**User updates their profile. The UI still shows old data for a few minutes. What’s the reason?**

**You use Redis for caching and Kafka for event propagation.**

**Possible cause:** **Cache invalidation delay** or missed event.

**Fixes**:

* Ensure **cache eviction** happens on profile update.
* Use **write-through or write-behind** strategies.
* Add **TTL** and **versioning** to cached objects.

Cache invalidation is one of the hardest problems — mention it upfront

**Same User Sends Repeated Requests.  
User refreshes page 10 times in 1 second, causing duplicate orders. How to handle this?**

Use **idempotency keys**.

* Client generates a UUID per operation.
* Backend stores UUIDs to prevent re-processing.
* Use a **distributed store (Redis)** to keep idempotency tokens with TTL.

**Inconsistent Data Across Services.  
OrderService and PaymentService show inconsistent states. Payment shows success, but order is not marked complete. How do you ensure consistency?**

* Use **choreography-based Saga**:
* PaymentService emits PaymentSuccess → OrderService listens and updates state.
* Or **orchestration-based Saga** using a **Saga orchestrator** (e.g., Temporal.io, Camunda).
* Use **outbox + CDC** for guaranteed delivery of events.
* Have a **reconciliation job** to fix eventual mismatches.

**A service goes down. Upstream services retry automatically. The moment it comes up, it crashes again. Why?**

**System crashes again and again despite restarts.**

* This is a classic **retry storm** or **thundering herd problem**.
* Use **exponential backoff** and **jitter** in retry policies.
* Implement **circuit breakers** to short-circuit repeated failures.
* Add **rate limiters** (like Bucket4j or Redis Leaky Bucket).

Mention how chaos testing (like Gremlin) can reveal such patterns early.

**Large Payload Is Crashing Services .An upstream service sends a large JSON payload. Downstream services crash or become slow. How to safeguard?**

* Add a **request size limit** at:
* API Gateway / LB (e.g., Nginx client\_max\_body\_size)
* Controller level (Spring Boot has spring.servlet.multipart.max-request-size)
* Apply **schema validation** early (JSON schema, custom validators).
* Use **streaming or pagination** for large responses (e.g., product catalog).

**Your client app hits an API Gateway that calls 3 downstream services. Sometimes the user gets a timeout even when the downstream services are healthy. What could be wrong?**

**Assume no load balancer logs are available.**

**Possible causes:**

* One of the downstream calls is **slower intermittently**, leading to a **gateway-level timeout**.
* **Synchronous chaining** increases overall latency.
* Gateway’s **timeout is shorter** than the slowest service’s response time.

**Fixes**:

* Implement **circuit breakers** and **fallbacks** using Resilience4j/Hystrix.
* Apply **timeouts per downstream call** (not globally).
* Use **asynchronous parallel calls** via CompletableFuture.allOf() or WebClient.

Show how tracing tools (e.g., Zipkin, OpenTelemetry) can help isolate the problem