., Authorization: Bearer <token>).
  + Server **verifies the token signature** but **does not need to store session state**.
* **Pros:**
  + Stateless; server doesn’t store session (scales easily).
  + Works well for APIs and microservices.
  + Easier for mobile apps and cross-domain communication.
* **Cons:**
  + Token revocation is harder (can’t easily invalidate before expiry without additional logic).
  + Must be handled securely to prevent theft.

1. What is HttpOnly and Secure flag in cookies? Why are they important?

**3) Database & ORMs**

1. How do you connect FastAPI with PostgreSQL?
2. What is SQLAlchemy ORM? What are models and schemas?
3. Difference between ORM and raw SQL?
4. What are migrations? How do you manage them in FastAPI?
5. How would you design a User and Post table with relationships?
6. What is connection pooling? Why is it needed?

**4) API & Real-World Scenarios**

1. How do you design an API for uploading files (e.g., images)?
2. How do you handle rate limiting (prevent abuse of your API)?
3. How would you log in from multiple devices? (We discussed this.)
4. How do you implement pagination for GET APIs?
5. How do you handle concurrent requests safely?
6. How do you design an API to search and filter data?

**5) DevOps & Deployment (Basic Level)**

1. How do you deploy a FastAPI app?
2. What is Docker? Why do we use it?
3. What is .env file and why is it important?
4. How do you store secrets like DB passwords safely?
5. What is versioning of APIs and why is it important?

**6) General CS Concepts (Common Internship Questions)**

1. Difference between GET and POST.
2. Difference between PUT and PATCH.
3. What is REST API? What are its principles?
4. What is the difference between monolithic vs microservices?
5. What is caching and why is it important?
6. What is rate limiting and throttling?
7. What are HTTP status codes? Name 5 important ones (200, 201, 400, 401, 404, 500).
8. What is hashing vs encryption?
9. What is bcrypt? Why do we hash passwords?
10. Explain SOLID principles (at least S and O).

**1. Path Parameters**

* Used for **required values** that are part of the URL path.
* Declared directly in the path and function signature.

from fastapi import FastAPI

app = FastAPI()

@app.get("/items/{item\_id}")

def get\_item(item\_id: int):

return {"item\_id": item\_id}

* URL: /items/5 → item\_id is parsed as 5 (int).
* FastAPI validates and converts it to int automatically. If a non-int is provided, it returns a 422 error.

**2. Query Parameters**

* Used for **optional or filtering values** after ? in the URL.
* Declared as function arguments **with default values**.

@app.get("/items/")

def get\_items(skip: int = 0, limit: int = 10):

return {"skip": skip, "limit": limit}

* URL: /items/?skip=5&limit=20  
  → skip=5, limit=20.
* If not provided, defaults are used.

**Optional parameters:**

from typing import Optional

@app.get("/search/")

def search\_items(q: Optional[str] = None):

return {"query": q}

* /search/?q=laptop → {"query": "laptop"}
* /search/ → {"query": null}

**3. Request Body**

* Used when sending **JSON payloads** (POST, PUT, PATCH).
* Declared using **Pydantic models** for validation.

from pydantic import BaseModel

class Item(BaseModel):

name: str

price: float

is\_available: bool = True

@app.post("/items/")

def create\_item(item: Item):

return {"name": item.name, "price": item.price}

* Request body (JSON):

{

"name": "Laptop",

"price": 999.99,

"is\_available": true

}

* FastAPI validates automatically and converts to an Item object.

**Combining All Three**

You can use **path, query, and body together**:

@app.put("/users/{user\_id}")

def update\_user(

user\_id: int, # Path parameter

active: bool = True, # Query parameter

user: Item = None # Request body (Pydantic model)

):

return {

"user\_id": user\_id,

"active": active,

"user\_data": user

}