Spring & Spring Boot - Quiz

1. Cos'è un framework?
2. Qual è l'obiettivo principale di Spring?
3. In quale anno è stato creato Spring?
4. Quale tipo di applicazioni è il focus di Spring?
5. Cosa semplifica Spring Boot rispetto a Spring puro?
6. Cos'è il 'boilerplate code'?
7. Come Spring riduce il codice boilerplate?
8. Cosa sono i moduli Core e Beans?
9. Che ruolo ha il modulo Context in Spring?
10. Che cos'è l'Expression Language in Spring?
11. Cosa offre il modulo JDBC?
12. Qual è la funzione del modulo ORM?
13. Qual è lo scopo del modulo JMS?
14. Cosa fa il modulo Transaction?
15. Cosa permette il modulo Web di Spring Boot?
16. Cosa consente il modulo Web-Sockets?
17. Che cos’è Spring Initializr?
18. Quali sono le opzioni configurabili in Spring Initializr?
19. Cosa sono i Bean in Spring?
20. Cos'è lo Spring Container?

Spring & Spring Boot - Risposte

1. Un framework è un insieme di librerie e strumenti per semplificare lo sviluppo di applicazioni.
2. Semplificare lo sviluppo di applicazioni Java enterprise.
3. Nel 2002.
4. Applicazioni enterprise, incluse web, cloud e di sicurezza.
5. Spring Boot automatizza la configurazione e facilita l'avvio dei progetti.
6. Codice ripetitivo e standard richiesto da molte tecnologie.
7. Con l'uso di template e classi di utilità.
8. Moduli fondamentali per IoC e Dependency Injection.
9. Gestisce il ciclo di vita degli oggetti e fornisce un ambiente modulare.
10. Un linguaggio usato per configurazione e manipolazione dati.
11. Fornisce un'astrazione per semplificare l'accesso ai database.
12. Permette l'uso di JPA, Hibernate, iBatis con Spring.
13. Gestisce messaggi in entrata e uscita in applicazioni.
14. Gestisce la transazionalità indipendentemente dal tipo di persistenza.
15. Fornisce funzionalità web di base e supporto MVC.
16. Permette la comunicazione bidirezionale in tempo reale.
17. Uno strumento per creare progetti Spring Boot con dipendenze.
18. Linguaggio, versione Spring, dipendenze, packaging, build system.
19. Oggetti gestiti dal container Spring.
20. Componente che gestisce il ciclo di vita dei Bean e le loro dipendenze.

Componenti, Annotazioni e Configurazione - Quiz

1. Cosa rappresenta l'annotazione @Component?
2. A cosa serve l'annotazione @Autowired?
3. Quali sono i tre tipi di Dependency Injection in Spring?
4. Cosa fa l'annotazione @Configuration?
5. A cosa serve l'annotazione @Bean?
6. Qual è la differenza tra @Component e una classe @Configuration?
7. Cosa succede se @Component è in un package non figlio del package principale?
8. A cosa serve l'annotazione @Primary?
9. Quando si usa @Qualifier?
10. Cosa rappresenta il concetto di Loose Coupling?
11. Come Spring promuove il Loose Coupling?
12. Qual è il vantaggio principale del Dependency Injection?
13. Come si recupera un valore dal file application.properties?
14. Dove si trova di default il file application.properties in un progetto Spring Boot?
15. Qual è il ruolo di CommandLineRunner?
16. Quando viene eseguito il metodo run() di CommandLineRunner?
17. Perché è importante annotare un Runner con @Component?
18. Che annotazione unisce @Configuration, @ComponentScan e @EnableAutoConfiguration?
19. Cosa sono le Starter Dependencies?
20. Come si aggiungono dipendenze in un progetto Spring?

Componenti, Annotazioni e Configurazione - Risposte

1. Indica che una classe è un componente gestito da Spring.

2. Permette a Spring di iniettare automaticamente le dipendenze.
3. Constructor Injection, Setter Injection, Field Injection.
4. Dichiara una classe come contenente configurazioni di Bean.
5. Definisce un metodo che restituisce un Bean gestito da Spring.
6. @Component definisce componenti, @Configuration definisce configurazioni.
7. Il bean non viene creato automaticamente.
8. Indica quale Bean usare quando ce ne sono più dello stesso tipo.
9. Per scegliere esplicitamente quale Bean usare in caso di ambiguità.
10. Minimo accoppiamento tra componenti per maggiore flessibilità.
11. Attraverso l'uso di IoC e Dependency Injection.
12. Maggiore flessibilità, testabilità e riusabilità del codice.
13. Con l'annotazione @Value e la sintassi ${nomeParametro}.
14. Dentro la cartella src/main/resources.
15. Eseguire codice al termine dell'avvio dell'applicazione.
16. Subito dopo l'avvio completo del contesto Spring.
17. Per permettere l'esecuzione automatica del Runner.
18. @SpringBootApplication.
19. Dipendenze che includono automaticamente altre dipendenze necessarie.
20. Nel file pom.xml (Maven) o build.gradle (Gradle).

Spring Data - Quiz

1. Qual è lo scopo principale di Spring Data?
2. Quali due tecnologie supporta Spring Data per la persistenza?
3. Cos'è il pattern DAO?

4. Qual è il vantaggio di usare interfacce DAO?
5. Qual è la differenza principale tra Spring JDBC e JPA?
6. Cos'è il JDBC Template?
7. Cosa fa una classe RowMapper?
8. Che ruolo ha il file application.properties nella persistenza?
9. Cosa fa il modulo spring-boot-starter-data-jdbc?
10. Qual è il provider JPA di default in Spring?
11. Cosa permette JpaRepository?
12. Quali operazioni CRUD supporta JpaRepository?
13. Come si definisce un Repository JPA?
14. Che differenza c'è tra JdbcTemplate e JpaRepository?
15. Perché è utile seguire la notazione snake_case con JPA?
16. Come si mappa una classe su una tabella con JPA?
17. Cosa succede se cambio il JPA Provider?
18. Spring JPA consente query automatiche?
19. Come specificare il tipo di chiave primaria in un repository?
20. Qual è il vantaggio principale di usare Spring Data?

Spring Data - Risposte

1. Fornire strumenti per la gestione della persistenza usando i concetti Spring.
2. JDBC e JPA.
3. Separare la logica di business da quella di accesso ai dati.
4. Permette di cambiare implementazione senza modificare la logica.

5. JDBC è più diretto, JPA usa un livello di astrazione ORM.
6. Una classe che semplifica l'accesso ai dati tramite template methods.
7. Converte un ResultSet in oggetti Java.
8. Contiene i parametri di connessione e configurazione.
9. Includere tutte le dipendenze necessarie per JDBC.
10. Hibernate.
11. Gestire la persistenza con metodi CRUD automatici.
12. Create, Read, Update, Delete.
13. Estendendo JpaRepository e specificando entità e chiave primaria.
14. JdbcTemplate è per JDBC, JpaRepository è per ORM con JPA.
15. Evita la scrittura manuale di annotazioni di mapping.
16. Tramite l'annotazione @Entity e @Table.
17. Potrebbe essere necessario aggiornare le proprietà di configurazione.
18. Sì, tramite query derivate.
19. Usando i tipi generici: <Entità, TipoChiave>.
20. Standardizza e velocizza l'accesso ai dati con meno codice.

Spring Data JPA - Queries - Quiz

1. Cosa sono le query derivate in Spring Data JPA?
2. Come si scrive una query derivata?
3. Cosa significa findByNome?
4. Come si combinano condizioni multiple?
5. Cosa fa findByNomeAndCategoria?
6. Come si ordina una query derivata?

7. Come si limita il numero di risultati?
8. Cosa fa findFirstByCategoria?
9. Come si nega una condizione?
10. Cosa fa findByNomeNot?
11. Come si fanno ricerche parziali?
12. Cosa fa findByNomeLike?
13. Come si ignorano maiuscole/minuscole?
14. Come si filtra per range di valori?
15. Cosa fa findByPrezzoBetween?
16. Qual è un vantaggio delle query derivate?
17. Qual è uno svantaggio delle query derivate?
18. Cosa sono le query custom in JPQL?
19. Quando è preferibile usare query custom?
20. Quale interfaccia si estende per usare query derivate?

Spring Data JPA - Queries - Risposte

1. Query costruite automaticamente da Spring usando il nome del metodo.
2. Con una convenzione come findBy + nomeCampo.
3. Cerca per il campo nome.
4. Usando And o Or tra i nomi dei campi.
5. Cerca entità con nome e categoria specifici.
6. Con OrderBy e specifica Asc o Desc.
7. Con parole come First, Top, ecc.

8. Restituisce il primo elemento di una categoria.
9. Usando la parola chiave Not.
10. Cerca tutti tranne quelli con un certo nome.
11. Usando la parola chiave Like.
12. Cerca stringhe che contengono un pattern.
13. Con IgnoreCase nella query.
14. Con operatori Between, GreaterThan, LessThan.
15. Restituisce entità con valori in un intervallo.
16. Meno codice e maggiore leggibilità.
17. Meno controllo e prestazioni ridotte in query complesse.
18. Query scritte manualmente in JPQL o SQL nativo.
19. Quando serve maggiore flessibilità o prestazioni.
20. JpaRepository.

JUnit e Testing - Quiz

1. Cosa sono i test nel contesto dello sviluppo software?
2. Qual è l'obiettivo principale dei test?
3. Quali sono i principali tipi di test?
4. Cosa fa un Unit Test?
5. Cosa fa un Integration Test?
6. Cosa fa un test End-to-End?
7. Perché è utile automatizzare i test?
8. Cos'è JUnit?
9. Qual è la versione attuale di JUnit?

10. Cosa fa l’annotazione @Test?
11. Cosa fa assertEquals?
12. Cosa fa assertTrue?
13. Cosa fa assertThrows?
14. Cos'è @BeforeEach?
15. Cosa fa @AfterEach?
16. Cosa fa @SpringBootTest?
17. Cos'è la test coverage?
18. Cosa fa @DisplayName?
19. A cosa serve @ParameterizedTest?
20. Quali sono i vantaggi dei test parametrizzati?

JUnit e Testing - Risposte

1. Procedure per verificare il comportamento del software.
2. Assicurarsi che il software funzioni come previsto.
3. Unit Test, Integration Test, End-to-End Test, altri.
4. Verifica unità di codice isolate come metodi o classi.
5. Verifica l'interazione tra più componenti.
6. Simula l’intero flusso come farebbe un utente reale.
7. Permette di eseguire test ripetibili e integrati nei CI/CD.
8. Un framework per scrivere test in Java.
9. JUnit 5.
10. Identifica un metodo come test da eseguire.

11. Verifica che due valori siano uguali.
12. Verifica che una condizione sia vera.
13. Verifica che venga lanciata un'eccezione specifica.
14. Metodo eseguito prima di ogni test.
15. Metodo eseguito dopo ogni test.
16. Avvia il contesto Spring per i test.
17. Percentuale di codice eseguita durante i test.
18. Aggiunge un nome descrittivo ai test.
19. Permette di eseguire lo stesso test con più dati.
20. Verificano il comportamento del codice in scenari diversi con poco codice.