- 📘 JAVA (Core / JVM)
  - 1️⃣ JVM i Model Pamięci
  - 2️⃣ Współbieżność (Concurrency)
  - 3️⃣ Kolekcje i Struktury Danych
  - 4️⃣ Stream API
  - 5️⃣ Typy, OOP i Generics
  - 6️⃣ IO / NIO
  - 7️⃣ Wyjątki i API Design
- 🌱 SPRING ECOSYSTEM
  - 1️⃣ IoC i Dependency Injection
  - 2️⃣ @Transactional (Mechanizm Transakcji)
  - 3️⃣ Spring AOP
  - 🚀 4️⃣ Spring Boot
  - 🔐 5️⃣ Spring Security
- 🗄️ BAZY DANYCH I PERSISTENCJA
  - 1️⃣ Podstawy Teorii Transakcji
  - 2️⃣ JPA Internals
  - 3️⃣ SQL vs NoSQL i Architektoniczne Aspekty Danych
- 🏗️ ARCHITEKTURA APLIKACJI
  - 1️⃣ Fundamenty Architektoniczne
  - 2️⃣ Clean / Hexagonal / Onion
  - 3️⃣ Domain-Driven Design (DDD)
  - 4️⃣ Modularność i Skalowanie
- 🌐 MIKROSERWISY
  - 1️⃣ Fundamenty Mikroserwisów
  - 2️⃣ Odporność i Stabilność
  - 3️⃣ Spójność w systemie rozproszonym
  - 4️⃣ Obserwowalność
- 🔌 KOMUNIKACJA I INTEGRACJA
  - 1️⃣ REST i Semantyka HTTP
  - 2️⃣ gRPC
  - 3️⃣ Messaging: Kafka vs RabbitMQ
  - 4️⃣ Schema Evolution
- 🧰 DEVOPS / RUNTIME / PRODUKCJA
  - 1️⃣ Kontenery i Artefakty
  - 2️⃣ CI/CD
  - 3️⃣ Monitoring i Observability

📘 JAVA (Core / JVM)

1️⃣ JVM i Model Pamięci

🔷 1. Czym jest JVM i z jakich głównych obszarów pamięci się składa?

✅ **Odpowiedź**

JVM (Java Virtual Machine) to środowisko uruchomieniowe, które: - wykonuje bytecode (.class), - zarządza pamięcią, - zarządza wątkami, - zapewnia izolację i bezpieczeństwo.

Główne obszary pamięci:

🔶 **Heap** - Współdzielony między wątkami. - Przechowuje obiekty i ich pola. - Zarządzany przez Garbage Collector. - Podzielony na: - Young Generation (Eden + Survivor) - Old Generation

🔶 **Stack (dla każdego wątku osobny)** - Przechowuje: - zmienne lokalne, - referencje do obiektów, - ramki metod (stack frames). - Zarządzany automatycznie (LIFO). - Nie podlega GC.

🔶 3️⃣ **Metaspace** - Przechowuje metadane klas. - Zastąpił PermGen od Java 8. - Alokowany w pamięci natywnej (poza heapem).

🔶 4️⃣ **PC Register** - Wskaźnik aktualnie wykonywanej instrukcji dla wątku.

🔶 5️⃣ **Native Method Stack** - Dla metod natywnych (JNI).

🔷 2. Czym jest Java Memory Model (JMM)?

✅ **Odpowiedź**

JMM to specyfikacja opisująca:

- jak wątki widzą zmiany w pamięci,
- jakie operacje mogą być reorderyzowane przez kompilator/JIT/CPU,
- kiedy zmiany są widoczne między wątkami.

JMM definiuje: - reguły widoczności, - reguły synchronizacji, - relację happens-before.

Bez JMM kod wielowątkowy byłby nieprzewidywalny.

🔷 3. Co to jest reordering i dlaczego jest problemem?

✅ **Odpowiedź**

Reordering to zmiana kolejności instrukcji przez: - kompilator, - JIT, - CPU (out-of-order execution).

Jeśli inny wątek odczyta zmienne między operacjami, może zobaczyć niespójny stan.

Dlatego potrzebujemy: - synchronized, - volatile, - locków, - atomics.

🔷 4. Czym jest relacja happens-before?

✅ **Odpowiedź**

Happens-before to gwarancja JMM mówiąca:

> Jeśli operacja A happens-before operacji B, to wszystkie zmiany pamięci wykonane przez A są widoczne dla B.

Przykłady:

- Zakończenie metody synchronized happens-before wejście do synchronized na tym samym monitorze.
- Zapis do zmiennej volatile happens-before jej odczyt.
- Wywołanie start() happens-before kod w nowym wątku.

- Zakończenie wątku happens-before join().

🔹 5. Czym różni się visibility od atomicity?

✅ *Visibility (widoczność)*

Czy zmiana dokonana przez jeden wątek jest widoczna dla innego.

Zapewniają: - volatile - synchronized - locki

✅ *Atomicity (niepodzielność)*

Czy operacja wykona się w całości, bez możliwości przerwania.

Np operacja i++ nie jest atomowa, ponieważ składa się z: 1. odczytu 2. inkrementacji 3. zapisu

Atomowość zapewniają: - synchronized - Lock - klasy Atomic* - CAS (Compare-And-Swap)

🔹 6. Czym jest Escape Analysis?

✅ **Odpowiedź**

Escape Analysis to optymalizacja JIT.

Sprawdza, czy obiekt nie “ucieka” poza metodę (nie jest zwracany ani przekazywany dalej).

Jeśli nie ucieka: - może zostać zaalokowany na stosie zamiast na heapie, - może zostać wyeliminowany (scalar replacement), - może zostać usunięta synchronizacja (lock elision).

🔹 7. Jak działa ClassLoader?

✅ **Odpowiedź**

ClassLoader: - ładuje klasy do JVM, - zamienia bytecode na obiekt klasy.

Hierarchia: 1. Bootstrap ClassLoader 2. Platform ClassLoader 3. Application ClassLoader

Działa w modelu parent-first — najpierw pyta rodzica, jeśli rodzic nie znajdzie klasy, ładuje ją sam.

🔷 8. Czym jest JIT?

✅ **Odpowiedź**

JIT (Just-In-Time compiler): - kompiluje bytecode do natywnego kodu maszynowego podczas działania programu, - optymalizuje “gorące” fragmenty kodu (hot spots).

Optymalizacje obejmują: - inlining, - loop unrolling, - escape analysis, - dead code elimination.

🔷 9. Jak działa Garbage Collector w skrócie?

✅ **Odpowiedź**

GC usuwa obiekty, do których nie ma referencji.

Główne etapy: 1. Mark — oznaczenie obiektów osiągalnych 2. Sweep — usunięcie nieosiągalnych 3. Compact — defragmentacja

Nowoczesne GC: - G1 - ZGC - Shenandoah

2️⃣ Współbieżność (Concurrency)

🔷 1. Jak działa `synchronized` w JVM?

✅ **Odpowiedź**

`synchronized` używa monitora (monitor lock) przypisanego do obiektu.

Każdy obiekt w Javie posiada nagłówek (object header), który zawiera informacje o stanie blokady.

Co dzieje się przy wejściu do bloku synchronized?

1. Wątek próbuje przejąć monitor.
2. Jeśli monitor jest wolny — zostaje właścicielem.
3. Jeśli zajęty — wątek przechodzi w stan BLOCKED.
4. Po wyjściu z bloku monitor jest zwalniany.

Gwarancje:

- Mutual exclusion (tylko jeden wątek naraz).
- Visibility (flush do pamięci głównej przy wyjściu).
- Relacja happens-before między unlock → lock.

🔹 2. Czym jest `volatile` i co dokładnie gwarantuje?

✅ **Odpowiedź**

`volatile` zapewnia:

1️⃣ Visibility — zapis trafia bezpośrednio do pamięci głównej, a odczyt zawsze pobiera aktualną wartość.

2️⃣ Zakaz reordering wokół zmiennej volatile.

Nie zapewnia atomicity dla operacji złożonych (np. i++).

JVM generuje instrukcje z barierami pamięci (memory barriers), które wymuszają synchronizację cache CPU.

🔹 3. Dlaczego `i++` nie jest bezpieczne wielowątkowo?

✅ **Odpowiedź**

Operacja `i++` składa się z: 1. read 2. increment 3. write

Dwa wątki mogą odczytać tę samą wartość i nadpisać się nawzajem (lost update).

Rozwiązania: - synchronized - AtomicInteger - Lock

🔹 4. Czym jest CAS (Compare-And-Swap)?

✅ **Odpowiedź**

CAS to atomowa instrukcja procesora wykonująca operację:

Jeśli currentValue == expectedValue → ustaw newValue. W przeciwnym razie operacja się nie powiedzie.

Operacja jest niepodzielna na poziomie CPU.

W Javie używana przez: - AtomicInteger - ConcurrentHashMap - Struktury lock-free

---

🔹 5. Jak działają klasy Atomic*?

✅ **Odpowiedź**

Np. AtomicInteger: - przechowuje wartość jako volatile, - używa CAS w pętli retry.

Schemat działania: 1. Odczytaj aktualną wartość. 2. Oblicz nową wartość. 3. Spróbuj wykonać CAS. 4. Jeśli się nie uda — powtórz.

To jest podejście lock-free.

🔷 6. Czym różni się ReentrantLock od synchronized?

✅ **Odpowiedź**

ReentrantLock oferuje: - tryLock() - lockInterruptibly() - możliwość ustawienia fairness - condition variables

`synchronized`: - prostszy w użyciu - automatycznie zwalnia lock przy wyjątku

Oba zapewniają mutual exclusion i visibility.

🔷 7. Czym jest deadlock?

✅ **Odpowiedź**

Deadlock to sytuacja, gdy dwa lub więcej wątków czekają na zasoby trzymane przez siebie nawzajem.

Warunki deadlocka: 1. Mutual exclusion 2. Hold and wait 3. No preemption 4. Circular wait

---

🔷 8. Czym jest livelock?

✅ **Odpowiedź**

Wątki nie są zablokowane, ale stale reagują na siebie i nie wykonują postępu.

🔷 9. Czym jest starvation?

✅ **Odpowiedź**

Wątek nigdy nie otrzymuje zasobu, ponieważ inne wątki są preferowane.

🔷 10. Czym jest False Sharing?

✅ **Odpowiedź**

False sharing występuje, gdy dwie zmienne modyfikowane przez różne wątki znajdują się w tej samej cache line CPU.

Powoduje to częstą synchronizację cache między rdzeniami i spadek wydajności.

Rozwiązania: - padding - adnotacja @Contended (z odpowiednią flagą JVM)

🔹 11. Jaki jest cykl życia wątku w Javie?

✅ **Odpowiedź**

Stany wątku (Thread.State):

1. NEW — wątek utworzony, ale nieuruchomiony.
2. RUNNABLE — gotowy do wykonania lub wykonywany przez CPU.
3. BLOCKED — oczekuje na monitor (np. wejście do synchronized).
4. WAITING — czeka bez limitu czasu (np. wait(), join()).
5. TIMED_WAITING — czeka przez określony czas (sleep(), wait(timeout)).
6. TERMINATED — zakończył wykonanie.

Metoda start() powoduje przejście z NEW do RUNNABLE.

🔹 12. Dlaczego nie powinniśmy tworzyć wątków ręcznie (new Thread)?

✅ **Odpowiedź**

Tworzenie wątków jest kosztowne: - alokacja pamięci stosu, - rejestracja w systemie operacyjnym, - przełączanie kontekstu (context switching).

Tworzenie wielu krótkotrwałych wątków prowadzi do: - wysokiego narzutu systemowego, - spadku wydajności, - ryzyka OutOfMemoryError.

Dlatego stosuje się pule wątków.

🔹 13. Czym jest ExecutorService?

✅ **Odpowiedź**

ExecutorService to interfejs zarządzający wykonywaniem zadań asynchronicznych.

Oddziela: - tworzenie wątków, - zarządzanie nimi, - wykonywanie zadań.

Pozwala: - submit() — zwraca Future, - execute() — bez zwracania wyniku, - shutdown() — inicjuje zamknięcie, - shutdownNow() — próbuje przerwać wątki.

🔹 14. Jak działa ThreadPoolExecutor?

✅ **Odpowiedź**

ThreadPoolExecutor zarządza pulą wątków według parametrów:

1. corePoolSize — minimalna liczba wątków.
2. maximumPoolSize — maksymalna liczba wątków.
3. keepAliveTime — czas utrzymywania nadmiarowych wątków.
4. workQueue — kolejka zadań.

Schemat działania:

1. Jeśli liczba aktywnych wątków < corePoolSize → tworzony nowy wątek.
2. Jeśli osiągnięto corePoolSize → zadanie trafia do kolejki.
3. Jeśli kolejka pełna i liczba wątków < maximumPoolSize → tworzony nowy wątek.
4. Jeśli osiągnięto maximumPoolSize i kolejka pełna → uruchamiany RejectedExecutionHandler.

🔷 15. Jakie są typowe implementacje ExecutorService?

✅ **Odpowiedź**

- newFixedThreadPool(n) — stała liczba wątków.
- newCachedThreadPool() — dynamiczna liczba wątków, brak ograniczenia (może być niebezpieczne).
- newSingleThreadExecutor() — jeden wątek.
- newScheduledThreadPool(n) — zadania cykliczne/opóźnione.

W środowisku produkcyjnym zaleca się jawne konfigurowanie ThreadPoolExecutor.

🔷 16. Czym jest ForkJoinPool i do czego służy?

✅ **Odpowiedź**

ForkJoinPool to specjalizowana pula wątków zaprojektowana do zadań dzielonych rekurencyjnie (divide-and-conquer).

Kluczowe cechy: - Używa algorytmu work-stealing. - Każdy wątek posiada własną dwukierunkową kolejkę (deque). - Jeśli wątek kończy swoje zadania — “kradnie” zadania z końca kolejki innego wątku.

Dzięki temu minimalizuje bezczynność wątków i poprawia wykorzystanie CPU.

Używany przez: - Parallel Stream - CompletableFuture (domyślnie commonPool)

🔷 17. Czym jest work-stealing?

✅ **Odpowiedź**

Work-stealing to strategia równoważenia obciążenia:

- Wątek wykonuje zadania ze swojej kolejki (LIFO — dla lepszej lokalności cache).
- Jeśli nie ma zadań — kradnie z innego wątku (FIFO — z końca kolejki).

Zmniejsza contention na wspólnej kolejce i poprawia skalowalność.

🔷 18. Czym jest CompletableFuture?

✅ **Odpowiedź**

CompletableFuture to rozszerzenie Future pozwalające na: - asynchroniczne przetwarzanie, - łączenie operacji w pipeline, - obsługę wyjątków, - kompozycję wielu zadań.

Różnice względem Future: - Future jest blokujące (get()). - CompletableFuture umożliwia non-blocking chaining (thenApply, thenCompose).

Przykłady metod: - supplyAsync() - thenApply() - thenCompose() - thenCombine() - exceptionally()

🔹 19. Czym różni się thenApply od thenCompose?

✅ **Odpowiedź**

thenApply: - Przekształca wynik synchronizacyjnie. - Jeśli zwraca CompletableFuture — powstaje zagnieżdżenie (CompletableFuture<CompletableFuture>).

thenCompose: - Służy do spłaszczania zagnieżdżonych future. - Analogiczny do flatMap w Stream API.

🔹 20. Co oznacza blocking vs non-blocking?

✅ **Odpowiedź**

Blocking: - Wątek czeka na zakończenie operacji. - CPU może być bezczynne. - Przykład: Future.get().

Non-blocking: - Wątek nie czeka. - Rejestruje callback i wykonuje inne zadania. - Przykład: CompletableFuture.thenApply().

Non-blocking poprawia skalowalność przy operacjach IO.

🔹 21. Czym jest backpressure?

✅ **Odpowiedź**

Backpressure to mechanizm kontroli przepływu danych między producentem a konsumentem.

Problem: - Producent generuje dane szybciej niż konsument je przetwarza. - Może to prowadzić do przepełnienia pamięci.

Rozwiązania: - Ograniczone kolejki (bounded queue). - Odrzucanie zadań. - Reactive Streams (request(n)).

W kontekście ThreadPoolExecutor: - Ograniczenie workQueue zapobiega niekontrolowanemu wzrostowi pamięci.

3️⃣ Kolekcje i Struktury Danych

🔹 22. Jak działa HashMap krok po kroku?

✅ **Odpowiedź**

1. Obliczany jest hashCode() klucza.
2. Hash jest mieszany (bit spreading), aby lepiej rozłożyć bity.
3. Na podstawie hasha wyliczany jest indeks w tablicy: (n - 1) & hash.
4. Jeśli bucket jest pusty → tworzony nowy Node.
5. Jeśli kolizja:
   - porównanie equals(),
   - jeśli klucz istnieje → nadpisanie wartości,
   - jeśli nie → dodanie do listy lub drzewa.

Od Java 8: - Jeśli w bucket jest ≥ 8 elementów → lista zamieniana na Red-Black Tree. - Jeśli liczba spadnie < 6 → powrót do listy.

Resize następuje po przekroczeniu threshold = capacity × loadFactor.

🔹 23. Dlaczego equals() i hashCode() muszą być spójne?

✅ **Odpowiedź**

Kontrakt:

1. Jeśli a.equals(b) == true → a.hashCode() == b.hashCode().
2. Jeśli hashCode różne → obiekty na pewno różne.

Jeśli złamiemy kontrakt: - HashMap może nie znaleźć klucza, - dane mogą zostać “zgubione”.

Typowy błąd: nadpisanie equals bez hashCode.

🔹 24. Czym jest loadFactor i dlaczego ma znaczenie?

✅ **Odpowiedź**

loadFactor określa, przy jakim zapełnieniu nastąpi resize.

Domyślnie: 0.75.

- Niższy loadFactor → mniej kolizji, większe zużycie pamięci.
- Wyższy loadFactor → więcej kolizji, mniejsza pamięć.

🔹 25. Czym jest Red-Black Tree?

✅ **Odpowiedź**

Red-Black Tree to samobalansujące drzewo binarne.

Właściwości: 1. Każdy węzeł jest czerwony lub czarny. 2. Korzeń jest czarny. 3. Czerwony węzeł nie może mieć czerwonego dziecka. 4. Każda ścieżka do liścia ma tę samą liczbę czarnych węzłów.

Zapewnia operacje w czasie O(log n).

🔷 26. Czym różni się HashMap od ConcurrentHashMap?

✅ **Odpowiedź**

HashMap: - Nie jest thread-safe. - Może wejść w nieskończoną pętlę przy równoległym resize (Java < 8).

ConcurrentHashMap: - Thread-safe bez globalnego locka. - Używa CAS i synchronizacji na poziomie bucket. - Brak segmentów od Java 8 (wcześniej segment-based locking).

Operacje odczytu są w większości bezblokujące.

🔷 27. Co to jest segment-based locking?

✅ **Odpowiedź**

W starszych wersjach ConcurrentHashMap (Java 7): - Mapa była podzielona na segmenty. - Każdy segment miał własny lock. - Zmniejszało to contention względem jednego globalnego locka.

Od Java 8 zastąpione synchronizacją na bucket.

🔷 28. Czym jest CopyOnWriteArrayList?

✅ **Odpowiedź**

CopyOnWriteArrayList: - Przy każdej modyfikacji tworzy nową kopię tablicy. - Odczyty są bezblokujące.

Dobre dla: - wielu odczytów, - rzadkich zapisów.

Kosztowne przy częstych modyfikacjach.

🔷 29. Jakie są złożoności czasowe (Big-O) podstawowych kolekcji?

✅ **Odpowiedź**

HashMap: - get/put: O(1) średnio, O(log n) przy drzewie.

ArrayList: - get: O(1) - add (na końcu): O(1) amortyzowane - insert/remove w środku: O(n)

LinkedList: - get: O(n) - add/remove na początku: O(1)

TreeMap: - get/put: O(log n)

4️⃣ Stream API

🔹 30. Czym jest lazy evaluation w Stream API?

✅ **Odpowiedź**

Lazy evaluation oznacza, że operacje pośrednie (intermediate operations) nie są wykonywane od razu.

Są wykonywane dopiero, gdy pojawi się operacja terminalna (np. collect, forEach, reduce).

Pozwala to na: - optymalizację przetwarzania, - przetwarzanie elementów “na żądanie”, - łączenie operacji w jeden pipeline.

🔹 31. Czym różnią się operacje stateless i stateful?

✅ **Odpowiedź**

Stateless: - Nie zależą od innych elementów. - Przykład: map, filter. - Łatwe do równoległego przetwarzania.

Stateful: - Wymagają wiedzy o innych elementach. - Przykład: sorted, distinct. - Wymagają buforowania danych.

Operacje stateful są droższe i trudniejsze do zrównoleglenia.

🔹 32. Czym różni się map od flatMap?

✅ **Odpowiedź**

map: - Przekształca element na inny element. - 1 → 1

flatMap: - Przekształca element na strumień elementów. - 1 → N - Spłaszcza wynik.

flatMap jest odpowiednikiem flatMap z programowania funkcyjnego.

🔹 33. Dlaczego funkcja w reduce musi być asocjacyjna?

✅ **Odpowiedź**

Operacja asocjacyjna spełnia warunek:

(a op b) op c = a op (b op c)

W parallel stream: - Dane są dzielone na części. - Wyniki częściowe są łączone w dowolnej kolejności.

Jeśli operacja nie jest asocjacyjna — wynik może być niepoprawny.

🔷 34. Czym jest Spliterator?

✅ **Odpowiedź**

Spliterator to iterator zaprojektowany do przetwarzania równoległego.

Posiada metodę trySplit(), która: - dzieli dane na części, - umożliwia przetwarzanie równoległe.

Definiuje charakterystyki: - SIZED - ORDERED - DISTINCT - SORTED - IMMUTABLE

Stream używa Spliterator do podziału pracy w parallel stream.

🔷 35. Dlaczego parallel stream może być niebezpieczny?

✅ **Odpowiedź**

Problemy:

1. Side effects — współdzielony mutowalny stan może powodować race condition.
2. Operacje stateful są kosztowne.
3. Małe kolekcje — narzut przewyższa zysk.
4. Współdzielenie ForkJoinPool commonPool — może blokować inne zadania.

Parallel stream jest dobry dla: - dużych, CPU-bound operacji, - operacji asocjacyjnych, - braku efektów ubocznych.

🔷 36. Dlaczego efekty uboczne (side effects) łamią parallel stream?

✅ **Odpowiedź**

Side effect to modyfikacja współdzielonego stanu.

W parallel stream: - Elementy są przetwarzane równolegle. - Brak synchronizacji prowadzi do race condition.

Przykład niepoprawny:

list.parallelStream().forEach(e -> sharedList.add(e));

Poprawne podejście: - collect(Collectors.toList())

Stream powinien być funkcjonalny i bez efektów ubocznych.

🔷 37. Czym różni się forEach od forEachOrdered?

✅ **Odpowiedź**

forEach: - Nie gwarantuje kolejności w parallel stream. - Szybszy.

forEachOrdered: - Zachowuje kolejność źródła. - Może ograniczać równoległość.

W sequential stream działają identycznie.

5️⃣ Typy, OOP i Generics

🔷 38. Czym jest type erasure w Generics?

✅ **Odpowiedź**

Type erasure oznacza, że informacje o typach generycznych są usuwane w czasie kompilacji.

Przykład: List i List w runtime są widziane jako List.

Konsekwencje: - Brak informacji o typie w runtime. - Nie można tworzyć new T(). - Nie można tworzyć tablic generycznych (new T[]).

Kompilator dodaje casty w bytecode, aby zachować bezpieczeństwo typów.

🔷 39. Czym jest covariance i contravariance?

✅ **Odpowiedź**

Covariance — pozwala używać typu bardziej szczegółowego.

List<? extends Number> - Można czytać jako Number. - Nie można bezpiecznie dodawać elementów.

Contravariance — pozwala używać typu bardziej ogólnego.

List<? super Integer> - Można dodawać Integer. - Odczyt jako Object.

Zasada PECS: - Producer → extends - Consumer → super

🔷 40. Dlaczego immutable objects są bezpieczne wielowątkowo?

✅ **Odpowiedź**

Obiekt immutable: - Nie zmienia stanu po konstrukcji. - Wszystkie pola final. - Brak setterów.

Gwarancje: - Brak race condition. - Bezpieczne publikowanie (safe publication), jeśli pola są final.

Immutable upraszcza concurrency.

🔷 41. Czym różni się equals od ==?

✅ **Odpowiedź**

== porównuje referencje (czy to ten sam obiekt).

equals porównuje logiczną równość (zawartość).

Dla klas własnych należy nadpisać equals i hashCode.

🔷 42. Czym jest record w Javie?

✅ **Odpowiedź**

record to skrócona forma klasy immutable.

Automatycznie generuje: - konstruktor, - equals, - hashCode, - toString, - gettery.

Record jest final i przeznaczony do przechowywania danych.

🔷 43. Czym są sealed classes?

✅ **Odpowiedź**

Sealed class ogranicza, które klasy mogą ją rozszerzać.

Przykład: sealed class Shape permits Circle, Square {}

Pozwala: - kontrolować hierarchię dziedziczenia, - ułatwiać exhaustive switch.

🔷 44. Kiedy Optional jest dobrym pomysłem, a kiedy złym?

✅ **Odpowiedź**

Dobry: - jako typ zwracany z metody. - aby uniknąć null.

Zły: - jako pole w encji. - jako parametr metody. - w serializacji.

Optional nie jest zamiennikiem każdego null.

🔷 45. Czym są value-based classes?

✅ **Odpowiedź**

Value-based class: - Reprezentuje wartość, nie tożsamość. - Nie należy polegać na referencyjnej równości (==).

Przykład: Integer, Optional.

W przyszłości Project Valhalla wprowadzi value types bez narzutu obiektowego.

6️⃣ IO / NIO

🔹 46. Czym różni się IO od NIO?

✅ **Odpowiedź**

IO (java.io): - Model strumieniowy (InputStream/OutputStream). - Operacje blokujące. - Jeden wątek na jedno połączenie.

NIO (java.nio): - Kanały (Channel) i bufory (Buffer). - Możliwość trybu non-blocking. - Selector pozwala obsługiwać wiele połączeń jednym wątkiem.

NIO jest bardziej skalowalne przy dużej liczbie połączeń.

🔹 47. Czym jest Channel?

✅ **Odpowiedź**

Channel to dwukierunkowe połączenie do źródła danych.

Przykłady: - FileChannel - SocketChannel - ServerSocketChannel

Współpracuje z Buffer.

🔹 48. Czym jest Buffer?

✅ **Odpowiedź**

Buffer to kontener na dane z polami: - capacity — maksymalny rozmiar, - position — aktualna pozycja, - limit — granica odczytu/zapisu.

Tryb zapisu → flip() → tryb odczytu.

🔹 49. Czym jest Selector?

✅ **Odpowiedź**

Selector pozwala jednemu wątkowi monitorować wiele kanałów.

Działa poprzez: - rejestrowanie kanałów, - sprawdzanie gotowości do operacji (read, write, accept).

Umożliwia model event-driven.

🔹 50. Czym jest memory-mapped file?

✅ **Odpowiedź**

Memory-mapped file (MappedByteBuffer): - Mapuje plik bezpośrednio do pamięci. - OS zarządza synchronizacją z dyskiem.

Zalety: - Szybki dostęp do dużych plików. - Brak kopiowania danych między buforami.

Wady: - Trudniejsze zarządzanie pamięcią.

7️⃣ Wyjątki i API Design

🔹 51. Czym różnią się checked i unchecked exceptions?

✅ **Odpowiedź**

Checked: - Muszą być zadeklarowane lub obsłużone. - Reprezentują sytuacje, które można przewidzieć.

Unchecked (RuntimeException): - Nie wymagają deklaracji. - Reprezentują błędy programistyczne.

W nowoczesnym kodzie preferuje się unchecked.

🔹 52. Czym jest exception wrapping?

✅ **Odpowiedź**

Exception wrapping polega na opakowaniu niższego wyjątku w wyższy:

throw new CustomException(“msg”, cause);

Pozwala: - zachować stack trace, - oddzielić warstwy aplikacji.

🔹 53. Czym są suppressed exceptions?

✅ **Odpowiedź**

Powstają przy try-with-resources.

Jeśli: - wyjątek wystąpi w bloku try, - oraz podczas zamykania zasobu,

Wyjątek z close() jest suppressed i dostępny przez getSuppressed().

🔷 54. Czym jest defensive copying?

✅ **Odpowiedź**

Defensive copying polega na tworzeniu kopii mutowalnych obiektów przekazywanych do klasy.

Chroni przed: - niekontrolowaną modyfikacją stanu.

Stosowane w klasach immutable.

🔷 55. Jak projektować dobre API?

✅ **Odpowiedź**

Zasady: - Minimalna powierzchnia publiczna. - Brak null (Optional lub wyjątek). - Immutable gdy możliwe. - Spójne nazewnictwo. - Jasne kontrakty (JavaDoc). - Nie ujawniać implementacji.

Dobre API jest łatwe do użycia i trudne do użycia błędnie.

💡 Questions

🌱 SPRING ECOSYSTEM

1️⃣ IoC i Dependency Injection

🔷 1. Czym jest IoC (Inversion of Control)?

✅ **Odpowiedź**

Inversion of Control oznacza odwrócenie kontroli nad tworzeniem i zarządzaniem zależnościami.

Zamiast: - Klasa sama tworzy swoje zależności (new Service()),

To: - Kontener (Spring) tworzy obiekty i wstrzykuje je do klas.

Korzyści: - Luźne powiązania (loose coupling), - Łatwiejsze testowanie, - Możliwość podmiany implementacji.

🔷 2. Czym jest Dependency Injection?

✅ **Odpowiedź**

Dependency Injection to mechanizm dostarczania zależności do klasy z zewnątrz.

Rodzaje: - Constructor injection (zalecane), - Setter injection, - Field injection (niezalecane).

Constructor injection: - Zapewnia immutability zależności, - Ułatwia testowanie, - Wymusza kompletność obiektu.

🔹 3. Jak działa kontener Springa krok po kroku?

✅ **Odpowiedź**

1. Odczyt konfiguracji (@Configuration, @ComponentScan).
2. Rejestracja definicji beanów (BeanDefinition).
3. Tworzenie instancji beanów.
4. Wstrzykiwanie zależności.
5. Wywołanie callbacków (np. @PostConstruct).
6. Bean gotowy do użycia.

Spring używa refleksji do tworzenia i łączenia obiektów.

🔹 4. Jakie są scope beanów?

✅ **Odpowiedź**

- singleton (domyślny) — jedna instancja na kontekst.
- prototype — nowa instancja przy każdym pobraniu.
- request — jedna instancja na request HTTP.
- session — jedna instancja na sesję.
- application — jedna instancja na aplikację webową.

Singleton w Springu ≠ Singleton w sensie wzorca projektowego (jest per ApplicationContext).

🔹 5. Czym jest lifecycle beana?

✅ **Odpowiedź**

Etapy: 1. Instancja. 2. Wstrzyknięcie zależności. 3. BeanPostProcessor (before). 4. @PostConstruct. 5. Bean gotowy. 6. Przy zamknięciu kontekstu — @PreDestroy.

BeanPostProcessor umożliwia modyfikację beanów (np. tworzenie proxy).

🔹 6. Czym jest @Primary i @Qualifier?

✅ **Odpowiedź**

Jeśli istnieje wiele implementacji interfejsu:

@Primary — wskazuje domyślny bean. @Qualifier — pozwala wybrać konkretną implementację.

Zapobiega NoUniqueBeanDefinitionException.

2️⃣ @Transactional (Mechanizm Transakcji)

🔷 7. Jak działa @Transactional pod spodem?

✅ **Odpowiedź**

@Transactional działa poprzez AOP.

Spring: 1. Tworzy proxy dla beana (JDK dynamic proxy lub CGLIB). 2. Proxy przechwytuje wywołanie metody. 3. Otwiera transakcję (PlatformTransactionManager). 4. Wywołuje metodę. 5. Commit lub rollback w zależności od wyniku.

Jeśli metoda jest wywołana bezpośrednio (self-invocation) — proxy nie jest używane i transakcja nie zadziała.

🔷 8. Kiedy @Transactional NIE działa?

✅ **Odpowiedź**

- Metoda private.
- Metoda final (przy CGLIB ograniczenia proxy).
- Wywołanie self-invocation.
- Bean niezarządzany przez Spring.
- Brak skonfigurowanego TransactionManager.

🔷 9. Jak działają propagacje transakcji?

✅ **Odpowiedź**

Propagation określa zachowanie przy wywołaniu metody wewnątrz istniejącej transakcji.

REQUIRED (domyślne): - Dołącza do istniejącej lub tworzy nową.

REQUIRES_NEW: - Zawsze tworzy nową transakcję. - Zawiesza obecną.

MANDATORY: - Wymaga istniejącej transakcji. - Jeśli brak → wyjątek.

SUPPORTS: - Działa w transakcji jeśli istnieje. - Jeśli nie → bez transakcji.

NOT_SUPPORTED: - Zawiesza istniejącą transakcję.

NEVER: - Jeśli istnieje transakcja → wyjątek.

NESTED: - Tworzy savepoint (jeśli wspierane przez DB).

🔷 10. Jakie są poziomy izolacji?

✅ **Odpowiedź**

READ_UNCOMMITTED: - Możliwe dirty reads.

READ_COMMITTED: - Brak dirty reads. - Możliwe non-repeatable reads.

REPEATABLE_READ: - Brak dirty i non-repeatable reads. - Możliwe phantom reads.

SERIALIZABLE: - Najwyższa izolacja. - Zachowuje się jak wykonanie sekwencyjne. - Najmniej wydajny.

Dirty read — odczyt niezatwierdzonych danych. Non-repeatable read — ten sam rekord zwraca różne wartości. Phantom read — nowe rekordy pojawiają się w wyniku tego samego zapytania.

🔷 11. Kiedy rollback NIE nastąpi?

✅ **Odpowiedź**

Domyślnie rollback następuje tylko dla RuntimeException i Error.

Nie nastąpi dla: - Checked exception (chyba że rollbackFor ustawione). - Jeśli wyjątek został złapany i nie rzucony dalej.

Można wymusić rollback: @Transactional(rollbackFor = Exception.class)

🔷 12. Czym jest TransactionManager?

✅ **Odpowiedź**

PlatformTransactionManager zarządza cyklem życia transakcji.

Implementacje: - DataSourceTransactionManager (JDBC). - JpaTransactionManager (JPA). - JtaTransactionManager (rozproszone transakcje).

Odpowiada za: - begin, - commit, - rollback, - zarządzanie zasobami (Connection).

3️⃣ Spring AOP

🔷 13. Czym jest AOP (Aspect-Oriented Programming)?

✅ **Odpowiedź**

AOP pozwala wydzielić logikę przekrojową (cross-cutting concerns), taką jak: - transakcje, - logowanie, - bezpieczeństwo, - cache.

Zamiast umieszczać ją w każdej metodzie — definiuje się aspekt.

🔷 14. Jak działa AOP w Springu?

✅ **Odpowiedź**

Spring AOP działa w oparciu o proxy.

1. Tworzony jest proxy beana.
2. Proxy przechwytuje wywołanie metody.
3. Wykonywana jest logika aspektu (np. @Around).
4. Wywoływana jest metoda docelowa.

Spring AOP działa tylko na metodach publicznych beanów zarządzanych przez Spring.

🔷 15. Czym różni się JDK Dynamic Proxy od CGLIB?

✅ **Odpowiedź**

JDK Dynamic Proxy: - Tworzy proxy na podstawie interfejsu. - Wymaga, aby bean implementował interfejs.

CGLIB: - Tworzy klasę dziedziczącą po klasie docelowej. - Może działać bez interfejsu. - Nie może proxy'ować metod final.

Spring domyślnie używa: - JDK proxy, jeśli istnieje interfejs, - CGLIB w przeciwnym razie.

🔷 16. Czym jest weaving?

✅ **Odpowiedź**

Weaving to proces wstrzykiwania aspektów do kodu.

Rodzaje: - Compile-time weaving (AspectJ). - Load-time weaving. - Runtime weaving (Spring proxy).

Spring używa runtime weaving przez proxy.

🔷 17. Jakie są typy advice?

✅ **Odpowiedź**

- @Before — przed metodą.
- @After — po metodzie (zawsze).
- @AfterReturning — po sukcesie.
- @AfterThrowing — po wyjątku.
- @Around — otacza metodę (najbardziej elastyczne).

@Around pozwala kontrolować wywołanie proceed().

🔷 18. Czym jest pointcut?

✅ **Odpowiedź**

Pointcut definiuje, które metody mają być objęte aspektem.

Przykład: execution(* com.example.service..(..))

Może bazować na: - pakiecie, - nazwie metody, - adnotacji.

Pointcut + Advice = Aspect.

🚀4️⃣ Spring Boot

🔷 19. Czym jest auto-configuration w Spring Boot?

✅ **Odpowiedź**

Auto-configuration to mechanizm automatycznego konfigurowania beanów na podstawie: - obecności klas w classpath, - właściwości konfiguracyjnych, - istniejących beanów.

Spring Boot: 1. Skanuje plik META-INF/spring/org.springframework.boot.autoconfigure.AutoConfiguration.imports. 2. Rejestruje klasy konfiguracyjne. 3. Warunkowo tworzy beany.

Pozwala to uruchomić aplikację bez ręcznej konfiguracji infrastruktury.

🔷 20. Czym są adnotacje @Conditional?

✅ **Odpowiedź**

@Conditional pozwala tworzyć bean tylko jeśli spełniony jest warunek.

Najczęstsze warianty: - @ConditionalOnClass - @ConditionalOnMissingBean - @ConditionalOnProperty - @ConditionalOnWebApplication

Mechanizm ten jest fundamentem auto-config.

🔷 21. Czym jest starter w Spring Boot?

✅ **Odpowiedź**

Starter to zestaw zależności (dependency descriptor), który: - agreguje biblioteki, - dodaje odpowiednie auto-config.

Przykład: - spring-boot-starter-web - spring-boot-starter-data-jpa

Ułatwia konfigurację poprzez konwencję zamiast konfiguracji.

🔹 22. Czym różni się ApplicationContext od WebApplicationContext?

✅ **Odpowiedź**

ApplicationContext: - Ogólny kontener beanów.

WebApplicationContext: - Rozszerza ApplicationContext. - Dodaje wsparcie dla środowiska web (ServletContext).

W aplikacjach web tworzony jest kontekst webowy.

🔹 23. Jak działa binding właściwości (@ConfigurationProperties)?

✅ **Odpowiedź**

@ConfigurationProperties pozwala mapować właściwości z application.yml na obiekt.

Mechanizm: - Spring odczytuje konfigurację. - Tworzy bean. - Wstrzykuje wartości przez setter lub konstruktor.

Zalety: - Typowanie. - Walidacja. - Lepsza organizacja konfiguracji.

🔹 24. Czym są profile w Spring?

✅ **Odpowiedź**

Profile pozwalają aktywować różne konfiguracje zależnie od środowiska.

Aktywacja: - spring.profiles.active - @Profile(“dev”)

Umożliwiają separację konfiguracji dev/test/prod.

🔹 25. Czym jest Spring Boot Actuator?

✅ **Odpowiedź**

Actuator dostarcza endpointy monitorujące aplikację.

Przykłady: - /actuator/health - /actuator/metrics - /actuator/env - /actuator/info

Pozwala monitorować: - stan aplikacji, - metryki, - konfigurację.

Często używany z Prometheus.

🔐 5️⃣ Spring Security

🔹 26. Jak działa Spring Security wewnętrznie?

✅ **Odpowiedź**

Spring Security działa jako łańcuch filtrów (Security Filter Chain).

1. Żądanie HTTP trafia do serwera.
2. Przechodzi przez kolejne filtry (np. authentication, authorization).
3. Jeśli uwierzytelnienie się powiedzie — tworzony jest Authentication.
4. Obiekt Authentication trafia do SecurityContext.

SecurityContext przechowywany jest w ThreadLocal (SecurityContextHolder).

🔹 27. Czym różni się Authentication od Authorization?

✅ **Odpowiedź**

Authentication (uwierzytelnienie): - Kim jesteś? - Weryfikacja tożsamości (login/hasło, token).

Authorization (autoryzacja): - Co możesz zrobić? - Sprawdzenie uprawnień (role, authorities).

Authentication poprzedza Authorization.

🔹 28. Czym jest SecurityContext?

✅ **Odpowiedź**

SecurityContext przechowuje informacje o aktualnie uwierzytelnionym użytkowniku.

Zawiera: - principal (użytkownik), - credentials, - authorities.

Domyślnie przechowywany w ThreadLocal.

🔹 29. Jak działa JWT w Spring Security?

✅ **Odpowiedź**

JWT (JSON Web Token): - Token zawiera zakodowane dane użytkownika. - Składa się z header.payload.signature.

Proces: 1. Użytkownik się loguje. 2. Serwer generuje podpisany token. 3. Klient wysyła token w nagłówku Authorization. 4. Serwer weryfikuje podpis i tworzy Authentication.

JWT jest stateless — serwer nie przechowuje sesji.

🔹 30. Czym jest OAuth2?

✅ **Odpowiedź**

OAuth2 to protokół autoryzacji.

Role: - Resource Owner - Client - Authorization Server - Resource Server

Flow (np. Authorization Code): 1. Przekierowanie do Authorization Server. 2. Użytkownik się loguje. 3. Client otrzymuje authorization code. 4. Wymienia go na access token.

Spring Security może działać jako Resource Server lub OAuth2 Client.

🔹 31. Czym jest CSRF i jak Spring go chroni?

✅ **Odpowiedź**

CSRF (Cross-Site Request Forgery): - Atak polegający na wysłaniu żądania w imieniu użytkownika bez jego wiedzy.

Spring generuje CSRF token: - Musi być dołączony do żądań modyfikujących (POST/PUT/DELETE). - Weryfikowany po stronie serwera.

W aplikacjach stateless (JWT) często wyłączany.

🔹 32. Czym jest CORS?

✅ **Odpowiedź**

CORS (Cross-Origin Resource Sharing): - Mechanizm przeglądarki kontrolujący dostęp między domenami.

Serwer musi zwrócić odpowiednie nagłówki: - Access-Control-Allow-Origin - Access-Control-Allow-Methods

Spring pozwala konfigurować CORS globalnie lub per endpoint.

🔹 33. Jak bezpiecznie przechowywać hasła?

✅ **Odpowiedź**

Hasła nie powinny być szyfrowane, lecz haszowane.

Algorytmy: - bcrypt - argon2 - scrypt

Cechy: - Salt (losowa wartość). - Koszt obliczeniowy (work factor).

Spring Security dostarcza PasswordEncoder.

💡 Questions

🗄️ BAZY DANYCH I PERSISTENCJA

1️⃣ Podstawy Teorii Transakcji

🔷 1. Czym jest ACID?

✅ **Odpowiedź**

ACID to cztery właściwości transakcji:

Atomicity (Atomowość) - Transakcja wykonuje się w całości albo wcale. - W przypadku błędu następuje rollback.

Consistency (Spójność) - Po zakończeniu transakcji dane spełniają wszystkie ograniczenia (constraints, reguły biznesowe).

Isolation (Izolacja) - Równoległe transakcje nie powinny wpływać na siebie w sposób łamiący spójność.

Durability (Trwałość) - Po commit dane są zapisane trwale (np. w WAL — Write-Ahead Log).

🔹 2. Czym są poziomy izolacji i jakie anomalie eliminują?

✅ **Odpowiedź**

READ_UNCOMMITTED - Pozwala na dirty reads (odczyt niezatwierdzonych danych).

READ_COMMITTED - Brak dirty reads. - Możliwe non-repeatable reads.

REPEATABLE_READ - Brak dirty reads. - Brak non-repeatable reads. - Możliwe phantom reads.

SERIALIZABLE - Najwyższa izolacja. - Eliminuje wszystkie powyższe anomalie. - Najmniej wydajny.

Anomalie: - Dirty read — odczyt danych, które mogą zostać wycofane. - Non-repeatable read — ten sam rekord zwraca różne wartości. - Phantom read — nowe rekordy pojawiają się w zakresie zapytania.

🔹 3. Czym jest optimistic locking?

✅ **Odpowiedź**

Optimistic locking zakłada, że konflikt jest rzadki.

Mechanizm: - Wiersz posiada pole version. - Przy update sprawdzana jest wersja. - Jeśli zmieniła się — wyjątek (OptimisticLockException).

Brak blokad na poziomie bazy.

🔹 4. Czym jest pessimistic locking?

✅ **Odpowiedź**

Pessimistic locking zakłada, że konflikt jest prawdopodobny.

Mechanizm: - SELECT … FOR UPDATE - Wiersz jest blokowany. - Inne transakcje czekają.

Większe bezpieczeństwo, mniejsza skalowalność.

🔹 5. Czym jest indeks i jak działa B-Tree?

✅ **Odpowiedź**

Indeks to struktura danych przyspieszająca wyszukiwanie.

B-Tree: - Samobalansujące drzewo. - Każdy węzeł zawiera wiele kluczy. - Wyszukiwanie w czasie O(log n).

Indeks przyspiesza SELECT, ale spowalnia INSERT/UPDATE.

🔹 6. Czym jest execution plan?

✅ **Odpowiedź**

Execution plan to plan wykonania zapytania generowany przez optimizer.

Zawiera: - sposób użycia indeksów, - join strategy (nested loop, hash join), - koszt operacji.

Analizowany przez EXPLAIN.

🔹 7. Czym jest connection pool?

✅ **Odpowiedź**

Connection pool zarządza pulą połączeń do bazy.

Zamiast tworzyć połączenie przy każdym zapytaniu: - połączenia są utrzymywane i ponownie wykorzystywane.

Najpopularniejszy: HikariCP.

Parametry: - maximumPoolSize - connectionTimeout - idleTimeout

Zmniejsza koszt tworzenia połączeń.

2️⃣ JPA Internals

🔹 8. Czym jest Persistence Context?

✅ **Odpowiedź**

Persistence Context to kontekst zarządzania encjami przez EntityManager.

Zawiera: - zarządzane encje (managed entities), - mechanizm śledzenia zmian, - cache pierwszego poziomu.

Każda encja w kontekście ma tylko jedną reprezentację (identity guarantee).

🔹 9. Jakie są stany encji w JPA?

✅ **Odpowiedź**

1. Transient — nowa encja, niezarządzana.
2. Managed — zarządzana przez Persistence Context.
3. Detached — była zarządzana, ale kontekst został zamknięty.
4. Removed — oznaczona do usunięcia.

Tylko encje Managed podlegają dirty checking.

🔹 10. Czym jest 1st level cache?

✅ **Odpowiedź**

1st level cache to cache w ramach Persistence Context.

Cechy: - Domyślnie włączony. - Unikalna instancja encji na klucz. - Zapobiega wielokrotnym zapytaniom do DB w jednej transakcji.

Jest powiązany z EntityManager.

🔹 11. Czym jest dirty checking?

✅ **Odpowiedź**

Dirty checking to mechanizm automatycznego wykrywania zmian w encjach.

Proces: 1. Przy załadowaniu encji tworzony jest snapshot. 2. Przy flush porównywany jest aktualny stan z snapshotem. 3. Jeśli są różnice — generowany jest UPDATE.

Działa tylko dla encji Managed.

🔹 12. Czym jest flush?

✅ **Odpowiedź**

Flush synchronizuje stan Persistence Context z bazą danych.

Nie oznacza commit.

Może nastąpić: - automatycznie przed zapytaniem JPQL, - przy commit, - ręcznie (entityManager.flush()).

🔹 13. Czym jest N+1 problem?

✅ **Odpowiedź**

N+1 występuje gdy: - Pobieramy listę encji (1 zapytanie), - Dla każdej wykonywane jest osobne zapytanie (N zapytań).

Powód: - Lazy loading relacji.

Rozwiązania: - JOIN FETCH - EntityGraph - Batch fetching

🔹 14. Czym różni się FetchType.LAZY od EAGER?

✅ **Odpowiedź**

LAZY: - Relacja ładowana przy pierwszym użyciu. - Domyślne dla @OneToMany.

EAGER: - Relacja ładowana od razu. - Może powodować nadmiarowe zapytania.

EAGER często prowadzi do problemów wydajnościowych.

🔹 15. Czym jest LazyInitializationException?

✅ **Odpowiedź**

Występuje gdy: - Próbujemy odczytać relację LAZY, - Po zamknięciu Persistence Context.

Typowy przypadek: - Dostęp do relacji poza transakcją.

Rozwiązania: - DTO projection, - JOIN FETCH, - OpenSessionInView (niezalecane w architekturze czystej).

3️⃣ SQL vs NoSQL i Architektoniczne Aspekty Danych

🔹 16. Czym jest normalizacja i denormalizacja?

✅ **Odpowiedź**

Normalizacja: - Proces organizowania danych w celu eliminacji redundancji. - Dane podzielone na wiele tabel. - Spójność i mniejsze ryzyko anomalii aktualizacji.

Denormalizacja: - Celowe wprowadzanie redundancji. - Mniej joinów. - Lepsza wydajność odczytu kosztem spójności i większej złożoności zapisu.

Systemy OLTP → zwykle normalizacja. Systemy analityczne / read-heavy → często denormalizacja.

🔹 17. Czym jest twierdzenie CAP?

✅ **Odpowiedź**

CAP mówi, że w systemie rozproszonym można mieć maksymalnie dwie z trzech cech:

Consistency — wszystkie węzły widzą te same dane w tym samym czasie. Availability — każdy request otrzymuje odpowiedź. Partition Tolerance — system działa mimo podziałów sieci.

W praktyce w systemach rozproszonych zawsze zakładamy Partition Tolerance.

Wybór jest więc między: - CP (Consistency + Partition Tolerance) - AP (Availability + Partition Tolerance)

🔹 18. Czym różni się SQL od NoSQL?

✅ Odpowiedź

SQL: - Relacyjny model danych. - Silna spójność. - Transakcje ACID. - Schemat zdefiniowany z góry.

NoSQL: - Dokumentowe, klucz-wartość, kolumnowe, grafowe. - Często eventual consistency. - Elastyczny schemat. - Łatwiejsze skalowanie poziome.

SQL lepszy dla: - złożonych relacji, - transakcji finansowych.

NoSQL lepszy dla: - dużej skali, - dynamicznych struktur danych, - systemów event-driven.

🔹 19. Czym jest eventual consistency?

✅ Odpowiedź

Eventual consistency oznacza, że: - System może chwilowo zwracać niespójne dane, - Ale ostatecznie wszystkie repliki się zsynchronizują.

Typowe w systemach AP.

Wymaga projektowania systemu z myślą o: - idempotency, - retry, - mechanizmach kompensacji.

🔹 20. Czym jest replikacja?

✅ Odpowiedź

Replikacja to kopiowanie danych na wiele węzłów.

Rodzaje: - Master-Slave (Primary-Replica). - Multi-Master.

Zalety: - Większa dostępność. - Skalowanie odczytu.

Wady: - Opóźnienia synchronizacji.

🔷 21. Czym jest sharding?

✅ **Odpowiedź**

Sharding to podział danych między wiele węzłów według klucza (shard key).

Cel: - Skalowanie zapisu. - Skalowanie pojemności.

Wyzwania: - Wybór shard key. - Migracja shardów. - Zapytania cross-shard.

🔷 22. Jak skalować bazę danych?

✅ **Odpowiedź**

Vertical scaling: - Więcej CPU/RAM. - Ograniczony przez sprzęt.

Horizontal scaling: - Replikacja (read scaling). - Sharding (write scaling).

W systemach o dużej skali często łączy się oba podejścia.

💡 Questions

🏗️ ARCHITEKTURA APLIKACJI

1️⃣ Fundamenty Architektoniczne

🔷 1. Czym jest Separation of Concerns?

✅ **Odpowiedź**

Separation of Concerns (SoC) oznacza rozdzielenie systemu na części odpowiedzialne za różne aspekty.

Przykłady warstw: - Warstwa prezentacji - Warstwa aplikacyjna - Warstwa domenowa - Warstwa infrastruktury

Celem jest: - mniejsza złożoność, - większa testowalność, - łatwiejsza wymiana technologii.

🔷 2. Czym jest Dependency Inversion Principle?

✅ **Odpowiedź**

Dependency Inversion Principle (DIP):

- Moduły wysokiego poziomu nie powinny zależeć od modułów niskiego poziomu.
- Oba powinny zależeć od abstrakcji.

Przykład: - Serwis biznesowy zależy od interfejsu Repozytorium. - Implementacja repozytorium zależy od bazy danych.

Zmniejsza sprzężenie i ułatwia testowanie.

2️⃣ Clean / Hexagonal / Onion

🔷 3. Czym jest Clean Architecture?

✅ **Odpowiedź**

Clean Architecture dzieli system na warstwy koncentryczne:

- Entities (logika biznesowa)
- Use Cases
- Interface Adapters
- Frameworks & Drivers

Zasada: - Zależności skierowane do środka. - Rdzeń domeny nie zna frameworków.

🔷 4. Czym jest Hexagonal Architecture?

✅ **Odpowiedź**

Hexagonal (Ports & Adapters):

- Domena w centrum.
- Porty (interfejsy) definiują komunikację.
- Adaptery implementują porty.

Pozwala oddzielić: - logikę biznesową, - technologię (DB, HTTP, messaging).

🔷 5. Czym jest Onion Architecture?

✅ **Odpowiedź**

Onion Architecture: - Warstwy koncentryczne. - Rdzeń domenowy w środku. - Zależności skierowane do środka.

Podobna do Clean i Hexagonal — różni się nazewnictwem i akcentem.

3️⃣ Domain-Driven Design (DDD)

🔷 6. Czym jest Bounded Context?

✅ **Odpowiedź**

Bounded Context to granica modelu domenowego.

W obrębie kontekstu: - Model ma jednoznaczne znaczenie.

Między kontekstami: - Możliwe różne definicje tego samego pojęcia.

Ułatwia modularność i skalowanie organizacyjne.

🔷 7. Czym jest Aggregate?

✅ **Odpowiedź**

Aggregate to klaster encji traktowany jako jedna jednostka spójności.

Cechy: - Ma Aggregate Root. - Tylko Root jest dostępny z zewnątrz. - Spójność utrzymywana w obrębie agregatu.

Transakcje powinny obejmować jeden agregat.

🔷 8. Czym jest CQRS?

✅ **Odpowiedź**

CQRS (Command Query Responsibility Segregation):

- Oddziela operacje zapisu (Command) od odczytu (Query).

Zalety: - Możliwość optymalizacji odczytu i zapisu niezależnie. - Lepsza skalowalność.

Wady: - Większa złożoność.

🔹 9. Czym jest Event Sourcing?

✅ **Odpowiedź**

Event Sourcing: - Stan systemu przechowywany jako sekwencja zdarzeń. - Aktualny stan rekonstruowany przez odtworzenie zdarzeń.

Zalety: - Pełna historia. - Audytowalność.

Wyzwania: - Migracje eventów. - Złożoność.

4️⃣ Modularność i Skalowanie

🔹 10. Czym jest Modular Monolith?

✅ **Odpowiedź**

Modular Monolith: - Jedna aplikacja. - Silna modularność wewnętrzna. - Wyraźne granice między modułami.

Zalety: - Prostota wdrożenia. - Brak kosztów sieciowych.

Może być etapem przed mikroserwisami.

🔹 11. Jak podejmować decyzje architektoniczne?

✅ **Odpowiedź**

Architektura to kompromisy.

Należy analizować: - Skalowalność - Złożoność - Koszt utrzymania - Wymagania niefunkcjonalne

Nie istnieje jedna “najlepsza” architektura — zależy od kontekstu.

💡 Questions

🌐 MIKROSERWISY

- 🌐 MIKROSERWISY

1️⃣ Fundamenty Mikroserwisów

🔷 1. Jak wyznaczać granice mikroserwisów?

✅ **Odpowiedź**

Granice mikroserwisów powinny wynikać z domeny biznesowej, nie z podziału technicznego.

Typowe podejścia: - Bounded Context (DDD) jako granica serwisu. - Wysoka spójność wewnątrz serwisu, niskie sprzężenie między serwisami.

Zły znak: - Silne transakcje rozproszone. - Wspólna baza danych między serwisami.

🔷 2. Synchroniczna vs asynchroniczna komunikacja — kiedy co?

✅ **Odpowiedź**

Synchroniczna (HTTP/gRPC): - Prostota. - Natychmiastowa odpowiedź. - Ryzyko propagacji awarii i większa latencja.

Asynchroniczna (Kafka/RabbitMQ): - Luźniejsze powiązania. - Lepsza odporność. - Eventual consistency. - Większa złożoność (idempotency, retry, ordering).

W praktyce często miesza się oba podejścia.

🔹 3. Czym jest API Gateway?

✅ **Odpowiedź**

API Gateway to punkt wejścia do systemu mikroserwisów.

Funkcje: - routing, - auth, - rate limiting, - agregacja danych, - observability.

Zaleta: - klient nie musi znać topologii mikroserwisów.

Ryzyko: - single point of failure (trzeba skalować i zabezpieczać).

2️⃣ Odporność i Stabilność

🔹 4. Czym jest Circuit Breaker?

✅ **Odpowiedź**

Circuit Breaker chroni system przed kaskadową awarią.

Stany: - Closed — ruch przechodzi. - Open — ruch blokowany, szybka porażka (fail fast). - Half-open — testowe requesty.

Stosowany przy komunikacji synchronicznej.

🔹 5. Czym jest retry i jakie są ryzyka?

✅ **Odpowiedź**

Retry to ponawianie żądania po błędzie.

Ryzyka: - powielanie operacji (brak idempotency), - retry storm (wiele usług retry jednocześnie), - wzrost obciążenia i pogorszenie awarii.

Dobre praktyki: - exponential backoff, - jitter, - limit prób, - idempotency key.

🔹 6. Czym jest idempotency i jak ją osiągnąć?

✅ **Odpowiedź**

Idempotency oznacza, że wielokrotne wykonanie tej samej operacji daje ten sam efekt.

Przykład: - PUT jest idempotentny, - POST zwykle nie.

Techniki: - Idempotency-Key przechowywany po stronie serwera, - deduplikacja komunikatów, - naturalny klucz biznesowy.

3️⃣ Spójność w systemie rozproszonym

🔹 7. Czym jest Saga?

✅ **Odpowiedź**

Saga to wzorzec realizacji transakcji rozproszonej przez sekwencję lokalnych transakcji.

Każdy krok ma akcję kompensacyjną.

Rodzaje: - Orchestrated saga — centralny orchestrator. - Choreographed saga — serwisy reagują na eventy.

🔹 8. Czym jest Outbox Pattern?

✅ **Odpowiedź**

Outbox Pattern rozwiązuje problem atomowości między: - zapisem do bazy, - publikacją eventu.

Mechanizm: 1. Zapis danych i eventu w tej samej transakcji w DB (tabela outbox). 2. Osobny proces publikuje eventy do brokera. 3. Eventy są oznaczane jako wysłane.

Zapobiega utracie eventów.

🔹 9. Dlaczego współdzielona baza danych między mikroserwisami jest problemem?

✅ **Odpowiedź**

Współdzielona baza: - zwiększa sprzężenie, - wymusza wspólne wdrożenia, - utrudnia niezależną ewolucję schematu, - zwiększa ryzyko blokad i konfliktów.

Zasada: Database per service.

4️⃣ Obserwowalność

🔹 10. Czym jest observability w mikroserwisach?

✅ **Odpowiedź**

Observability to zdolność zrozumienia stanu systemu na podstawie sygnałów.

Trzy filary: - Logs (najlepiej structured) - Metrics (np. Prometheus) - Traces (distributed tracing)

Bez observability mikroserwisy są trudne w utrzymaniu.

💡 Questions

🔌 KOMUNIKACJA I INTEGRACJA

1️⃣ REST i Semantyka HTTP

🔷 1. Co oznacza idempotency w HTTP i które metody są idempotentne?

✅ **Odpowiedź**

Idempotency oznacza, że wielokrotne wykonanie tego samego requestu daje ten sam efekt końcowy.

Metody idempotentne: - GET - PUT - DELETE (efekt końcowy ten sam: zasób usunięty) - HEAD - OPTIONS

Metody nie-idempotentne: - POST (zwykle tworzy nowy zasób przy każdym wywołaniu).

Idempotency jest kluczowa dla retry.

🔷 2. Jakie są najważniejsze klasy kodów HTTP i jak ich używać?

✅ **Odpowiedź**

2xx — sukces - 200 OK - 201 Created - 204 No Content

3xx — przekierowania - 301/302

4xx — błąd klienta - 400 Bad Request - 401 Unauthorized (brak uwierzytelnienia) - 403 Forbidden (brak uprawnień) - 404 Not Found - 409 Conflict

5xx — błąd serwera - 500 Internal Server Error - 503 Service Unavailable

Poprawne statusy zwiększają czytelność API i ułatwiają integracje.

🔷 3. Czym jest caching w HTTP?

✅ **Odpowiedź**

Caching pozwala ograniczyć liczbę requestów i poprawić wydajność.

Mechanizmy: - Cache-Control - ETag + If-None-Match - Last-Modified + If-Modified-Since

ETag pozwala na walidację wersji zasobu (304 Not Modified).

2️⃣ gRPC

🔷 4. Czym jest gRPC i kiedy jest lepsze od REST?

✅ **Odpowiedź**

gRPC: - Protokół RPC oparty o HTTP/2. - Używa Protobuf. - Wspiera streaming.

Lepsze niż REST gdy: - potrzebujemy niskiej latencji, - komunikacja serwis-serwis, - silne kontrakty typów, - streaming.

REST częściej dla komunikacji z frontendem (łatwiejsze debugowanie i kompatybilność).

3️⃣ Messaging: Kafka vs RabbitMQ

🔷 5. Kafka vs RabbitMQ — kluczowe różnice

✅ **Odpowiedź**

Kafka: - Log zdarzeń (append-only). - Konsumenci trzymają offset. - Bardzo dobre do streamingu i dużej skali.

RabbitMQ: - Klasyczna kolejka. - Broker zarządza dostarczaniem. - Dobre do task queue i routing.

Kafka świetna do event-driven i audytu. RabbitMQ świetny do work distribution.

🔷 6. Co oznacza at-least-once, at-most-once, exactly-once delivery?

✅ **Odpowiedź**

At-most-once: - Wiadomość może zginąć. - Nigdy nie będzie duplikatu.

At-least-once: - Wiadomość nie zginie. - Mogą pojawić się duplikaty.

Exactly-once: - Brak utraty i brak duplikatów.

W praktyce exactly-once jest trudne i zwykle osiąga się je przez: - idempotency, - deduplikację, - transakcyjność brokera/producenta.

🔷 7. Jak radzić sobie z duplikatami wiadomości?

✅ **Odpowiedź**

Ponieważ at-least-once jest częste, system powinien tolerować duplikaty.

Techniki: - deduplikacja po messageId / idempotency key, - “upsert” zamiast “insert”, - przechowywanie historii przetworzonych eventów.

🔷 8. Czym jest ordering i dlaczego jest trudny?

✅ **Odpowiedź**

Ordering oznacza zachowanie kolejności zdarzeń.

Wyzwania: - partycje w Kafka — ordering jest gwarantowany tylko w obrębie partycji. - równoległe konsumpcje.

Rozwiązanie: - klucz partycji oparty o agregat (np. aggregateId).

4️⃣ Schema Evolution

🔷 9. Czym jest schema evolution?

✅ **Odpowiedź**

Schema evolution to zmiana formatu danych bez psucia kompatybilności.

Dobre praktyki: - dodawanie pól jako opcjonalne, - unikanie zmiany znaczenia pól, - wersjonowanie kontraktów.

Protobuf/Avro wspierają kompatybilne zmiany lepiej niż “goły” JSON.

💡 Questions

🧰 DEVOPS / RUNTIME / PRODUKCJA

1️⃣ Kontenery i Artefakty

🔷 1. Czym jest kontener (Docker) i czym różni się od VM?

✅ **Odpowiedź**

Kontener: - Izoluje procesy na poziomie systemu operacyjnego (namespaces, cgroups). - Współdzieli kernel z hostem. - Jest lekki i szybko startuje.

VM: - Wirtualizuje sprzęt. - Każda VM ma własny system operacyjny. - Jest cięższa i wolniej startuje.

Kontenery są lepsze do skalowania aplikacji, VM częściej do silnej izolacji.

🔷 2. Co powinno znaleźć się w dobrym Dockerfile dla aplikacji Java?

✅ **Odpowiedź**

Dobre praktyki: - Multi-stage build (osobno build i runtime). - Użycie lekkiego obrazu runtime (np. JRE zamiast JDK). - Uruchamianie jako non-root. - Stabilne warstwy (kopiuj najpierw pliki zależności, potem kod). - Parametry JVM dostosowane do kontenera (pamięć, GC).

Cel: mały obraz, szybki build, bezpieczeństwo.

2️⃣ CI/CD

🔹 3. Czym jest CI/CD i jakie są typowe etapy pipeline?

✅ Odpowiedź

CI (Continuous Integration): - Częste integrowanie zmian. - Automatyczne budowanie i testy.

CD (Continuous Delivery/Deployment): - Automatyczne dostarczanie na środowiska. - Deployment — automatyczne wdrożenie na produkcję.

Typowe etapy: - build - unit tests - static analysis (np. Sonar) - integration tests (np. Testcontainers) - package (jar/docker) - deploy

---

3️⃣ Monitoring i Observability

🔹 4. Czym różni się monitoring od observability?

✅ Odpowiedź

Monitoring: - Sprawdza znane metryki i alertuje na znane problemy.

Observability: - Umożliwia diagnozę nieznanych problemów na podstawie sygnałów.

Trzy filary observability: - Logs - Metrics - Traces

🔹 5. Co to jest structured logging i dlaczego jest ważny?

✅ Odpowiedź

Structured logging to logowanie w formacie maszynowo czytelnym (np. JSON), z polami: - timestamp - level - message - traceId - spanId - userId / requestId

Ułatwia: - filtrowanie, - agregację, - korelację logów w systemach rozproszonych.

🔹 6. Czym jest distributed tracing?

✅ Odpowiedź

Distributed tracing śledzi przebieg jednego requestu przez wiele usług.

Pojęcia: - Trace — całe żądanie end-to-end. - Span — pojedynczy krok (np. HTTP call, DB query).

Wymaga propagacji: - traceId - spanId

Pozwala diagnozować: - wąskie gardła, - opóźnienia, - błędy w komunikacji.

4️⃣ Health Checks i Deploy

🔷 7. Czym różni się liveness od readiness?

✅ **Odpowiedź**

Liveness: - Czy aplikacja żyje (nie zawiesiła się)? - Jeśli nie — restart.

Readiness: - Czy aplikacja jest gotowa przyjmować ruch? - Jeśli nie — wyłączenie z load balancera (bez restartu).

W Kubernetes są to osobne probe.

🔷 8. Blue/Green vs Canary — czym się różnią?

✅ **Odpowiedź**

Blue/Green: - Dwa środowiska: stare (blue) i nowe (green). - Przełączenie ruchu na green. - Szybki rollback przez powrót na blue.

Canary: - Nowa wersja dostaje mały procent ruchu. - Stopniowe zwiększanie. - Pozwala wykryć problemy zanim dotkną wszystkich.

5️⃣ Skalowanie

🔷 9. Horizontal vs Vertical scaling — różnice i konsekwencje

✅ **Odpowiedź**

Vertical scaling: - Większa maszyna (CPU/RAM). - Proste, ale ma limit sprzętowy.

Horizontal scaling: - Więcej instancji. - Wymaga stateless aplikacji lub zewnętrznego stanu. - Lepsza dostępność i skalowalność.

W praktyce preferuje się horizontal scaling.

💡 Questions

🛡️ BEZPIECZEŃSTWO

1️⃣ Podstawy Kryptografii

🔷 1. Czym różni się szyfrowanie od haszowania?

✅ **Odpowiedź**

Szyfrowanie: - Proces odwracalny. - Używa klucza do zaszyfrowania i odszyfrowania danych. - Stosowane do ochrony poufności (np. TLS).

Haszowanie: - Proces nieodwracalny. - Generuje skrót o stałej długości. - Stosowane do przechowywania haseł.

Hasło nie powinno być szyfrowane — powinno być haszowane.

🔷 2. Czym jest salt i dlaczego jest ważny?

✅ **Odpowiedź**

Salt to losowa wartość dodawana do hasła przed haszowaniem.

Zapobiega: - atakom rainbow table, - identycznym hashom dla tych samych haseł.

Każde hasło powinno mieć unikalny salt.

2️⃣ TLS i Transport

🔷 3. Jak działa TLS w skrócie?

✅ **Odpowiedź**

TLS zapewnia poufność i integralność komunikacji.

Proces: 1. Handshake. 2. Wymiana kluczy (asymetryczna kryptografia). 3. Ustalenie klucza symetrycznego. 4. Szyfrowana komunikacja.

Chroni przed: - podsłuchem, - modyfikacją danych.

3️⃣ OWASP i Najczęstsze Ataki

🔹 4. Czym jest SQL Injection?

✅ **Odpowiedź**

SQL Injection to wstrzyknięcie złośliwego kodu SQL do zapytania.

Przyczyna: - Konkatenacja stringów zamiast parametrów.

Zapobieganie: - PreparedStatement, - ORM, - walidacja danych.

🔹 5. Czym jest XSS?

✅ **Odpowiedź**

XSS (Cross-Site Scripting): - Wstrzyknięcie złośliwego skryptu do strony.

Rodzaje: - Stored - Reflected - DOM-based

Zapobieganie: - Escaping danych, - CSP (Content Security Policy).

🔹 6. Czym jest CSRF?

✅ **Odpowiedź**

CSRF (Cross-Site Request Forgery): - Wysłanie żądania w imieniu zalogowanego użytkownika bez jego wiedzy.

Ochrona: - CSRF token, - SameSite cookies.

4️⃣ Autoryzacja i Dostęp

🔹 7. Czym jest RBAC vs ABAC?

✅ **Odpowiedź**

RBAC (Role-Based Access Control): - Uprawnienia przypisane do ról.

ABAC (Attribute-Based Access Control): - Decyzja na podstawie atrybutów (rola, czas, lokalizacja).

ABAC daje większą elastyczność kosztem złożoności.

🔷 8. Jak zabezpieczać sekrety w systemie?

✅ **Odpowiedź**

Nie powinny być: - w kodzie, - w repozytorium, - w obrazach Docker.

Powinny być: - w Secret Manager (Vault, AWS Secrets Manager), - w zmiennych środowiskowych, - rotowane cyklicznie.

Zasada: najmniejsze możliwe uprawnienia (least privilege).

💡 Questions

🧪 TESTOWANIE

1️⃣ Fundamenty Testowania

🔷 1. Czym różni się test jednostkowy od integracyjnego?

✅ **Odpowiedź**

Test jednostkowy (unit test): - Testuje pojedynczą klasę lub metodę. - Izoluje zależności (mocki, stuby). - Szybki i deterministyczny.

Test integracyjny: - Testuje współpracę komponentów. - Może używać prawdziwej bazy, brokera, kontekstu Spring. - Wolniejszy, ale bliższy rzeczywistości.

🔹 2. Czym jest test end-to-end (E2E)?

✅ **Odpowiedź**

Test E2E testuje cały system jako całość.

- Obejmuje warstwę HTTP, bazę danych, integracje.
- Symuluje zachowanie użytkownika.
- Najbardziej realistyczny, ale najwolniejszy i najbardziej kruchy.

2️⃣ Mockowanie i Izolacja

🔹 3. Czym jest mock, stub i spy?

✅ **Odpowiedź**

Mock: - Obiekt symulujący zachowanie zależności. - Pozwala weryfikować interakcje.

Stub: - Zwraca z góry ustalone dane. - Nie weryfikuje interakcji.

Spy: - Częściowo prawdziwy obiekt. - Pozwala nadpisać wybrane metody.

🔹 4. Kiedy mockowanie jest złym pomysłem?

✅ **Odpowiedź**

- Gdy test staje się testem implementacji zamiast zachowania.
- Gdy mockujemy zbyt wiele zależności.
- Gdy test jest kruchy przy refaktorze.

Preferować testowanie zachowania, nie implementacji.

3️⃣ Testy Integracyjne i Środowisko

🔹 5. Czym jest Testcontainers i dlaczego jest użyteczny?

✅ **Odpowiedź**

Testcontainers pozwala uruchomić prawdziwe zależności (DB, Kafka) w Dockerze podczas testów.

Zalety: - Realne środowisko. - Brak zależności od lokalnej konfiguracji. - Reproducibility.

🔷 6. Czym jest Contract Testing?

✅ **Odpowiedź**

Contract Testing weryfikuje zgodność między serwisami.

- Consumer definiuje kontrakt.
- Provider musi go spełniać.

Zapobiega breaking changes w mikroserwisach.

4️⃣ Zaawansowane Techniki

🔷 7. Czym jest TDD?

✅ **Odpowiedź**

TDD (Test-Driven Development):

1. Napisz test.
2. Napisz minimalny kod, by test przeszedł.
3. Refaktor.

Cel: - Lepszy design. - Większa pewność zmian.

🔷 8. Czym jest property-based testing?

✅ **Odpowiedź**

Property-based testing polega na testowaniu ogólnych właściwości funkcji zamiast konkretnych przypadków.

Przykład: - Funkcja sortująca powinna zwracać listę uporządkowaną rosnąco.

Generowane są losowe dane wejściowe.

🔷 9. Czym jest mutation testing?

✅ **Odpowiedź**

Mutation testing sprawdza jakość testów poprzez wprowadzanie małych zmian w kodzie (mutacje).

Jeśli testy nie wykryją zmiany — są niewystarczające.

Pozwala ocenić skuteczność testów, nie tylko ich pokrycie.

💡 Questions

📈 WYDAJNOŚĆ I DIAGNOSTYKA

1️⃣ Zrozumienie Wydajności

🔷 1. Czym różni się latency od throughput?

✅ **Odpowiedź**

Latency: - Czas obsługi pojedynczego żądania. - Mierzona np. w milisekundach.

Throughput: - Liczba obsłużonych żądań w jednostce czasu. - Mierzona np. w requests/second.

System może mieć: - niską latency i niski throughput, - wysoką latency i wysoki throughput.

Optymalizacja zależy od wymagań biznesowych.

🔷 2. Czym jest bottleneck i jak go znaleźć?

✅ **Odpowiedź**

Bottleneck to element systemu ograniczający wydajność całości.

Może to być: - CPU, - baza danych, - I/O, - lock contention, - sieć.

Identyfikacja: - Profilowanie CPU, - analiza metryk, - distributed tracing, - analiza GC.

Optymalizuje się wąskie gardło, nie wszystko naraz.

2️⃣ JVM Profilowanie

🔹 3. Czym jest heap dump i kiedy go używać?

✅ **Odpowiedź**

Heap dump to zrzut pamięci heap w danym momencie.

Używany do: - analizy wycieków pamięci, - identyfikacji dużych obiektów, - analizy referencji.

Narzędzia: - VisualVM - Eclipse MAT

🔹 4. Czym jest thread dump?

✅ **Odpowiedź**

Thread dump pokazuje stan wszystkich wątków.

Używany do: - analizy deadlock, - blokad, - wysokiego zużycia CPU.

Zawiera stack trace każdego wątku.

🔹 5. Jak działa Garbage Collection i kiedy może być problemem?

✅ **Odpowiedź**

GC usuwa nieosiągalne obiekty.

Problem gdy: - długie pauzy (Stop-The-World), - zbyt częste kolekcje, - wysokie allocation rate.

Rozwiązania: - zmiana GC (G1, ZGC), - tuning heap size, - zmniejszenie tworzenia obiektów.

3️⃣ Analiza Algorytmiczna

🔹 6. Dlaczego Big-O jest ważne w systemach backendowych?

✅ **Odpowiedź**

Big-O opisuje złożoność algorytmu względem rozmiaru danych.

Przykład: - O(1) — stały czas, - O(log n) — logarytmiczny, - O(n) — liniowy, - $O(n^2)$ — kwadratowy.

Przy dużej skali różnice stają się krytyczne.

4️⃣ Load Testing

🔷 7. Czym jest load testing i czym różni się od stress testing?

✅ **Odpowiedź**

Load testing: - Test przy oczekiwanym obciążeniu. - Sprawdza czy system spełnia SLA.

Stress testing: - Test powyżej zakładanego obciążenia. - Sprawdza jak system zachowuje się przy przeciążeniu.

Celem jest poznanie granic systemu i jego zachowania w awarii.

💡 Questions

🧠 SYSTEM DESIGN / THINKING

1️⃣ Myślenie Architektoniczne

🔷 1. Czym są trade-offs w architekturze?

✅ **Odpowiedź**

Każda decyzja architektoniczna to kompromis między: - wydajnością, - skalowalnością, - złożonością, - kosztem utrzymania, - czasem dostarczenia.

Nie istnieje rozwiązanie idealne — istnieje rozwiązanie najlepsze w danym kontekście.

Architekt powinien jasno rozumieć konsekwencje wyborów.

🔷 2. Jak rozumieć CAP w praktyce projektowej?

✅ **Odpowiedź**

W systemach rozproszonych Partition Tolerance jest obowiązkowe.

Pozostaje wybór między: - Consistency (CP) - Availability (AP)

System finansowy → preferuje CP. System social media → często AP.

Projekt musi uwzględniać konsekwencje chwilowej niespójności.

2️⃣ Projektowanie pod Skalę

🔷 3. Czym jest skalowalność funkcjonalna vs techniczna?

✅ **Odpowiedź**

Skalowalność techniczna: - Więcej instancji, więcej zasobów.

Skalowalność funkcjonalna: - Możliwość rozwijania systemu bez eksplozji złożoności. - Modularność, bounded contexts.

Obie są kluczowe dla systemów długowiecznych.

🔷 4. Czym jest latency budget?

✅ **Odpowiedź**

Latency budget to maksymalny czas odpowiedzi systemu rozłożony na komponenty.

Przykład: - API ma 300ms SLA. - DB może użyć 100ms. - Zewnętrzny serwis 80ms. - Reszta to logika.

Pomaga kontrolować zależności i unikać kaskadowych opóźnień.

3️⃣ Projektowanie na Awarię

🔷 5. Co oznacza “design for failure”?

✅ **Odpowiedź**

W systemach rozproszonych zakładamy, że: - sieć zawiedzie, - serwis zawiedzie, - baza zawiedzie.

System powinien: - degradować się łagodnie (graceful degradation), - mieć timeouty, - mieć retry z ograniczeniami, - stosować circuit breaker.

🔹 6. Czym jest graceful degradation?

✅ **Odpowiedź**

Graceful degradation oznacza, że przy awarii części systemu: - system nadal działa, - ale z ograniczoną funkcjonalnością.

Przykład: - brak rekomendacji, ale działa koszyk zakupowy.

4️⃣ Backpressure i Przepływ Danych

🔹 7. Czym jest backpressure na poziomie systemowym?

✅ **Odpowiedź**

Backpressure to kontrola przepływu danych między komponentami.

Bez backpressure: - kolejki rosną, - pamięć się wyczerpuje, - system się zapada.

Mechanizmy: - bounded queues, - rate limiting, - reactive streams (request(n)).

5️⃣ Failure Modes i Analiza Ryzyka

🔹 8. Jak analizować failure modes systemu?

✅ **Odpowiedź**

Należy zidentyfikować: - pojedyncze punkty awarii (SPOF), - zależności zewnętrzne, - operacje długotrwałe, - miejsca blokad.

Stosowane techniki: - Chaos engineering, - testy awarii, - analiza scenariuszy (“what if”).

Celem jest zwiększenie odporności systemu.

6️⃣ Myślenie Długoterminowe

🔹 9. Jak podejmować decyzje technologiczne długoterminowo?

✅ **Odpowiedź**

Należy brać pod uwagę: - dojrzałość technologii, - wsparcie społeczności, - koszt utrzymania, - łatwość rekrutacji, - vendor lock-in.

Najlepsza technologia to taka, którą zespół potrafi utrzymać przez lata.