

# Spring Boot 3

Introduzione al Corso





Nicola La Rocca

# Il Docente

- Analista Sviluppatore multilinguaggio
- Oltre **20 anni** di esperienza nell'ICT
- Analisi e creazione di progetti C# e Java e SQL
- Analisi e creazione di progetti basati su piattaforma cloud AWS e Azure
- Creazione di progetti Mobile in Android
- Creazione di **ecosistemi microservices** cloud ready in AWS, GCS, Azure
- Esperienza nella progettazione e creazione di database e DWH con i principali DBMS SQL
- Docente Udemy

# Obiettivi del corso

- Imparare ad usare il **Framework Spring Boot 3** per sviluppare applicazioni web di diversa natura e tipologia.





# Destinatari

- Studenti di informatica
- Professionisti dell'IT che vogliono usare il framework Spring Boot 3
- Coloro i quali vogliono sviluppare applicazioni web usando il linguaggio Java



# I Requisiti

- Nessuna Conoscenza Pregressa di Spring
- Basi di Java
- Connessione a internet
- PC con almeno 8GB di memoria RAM

# Cosa Otterrai Iscrivendoti al Corso

- Accesso illimitato on demand a tutte le lezioni del corso
- Accesso alla sezione Domande & Risposte (D&R)
- Aggiornamenti e Integrazioni Future
- Accesso a tutto il materiale delle lezioni (slide, codice sorgente)
- Certificato di fine corso



# Spring Boot 3

Introduzione al framework



```
background: transparent;
background-size: 100vw 100vh;
}
.box{
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
  width: 400px;
  padding: 40px;
  background: #fff;
  box-sizing: border-box;
  box-shadow: 0 15px 25px #000;
  border-radius: 10px;
}
.box h2{
  margin: 0 0 30px;
  padding: 0;
  color: #fff;
  text-align: center;
}
.box h3{
  margin: 0 0 10px;
  padding: 0;
  color: #fff;
  text-align: center;
}
.box .inputs
```



## Introduzione all'Application Framework

- Un "application framework" (framework per applicazioni) è un insieme di librerie, strumenti e convenzioni di programmazione progettati per facilitare lo sviluppo di software, in particolare applicazioni software complesse o specifiche di un dominio.
- Questi framework offrono una serie di funzionalità comuni, componenti e modelli di progettazione, consentendo agli sviluppatori di concentrarsi sulla logica specifica dell'applicazione anziché sulla scrittura di codice di base o di infrastruttura comune.
- Gli application framework riducono quindi il tempo e lo sforzo necessari per creare un'applicazione e agevolano la sua manutenzione e l'aggiornamento nel tempo.
- Uno dei principali vantaggi degli application framework è che consentono agli sviluppatori di seguire una struttura organizzativa predefinita e una metodologia di sviluppo coerente.



## Introduzione all'Application Framework

- Questo rende il codice più **ordinato, leggibile e facilmente manutenibile**, soprattutto quando più sviluppatori lavorano allo stesso progetto.
- Inoltre, l'uso di un framework **facilita la collaborazione tra gli sviluppatori**, in quanto i componenti comuni e i modelli di progettazione standardizzati **facilitano la comprensione e la modifica del codice** da parte di altri membri del team.
- Gli application framework possono fornire anche librerie che semplificano l'interazione con database, la gestione di sessioni e autenticazioni, la manipolazione di input/output sui dispositivi, la gestione di eventi e errori, la creazione di interfacce utente, ed altro.
- Queste librerie predefinite **riducono notevolmente il tempo richiesto per lo sviluppo di queste funzionalità di base**, poiché l'implementazione è già pronta e i sviluppatori possono semplicemente utilizzare le funzioni esistenti anziché scrivere da zero.



## Introduzione all'Application Framework

- Inoltre, gli application framework spesso offrono anche strumenti e risorse, come IDE (Integrated Development Environment), simulatori e documentazione, che rendono più agevole e produttiva l'attività di sviluppo.
- Alcuni dei framework di applicazione più popolari includono Django per lo sviluppo di applicazioni web in Python, Ruby on Rails per lo sviluppo di applicazioni web in Ruby, Angular per la creazione di interfacce utente dinamiche in TypeScript, e naturalmente lo Spring Framework che permette lo sviluppo di applicazioni Java.
- Non c'è bisogno di conoscere ogni elemento di un framework ma è essenziale saper utilizzare quelle parti che possono essere utili alla attività di sviluppo dell'applicazione.



## Introduzione al Framework Spring

- Il Spring Framework è un ecosistema di framework basato su Java che offre **una vasta gamma di strumenti e funzionalità** per semplificare lo sviluppo di applicazioni aziendali robuste e scalabili.
- Di seguito sono elencate le principali caratteristiche del Spring Framework:
  - **Inversion of Control (IoC) e Dependency Injection (DI):** Il Spring Framework utilizza il principio dell'Inversione di Controllo per gestire la creazione e la gestione degli oggetti nelle applicazioni. Attraverso il meccanismo di Dependency Injection, le dipendenze tra gli oggetti vengono risolte dal container di Spring, consentendo di scrivere codice modularizzato e facilmente testabile.
  - **Spring Container:** Il Spring Container è il componente centrale del framework che gestisce e controlla tutti gli oggetti creati all'interno di un'applicazione Spring. Il container crea gli oggetti, gestisce le loro dipendenze e li fornisce quando richiesto.

# Introduzione al Framework Spring

- **Spring MVC:** Il modulo Spring MVC (Model-View-Controller) offre un framework per lo sviluppo di applicazioni web basate su **modello MVC**. Fornisce una robusta infrastruttura per la gestione delle richieste HTTP, la gestione delle view e la gestione della logica di business.
- **Security:** Il modulo **Spring Security** fornisce funzionalità per la gestione dell'autenticazione, dell'autorizzazione e della protezione dell'applicazione da attacchi comuni come **Cross-Site Scripting (XSS)** e **Cross-Site Request Forgery (CSRF)**.
- **Persistence:** Il Spring Framework supporta l'integrazione con diversi framework di persistenza dati come **Java Persistence API (JPA)**, **Hibernate** e **MyBatis**. Offre anche il supporto per il database transaction management utilizzando il modulo **Spring Transaction Management**.



# Introduzione al Framework Spring

- **AOP (Aspect-Oriented Programming):** Il framework Spring integra il concetto di AOP, che consente di separare le funzionalità trasversali dalla logica di business di un'applicazione. Questo consente di implementare in modo pulito e modulare funzionalità come il logging, il caching e la transazione.
- **Testing:** Il framework Spring supporta il testing delle applicazioni. Fornisce un ambiente di test integrato che facilita la scrittura di test unitari e di integrazione.
- **Gestione delle transazioni:** Spring offre un modulo per la gestione delle transazioni, che semplifica l'implementazione delle funzionalità di transazione nelle applicazioni. Supporta transazioni locali e distribuite, consentendo di gestire facilmente le transazioni di database.





## Introduzione al Framework Spring

- **Modularità:** Spring è altamente modulare, consentendo ai developer di utilizzare solo i moduli di cui hanno bisogno per le proprie applicazioni. Ciò favorisce la flessibilità, facilitando l'integrazione con altri framework e librerie di terze parti.
- **Facilità di integrazione:** Il Spring Framework semplifica l'integrazione con altri framework e componenti di terze parti. Offre connettori per l'integrazione con diverse tecnologie come Struts, JSF, JMS, REST, SOAP, ecc.
- **Gestione delle Eccezioni =** Spring semplifica la gestione delle eccezioni attraverso l'uso di annotazioni o configurazioni XML, consentendo il controllo dichiarativo delle eccezioni.
- **Internazionalizzazione (i18n) e Localizzazione (l10n) =** Spring fornisce supporto per l'internazionalizzazione e la localizzazione delle applicazioni, consentendo di supportare più lingue e culture.

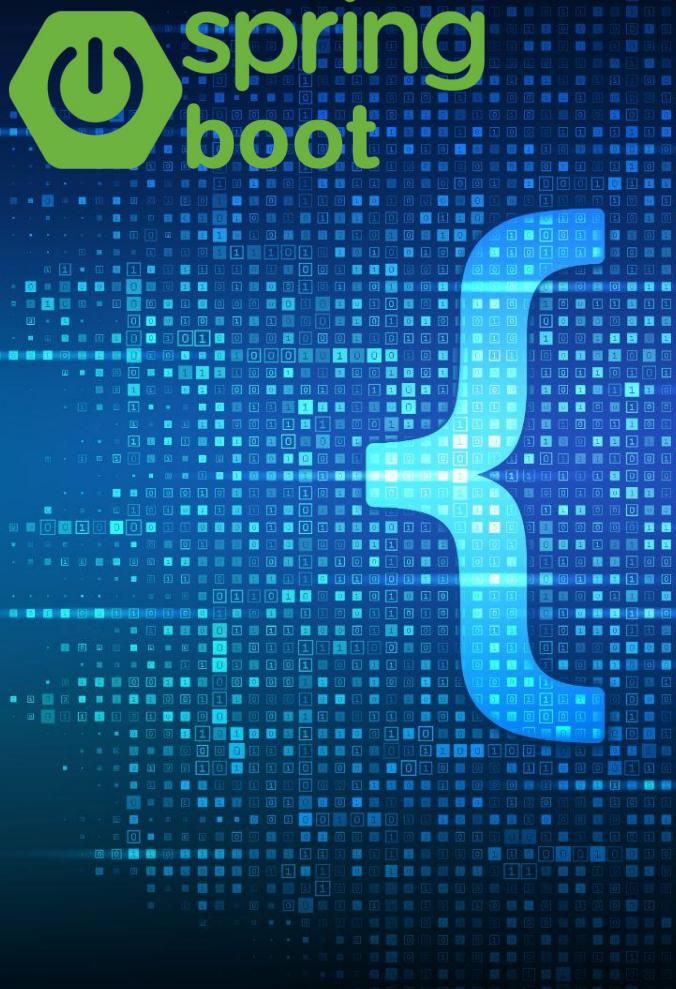
The background of the slide features the Spring Framework logo on the left, which consists of a green leaf icon followed by the word "spring" in a lowercase sans-serif font. To the right of the logo is a blue digital grid pattern.

# Introduzione al Framework Spring

- **Spring Boot** = Spring Boot semplifica la creazione di applicazioni Spring stand-alone, con configurazione automatica e avvio rapido.
- **Spring Cloud** = Spring Cloud fornisce strumenti per la creazione di applicazioni cloud-native, inclusi servizi di registrazione, bilanciamento del carico, configurazione esterna e altro ancora.
- **Comunità Attiva e Documentazione** = Spring ha una vasta comunità di sviluppatori e offre documentazione completa e risorse per gli stessi. Secondo molteplici sondaggi sui sviluppatori Java, Spring è il framework Java maggiormente utilizzato (<https://www.jrebel.com/blog/2020-java-technology-report>)

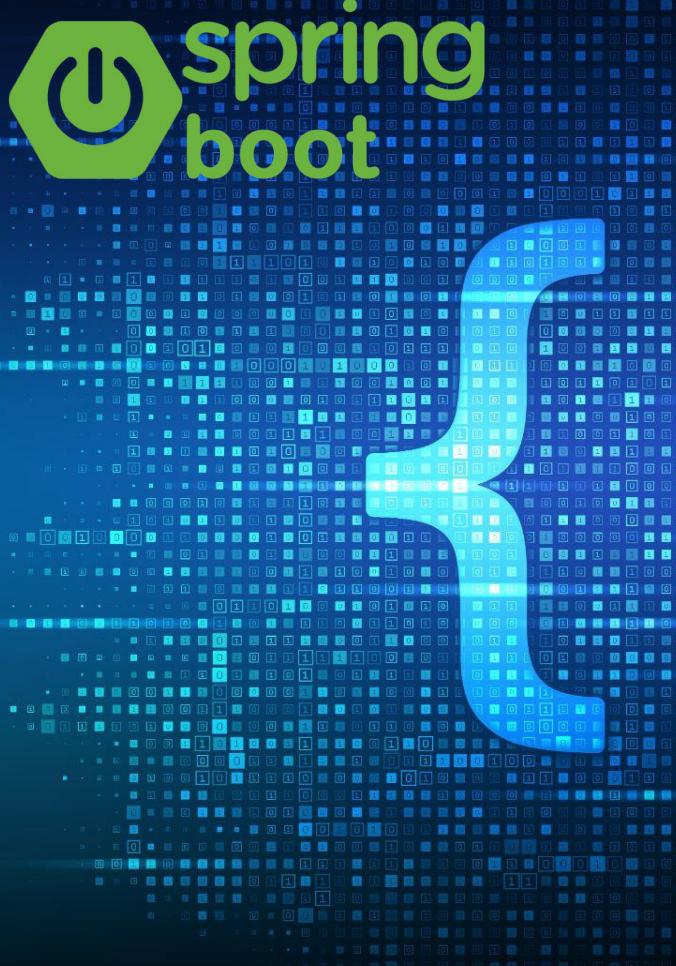
# Spring Boot 3

Introduzione al framework  
Spring Boot 3



## Limiti dello Spring Framework

- La configurazione dei progetti NON è una operazione semplice.
- Lo sviluppatore deve investire molto tempo nella fase di configurazione e implementazione del progetto creato con Spring Framework
- Ogni specifica funzionalità richiede una o più dipendenze delle quali si deve conoscere il nome e la versione.
- Se si vogliono attivare gli unit test, anche questa funzione dovrà essere configurata ed implementata.



## Introduzione al Framework Spring Boot

- Spring Boot è un **framework open source** di sviluppo di applicazioni in Java basato sul framework Spring e che semplifica notevolmente il processo di creazione e configurazione di applicazioni web, di servizi web e di applicazioni autonome.
- Spring Boot semplifica lo sviluppo di applicazioni Java fornendo un'**infrastruttura preconfigurata e automatizzata**, consentendo agli sviluppatori di concentrarsi sulla **business logic** invece di dover gestire complessità e configurazioni di basso livello.
- Il framework Spring Boot prevede un **sistema di avvio rapido (bootstrap)** che configura automaticamente l'ambiente in cui l'applicazione viene eseguita.
- Questo è un enorme vantaggio, poiché si può così **inizializzare l'applicazione in pochi secondi** e senza dover effettuare manualmente tutte le configurazioni necessarie.



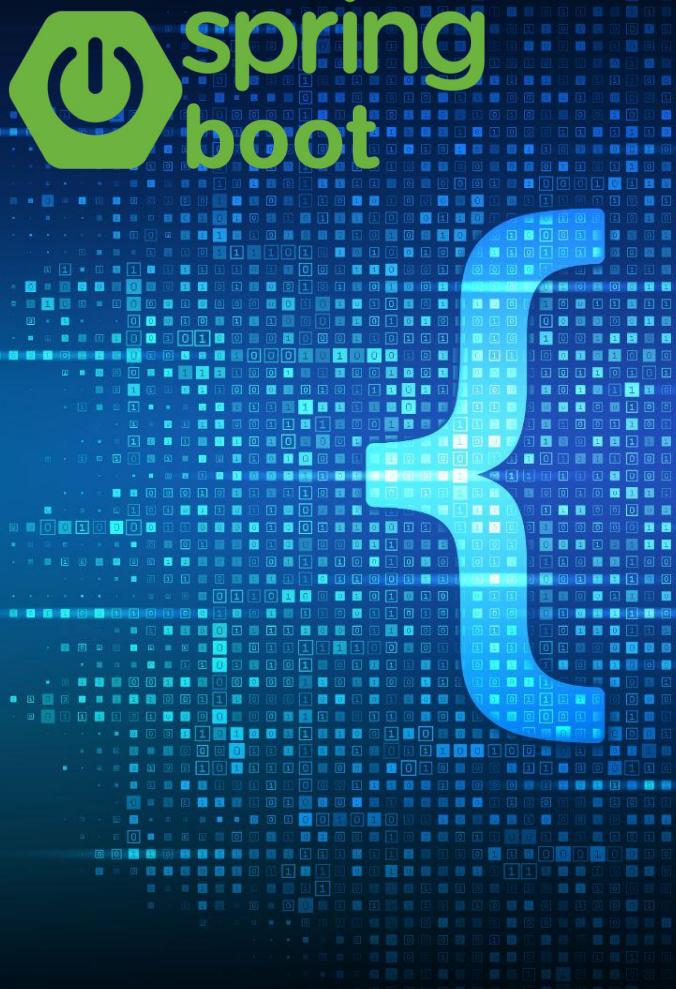
## Introduzione al Framework Spring Boot

- Spring Boot è un **framework open source** di sviluppo di applicazioni in Java basato sul framework Spring e che semplifica notevolmente il processo di creazione e configurazione di applicazioni web, di servizi web e di applicazioni autonome.
- Spring Boot semplifica lo sviluppo di applicazioni Java fornendo un'**infrastruttura preconfigurata e automatizzata**, consentendo agli sviluppatori di concentrarsi sulla **business logic** invece di dover gestire complessità e configurazioni di basso livello.
- Il framework Spring Boot prevede un **sistema di avvio rapido (bootstrap)** che configura automaticamente l'ambiente in cui l'applicazione viene eseguita.
- Questo è un enorme vantaggio, poiché si può così **inizializzare l'applicazione in pochi secondi** e senza dover effettuare manualmente tutte le configurazioni necessarie.



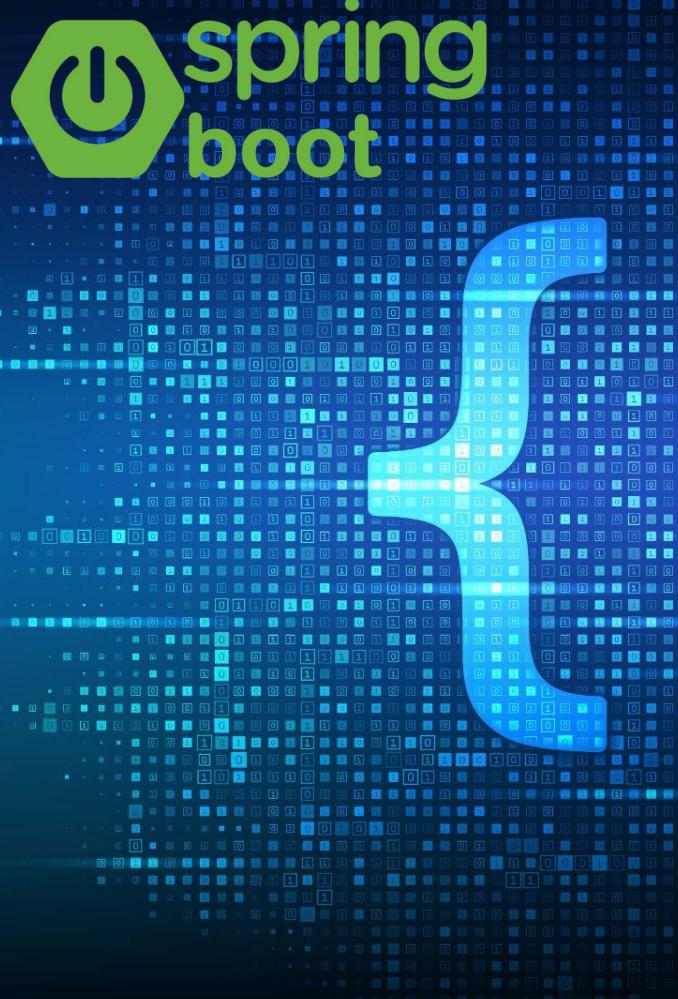
## Introduzione al Framework Spring Boot

- Inoltre, Spring Boot **consente la creazione di applicazioni standalone**, rendendo possibile l'esecuzione dell'applicazione senza la necessità di un server esterno.
- Spring Boot fornisce anche un **sistema di gestione delle dipendenze** (Maven o Gradle), che facilita il processo di creazione, test e distribuzione dell'applicazione.
- Spring Boot **prevede un'eccellente integrazione** con altre tecnologie utilizzate nel contesto delle applicazioni web, come ad esempio Thymeleaf per la gestione delle viste, **Spring Data** per l'accesso ai database, o **Spring Security** per la sicurezza delle applicazioni.



## Introduzione al Framework Spring Boot

- Concetti chiave dello Spring Boot:
- **Convenzione su configurazione:** Spring Boot adotta il principio della "convenzione su configurazione". Ciò significa che il framework fornisce delle impostazioni predefinite intelligenti e un'ampia configurazione automatica, permettendo agli sviluppatori di iniziare rapidamente con un'applicazione funzionante senza dover configurare manualmente molte opzioni.
- **Autoconfigurazione:** Spring Boot offre una potente funzionalità di autoconfigurazione. Basandosi sulle dipendenze presenti nel classpath dell'applicazione, Spring Boot può rilevare automaticamente quali funzionalità e librerie sono disponibili e configuralle di conseguenza. Questo riduce notevolmente la quantità di configurazione manuale richiesta.



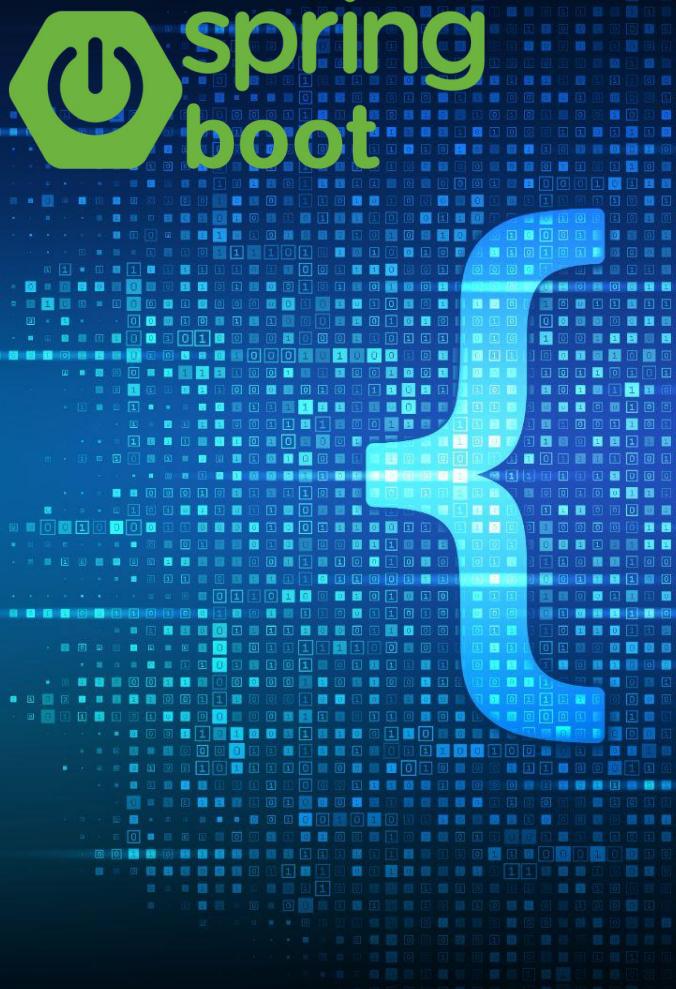
## Introduzione al Framework Spring Boot

- **Embedded Server:** Spring Boot fornisce un **server Web incorporato**, come ad esempio Tomcat o Jetty, che può essere eseguito direttamente dall'applicazione **senza dover configurare un server esterno**. Questo semplifica il processo di distribuzione dell'applicazione, in quanto tutto ciò che serve è un semplice file eseguibile JAR.
- **Spring Boot Starter:** Spring Boot Starter è un concetto chiave che semplifica la gestione delle dipendenze dell'applicazione. I Starter **sono una serie di dipendenze preconfigurate che vengono fornite in bundle** per supportare specifiche funzionalità, come ad esempio l'integrazione con database, la gestione delle sessioni, la sicurezza e molto altro. Aggiungendo un Starter al progetto, Spring Boot gestisce automaticamente la configurazione delle dipendenze correlate.



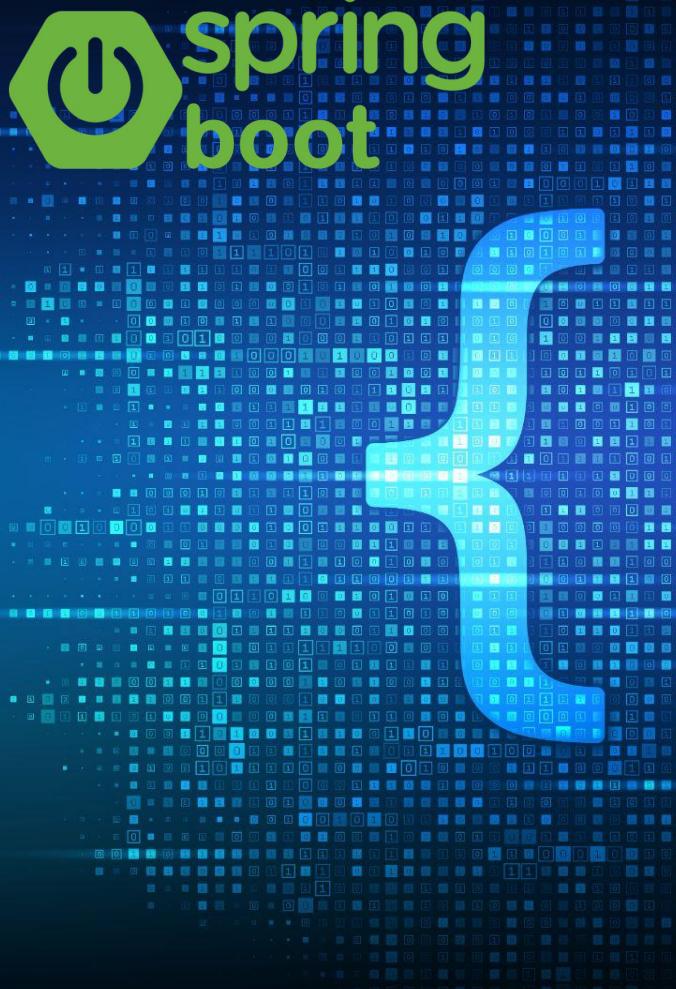
## Introduzione al Framework Spring Boot

- Actuator: Spring Boot Actuator fornisce funzionalità di monitoraggio e gestione dell'applicazione. Attraverso Actuator, è possibile ottenere informazioni sullo stato dell'applicazione, monitorare le metriche di performance, controllare e gestire gli endpoint e molto altro. Questo facilita il monitoraggio e la gestione delle applicazioni Spring Boot in produzione.
- In sintesi, Spring Boot è un framework Java altamente efficiente e conveniente per la creazione di applicazioni web altamente performanti e di qualità, che semplifica il processo di sviluppo ed evita le complessità della configurazione manuale.
- Inoltre, la sua vasta comunità e la sua ampia documentazione lo rendono uno strumento altamente accessibile e è considerato uno dei migliori frameworks disponibili per lo sviluppo di applicazioni Java.



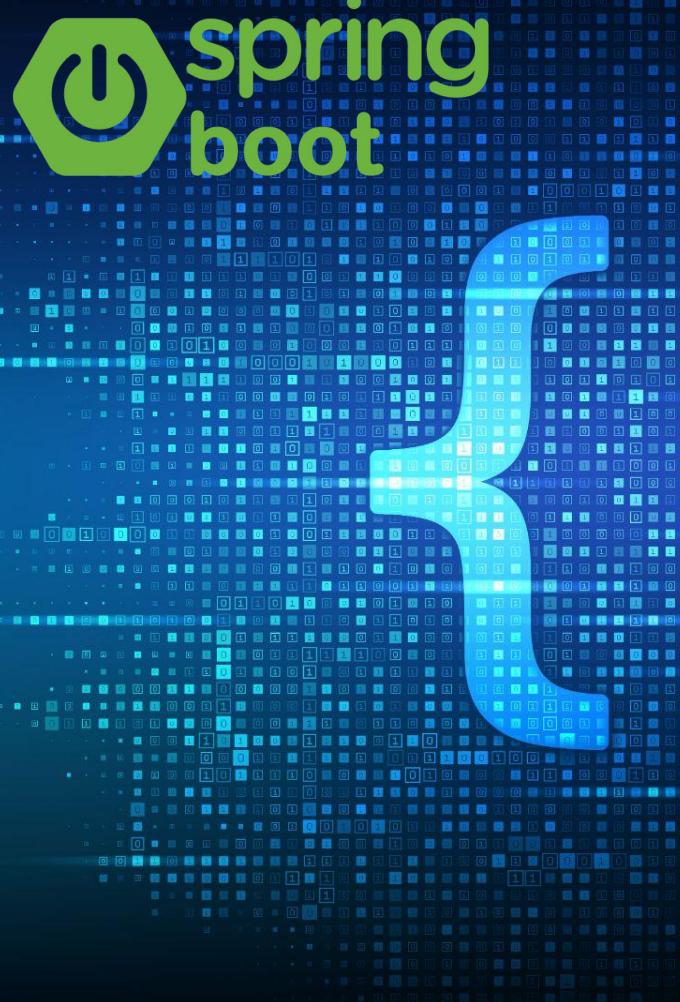
## Il modello MVC in Spring

- Il modello **MVC** (Model-View-Controller) in Spring è un design pattern che consente di organizzare e strutturare una applicazione web in modo modulare e separato per favorire la manutenibilità e la scalabilità del codice.
- Nel modello MVC di Spring, i tre componenti principali sono:
- **Model**: rappresenta la **business logic** dell'applicazione. Il model contiene i dati e le operazioni che possono essere eseguite su tali dati. Questi dati possono provenire da una base di dati, da un altro servizio esterno o da altre fonti di dati. Nel caso di Spring, il modello può essere rappresentato da classi di dominio annotate come **@Entity** o semplici classi di **POJO** (Plain Old Java Object).

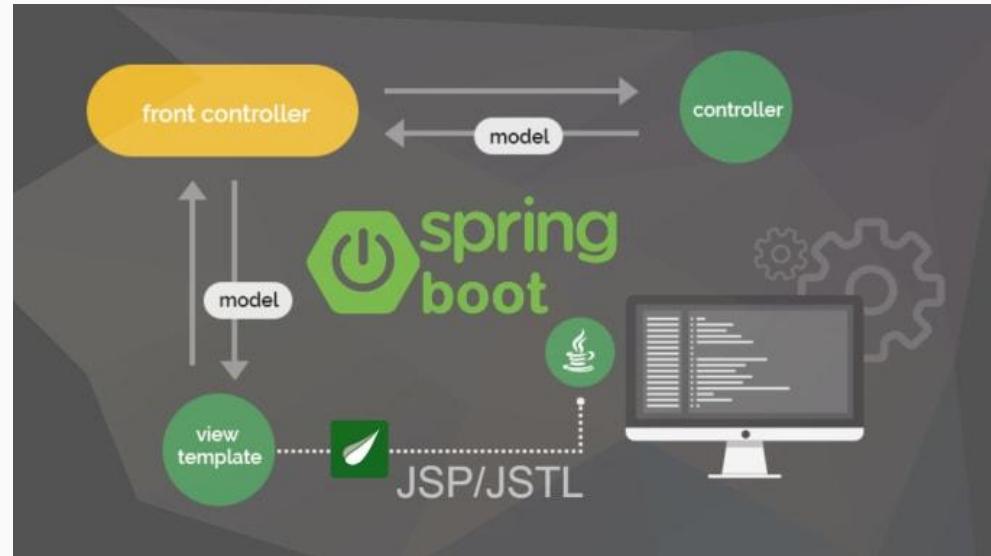


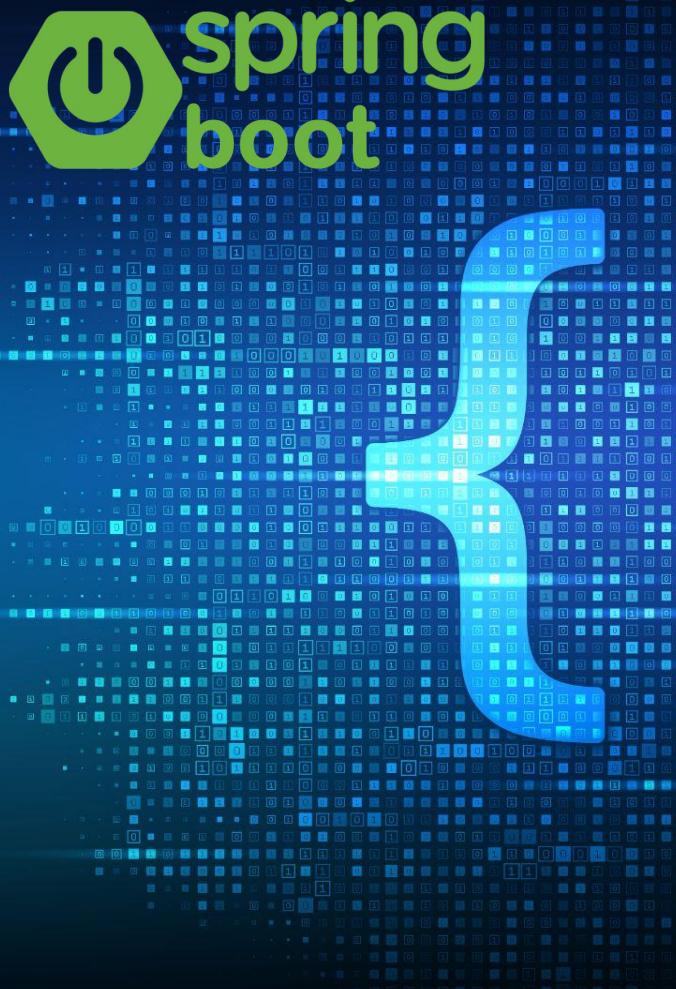
## Il modello MVC in Spring

- **View:** rappresenta l'interfaccia utente dell'applicazione. La view è responsabile per la **presentazione** dei dati al client. Nel caso di Spring, la view può essere rappresentata da template come ad esempio **JSP** (Java Server Pages) o **Thymeleaf**.
- **Controller:** funge da **intermediario** tra il **modello** e la **view**. Il controller riceve le richieste del client, interroga il modello per ottenere i dati necessari e li passa alla view corrispondente per la presentazione. Nel caso di Spring, i controller sono gestiti dal framework Spring MVC e possono essere annotati con **@Controller**.
- Quando una richiesta arriva al server, **il controller appropriato viene attivato per gestire la richiesta**. Il controller interagisce con il modello per ottenere i dati necessari e li passa alla vista per la presentazione. Una volta che la vista ha presentato i dati al cliente, il ciclo di richiesta-risposta è completato.



## Il modello MVC in Spring





## Introduzione alla classe Controller

- In Spring Boot, la classe **Controller** è una componente chiave per la creazione di servizi e web app in Spring Boot.
- Un controller **rappresenta la logica di gestione delle richieste e delle risposte HTTP**, consentendo di definire l'interfaccia di comunicazione tra il client e il server.
- La classe controller in Spring è una **classe Java** responsabile per la gestione delle richieste HTTP e per l'inoltro delle risposte al client. È una componente essenziale nell'architettura del pattern **MVC (Model-View-Controller)**, in cui il controller si occupa di comunicare con il model e con la view.
- Nella pratica, il compito di un **Controller** è quello di esporre una serie di endpoint HTTP (API REST) e di gestire le richieste in arrivo, definendo la business logic necessaria per soddisfare la richiesta.



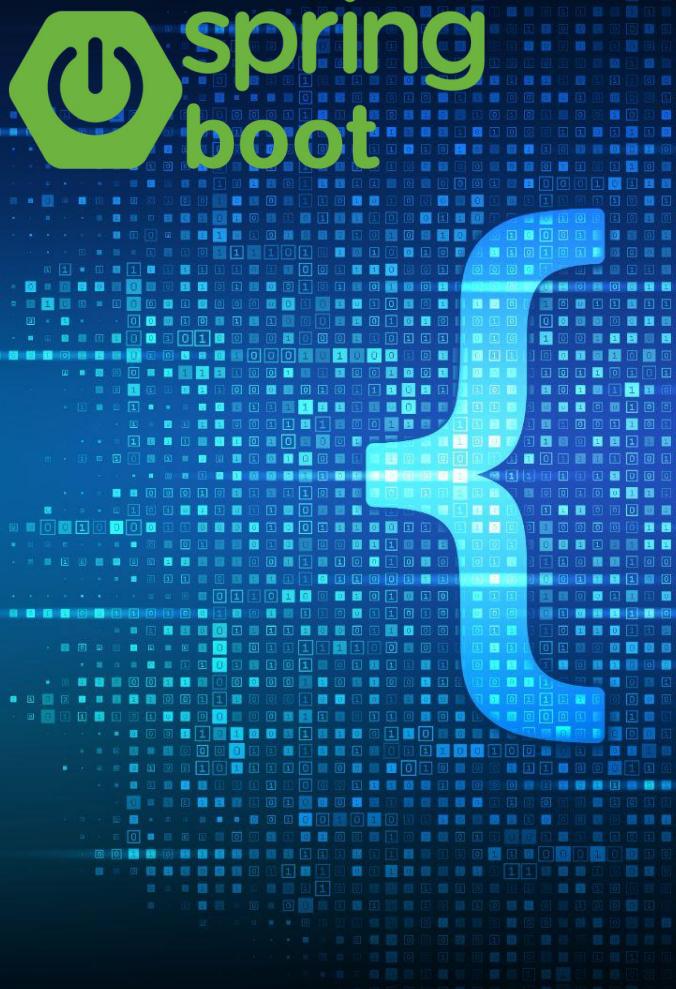
## Introduzione alla classe Controller

- In particolare una classe controller `composta:
- Annotazione `@Controller`: Per definire una classe come controller in Spring Boot, è necessario usare l'annotazione `@Controller`. Questa annotazione indica a Spring che la classe contiene la logica di gestione delle richieste e delle risposte HTTP.
- Metodi annotati con `@RequestMapping`: I metodi all'interno di un controller sono annotati con l'annotazione `@RequestMapping` o altre annotazioni come `@GetMapping`, `@PostMapping`, `@PutMapping`, `@DeleteMapping`, che indicano quale tipo di richiesta HTTP il metodo deve gestire. L'annotazione, inoltre, specifica l'URI relativo a cui il metodo deve rispondere.



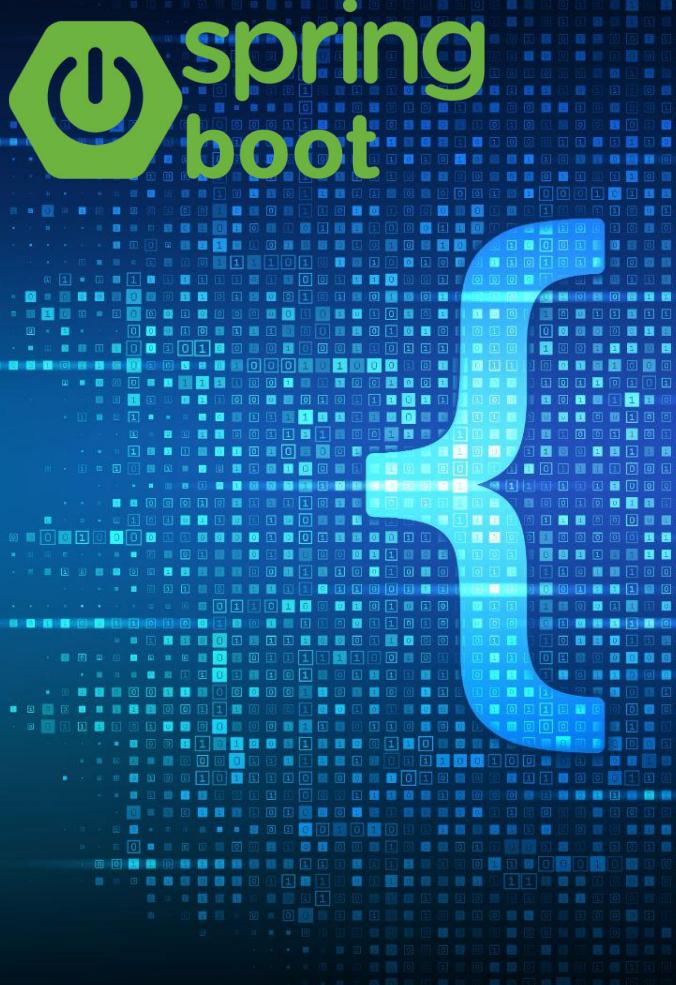
## Introduzione alla classe Controller

- **Parametri del metodo:** I parametri dei metodi possono essere utilizzati per accedere ai dati inviati nella richiesta HTTP, come i parametri della query, i parametri del percorso, i dati del corpo della richiesta, gli header, ecc. Questi parametri possono essere definiti utilizzando le annotazioni come `@RequestParam`, `@PathVariable`, `@RequestBody`, `@RequestHeader`, ecc.
- **Restituzione dei dati:** I metodi del controller possono restituire diversi tipi di dati come risposta HTTP, come oggetti Java, rappresentazioni JSON o XML, oppure viste (HTML). La restituzione dei dati può essere fatta utilizzando l'annotazione `@ResponseBody` per restituire dati direttamente nel corpo della risposta o restituire un oggetto `ModelAndView` per la generazione di una vista.



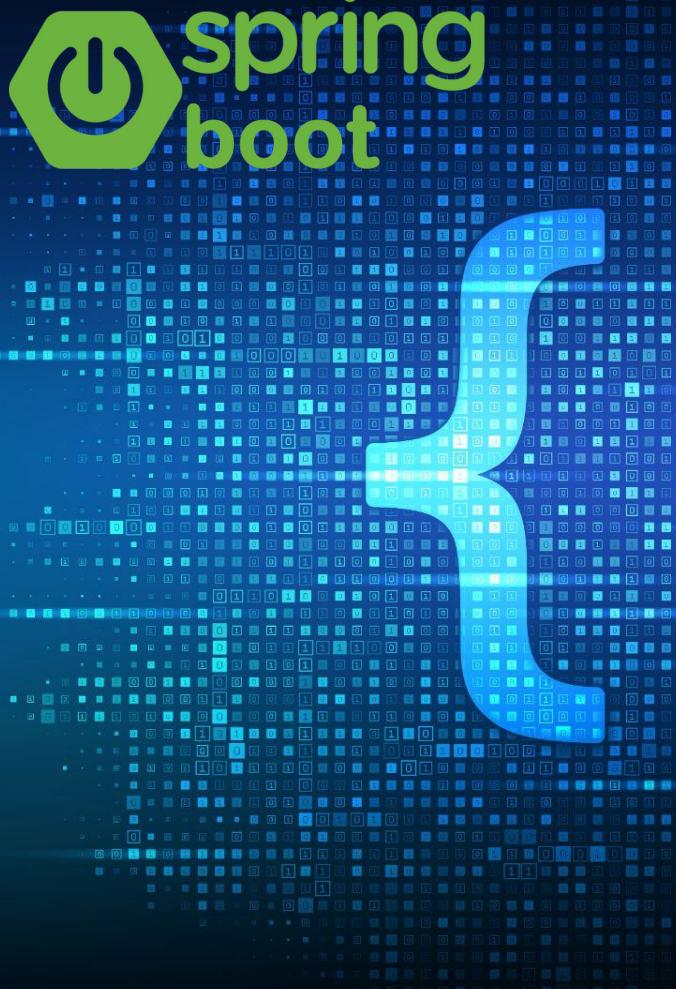
## Introduzione alla Java Server Page (JSP)

- La JSP (JavaServer Pages) è una tecnologia utilizzata per creare pagine web dinamiche. È una tecnologia che unisce HTML e Java per consentire l'inserimento di contenuti dinamici nelle pagine web.
- Le pagine JSP sono simili alle pagine HTML, ma contengono anche del codice Java all'interno dei tag delimitati da "<% %>". Questo codice Java viene eseguito dal server web quando la pagina viene richiesta dal client.
- Quando una pagina JSP viene richiesta, il server web interpreta e esegue il codice Java all'interno della pagina e quindi genera un HTML statico che viene inviato al client. Questo significa che il client non è a conoscenza dell'esistenza di JSP o del codice Java che è stato utilizzato per generare la pagina.



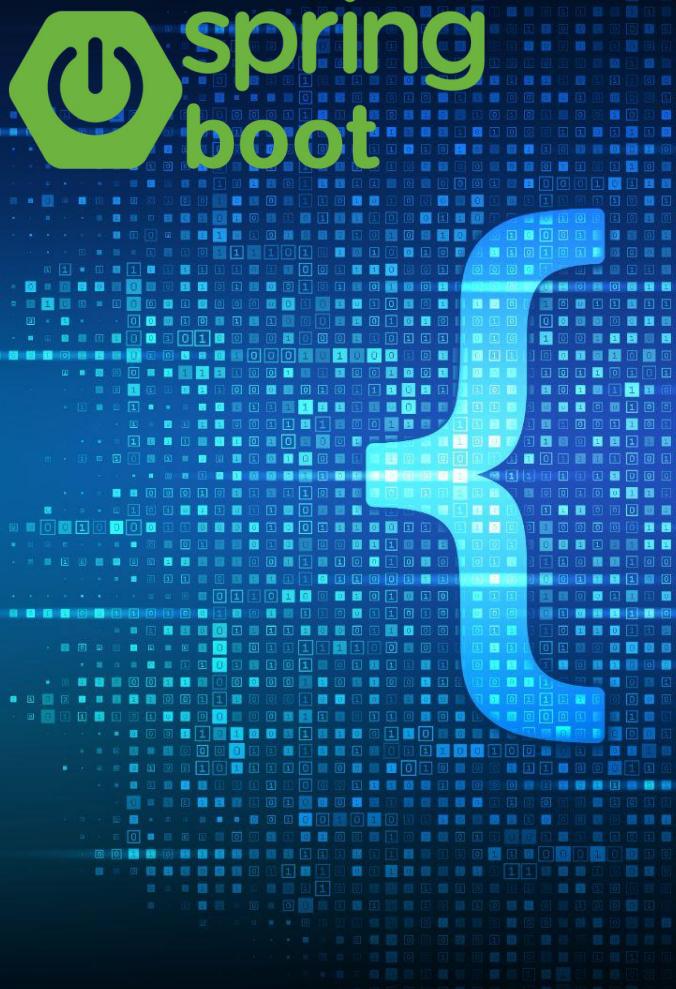
## La classe ModelMap

- Il **ModelMap** è una classe dell'ecosistema Spring Framework utilizzata per condividere dati tra i componenti del controller (ad esempio metodi di gestione delle richieste HTTP) e la vista (template o pagine frontend) all'interno di un'applicazione web. In altre parole, il **ModelMap** funge da contenitore per memorizzare dati che verranno visualizzati o utilizzati dalla vista.
- Sia **Model** che **ModelMap** sono classi utilizzate in Spring Framework per memorizzare dati da passare dalla parte del controller alla vista nelle applicazioni web.
- **ModelMap** è una classe concreta che estende le funzionalità di **Model** aggiungendo più metodi utili. Oltre al metodo `addAttribute()`, fornisce metodi come `addAllAttributes()`, `mergeAttributes()`, `removeAttribute()`, `containsAttribute()` e altri per manipolare i dati in modo più avanzato.



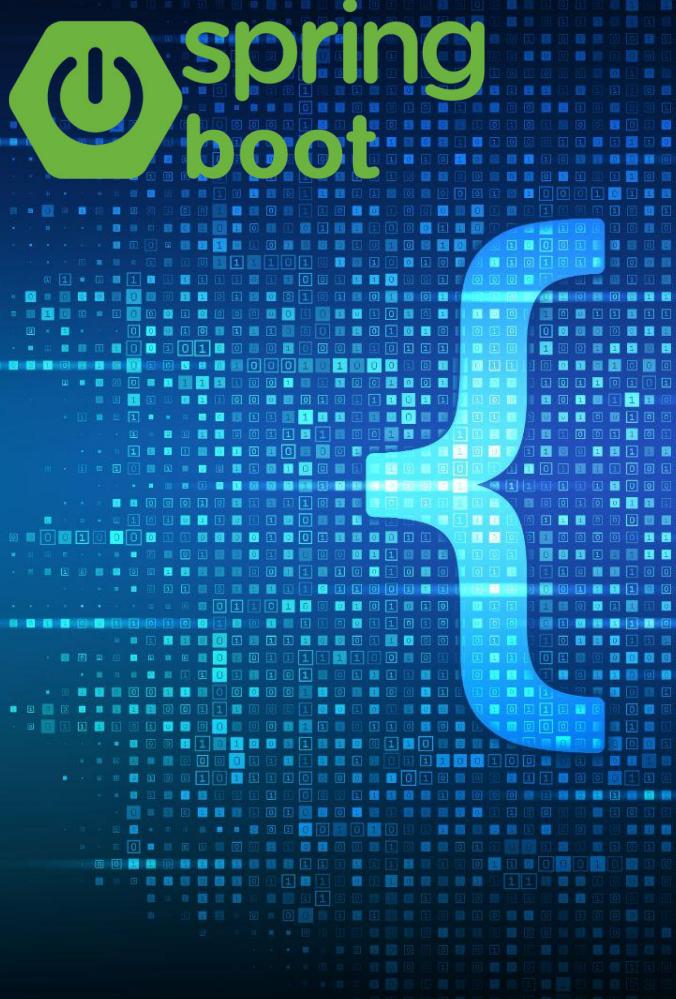
## La notazione @RequestParam

- La notazione `@RequestParam` è un'annotazione di Spring Framework utilizzata per recuperare i parametri di una richiesta HTTP (come quelli provenienti da un URL o un form HTML) e mapparli a parametri di un metodo del controller.
- Quando si definisce un metodo all'interno di una classe annotata come controller di Spring, è possibile utilizzare `@RequestParam` per dichiarare i parametri che si desidera estrarre dalla richiesta HTTP.
- Questa annotazione permette a Spring di leggere i parametri della richiesta e di passarli al metodo del controller.



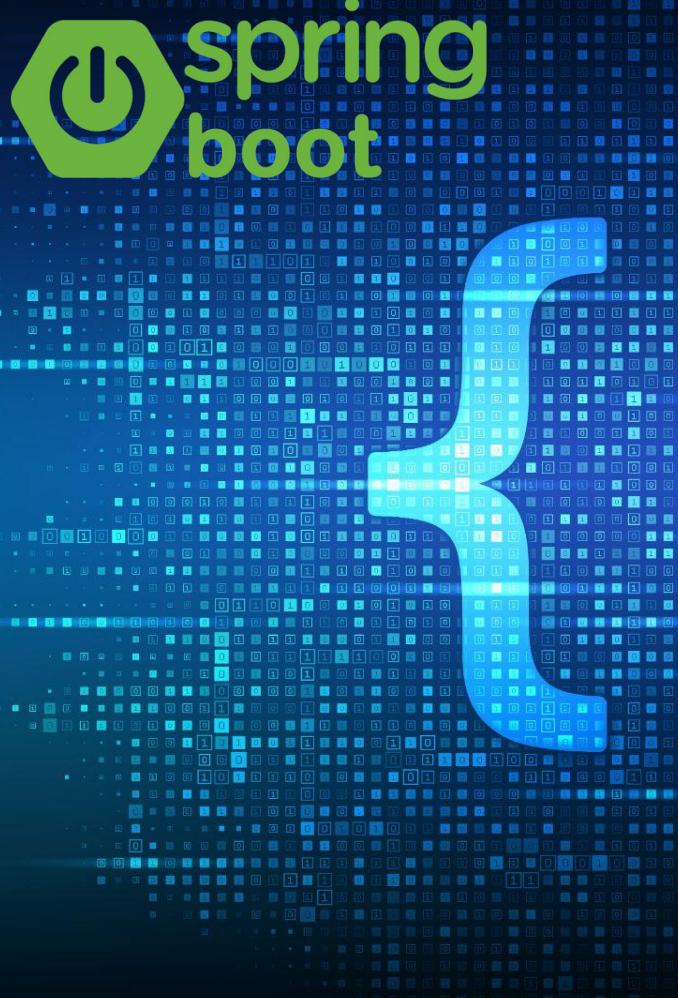
## La notazione @SessionAttributes

- L'annotazione `@SessionAttribute` è una delle annotazioni fornite dal framework Spring per gestire gli attributi di sessione all'interno dei controller delle applicazioni web.
- Quando si lavora con applicazioni web, spesso esistono dati che devono essere memorizzati tra più richieste HTTP per la stessa sessione utente. Questi dati possono essere memorizzati nell'oggetto `HttpSession`, che è una struttura di dati fornita dal container Servlet per gestire lo stato della sessione utente.
- L'annotazione `@SessionAttribute` viene utilizzata per collegare un attributo dell'oggetto `HttpSession` con un parametro di un metodo del controller. In questo modo, i dati possono essere automaticamente estratti dalla sessione e passati come argomenti al metodo del controller durante le richieste HTTP.



## Introduzione al JSTL

- JSTL, acronimo di **JavaServer Pages Standard Tag Library**, è una libreria di tag per semplificare e migliorare la creazione di pagine web dinamiche utilizzando JavaServer Pages (JSP) e Java Servlet. JSTL è un componente standard fornito da Java EE (Java Platform, Enterprise Edition) e fa parte della specifica JavaServer Pages (JSP).
- JSTL fornisce una serie di tag predefiniti che possono essere utilizzati direttamente all'interno di un file JSP per eseguire azioni comuni, come **iterazioni**, **condizioni**, **manipolazione di stringhe** e **accesso a oggetti Java**. Con l'uso dei tag JSTL, il codice JSP diventa più leggibile, manutenibile e modulare.



## Introduzione al JSTL

- Alcuni dei tag JSTL più utilizzati includono:
- **Core Tags:** Questi tag forniscono funzionalità essenziali, come iterazioni (`<c:forEach>`), condizioni (`<c:if>`, `<c:choose>`, `<c:when>`, `<c:otherwise>`), gestione delle variabili (`<c:set>`, `<c:remove>`), e altro ancora.
- **Format Tags:** Questi tag consentono la formattazione dei numeri e delle date (`<fmt:formatNumber>`, `<fmt:formatDate>`).
- **XML Tags:** Consentono l'elaborazione di documenti XML all'interno di un JSP (`<x:parse>`, `<x:transform>`).



## Introduzione al JSTL

- Per utilizzare JSTL, è necessario prima importare la libreria JSTL nel progetto tramite dipendenza nel file POM.XML.
- Sarà necessario, inoltre, inserire le dichiarazioni di importazione JSTL nel file JSP come le seguenti:

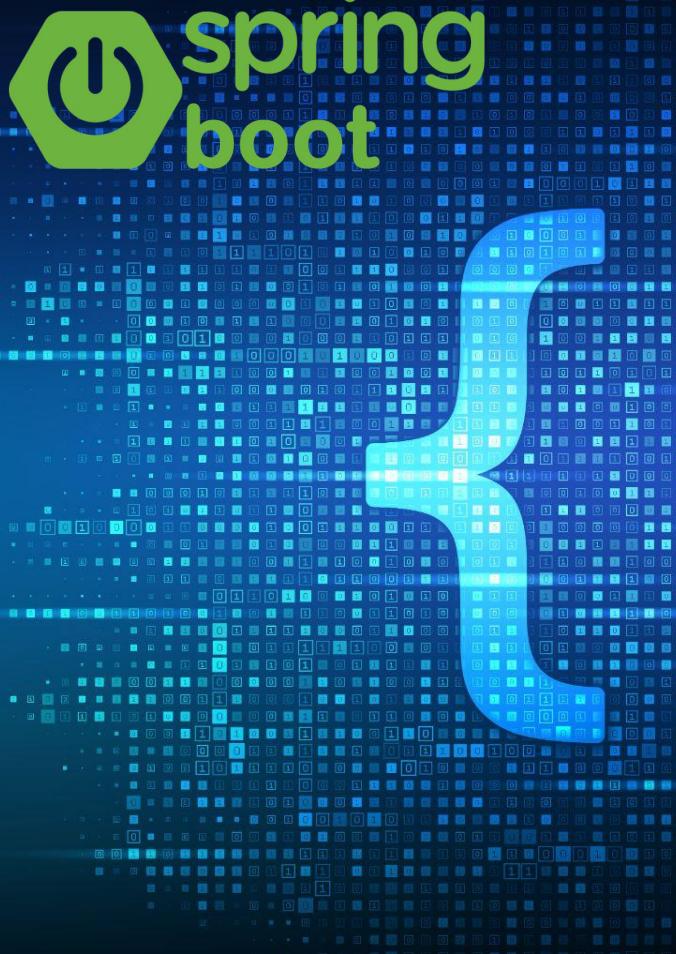
```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
```

- Una volta importata la libreria, sarà possibile utilizzare i tag JSTL nelle pagine JSP.



## Introduzione al DevTool

- In Spring Boot, i "DevTool" si riferisce a un insieme di funzionalità e strumenti che **migliorano l'esperienza di sviluppo** durante la creazione di applicazioni Spring Boot.
- Questi strumenti sono progettati per **accelerare il processo di sviluppo**, **semplificare la gestione dell'applicazione** e **fornire funzionalità utili** durante lo sviluppo.
- Le funzionalità principali fornite dai DevTool di Spring Boot includono:
  - **Riavvio automatico dell'applicazione:** Questa funzionalità permette di **ridurre il tempo di sviluppo** evitando di dover **avviare manualmente** l'applicazione ogni volta che si **apportano modifiche** al codice. Quando si utilizzano i DevTool, l'applicazione viene riavviata automaticamente quando vengono rilevati cambiamenti nel codice sorgente, consentendo di visualizzare immediatamente le modifiche senza dover interrompere e riavviare manualmente l'applicazione.



## Introduzione al DevTool

- **Hot Swapping:** In combinazione con il riavvio automatico, i DevTool di Spring Boot supportano il "hot swapping", che permette di ricaricare parti specifiche del codice senza riavviare l'intera applicazione. Questo è particolarmente utile per cambiamenti rapidi e leggeri, come modifiche al codice all'interno di un metodo o di una classe.
- **Proprietà di configurazione dinamica:** I DevTool consentono di modificare le proprietà di configurazione dell'applicazione senza dover ricompilare o riavviare l'applicazione. È possibile cambiare le proprietà di configurazione durante l'esecuzione dell'applicazione, ad esempio, impostando nuovi valori nel file `application.properties` o `application.yml`, e l'applicazione le rileverà automaticamente.
- **Gestione delle dipendenze:** I DevTool semplificano l'aggiornamento e la gestione delle dipendenze del progetto. Ad esempio, se si desidera utilizzare una versione più recente di una dipendenza, i DevTool consentono di farlo senza dover apportare modifiche manuali nel file di configurazione.



## Introduzione ai Database

- I database sono software che hanno lo scopo di memorizzare e gestire quantità più o meno elevate di dati in maniera organizzata e facilmente accessibile.
- Rispetto ad inserire i dati in file, i databases permettono di strutturare i dati in specifiche entità (tabelle, documenti), inserire indici in uno o più campi chiave al fine di minimizzare i tempi di ricerca del dato e impiegare **uno specifico linguaggio** (SQL) e l'uso di software creati ad hoc per ottenere e gestire i dati in maniera strutturata e flessibile
- I Database vengono usati per uno specifico scopo (base dati applicazione, data warehouse, cache, altro) e possono essere di diverse tipologie (relazionali, nosql, altro)



## I Database Relazionali

- Il concetto di database relazione risale alla fine degli anni 70 ed è tutt'ora il **tipo di database maggiormente utilizzato e sviluppato**.
- I database relazionali sono un tipo di database che **organizza e conserva dati in tabelle e relazioni**. Questo tipo di database si basa sul modello relazionale, che è una struttura composta da tabelle bidimensionali.
- Nel modello relazionale, i dati sono organizzati all'interno di **tabelle**, che sono **composte da righe e colonne**. Ogni riga rappresenta una singola istanza o **record dei dati**, mentre le colonne rappresentano gli **attributi o campi dei dati**. Ogni tabella ha una **chiave primaria**, che è un attributo univoco per identificare ogni record nella tabella.



## I Database Relazionali

- La relazione tra le tabelle è stabilita tramite chiavi esterne. Una chiave esterna in una tabella fa riferimento alla chiave primaria di un'altra tabella, creando una relazione tra i dati in entrambe le tabelle. Questo permette di collegare i dati in modo significativo e coerente.
- L'accesso ai dati all'interno di un database relazionale avviene tramite il linguaggio SQL. Il SQL permette di interrogare e manipolare i dati all'interno delle tabelle, eseguire operazioni di inserimento, aggiornamento e cancellazione, e creare ricerche complesse.



## Esempio di tabelle nel db relazionale

- Esempio di due tabelle con relativi dati, chiavi e relazioni.

Autori

	Id	Età	Sesso	Nome
1	19	46	F	Leigh Bardugo
2	20	35	F	Sarah J. Maas

Romanzi

	Id	ISBN	Titolo	Genere	IdAutore
1	39	B08KH1B786	Grishaverse - Tenebre e ossa	Fantasy	19
2	40	B08R5MD353	Grishaverse - Assedio e tempesta	Fantasy	19
3	41	B08XWV5SMR	Grishaverse - Rovina e ascesa	Fantasy	19
4	42	B07PB3LNY9	La corte di rose e spine	Fantasy	20
5	43	B07RX2BX6V	La corte di nebbia e furia	Fantasy	20
6	44	B07VWZYQLY	La corte di ali e rovina	Fantasy	20
7	45	B08VWMGH3N	La corte di fiamme e argento	Fantasy	20



## I Database NoSQL

- I database relazionali sono ideali quando i dati possono essere facilmente strutturati in schemi fissi.
- Quando si ha l'esigenza di avere maggiore flessibilità nella determinazione della struttura dei dati è necessario utilizzare i database No-SQL
- I database NoSQL (Not only SQL) sono sistemi di gestione dei dati che differiscono dai tradizionali database relazionali.
- Questi database sono progettati per gestire grandi quantità di dati non strutturati o semi-strutturati, che possono essere difficili da organizzare in un tradizionale schema tabellare.



## I Database NoSQL

- Alcuni delle principale caratteristiche dei database No-SQL sono:
- Struttura flessibile: A differenza dei database relazionali, i database NoSQL non richiedono uno schema predefinito. Ciò significa che è possibile modificare la struttura dei dati senza dover riscrivere l'intero schema del database.
- Scalabilità orizzontale: I database NoSQL sono progettati per gestire quantità enormi di dati in modo scalabile. Possono essere distribuiti su più server, consentendo un'elaborazione parallela dei dati e una maggiore capacità di archiviazione.
- Modello di dati diversi: I database NoSQL utilizzano una varietà di modelli di dati, tra cui database a grafo, chiavi-valore, colonne familiari e documenti. Ognuno di questi modelli è ottimizzato per diversi tipi di dati e applicazioni.



## I Database NoSQL

- Adatto per dati non strutturati: Con i database NoSQL, è possibile archiviare e gestire facilmente dati non strutturati o semi-strutturati, come documenti JSON, dati XML, grafici complessi, dati multimediali e altro ancora.
- Prestazioni elevate: I database NoSQL sono progettati per offrire elevate prestazioni su un'ampia gamma di carichi di lavoro. Possono gestire velocemente le operazioni di lettura e scrittura, riducendo il tempo di latenza e migliorando l'esperienza dell'utente finale.
- Alta disponibilità: I database NoSQL sono spesso progettati per garantire un'elevata disponibilità dei dati. Utilizzano la replica dei dati su più server, riducendo il rischio di perdita di dati in caso di guasti hardware o problemi di rete.



## I Database NoSQL

- Adatto per architetture moderne: I database NoSQL sono spesso utilizzati nell'ambito delle architetture moderne come le applicazioni web, mobile e cloud. Possono integrarsi facilmente con altre tecnologie come i servizi di cloud computing e possono adattarsi alle esigenze di scalabilità e flessibilità richieste da queste architetture.
- In sostanza, i database NoSQL offrono un'alternativa flessibile e scalabile ai tradizionali database relazionali. Sono progettati per gestire grandi quantità di dati non strutturati, offrendo elevate prestazioni e disponibilità.



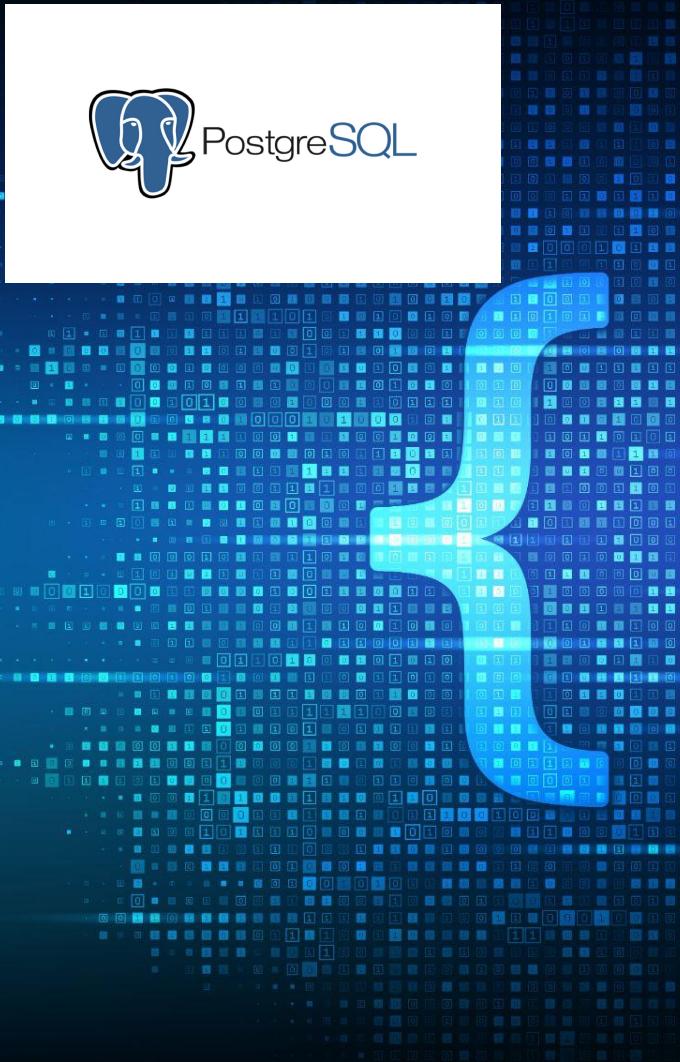
## Introduzione alle PostgreSQL

- PostgreSQL, anche noto come Postgres, è un sistema di gestione di database relazionali (RDBMS) open-source e gratuito che offre una vasta gamma di funzionalità e caratteristiche.
- Di seguito sono elencate alcune delle caratteristiche principali di PostgreSQL:
- Open Source: PostgreSQL è rilasciato sotto la licenza PostgreSQL, una licenza open source che permette un utilizzo gratuito, la modifica e la distribuzione del software.
- Architettura client-server: PostgreSQL utilizza un modello client-server, dove i client si connettono al server che gestisce effettivamente il database. Ciò consente la gestione centralizzata dei dati e la condivisione dei dati tra diversi client.



# Introduzione alle PostgreSQL

- **Supporto per dati strutturati e non strutturati:** PostgreSQL supporta sia **dati strutturati**, come tabelle e colonne, che **dati non strutturati**, come JSON e XML. Ciò offre la flessibilità di archiviare e gestire diversi tipi di dati.
- **Transazioni ACID:** PostgreSQL garantisce la conformità alle proprietà **ACID** (Atomicity, Consistency, Isolation, Durability) per le transazioni. Ciò significa che le transazioni sono affidabili, coerenti e isolate da altre transazioni.
- **Integrità dei dati:** PostgreSQL supporta vincoli per garantire l'integrità dei dati nel database. È possibile definire vincoli di chiave primaria, chiave esterna, unicità e altro ancora per garantire che i dati siano validi.



## Introduzione alle PostgreSQL

- **Linguaggio di query avanzato:** PostgreSQL supporta il linguaggio di query **SQL standard**, consentendo di eseguire query complesse sui dati. Inoltre, offre anche **estensioni al linguaggio SQL**, come le funzioni e i tipi definiti dall'utente.
- **Funzioni avanzate:** PostgreSQL offre un'ampia gamma di funzioni integrate, tra cui funzioni matematiche, funzioni di stringa, funzioni di data/ora e molto altro ancora. Inoltre, supporta anche la creazione di funzioni definite dall'utente per estendere le funzionalità del database.
- **Scalabilità:** PostgreSQL è in grado di gestire **grandi volumi di dati e di scalare in modo efficiente**. Supporta la replica dei dati per consentire la distribuzione dei carichi di lavoro su più server.



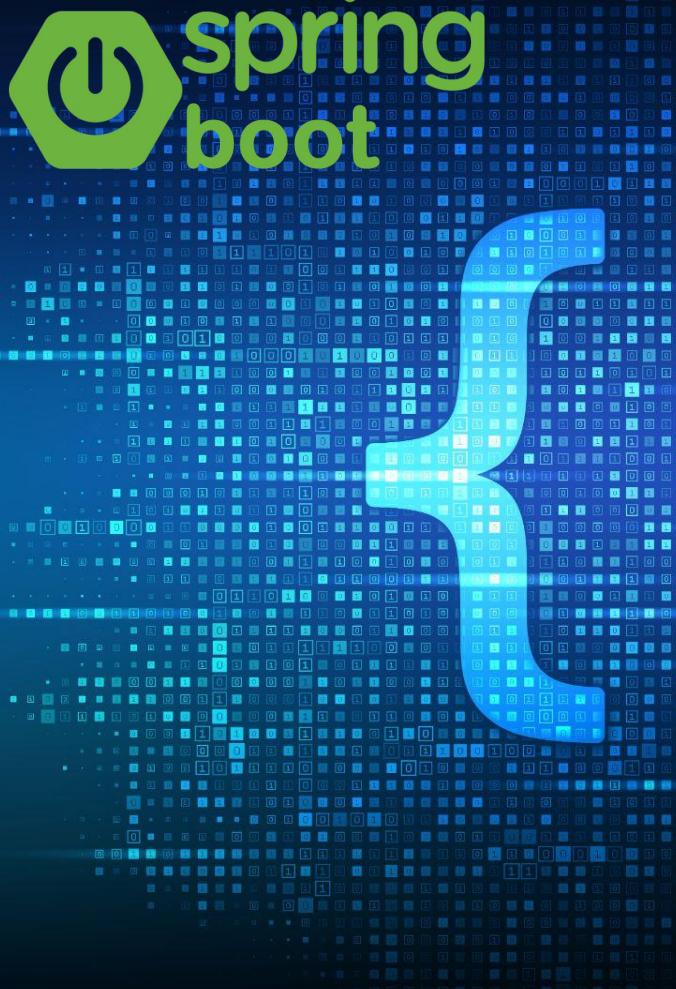
# Introduzione alle PostgreSQL

- **Sicurezza dei dati:** PostgreSQL offre una robusta sicurezza dei dati con autenticazione e autorizzazione basate su ruoli. I privilegi possono essere concessi o revocati in modo granulare a utenti e ruoli specifici per garantire la protezione dei dati sensibili.
- **Estensibilità:** PostgreSQL è altamente estensibile grazie al supporto per le estensioni. Gli sviluppatori possono creare e installare estensioni per aggiungere nuove funzionalità al database.
- **Supporto per Indici Avanzati:** PostgreSQL offre una varietà di tipi di indici, tra cui B-tree, GiST, GIN, SP-GiST e altri. Questi indici permettono di migliorare le prestazioni delle query su grandi dataset.
- **Replicazione:** PostgreSQL supporta la replicazione dei dati sia master-slave che multi-master, permettendo la creazione di ambienti di alta disponibilità e bilanciamento del carico.



## Introduzione alle PostgreSQL

- **Compatibilità:** PostgreSQL è compatibile con molti linguaggi di programmazione e framework, rendendolo una scelta popolare per lo sviluppo di applicazioni web e aziendali.
- **Pluralità dei linguaggi di programmazione:** È possibile scrivere funzioni server-side in diversi linguaggi come SQL, PL/pgSQL, Python, Java, Ruby, e molti altri.
- **Gestione avanzata delle transazioni:** PostgreSQL offre funzionalità avanzate per la gestione delle transazioni, tra cui transazioni annidate, transazioni salvabili e gestione delle transazioni distribuite.
- **Comunità attiva e supporto:** PostgreSQL ha una vasta comunità di sviluppatori che forniscono supporto e continuano a migliorare il software. Ci sono innumerevoli risorse online, documentazione e forum di discussione disponibili per risolvere i problemi e ottenere assistenza.



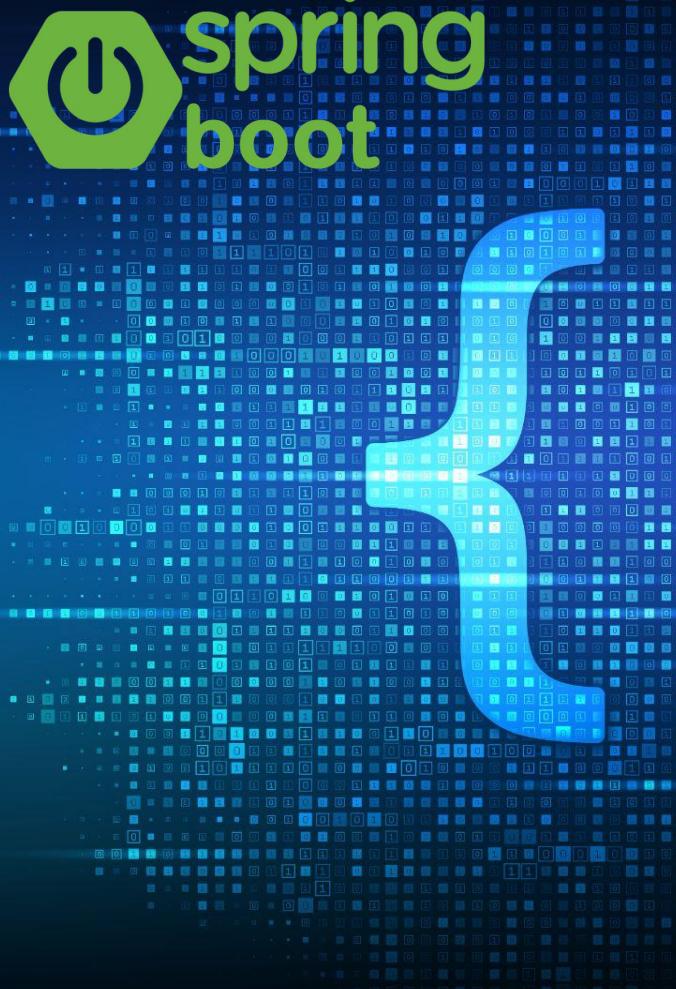
## Introduzione alle classi Entity

- Nello sviluppo di applicazioni basate su database in Spring Boot, le classi Entity rappresentano gli oggetti persistenti che vengono memorizzati nel database.
- Le classi Entity sono classi Java che mappano oggetti ricavati dal database o da altre fonti di dati.
- Nello specifico di Spring Boot, le classi Entity mappano i record di una tabella di un database relazionale in oggetti Java.
- Nel dettaglio in una classe entity possiamo avere:
- Annotazione `@Entity`: Per dichiarare una classe come Entity in Spring Boot, è necessario annotarla con l'annotazione `@Entity`. Questa annotazione indica a Spring che la classe rappresenta una tabella nel database.



## Introduzione alle classi Entity

- **Annotazione @Table:** L'annotazione `@Table` può essere utilizzata per specificare il nome della tabella a cui l'entità è associata nel database. Se il nome della classe coincide con il nome della tabella, questa annotazione può essere omessa.
- **Attributi dell'Entity:** Gli attributi della classe Entity rappresentano le colonne della tabella nel database. Ogni attributo deve essere annotato con l'annotazione appropriata per indicare il suo mapping al database. Ad esempio, l'annotazione `@Id` viene utilizzata per identificare un attributo come chiave primaria, mentre `@Column` viene utilizzata per specificare il nome della colonna e le proprietà di mappatura.
- **Getter e Setter:** Le classi Entity devono fornire metodi getter e setter per gli attributi, in modo che Spring possa accedere e modificare i dati delle entità. Possono essere autogenerati con l'uso del tool `Lombok`.



## Introduzione alle classi Entity

- **Relazioni tra Entity:** Le classi Entity possono avere relazioni tra loro, come relazioni uno-a-uno, uno-a-molti o molti-a-molti. Per gestire queste relazioni, vengono utilizzate annotazioni come `@OneToOne`, `@OneToMany`, `@ManyToOne` e `@ManyToMany`. Queste annotazioni definiscono il tipo di relazione e specificano le proprietà di mappatura tra le entità coinvolte.
- Nella definizione delle classi Entity, è possibile specificare varie informazioni aggiuntive come la chiave primaria, le relazioni con altre tabelle, la lunghezza massima dei campi, se sono richiesti o meno e altre informazioni di validità dei dati per i campi.
- Le classi Entity in Spring Boot consentono di mappare oggetti Java direttamente alle tabelle del database e di utilizzare le funzionalità ORM (Object-Relational Mapping) fornite da Spring per accedere e manipolare i dati nel database in modo semplice e intuitivo.



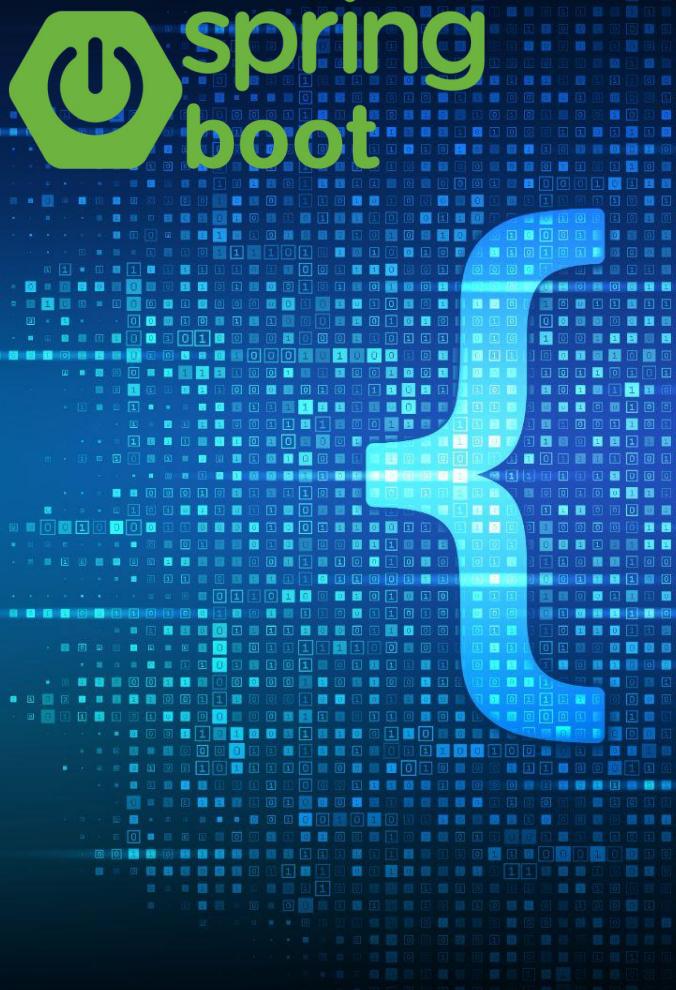
## Introduzione alle relazione tra Entity

- In Spring Boot, è possibile stabilire **relazioni** tra le classi di entità utilizzando le annotazioni fornite dal framework di persistenza **JPA** (Java Persistence API).
- Le relazioni possono essere di **diversi tipi**, come ad esempio **uno a uno**, **uno a molti** o **molti a molti**. Vediamo nel dettaglio come definire queste relazioni utilizzando esempi concreti.
- La notazione **@OneToOne** di Spring Boot è utilizzata per **stabilire una relazione uno-a-molti** tra due entità nel contesto di un'implementazione JPA (Java Persistence API).
- Questa annotazione **definisce un'associazione** in cui **un'entità ha una relazione "uno" con un'altra entità che ha una relazione "molti"**.
- Nel dettaglio le fasi da porre in essere sono:



## Introduzione alle relazione tra Entity

- Definizione delle entità:
- Si hanno due entità coinvolte nella relazione uno-a-molti: l'entità "uno" (solitamente chiamata entità principale) e l'entità "molti" (solitamente chiamata entità figlia).
- Nell'entità principale, si utilizza la notazione `@OneToMany` per definire la relazione. Questa notazione indica che un'istanza dell'entità principale può essere associata a molteplici istanze dell'entità figlia.
- Nell'entità figlia, si utilizza la notazione `@ManyToOne` per definire la relazione inversa. Questa notazione indica che molteplici istanze dell'entità figlia possono essere associate a un'unica istanza dell'entità principale.



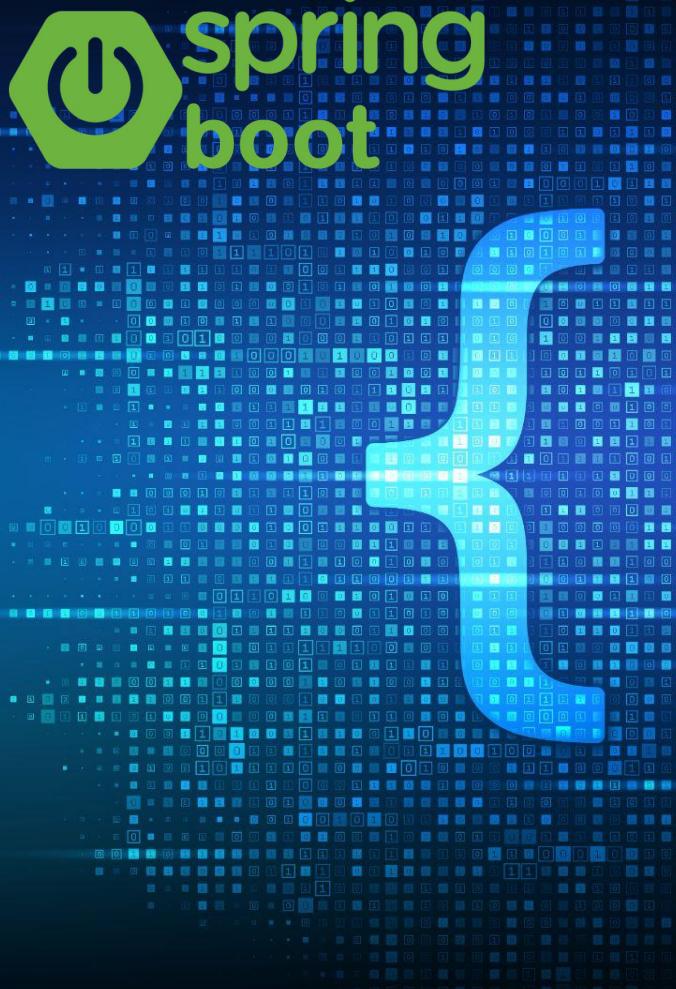
## Introduzione alle relazione tra Entity

- Parametri della notazione @OneToMany:
  - **mappedBy**: Questo parametro specifica il nome del campo nella classe figlia che mappa la relazione. Viene utilizzato per creare il collegamento tra le due entità. Il valore di mappedBy dovrebbe corrispondere al nome del campo nella classe figlia che rappresenta la relazione inversa con l'entità principale.
  - **cascade**: Questo parametro specifica le operazioni di cascata che devono essere applicate alla relazione. Ad esempio, se si desidera che le operazioni di salvataggio o eliminazione sulla classe principale si propaghino automaticamente alla classe figlia, si può specificare **CCascadeType.ALL** o un'altra combinazione appropriata di opzioni di cascata.
- Altri parametri opzionali includono **fetch** per specificare il tipo di caricamento delle entità figlie, **orphanRemoval** per rimuovere automaticamente le entità figlie orfane e altri parametri di configurazione specifici.



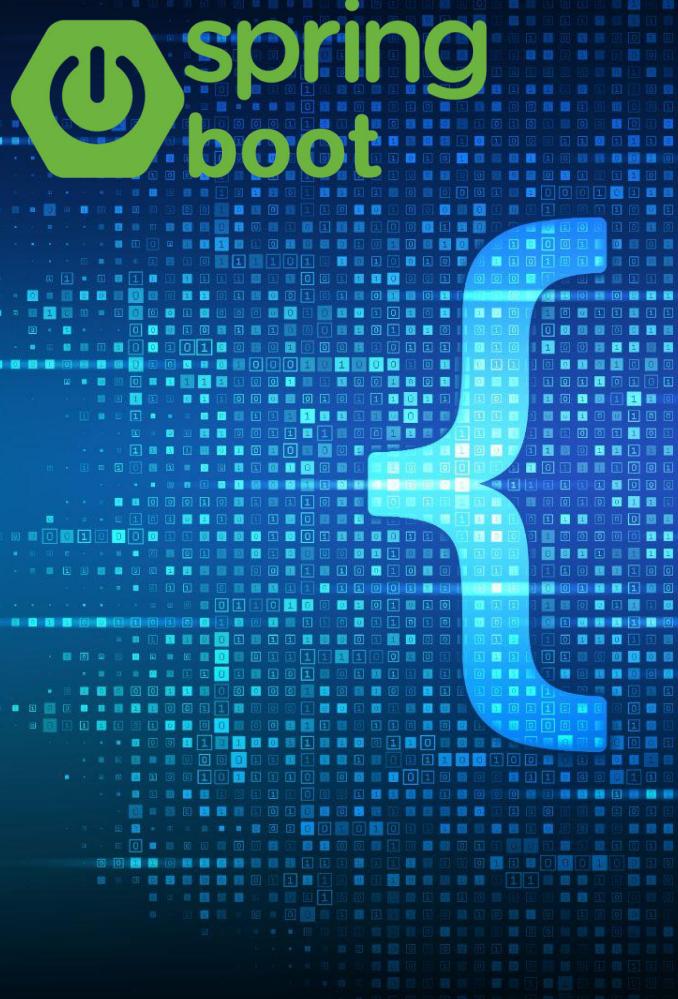
## Approfondimento parametro Fetch

- Quando si utilizza `fetch = FetchType.LAZY`, l'associazione viene considerata come un "proxy" o "riferimento" all'entità associata. L'associazione non viene caricata immediatamente quando si recupera l'entità principale dal database.
- L'associazione viene caricata solo quando si accede effettivamente ad essa tramite un'operazione di accesso (ad esempio, chiamando un metodo getter per l'associazione).
- Questo approccio consente di ridurre il traffico di rete e migliorare le prestazioni dell'applicazione, poiché solo le associazioni necessarie vengono recuperate dal database.



## Approfondimento parametro Fetch

- Quando si utilizza `fetch = FetchType.EAGER`, l'associazione viene caricata immediatamente insieme all'entità principale quando viene recuperata dal database.
- Ciò significa che tutte le associazioni contrassegnate come EAGER vengono recuperate anche se non vengono effettivamente utilizzate nel contesto corrente. Questo può causare un carico aggiuntivo sulla rete e sulle risorse del database, specialmente se ci sono molte associazioni o se l'associazione stessa è complessa o contiene molti dati.
- La scelta tra il caricamento Lazy ed Eager dipende dalle esigenze specifiche dell'applicazione.
- Di solito, il caricamento Lazy viene preferito quando si tratta di associazioni grandi o complesse che potrebbero non essere necessarie in tutte le operazioni, o quando si vuole ottimizzare le prestazioni evitando di recuperare dati inutilizzati.



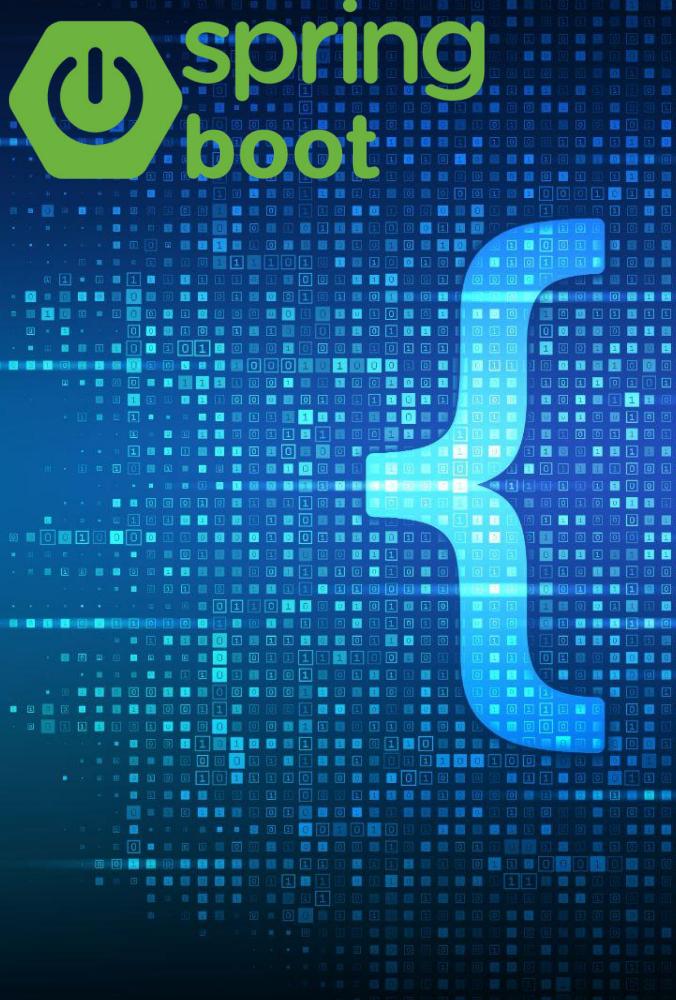
## Approfondimento parametro Fetch

- D'altra parte, il caricamento Eager viene utilizzato quando si ha la certezza che l'associazione sarà sempre necessaria o quando si desidera semplificare l'accesso ai dati associati senza dover eseguire query aggiuntive.



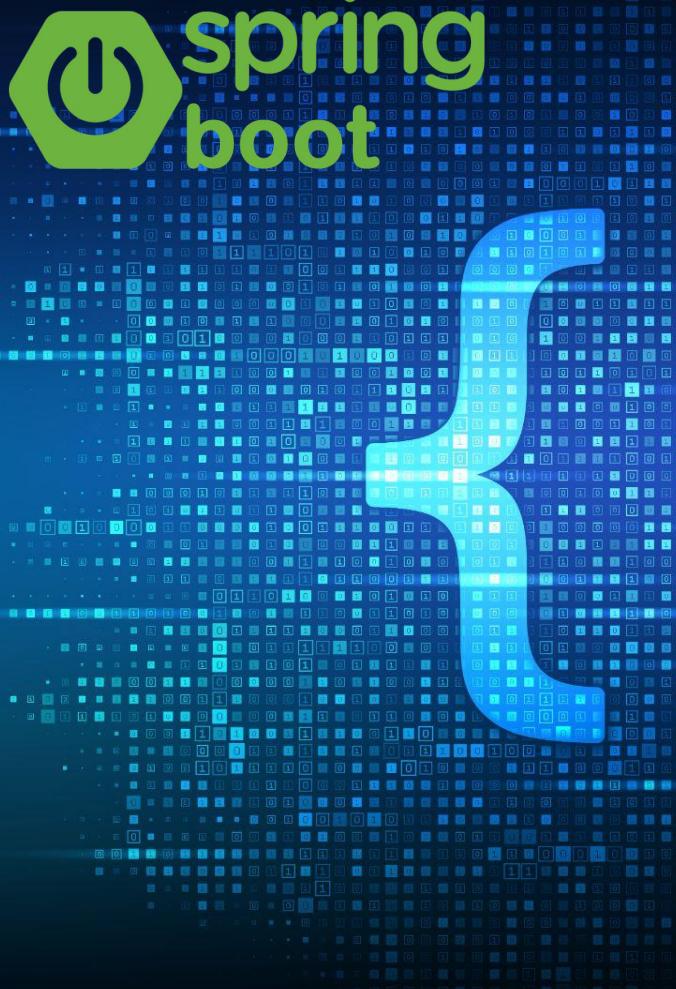
## Approfondimento parametro Cascade

- Il parametro **cascade** è un'opzione usata per specificare il comportamento di cascata delle operazioni di persistenza su un'entità padre e le sue entità figlie correlate.
- Indicando il parametro **cascade** su una relazione tra entità, è possibile specificare che **un'operazione di persistenza**, come ad esempio l'inserimento o l'aggiornamento, **deve essere propagata automaticamente anche alle entità collegate**.
- Nel contesto di Spring Boot, il parametro **cascade** viene spesso utilizzato in combinazione con le annotazioni di persistenza JPA, come ad esempio **@OneToMany**, **@ManyToOne** o **@ManyToMany**, per definire il comportamento di cascata delle operazioni.



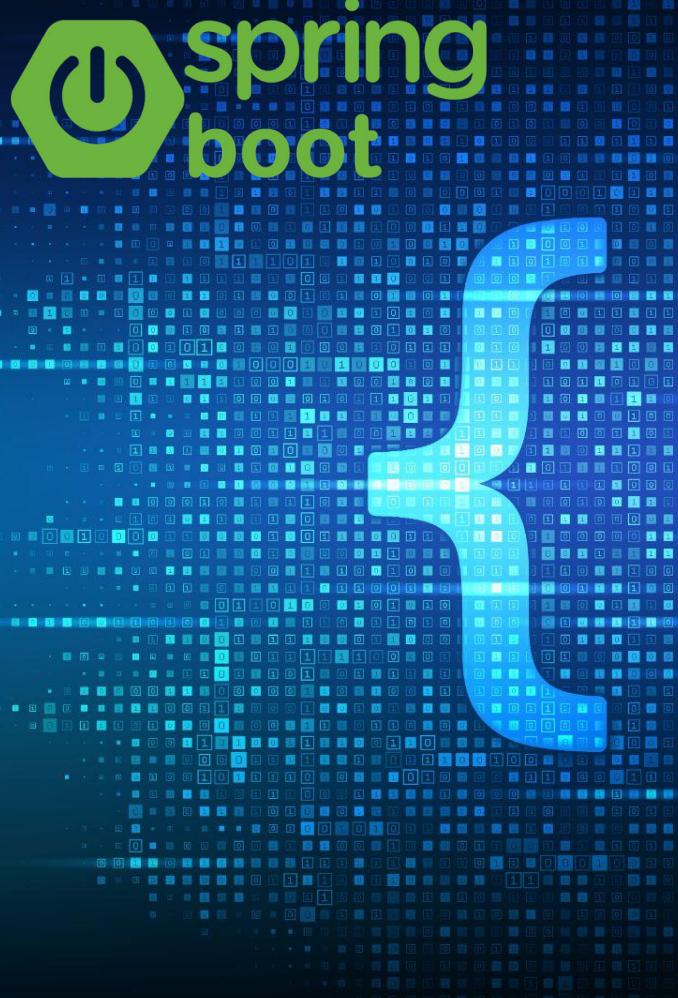
## Approfondimento parametro Cascade

- Ecco alcuni dei valori comuni che possono essere utilizzati per il parametro cascade:
- **CascadeType.ALL**: Questo valore indica che tutte le operazioni di persistenza (inserimento, aggiornamento, eliminazione) eseguite sull'entità padre devono essere propagate automaticamente alle entità figlie correlate.
- **CascadeType.PERSIST**: Questo valore indica che l'operazione di persistenza (inserimento) eseguita sull'entità padre deve essere propagata alle entità figlie correlate.
- **CascadeType.MERGE**: Questo valore indica che l'operazione di merge (aggiornamento) eseguita sull'entità padre deve essere propagata alle entità figlie correlate.
- **CascadeType.REMOVE**: Questo valore indica che l'operazione di rimozione eseguita sull'entità padre deve essere propagata alle entità figlie correlate.



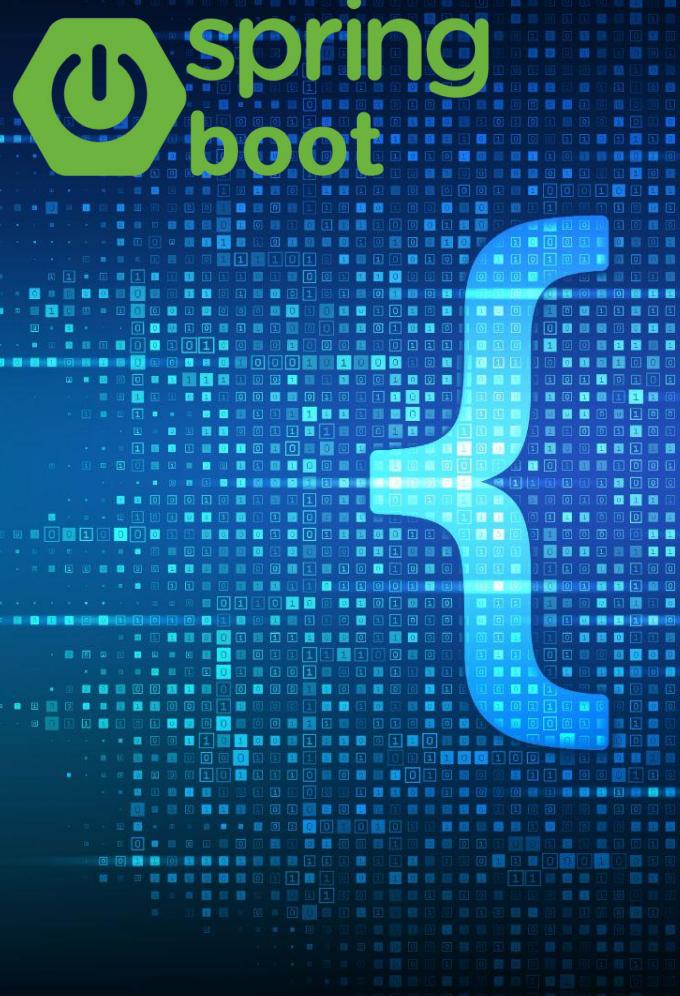
## Approfondimento parametro Cascade

- **CascadeType.REFRESH**: Questo valore indica che l'operazione di refresh (ricaricamento) eseguita sull'entità padre deve essere propagata alle entità figlie correlate.
- **CascadeType.DETACH**: Questo valore indica che l'operazione di detach (distacco) eseguita sull'entità padre deve essere propagata alle entità figlie correlate.
- **CascadeType.REPLICATE**: Questo valore indica che l'operazione di replicazione eseguita sull'entità padre deve essere propagata alle entità figlie correlate.
- Questi sono solo alcuni dei valori comuni per il parametro cascade. È possibile specificare **più valori separati da virgola** per indicare combinazioni di comportamenti di cascata desiderati.



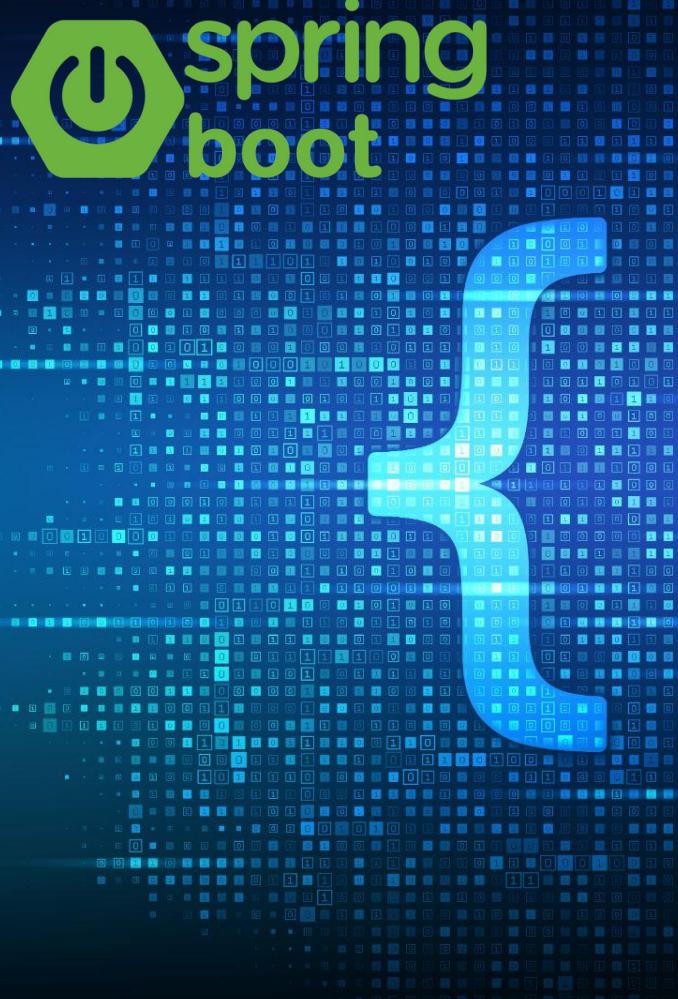
## Introduzione alle relazione tra Entity

- La notazione `@OneToOne` di Spring Boot viene utilizzata per stabilire una relazione uno a uno tra due entità.
- Questa annotazione viene utilizzata insieme alle annotazioni fornite da JPA (Java Persistence API) per definire la mappatura delle relazioni e le colonne delle chiavi esterne.
- Definizione dell'annotazione `@OneToOne`:
- L'annotazione `@OneToOne` viene applicata a un campo o a un metodo getter nella classe di entità che rappresenta il lato proprietario della relazione.
- Questa annotazione indica che la relazione è uno a uno tra l'entità proprietaria e un'altra entità.
- L'annotazione può essere utilizzata con o senza l'annotazione `@JoinColumn`.



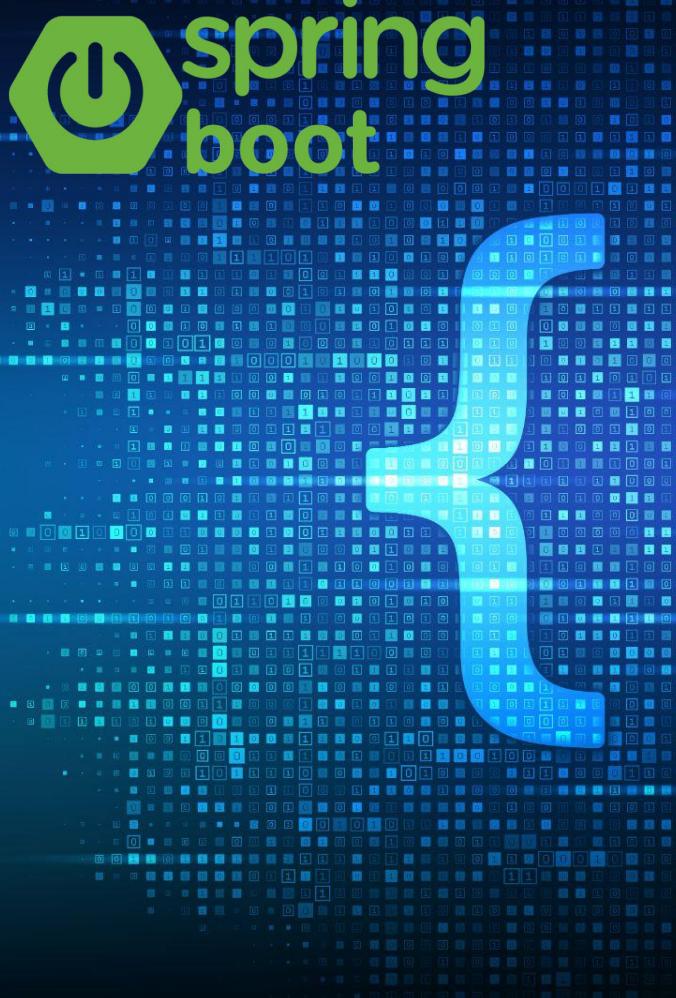
## Introduzione alle relazioni tra Entity

- Utilizzo di `@OneToOne` con `@JoinColumn`:
- Quando si utilizza `@OneToOne` con `@JoinColumn`, è possibile specificare i dettagli sulla colonna della chiave esterna.
- L'annotazione `@JoinColumn` viene utilizzata per specificare il nome della colonna della chiave esterna nella tabella dell'entità proprietaria.



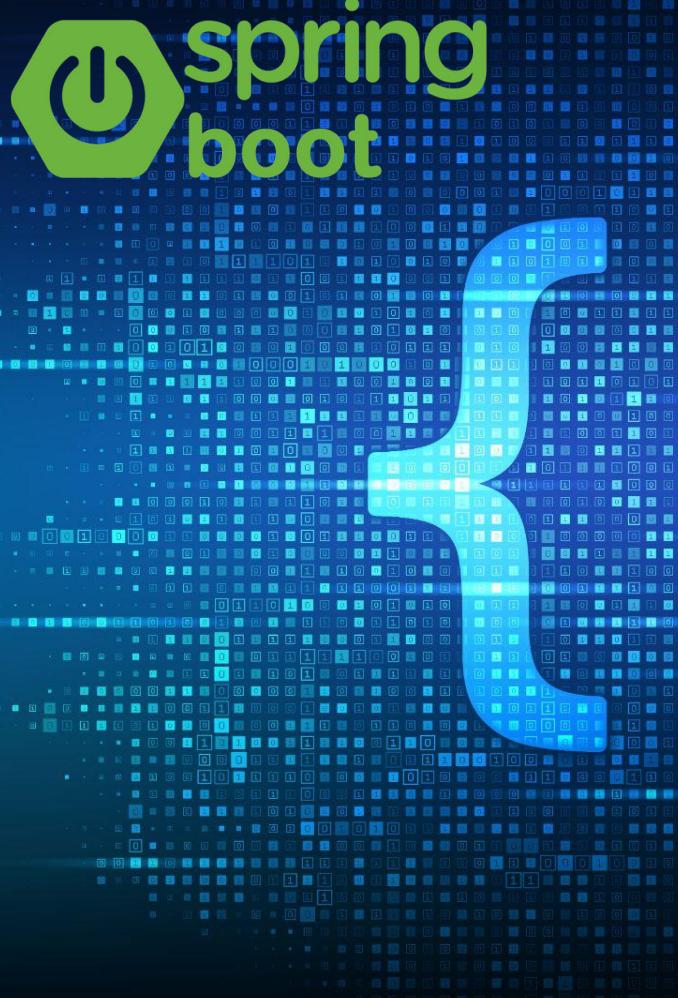
## Introduzione al Lombok

- Lombok è una tool Java che semplifica lo sviluppo delle applicazioni **gestendo automaticamente la generazione di codice boilerplate**, come getter, setter, costruttori, metodi equals e hashCode, toString e altri metodi comuni.
- Spring Boot è compatibile con Lombok e può sfruttarne le funzionalità **per ridurre la quantità di codice ripetitivo** che deve essere scritto manualmente.
- Lombok fornisce una serie di annotazioni che possono essere applicate alle classi per generare automaticamente il codice corrispondente. Alcune delle annotazioni più comuni includono:
  - **@Getter e @Setter:** Generano automaticamente i metodi getter e setter per i campi della classe.



## Introduzione al Lombok

- **@NoArgsConstructor**: Genera automaticamente un costruttore senza argomenti.
- **@AllArgsConstructor**: Genera automaticamente un costruttore con tutti gli argomenti.
- **@Data**: Combina **@Getter**, **@Setter**, **@ToString**, **@EqualsAndHashCode** e **@RequiredArgsConstructor** in un'unica annotazione.
- **@Builder**: Genera un builder pattern per la creazione di oggetti immutabili.
- Molte altre annotazioni sono disponibili per scopi specifici.
- <https://projectlombok.org/features/>



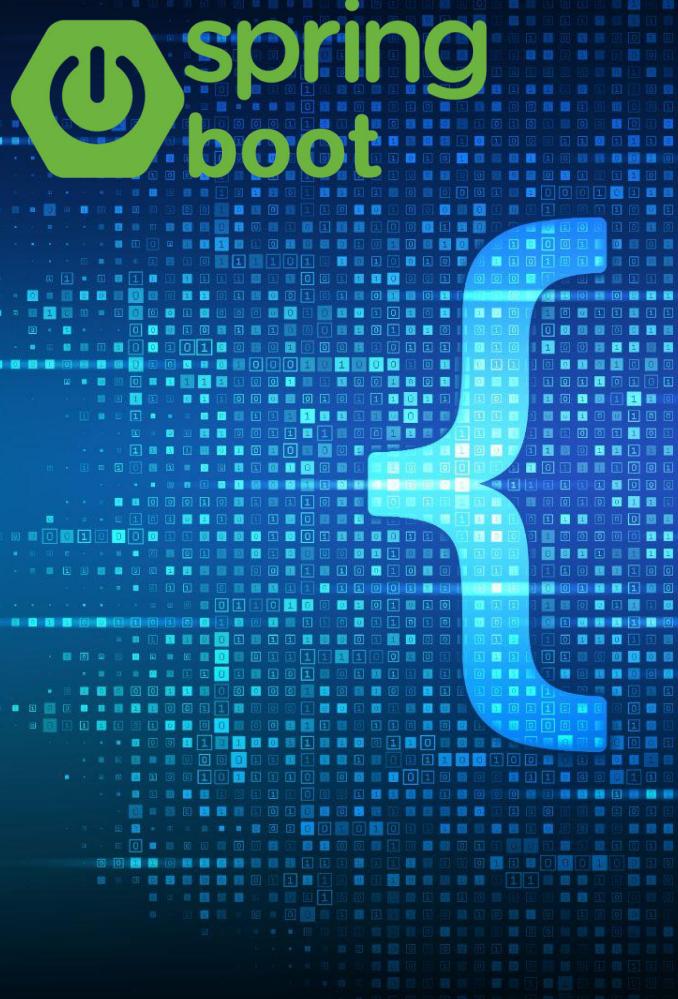
## Introduzione al Lombok

- Per utilizzare Lombok in un progetto Spring Boot, è necessario seguire alcuni passaggi:
- Aggiungere la dipendenza di Lombok nel file pom.xml (se si utilizza Maven) o nel file build.gradle (se si utilizza Gradle).
- Assicurarsi che il plugin Lombok sia installato nell'IDE utilizzato per lo sviluppo.
- Applicare le annotazioni di Lombok alle classi in cui si desidera generare automaticamente il codice.
- info: <https://www.baeldung.com/lombok-ide>



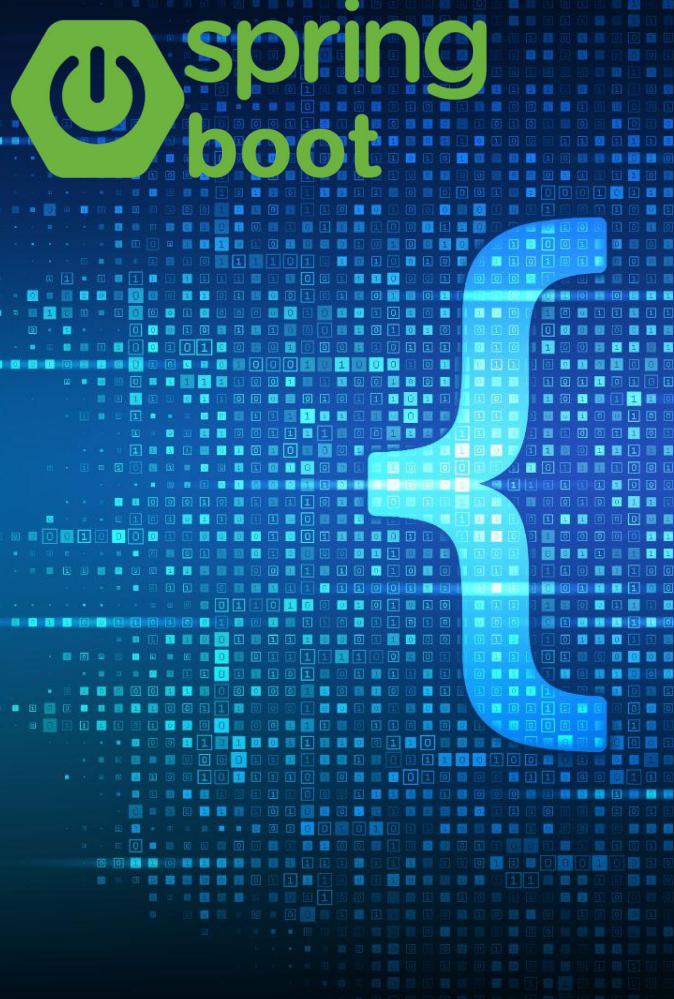
## Introduzione al Lombok

- L'utilizzo di Lombok offre diversi vantaggi:
- **Riduzione del codice ripetitivo:** Lombok genera automaticamente il codice comune, eliminando la necessità di scriverlo manualmente.
- **Maggiore leggibilità:** Rimuovendo il codice boilerplate, il codice diventa più pulito e focalizzato sulle funzionalità essenziali.
- **Manutenibilità migliorata:** Con meno codice da scrivere e mantenere, si riduce il rischio di errori e si semplifica la manutenzione del codice.
- **Risparmio di tempo nello sviluppo:** L'utilizzo di Lombok permette di sviluppare più velocemente, in quanto riduce il tempo necessario per scrivere il codice standard.



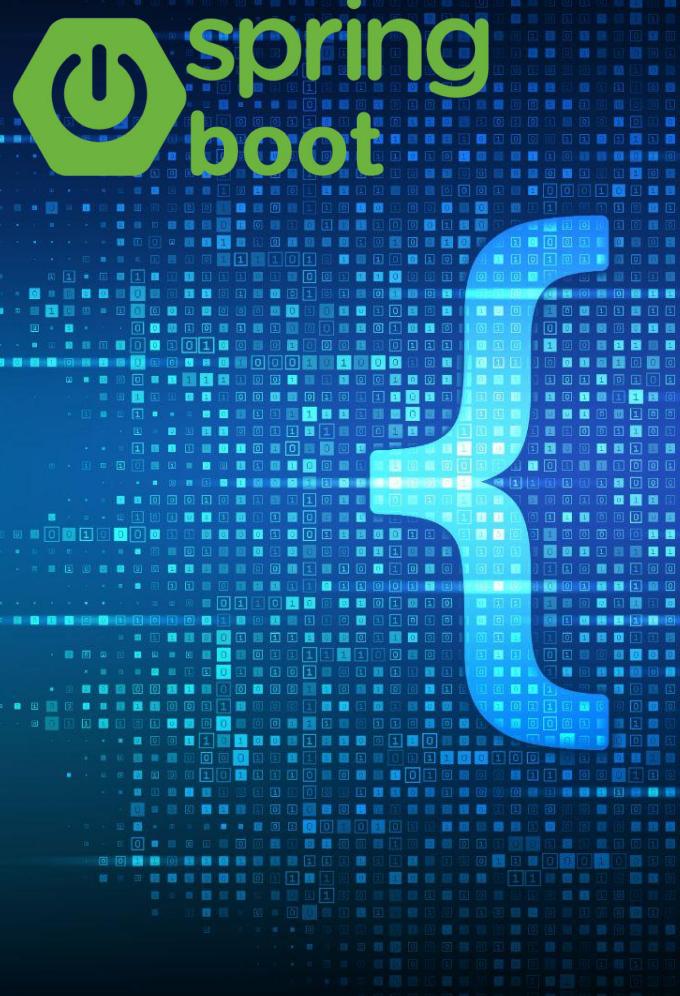
## Introduzione allo strato di persistenza

- Lo **strato di persistenza**, o strato di accesso ai dati, è uno dei componenti fondamentali di un progetto Spring Boot.
- Esso si occupa di interagire con il database o altre fonti di dati per l'archiviazione e il recupero delle informazioni.
- Nel contesto di un'applicazione Spring Boot, lo strato di persistenza viene comunemente implementato utilizzando l'approccio **Object-Relational Mapping (ORM)**, in cui le classi di entità Java sono mappate a tabelle nel database relazionale.
- Spring Boot fornisce diverse opzioni per la gestione della persistenza dei dati, tra cui JPA (Java Persistence API), Hibernate e **Spring Data JPA**.



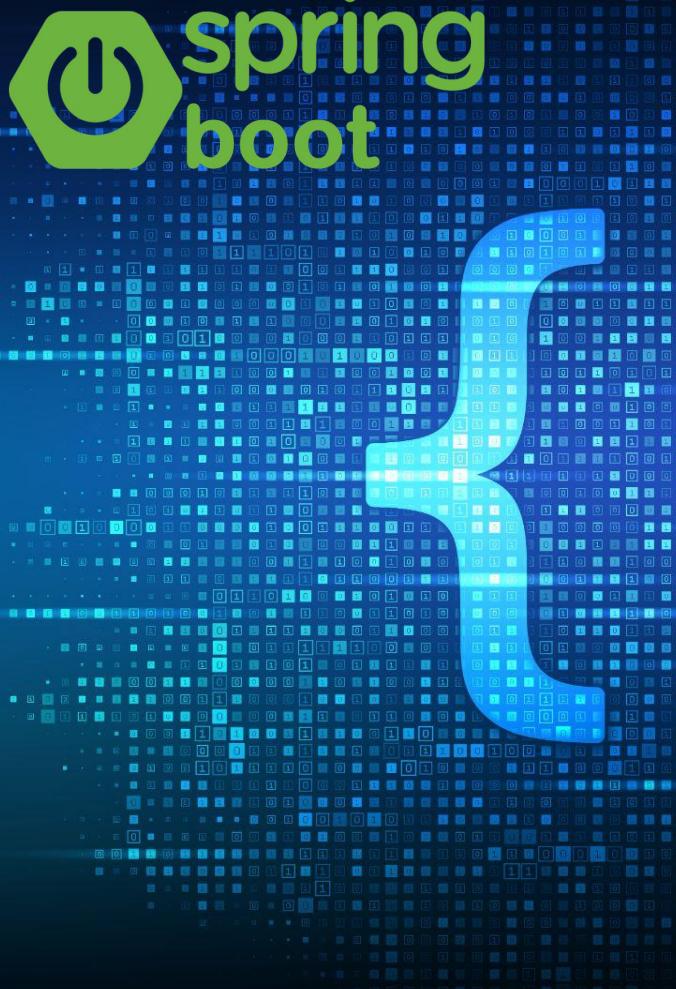
## Introduzione allo strato di persistenza

- I componenti principali che costituiscono lo strato di persistenza in un progetto Spring Boot sono:
  - **Classi di entità.**
  - **Repository** = I repository sono interfacce o classi che definiscono i metodi per l'accesso e la gestione dei dati. Questi metodi possono includere operazioni di base come `save()`, `delete()`, `findById()`, nonché metodi personalizzati definiti dall'utente. I repository vengono annotati con l'annotazione `@Repository` per consentire a Spring di rilevarli e gestirli automaticamente.
  - **JPA (Java Persistence API)** = JPA è una specifica standard per la gestione della persistenza dei dati in Java. Spring Boot integra JPA e fornisce implementazioni come Hibernate per l'implementazione dell'ORM. JPA definisce un set di annotazioni, come `@Entity`, `@Id`, `@GeneratedValue`, che vengono utilizzate per mappare le classi di entità alle tabelle del database e definire le relazioni tra le entità.



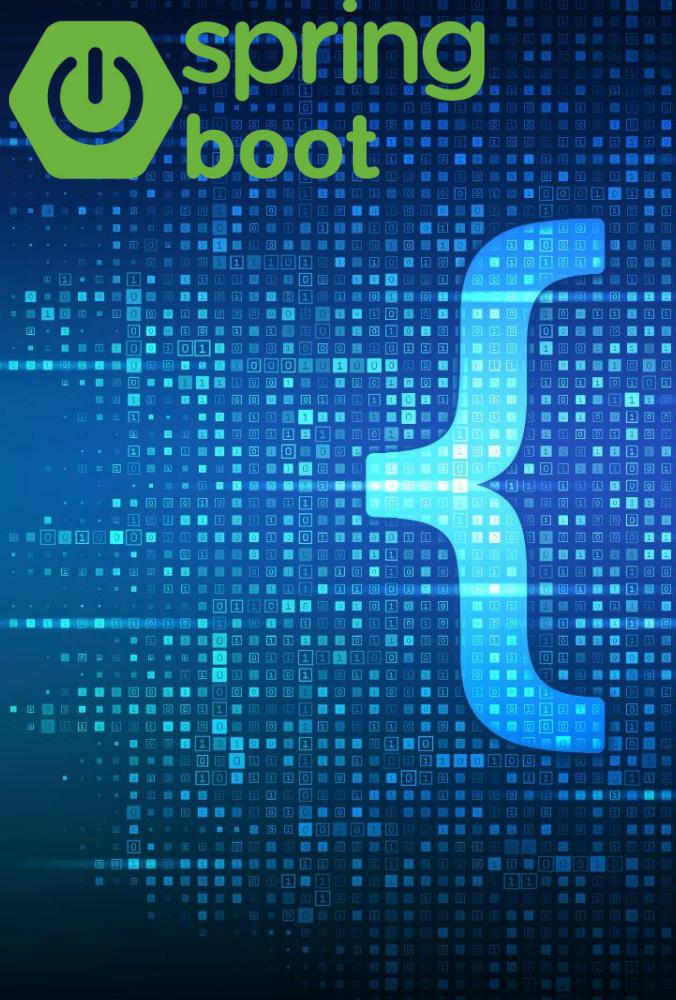
## Introduzione allo strato di persistenza

- **Configurazione del database** = Spring Boot semplifica la configurazione del database fornendo proprietà di configurazione nel file `application.properties` o `application.yml`. È possibile specificare il driver del database, l'URL di connessione, il nome utente, la password, ecc. in queste proprietà di configurazione.



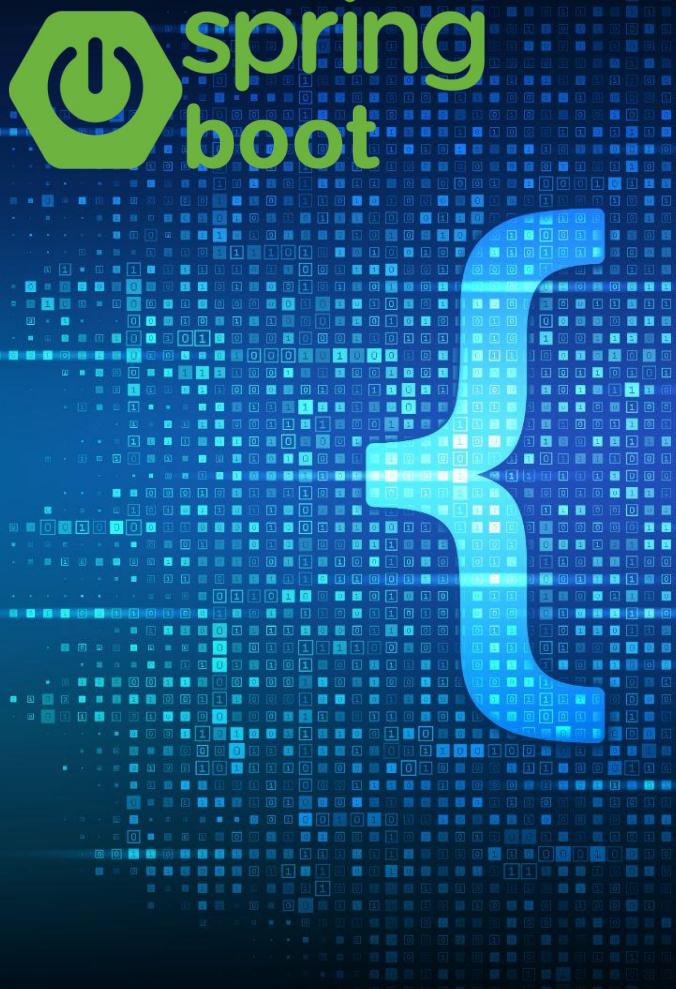
## Introduzione allo Spring Data JPA

- Spring Data JPA è un modulo di Spring Framework che semplifica lo sviluppo di applicazioni basate su JPA (Java Persistence API) per l'accesso ai dati.
- Offre un'astrazione aggiuntiva sopra il JPA, riducendo la quantità di codice boilerplate necessario per interagire con il database e semplificando le operazioni di persistenza dei dati.
- Le principali caratteristiche includono:
- Repository abstraction = Spring Data JPA fornisce un'interfaccia di repository generica chiamata **Repository** che offre operazioni **CRUD** (Create, Read, Update, Delete) di base e altre operazioni di query. Questa interfaccia può essere estesa per creare repository specifici per le entità dell'applicazione.



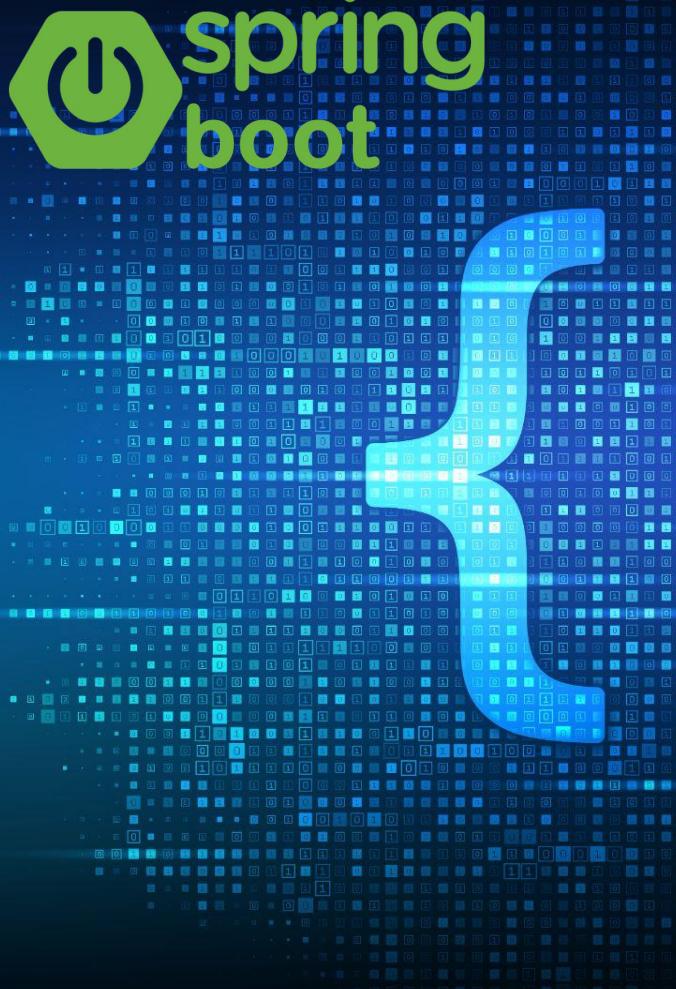
## Introduzione allo Spring Data JPA

- **Query methods** = Spring Data JPA consente di definire metodi di query all'interno dell'interfaccia del repository utilizzando una convenzione di denominazione basata su nomi. Ad esempio, definendo un metodo come `findByUsername(String username)` nel repository `UserRepository`, Spring Data JPA genererà automaticamente la query per cercare gli utenti in base al nome utente.
- **Query annotation** = Oltre ai query methods, è possibile utilizzare le annotazioni `@Query` per definire query personalizzate utilizzando il linguaggio **JPQL (Java Persistence Query Language)** o **SQL**.
- **Paginazione e ordinamento** = Spring Data JPA supporta la paginazione e l'ordinamento dei risultati delle query tramite le interfacce `Pageable` e `Sort`. Queste interfacce possono essere utilizzate come parametri nei metodi di query o nelle operazioni di repository per specificare l'ordine dei risultati e il numero di elementi per pagina.



## Introduzione allo Spring Data JPA

- **Supporto per diverse implementazioni di JPA** = Spring Data JPA è compatibile con diverse implementazioni di JPA come Hibernate, EclipseLink, OpenJPA, ecc. È possibile configurare l'implementazione desiderata tramite le proprietà di configurazione di Spring Boot.
- **Transazioni** = Spring Data JPA **supporta il controllo transazionale** tramite l'annotazione `@Transactional`. Questa annotazione può essere applicata **a livello di classe o di metodo** per definire il comportamento transazionale delle operazioni di repository.
- L'utilizzo di Spring Data JPA semplifica notevolmente lo sviluppo delle operazioni di persistenza dei dati in un progetto Spring Boot, riducendo la quantità di codice necessario per interagire con il database e offrendo una maggiore produttività nello sviluppo delle applicazioni.



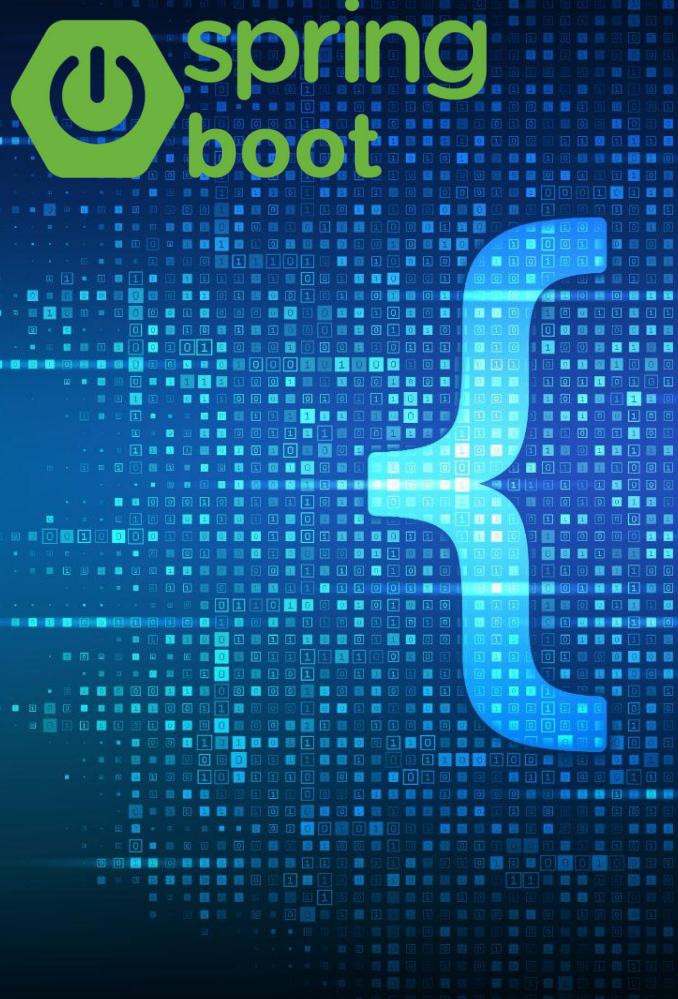
## Introduzione allo Spring Data JPA

- Spring Data JPA fornisce diverse interfacce che possono essere estese per creare i repository personalizzate.
- Di seguito sono elencate le principali interfacce offerte da Spring Data JPA:
- **Repository<T, ID>** = Questa è l'interfaccia di base da cui estendere i repository. Fornisce operazioni CRUD di base come save(), findById(), findAll(), delete(), ecc.
- **CrudRepository<T, ID>** = Estende l'interfaccia Repository e aggiunge operazioni di persistenza avanzate come saveAll(), deleteAll(), count(), ecc.
- **PagingAndSortingRepository<T, ID>** = Estende l'interfaccia CrudRepository e aggiunge il supporto per la paginazione e l'ordinamento dei risultati delle query.



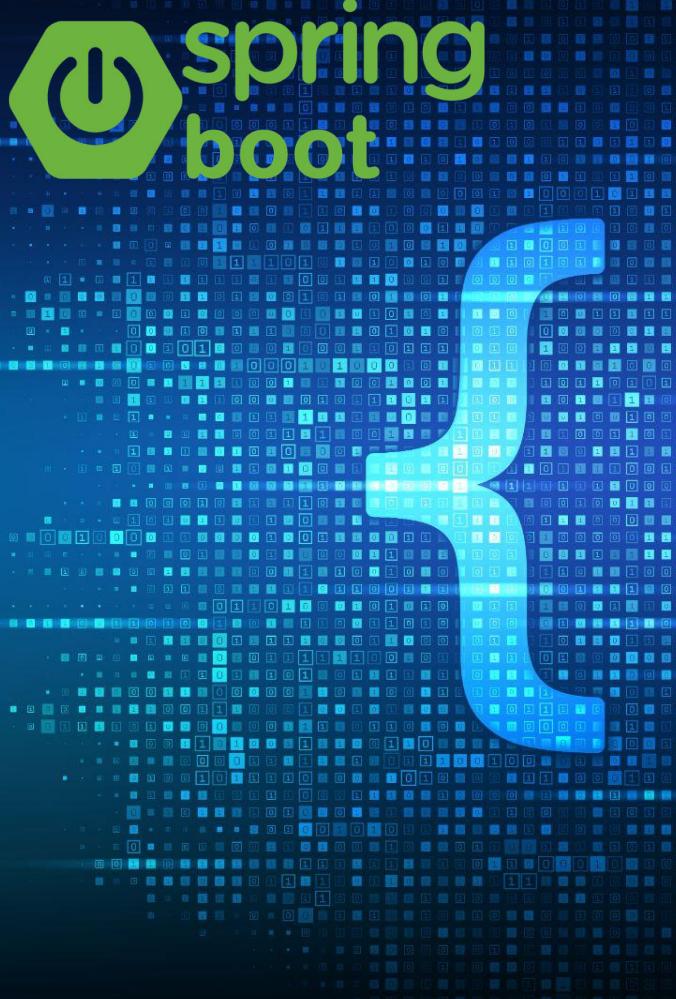
## Introduzione allo Spring Data JPA

- **JpaRepository<T, ID>** = Estende l'interfaccia PagingAndSortingRepository e aggiunge funzionalità specifiche di JPA come le operazioni di flush, refresh e detach.
- **JpaSpecificationExecutor<T>** = Questa interfaccia offre metodi per eseguire query basate su specifiche JPA, che consentono di definire condizioni di ricerca più complesse.
- **QueryByExampleExecutor<T>** = Questa interfaccia offre metodi per eseguire query utilizzando un esempio di entità come filtro di ricerca.
- **JpaRepositoryImplementation<T, ID>** = Questa interfaccia è utilizzata internamente da Spring Data JPA per implementare i repository personalizzati.
- È possibile estendere queste interfacce o combinarle per creare repository personalizzati che soddisfino le esigenze specifiche dell'applicazione.



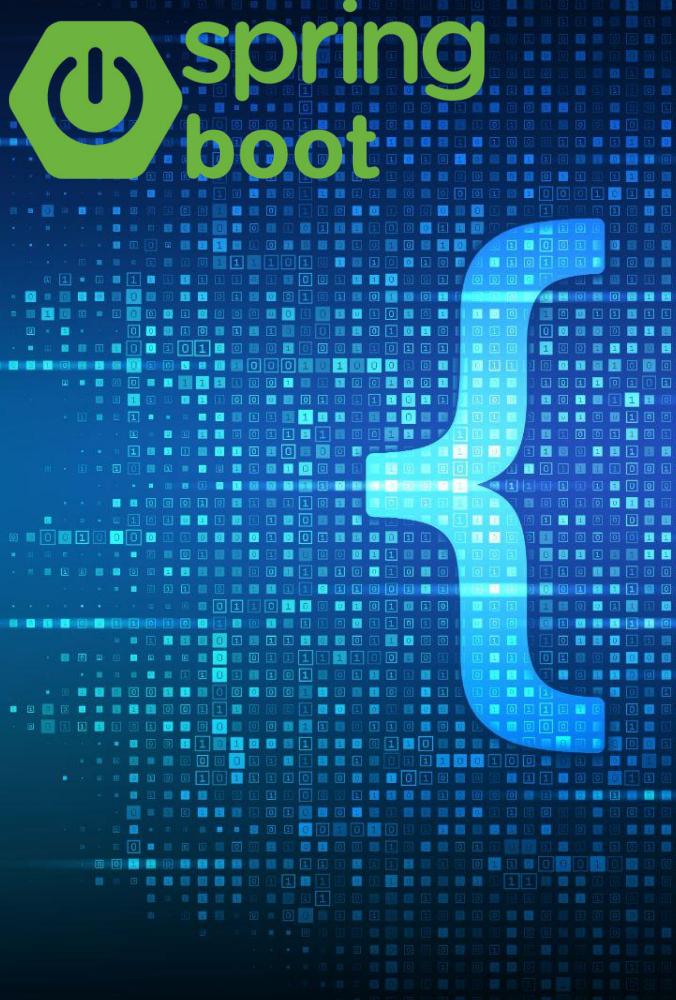
## Introduzione allo Strato di Servizio

- Lo **strato di servizio**, o **strato di business logic**, è uno dei componenti fondamentali di un'applicazione Spring Boot.
- Esso **rappresenta la logica dell'applicazione** e si **occupa di gestire le operazioni e le regole di business**.
- Lo strato di servizio agisce come **intermediario** tra lo **strato di presentazione (controller)** e lo **strato di persistenza dei dati (repository)**, garantendo che la logica di business venga eseguita in modo corretto e coerente.
- Nello specifico, lo strato di servizio di Spring Boot è responsabile di definire le operazioni e le funzionalità che possono essere eseguite sulle entità della base di dati, così come le procedure di business logic che orchestrano l'elaborazione dei dati e le regole di validazione e consentono di implementare il caching dei dati, la gestione delle transazioni, la sicurezza dell'applicazione, e così via.



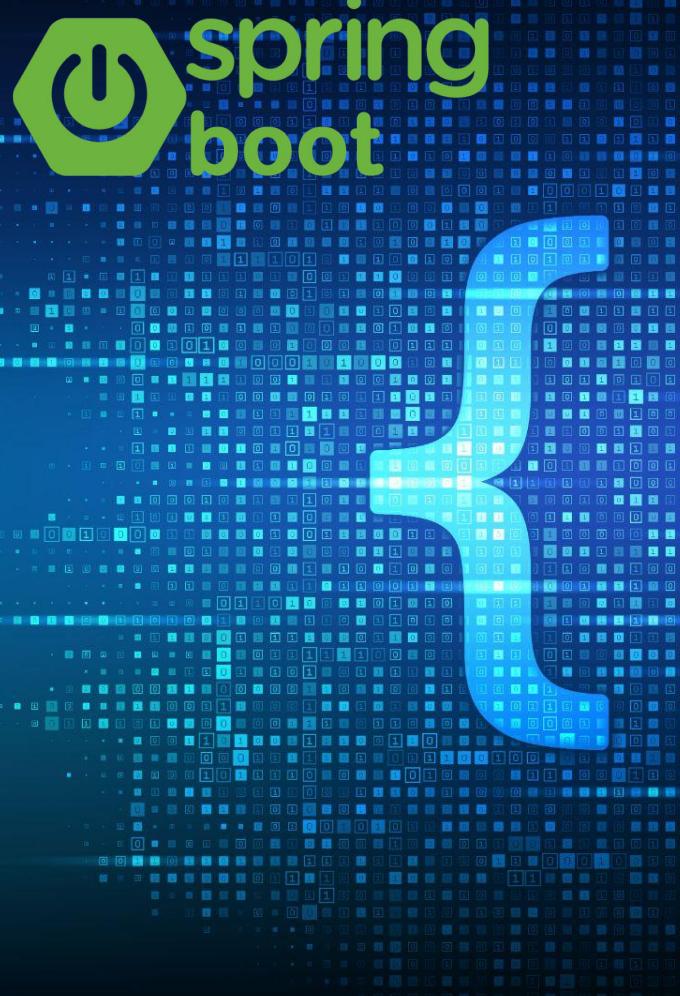
## Introduzione allo Strato di Servizio

- I punti chiave riguardanti lo strato di servizio in un'applicazione Spring Boot sono:
- **Responsabilità** = Lo strato di servizio si occupa di implementare le regole di business e le operazioni complesse dell'applicazione. Questo strato è responsabile di coordinare le operazioni tra più entità e di garantire che le operazioni siano eseguite in modo transazionale e coerente.
- **Indipendenza dallo strato di presentazione e dallo strato di persistenza dei dati** = Lo strato di servizio dovrebbe essere indipendente dagli aspetti specifici dell'interfaccia utente (come l'interfaccia grafica) e della persistenza dei dati (come il database). Ciò consente di separare la business logic dalle altre responsabilità dell'applicazione e di rendere il codice più modulare e riutilizzabile.



## Introduzione allo Strato di Servizio

- Chiamate ai repository = Lo strato di servizio comunica con lo strato di persistenza dei dati utilizzando i repository, richiamando i metodi dei repository per eseguire operazioni di lettura o scrittura sui dati. Ciò consente di separare la logica di accesso ai dati dallo strato di servizio e di delegare la gestione della persistenza dei dati ai repository.
- Transazioni = Lo strato di servizio può essere configurato per eseguire operazioni all'interno di transazioni, garantendo l'atomicità, la coerenza, l'isolamento e la durabilità delle operazioni. L'annotazione `@Transactional` può essere utilizzata per definire il comportamento transazionale dei metodi di servizio.
- Business Logic Complessa = Lo strato di servizio è il luogo in cui viene implementata la business logic più complessa dell'applicazione. Questa logica può includere operazioni di validazione, calcoli, elaborazioni, algoritmi complessi, ecc.



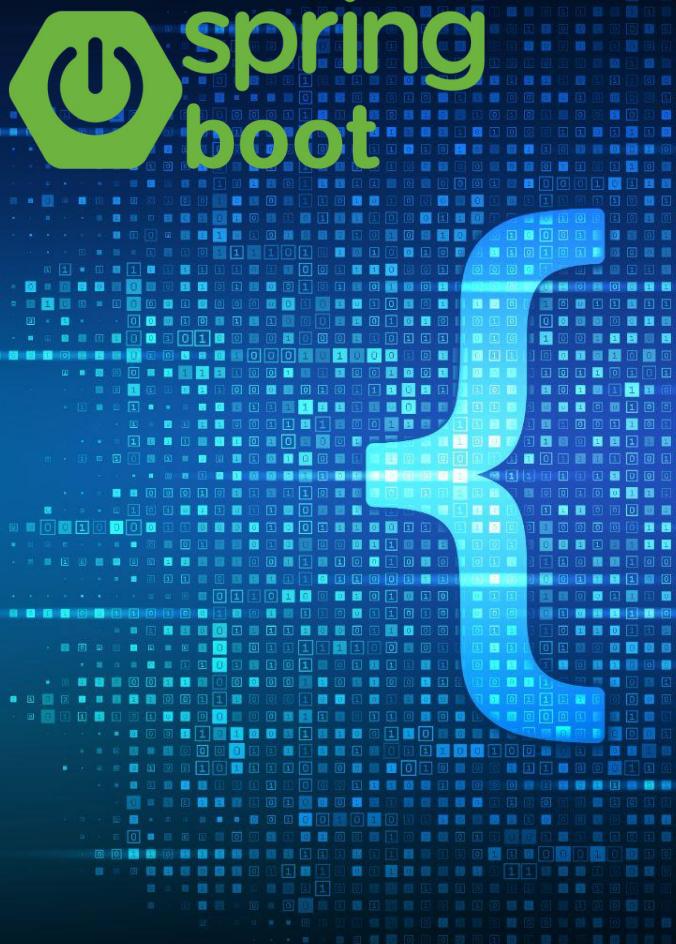
## Introduzione allo Strato di Servizio

- In sintesi, lo strato di servizio di Spring Boot rappresenta un punto focale cruciale per la definizione della logica di business e della gestione dei dati all'interno di un'applicazione basata su Spring Boot, offrendo un'organizzazione snella e scalabile dei componenti dell'applicazione.



## Introduzione alle classi DTO

- Le classi DTO (Data Transfer Object) in Spring Boot sono utilizzate per trasferire dati tra diversi strati o componenti dell'applicazione, come ad esempio tra lo strato di presentazione (controller) e lo strato di servizio o lo strato di persistenza dei dati (repository).
- Le classi DTO sono spesso utilizzate come **modello di dati semplice e leggero** per rappresentare un subset dei dati di un'entità.
- Le classi DTO sono solitamente progettate per rappresentare solo i dati che devono essere trasmessi tra il client e il server, e sono spesso utilizzate per ottimizzare la **velocità di risposta dell'applicazione**, riducendo il carico di dati trasferiti tra il client e il server.
- Le classi DTO possono anche essere utilizzate per effettuare la **validazione dei dati in ingresso**, prima che essi vengano salvati nel database, al fine di garantire la loro integrità e coerenza.



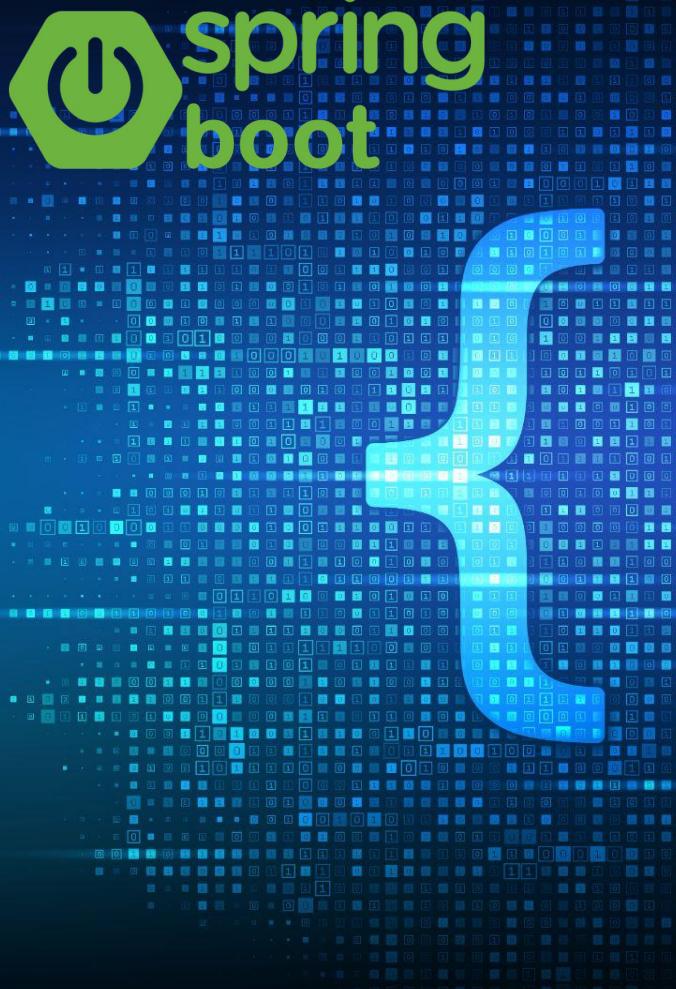
## Introduzione alle classi DTO

- Alcuni punti chiave sulle classi DTO in Spring Boot sono:
- **Separazione delle responsabilità:** L'utilizzo delle classi DTO consente di separare i modelli di dati dell'applicazione (entity) dai modelli di dati utilizzati per la comunicazione tra i diversi strati. Questa separazione aiuta a mantenere l'indipendenza tra gli strati e a evitare l'esposizione di dettagli interni dell'entità al di fuori del loro contesto.
- **Personalizzazione dei dati:** Le classi DTO permettono di personalizzare il set di dati trasmessi in una determinata situazione. Ad esempio, è possibile escludere alcuni campi sensibili o calcolati dalle risposte API o includere solo i campi necessari per una determinata operazione.



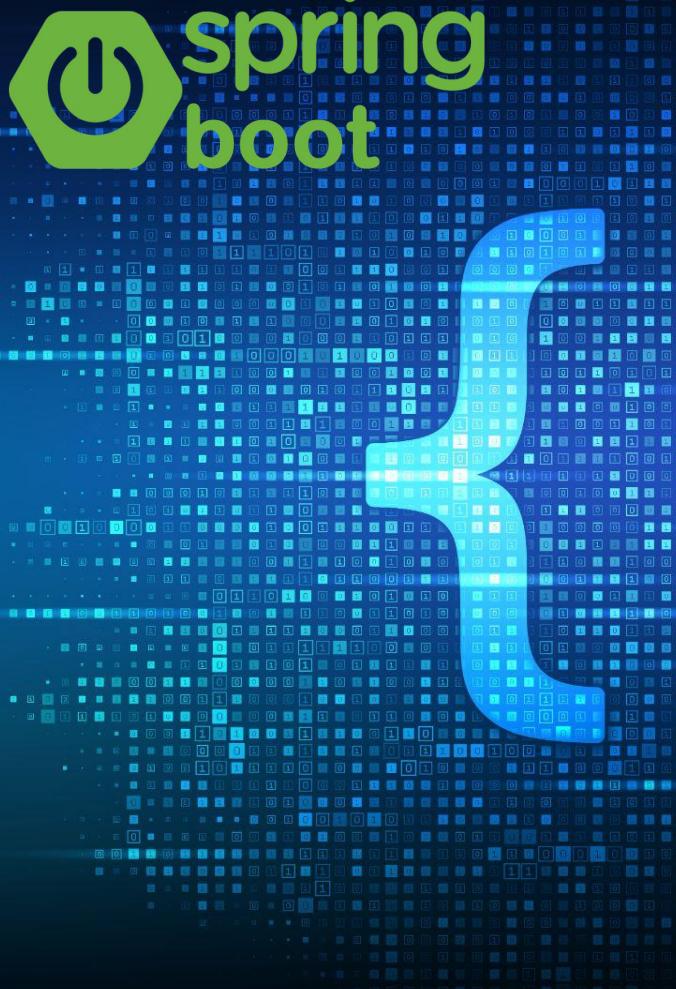
## Introduzione alle classi DTO

- Ottimizzazione delle prestazioni: Utilizzando classi DTO, è possibile ridurre il carico di dati trasmessi tra i componenti dell'applicazione. Questo può portare a prestazioni migliori, specialmente in scenari in cui si trasferiscono grandi quantità di dati o in reti a banda limitata.
- Mappatura dei dati: Le classi DTO vengono utilizzate per mappare i dati provenienti da oggetti più complessi, come entità JPA o oggetti restituiti da servizi esterni, a oggetti più semplici e specifici per uno scopo particolare. Questo processo di mappatura può essere eseguito manualmente o utilizzando framework di mappatura come ModelMapper o MapStruct.
- In sintesi, le classi DTO sono un modo utilizzato in Spring Boot per semplificare e velocizzare la gestione dei dati all'interno di un'applicazione, garantendo al contempo l'integrità e la coerenza dei dati.



## Introduzione alle ModelMapper

- Il ModelMapper è una libreria Java open source utilizzata per mappare oggetti di diversi tipi (DTO, Entity, Model) in modo automatico ed efficiente. In particolare, in un contesto Spring Boot, il ModelMapper viene utilizzato per semplificare il processo di conversione di oggetti da una classe a un'altra, ad esempio durante il trasferimento di dati tra il layer di business e quello di presentazione.
- Il ModelMapper fornisce una soluzione conveniente per la mappatura degli oggetti, consentendo di definire le regole di mappatura tra i campi dei diversi oggetti e di eseguire la conversione automatica dei valori corrispondenti. Semplifica il processo di mappatura evitando la scrittura manuale del codice di mappatura per ogni singolo campo.



## Introduzione alle ModelMapper

- Il ModelMapper è integrato perfettamente con Spring Boot e può essere facilmente configurato utilizzando i bean di Spring. Infatti, il ModelMapper può essere configurato mediante un bean all'interno del contesto Spring e utilizzato per tutte le conversioni tra gli oggetti.

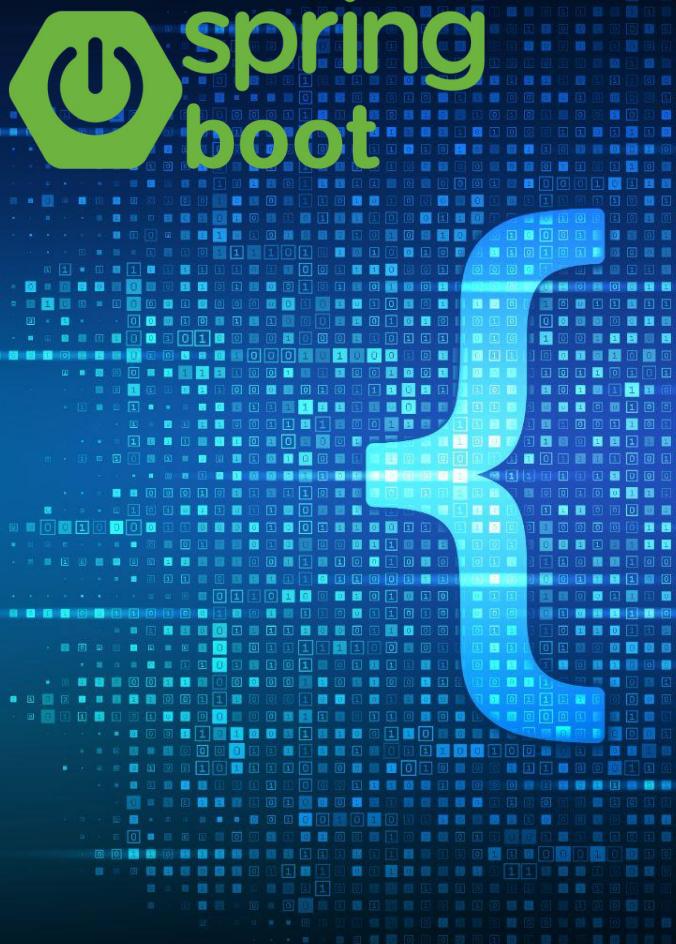


## Introduzione alla notazione @PathVariable

- Nel contesto di una richiesta HTTP, gli URL possono contenere parametri, chiamati **path variables**, che sono inseriti nell'URL stesso e **non nella query string**. Ad esempio, nell'URL "https://mywebsite.com/users/123", "123" è un esempio di path variable che rappresenta l'identificatore di un utente.
- Quando si crea un metodo di gestione delle richieste in Spring, è possibile utilizzare la notazione **@PathVariable** per mappare queste variabili di percorso agli argomenti del metodo. Ad esempio:

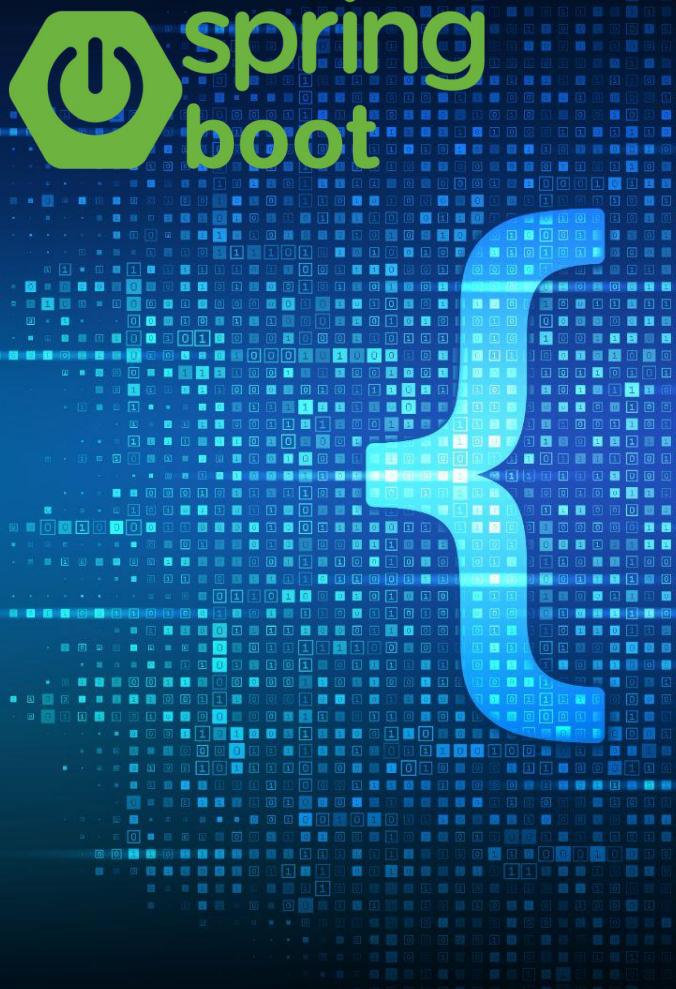
```
@GetMapping("/users/{id}")
public User getUser(@PathVariable("id") int id) {
    //... logica per ottenere l'utente con l'ID dato ...
    return user;
}
```

- La notazione **@PathVariable** supporta anche altre funzionalità, come la specifica di valori predefiniti e l'uso di espressioni regolari per vincolare i valori delle variabili di percorso.



## Introduzione al tag <form:form>

- Il tag <form:form> è un tag personalizzato fornito dal taglib Spring Form che semplifica la creazione di form HTML all'interno delle pagine JSP (JavaServer Pages) all'interno di applicazioni Spring.
- L'utilizzo di <form:form> consente di definire e configurare facilmente un form nel codice HTML utilizzando le funzionalità di binding dei dati di Spring.
- Questa notazione offre vari attributi che possono essere utilizzati per specificare l'azione del form, il metodo di invio, gli attributi del modello associato, i valori di default e così via.



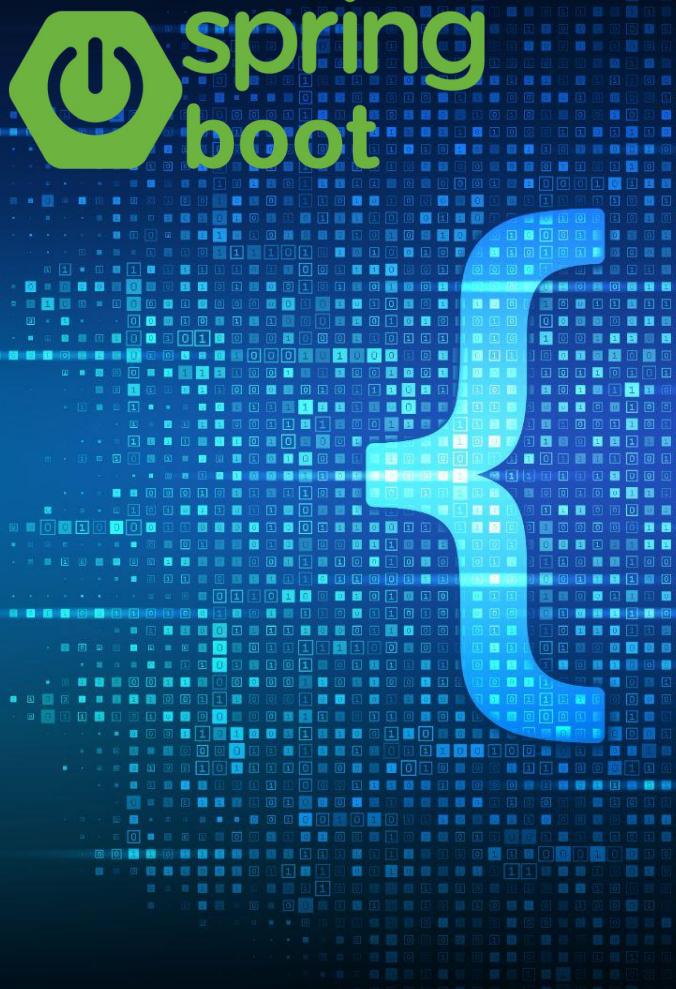
## Introduzione al tag <form:input>

- Il tag <form:input> è un tag specifico di Spring Framework che semplifica la creazione di campi di input HTML all'interno delle pagine JSP (JavaServer Pages)
- Questo tag è utilizzato principalmente all'interno di form creati con il tag <form:form> per legare i valori dei campi di input agli oggetti modello di Spring.
- Gli attributi più comuni utilizzati con il tag <form:input> sono:
  - **path:** specifica il percorso al campo nel modello. È molto simile all'attributo "name" di un campo HTML.
  - **id:** specifica l'ID del campo di input HTML generato. **Questo attributo è opzionale**, ma è spesso utile per scopi di stile o per associare label con input.



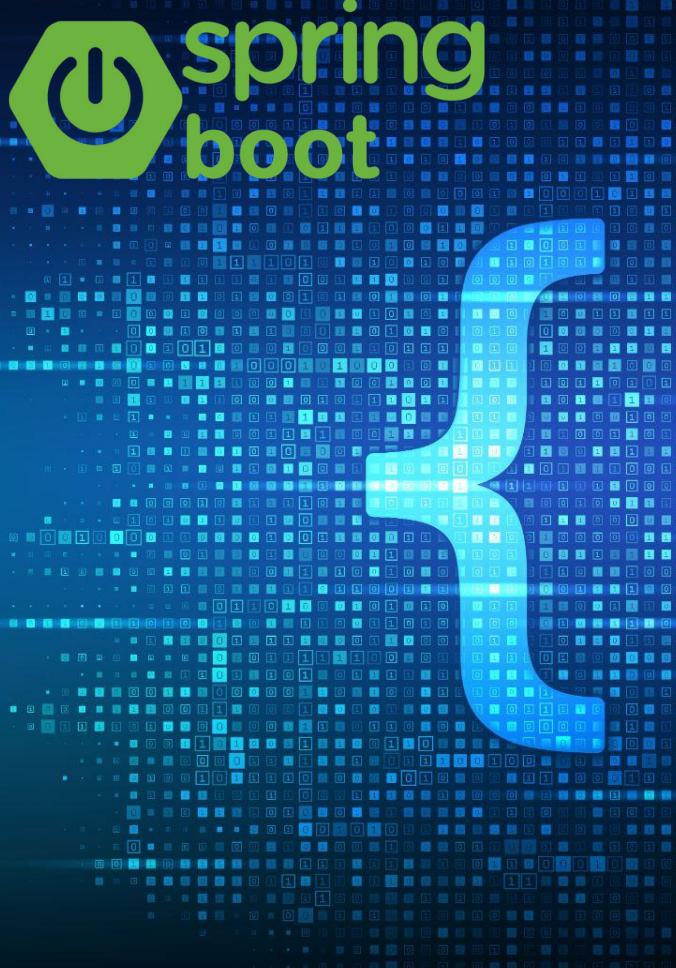
## Introduzione al tag <form:input>

- **cssClass**: specifica una o più classi CSS da applicare al campo di input generato.
- **size**: specifica il numero di caratteri visibili nel campo di input. Ad esempio, size="20" creerà un campo di input con una larghezza di 20 caratteri.
- **maxlength**: specifica il numero massimo di caratteri consentiti nel campo di input. Questo attributo è utile per limitare la lunghezza dei dati inseriti.
- **placeholder**: specifica un testo di esempio che viene visualizzato all'interno del campo di input vuoto. Questo attributo fornisce suggerimenti agli utenti su cosa inserire nel campo.
- **readonly**: imposta il campo di input in sola lettura, il che significa che l'utente non può modificarne il valore.



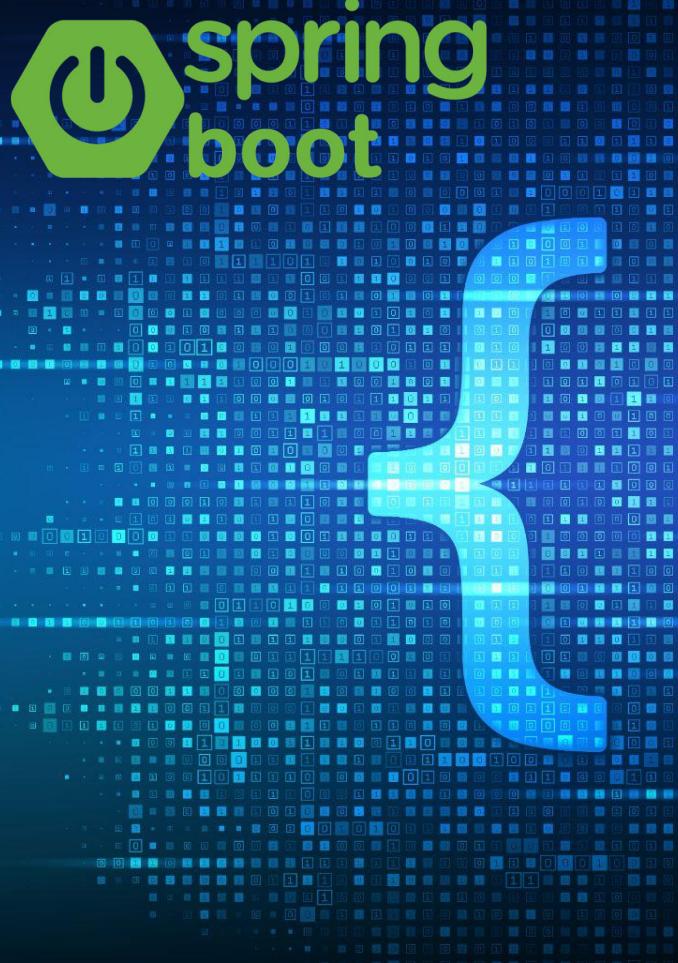
## Introduzione al tag <form:input>

- **disabled:** disabilita il campo di input, il che significa che l'utente non può interagire con esso.
- **value:** specifica il valore predefinito del campo di input. Ad esempio, value="Default value" mostrerà il testo "Default value" all'interno del campo di input.
- **Tipo di input:** Il tag <form:input> è utilizzato per generare campi di input HTML di diversi tipi, come testo (<input type="text">), password (<input type="password">), numeri (<input type="number">), ecc.
- **Validazione:** Se il form è configurato con la validazione, gli eventuali errori di validazione vengono visualizzati automaticamente accanto al campo di input corrispondente.



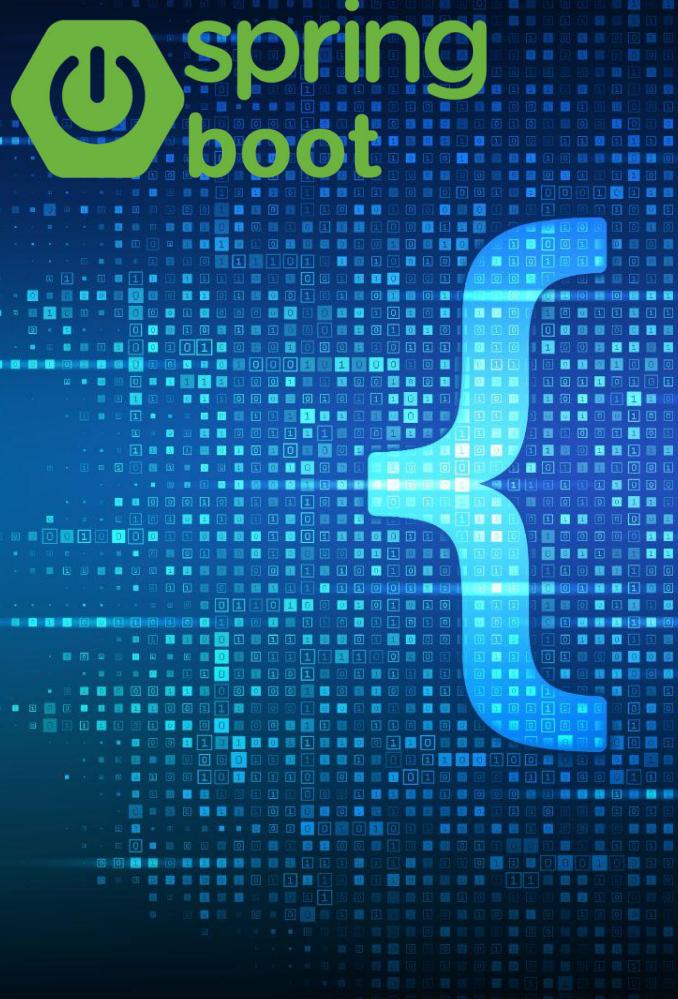
## Introduzione al tag <form:input>

- **Conversione dei tipi:** Spring esegue automaticamente la conversione dei tipi per i campi di input. Ad esempio, se l'attributo di un oggetto modello è di tipo int e il valore di input è una stringa numerica, Spring effettuerà la conversione in un valore intero.



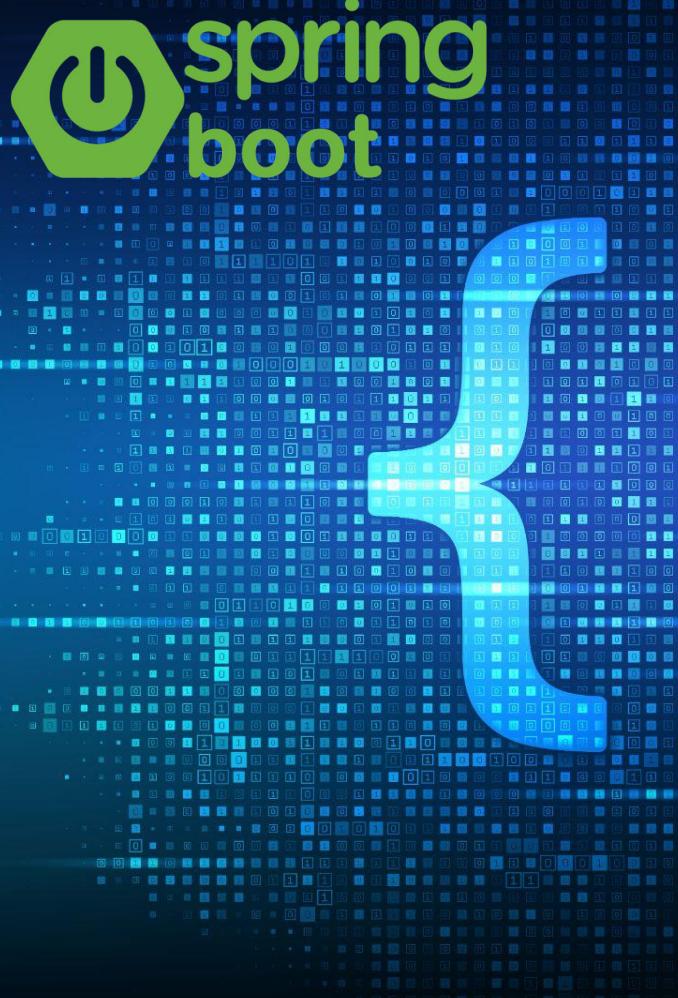
## Introduzione alla notazione @MatrixVariable

- La notazione `@MatrixVariable` è una specifica di annotazione utilizzata nel framework Spring per l'accesso a una variabile di matrice di una richiesta HTTP.
- Una **matrice di variabili** è un componente dell'URI di una richiesta HTTP che **consente di specificare dei parametri aggiuntivi** per specifiche risorse. Ad esempio, nell'URI `"/studenti;nome=John;età=20"`, `"nome=John"` e `"età=20"` sono matrici di variabili che possono essere utilizzate per filtrare i risultati di una richiesta per gli studenti.
- Nel contesto di Spring, `@MatrixVariable` può essere utilizzato come un'annotazione per un parametro del metodo di un controller **per estrarre i valori delle variabili di matrice dalla richiesta HTTP**.



## Introduzione alla notazione @MatrixVariable

- Dettaglio caratteristiche notazione `@MatrixVariable`:
- **Parametri di Matrice:** Un parametro di matrice è una coppia chiave-valore separata dal punto e virgola (;) che è inclusa all'interno di un percorso URL. Ad esempio, nell'URL `http://example.com/product;name=iphone;color=black`, `name=iphone` e `color=black` sono parametri di matrice.
- **Uso con `@RequestMapping`:** La notazione `@MatrixVariable` può essere utilizzata all'interno dei metodi annotati con `@RequestMapping` di un controller Spring. Consentono di estrarre i parametri di matrice direttamente dal percorso URL.
- **Sintassi:** La notazione `@MatrixVariable` può essere utilizzata su un parametro di metodo del controller. La sintassi è simile a quella di altre annotazioni di Spring, come `@RequestParam` o `@PathVariable`. Ad esempio: `@MatrixVariable(name = "paramName") String paramValue`.



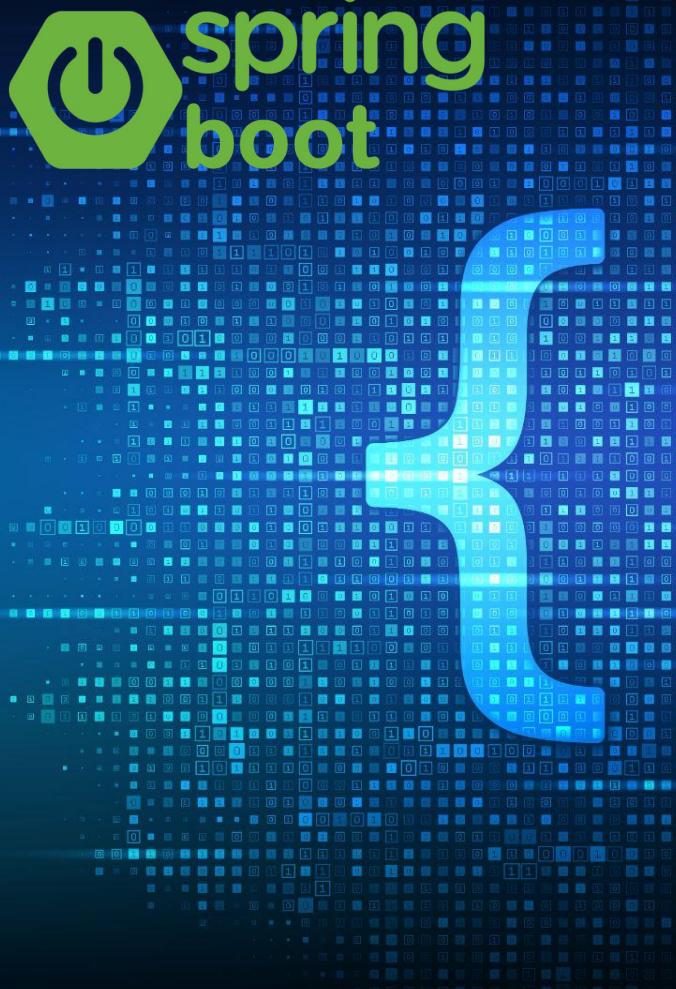
## Introduzione alla notazione @MatrixVariable

- **Nome del Parametro:** Il parametro `name` di `@MatrixVariable` specifica il nome del parametro di matrice da estrarre dall'URL.
- **Valore Predefinito:** È possibile fornire un `valore predefinito` utilizzando l'attributo `defaultValue` nel caso in cui il parametro di matrice non sia presente nell'URL.
- **Accesso Multiplo:** Se il parametro di matrice è presente più volte nell'URL, utilizzando l'attributo `pathVar` di `@MatrixVariable` è possibile indicare quale occorrenza si desidera estrarre.



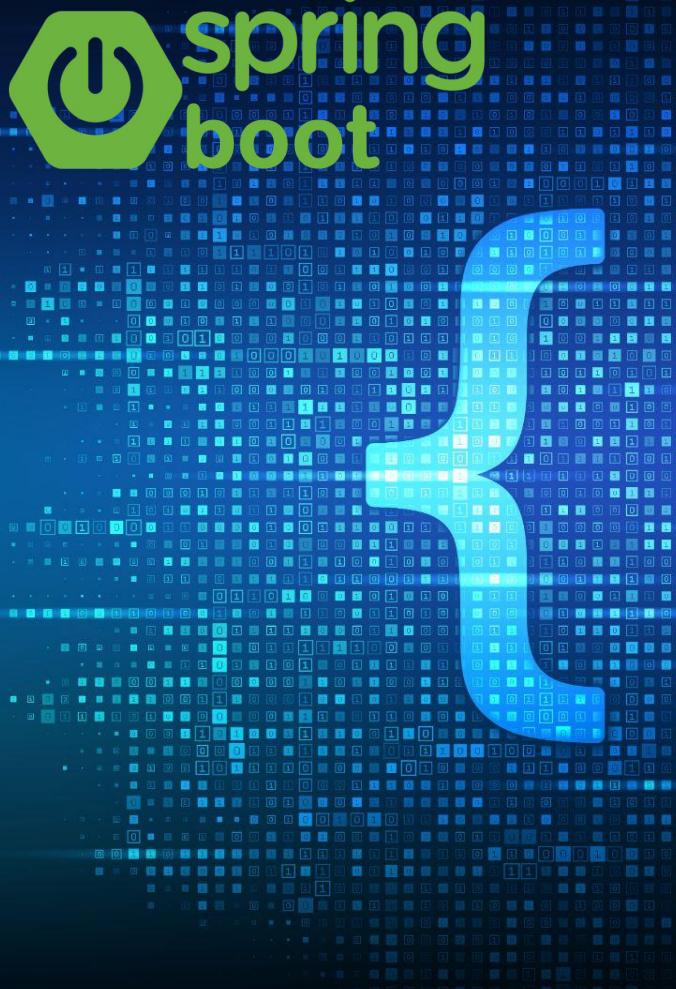
## Introduzione al tag <spring:url>

- Il tag <spring:url> è un tag personalizzato fornito da Spring Framework per generare URL o link con il supporto del contesto di Spring.
- Viene utilizzato all'interno delle pagine JSP per generare URL in modo dinamico e affidabile.
- Il tag <spring:url> ha diversi attributi che consentono di specificare le diverse parti dell'URL. Questi attributi includono:
  - **value**: l'attributo value accetta un'espressione EL (Expression Language) che può includere variabili o metodi per costruire l'URL. Ad esempio, `value="/path/{param}"`.
  - **var**: l'attributo var specifica il nome della variabile in cui verrà memorizzato l'URL generato. Ad esempio, `var="myUrl"`.



## Introduzione al tag <spring:url>

- **scope**: l'attributo scope specifica la portata della variabile definita nell'attributo var. I valori validi per questo attributo includono "page", "request", "session" e "application".
- **context**: l'attributo context specifica il contesto del server a cui fa riferimento l'URL. Ad esempio, context="/myapp".
- **params**: l'attributo params consente di specificare i parametri che devono essere inclusi nell'URL. Ad esempio,  
`params="param1=${value1}&param2=${value2}"`.
- **secure**: l'attributo secure specifica se l'URL generato deve essere sicuro oppure no. Il valore predefinito è false.
- **htmlEscape**: l'attributo htmlEscape specifica se l'URL generato deve essere escape per evitare attacchi di cross-site scripting (XSS). Il valore predefinito è true.

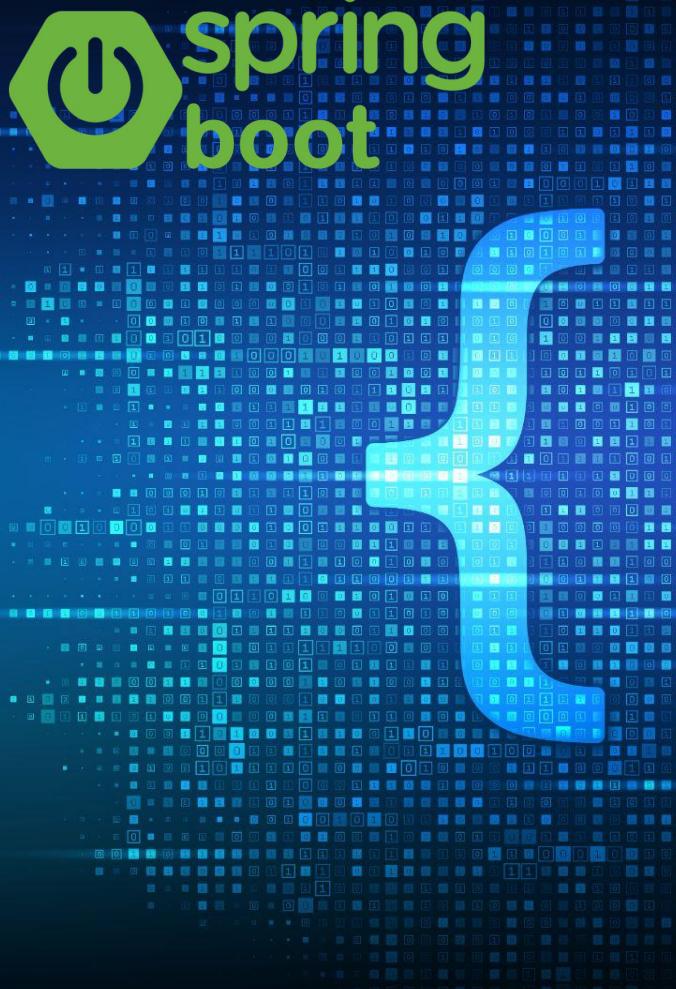


## Introduzione alla notazione @ModelAttribute

- La notazione `@ModelAttribute` è utilizzata in Spring Framework per associare un metodo o un parametro a un attributo del modello.
  - Questo attributo del modello sarà quindi disponibile per la vista in cui viene visualizzata l'informazione.
  - La notazione `@ModelAttribute` può essere utilizzata in due modi:
1. Quando viene utilizzata su un metodo, il valore restituito da quel metodo verrà aggiunto al modello con il nome specificato come parametro della notazione.

```
@ModelAttribute("product")
public Articoli getProduct(@RequestParam Long id) {
    // Logic to fetch product details by id
    Articoli product = articoliService.getProductById(id);
    return product;
}

@GetMapping("/view")
public String viewProductDetails(@ModelAttribute("product") Articoli product) {
    // Logic to display product details in the view
    return "productDetailsView";
}
```

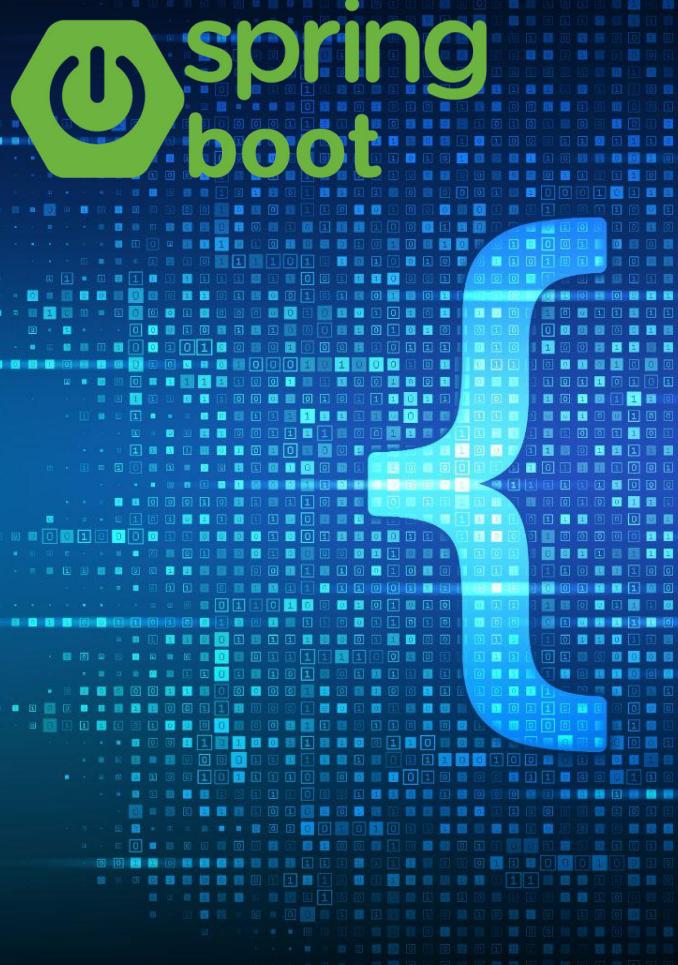


## Introduzione alla notazione @ModelAttribute

- Quando viene utilizzata **come un parametro di un metodo**, la notazione **@ModelAttribute** legge un attributo dal modello con il nome specificato come parametro della notazione e **lo passa al metodo come argomento**. Ad Esempio:

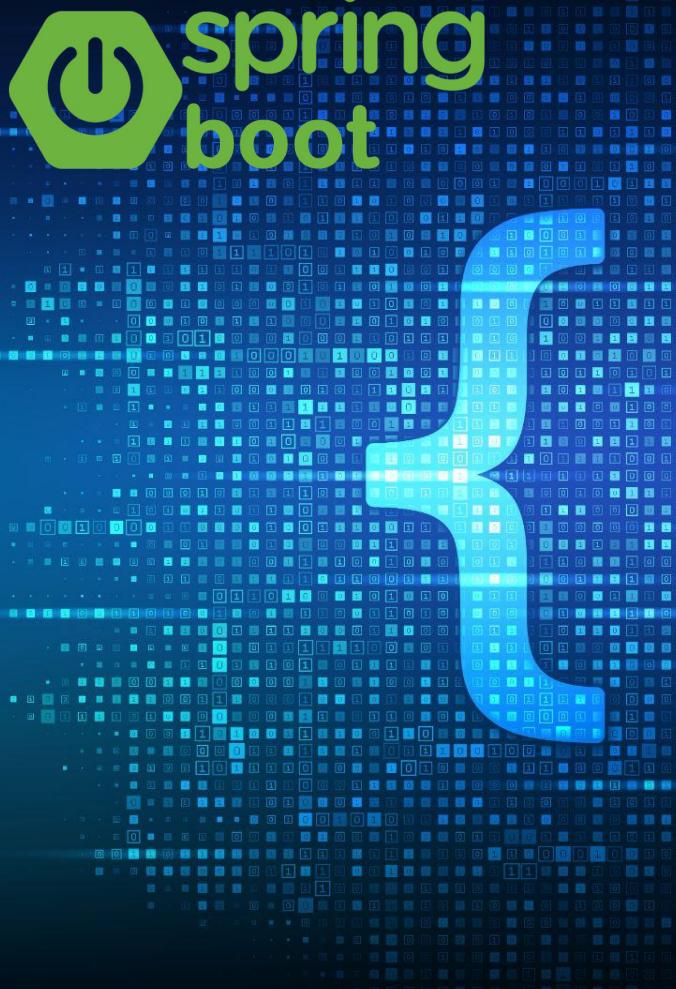
```
@PostMapping("/user")
public String saveUser(@ModelAttribute("user") User user){
    // Salva il nuovo utente nel database
    return "redirect:/users";
}
```

- Se il metodo del controller restituisce un oggetto modello, **@ModelAttribute** non è necessario perché **Spring associa automaticamente l'oggetto modello restituito al nome dell'attributo corrispondente nella vista**.



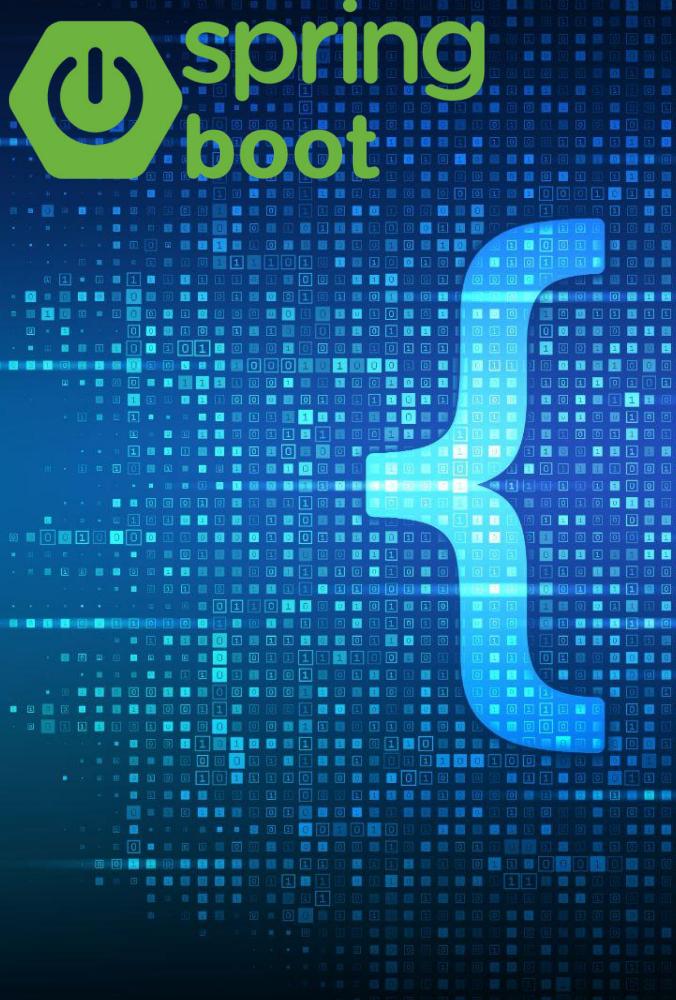
## Introduzione alla notazione @ModelAttribute

- Se il metodo annotato con `@ModelAttribute` accetta parametri aggiuntivi (ad esempio `@RequestParam`), questi parametri vengono utilizzati per popolare l'oggetto modello. Inoltre, i dati provenienti dalla vista (ad esempio da un form) vengono automaticamente associati all'oggetto modello in base ai nomi dei campi corrispondenti.
- E' possibile utilizzare `@ModelAttribute` per eseguire operazioni aggiuntive durante il popolamento dell'oggetto modello, come eseguire conversioni o applicare validazioni.



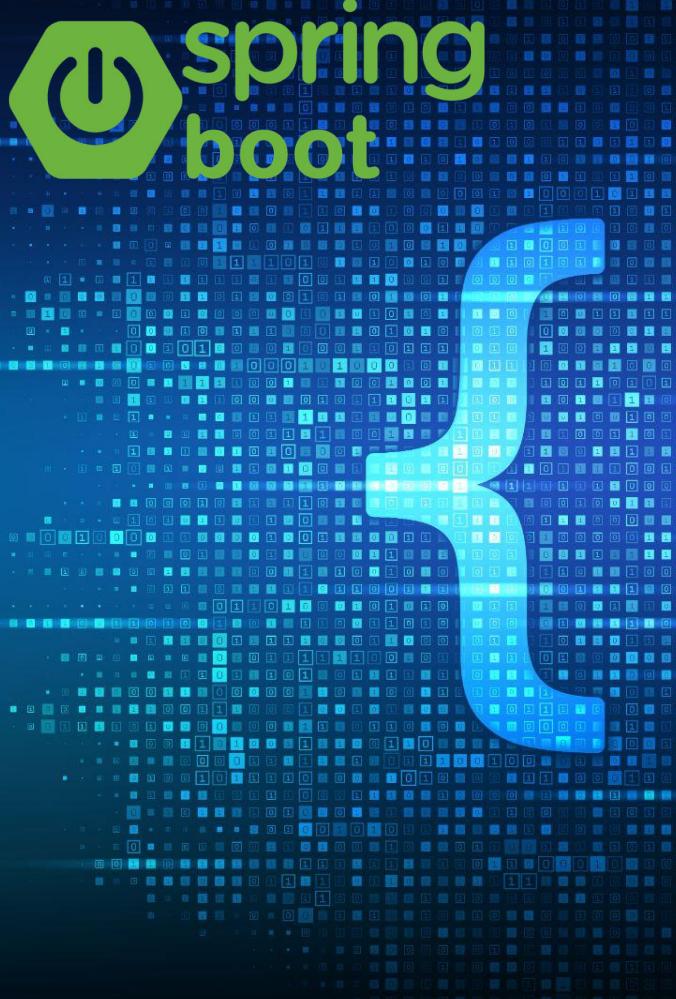
## Introduzione alla Binding Validation

- La Binding Validation in Spring Boot è un meccanismo che consente di applicare regole di validazione agli oggetti in fase di binding. Consiste nell'applicazione di vincoli e regole specifiche per garantire che i dati inseriti dagli utenti soddisfino determinati criteri di validità.
- Quando si riceve una richiesta HTTP, Spring Boot mappa automaticamente i dati inviati come parametri nella richiesta su un oggetto corrispondente. Questo processo viene chiamato "binding" e consente di associare i parametri della richiesta ai campi di un oggetto.
- La binding validation viene utilizzata per controllare che i valori dei campi dell'oggetto siano validi. Ciò può includere la verifica della presenza o meno di un valore, il controllo della lunghezza di una stringa, la validazione di un indirizzo email, la verifica di una password complessa e così via.



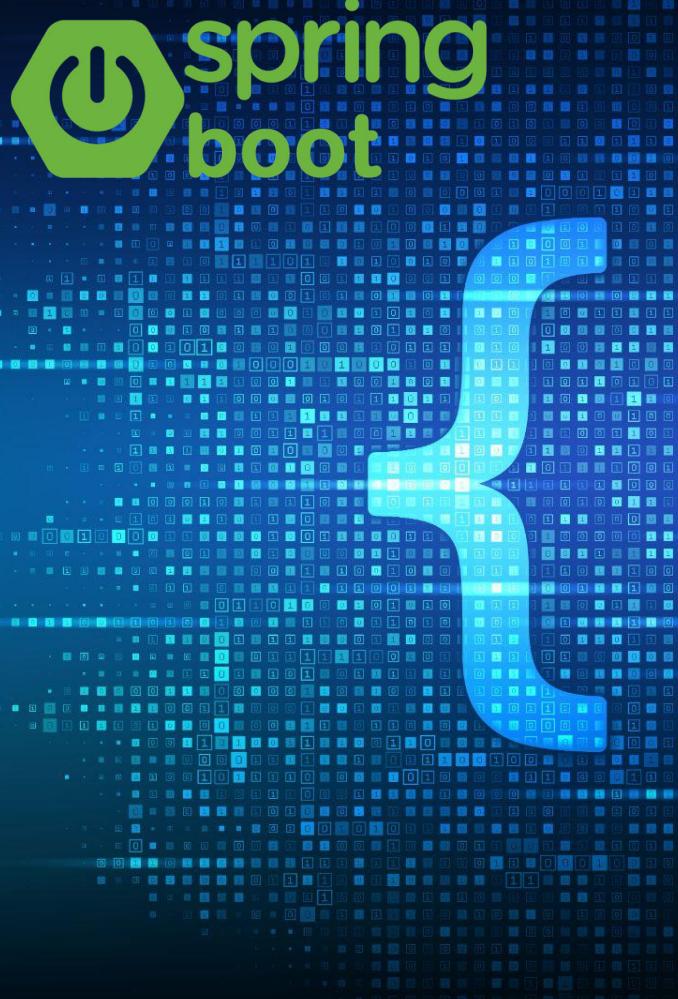
## Introduzione alla Binding Validation

- La Binding Validation avviene tramite:
- Definizione delle regole di validazione: Prima di tutto, è necessario definire le regole di validazione per l'oggetto che si desidera validare. Ciò può essere fatto creando una classe di validazione che implementi l'interfaccia **Validator** di Spring Framework. In questa classe, è possibile definire le regole di validazione utilizzando annotazioni come `@NotNull`, `@Size`, `@Pattern`, etc., o scrivendo una logica personalizzata nel metodo `validate()`.
- Associazione delle regole di validazione all'oggetto: Una volta definite le regole di validazione, è necessario associarle all'oggetto che si desidera validare. Questo viene fatto utilizzando l'annotazione `@Valid` sul parametro del metodo o come annotazione a livello di classe per l'oggetto stesso.



## Introduzione alla Binding Validation

- **Attivazione della validazione:** Per attivare la validazione durante il processo di binding, è possibile utilizzare l'annotazione `@Validated` a livello di classe o metodo nel controller che gestisce la richiesta. Questa annotazione indica a Spring di avviare la validazione per gli oggetti associati con l'annotazione `@Valid`.
- **Gestione degli errori di validazione:** Durante il processo di validazione, se vengono rilevati errori, Spring genera un oggetto `BindingResult` che contiene gli errori di validazione. È possibile accedere a questo oggetto e gestire gli errori di validazione, ad esempio restituendo un messaggio di errore personalizzato o reindirizzando l'utente a una pagina di errore.



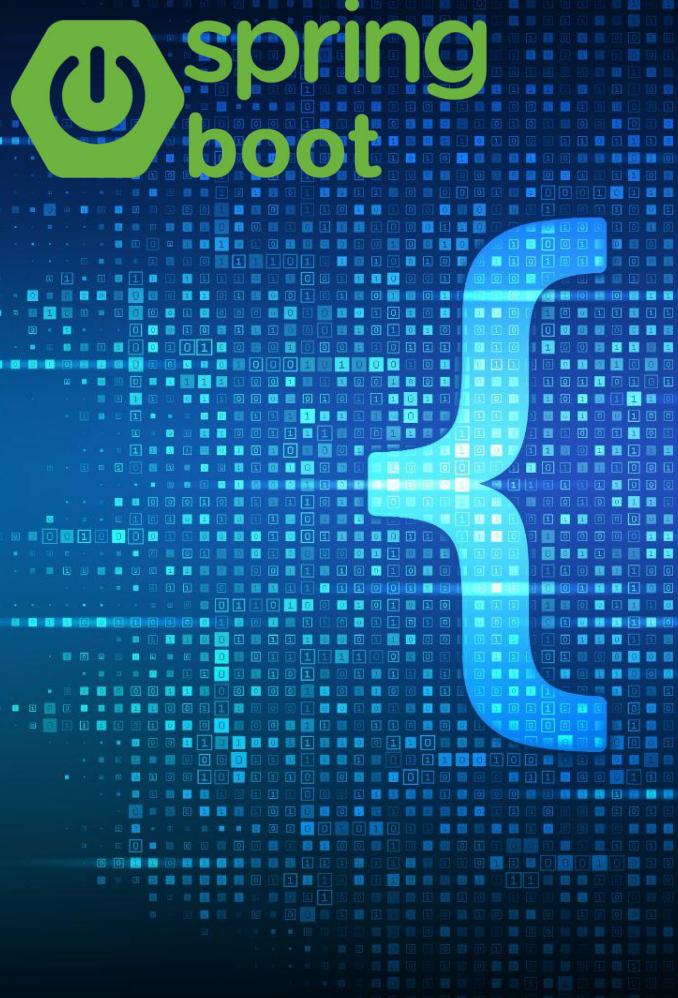
## Notazioni della Binding Validation

- Le annotazioni della binding validation sono utilizzate per definire le regole di validazione dei dati durante il processo di binding (associazione) dei dati in un'applicazione. Le annotazioni più comuni utilizzate per la binding validation sono:
- `@NotNull`: specifica che il valore non può essere nullo.
- `@NotEmpty`: specifica che il valore non può essere vuoto.
- `@NotBlank`: specifica che il valore non può essere vuoto o contenere solo spazi vuoti.
- `@Size`: specifica la dimensione minima e/o massima consentita per il valore.
- `@Min`: specifica il valore minimo consentito per il valore numerico



## Notazioni della Binding Validation

- **@Max:** specifica il valore massimo consentito per il valore numerico.
- **@Pattern:** specifica un pattern regex che il valore deve soddisfare.
- **@Email:** specifica che il valore deve essere un indirizzo email valido.
- **@Valid:** valida un oggetto complesso nidificato.
- **@AssertTrue:** specifica che il valore deve essere true.
- **@AssertFalse:** specifica che il valore deve essere false.
- **@DecimalMin:** specifica il valore minimo consentito per il valore numerico decimale.
- **@DecimalMax:** specifica il valore massimo consentito per il valore numerico decimale.



## Introduzione alla notazione @InitBinder

- L'annotazione `@InitBinder` è una funzionalità fornita da Spring Framework che consente di personalizzare il processo di inizializzazione del WebDataBinder utilizzato per convertire i dati di input del cliente in oggetti Java durante le richieste HTTP.
- Quando un client invia una richiesta HTTP ad un'applicazione web sviluppata con Spring, i parametri di quella richiesta vengono automaticamente estratti e mappati ai parametri del metodo del controller.
- Tuttavia, a volte si desidera personalizzare il processo di estrazione e mappatura dei parametri per soddisfare specifiche esigenze dell'applicazione.
- L'utilizzo della notazione `@InitBinder` consente di configurare un metodo all'interno di un controller per modificare il comportamento predefinito di Spring nella gestione dei parametri.

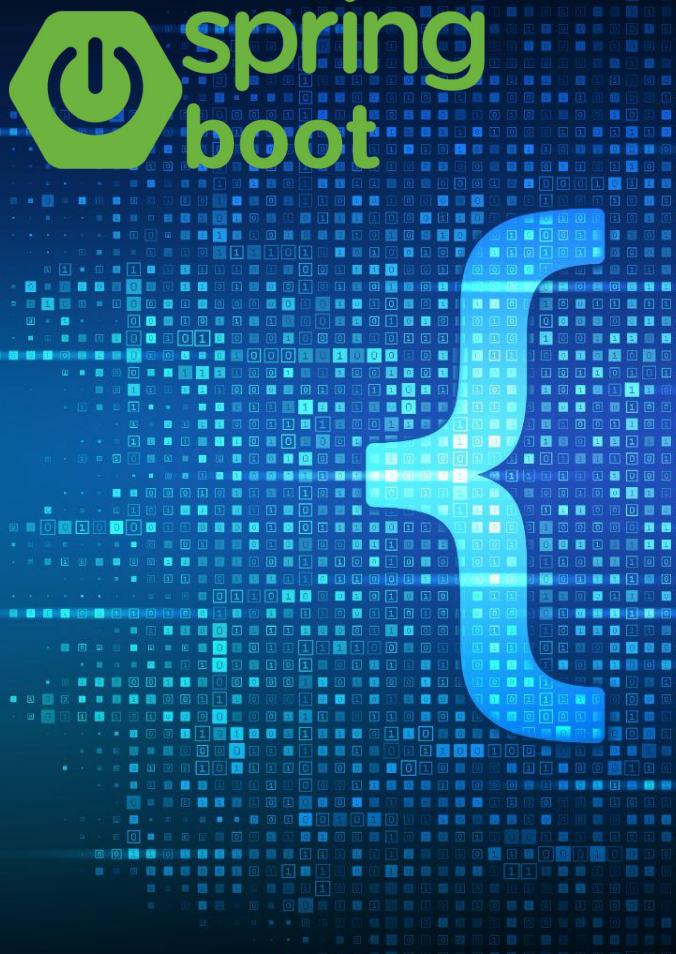


## Introduzione alla notazione @InitBinder

- Il metodo contrassegnato con `@InitBinder` viene invocato prima di ogni richiesta che coinvolge il controller, consentendo di eseguire operazioni di inizializzazione o configurazione necessarie per i parametri.
- Nel metodo annotato con `@InitBinder`, è possibile applicare personalizzazioni al WebDataBinder, ad esempio:
  - **Regole di validazione:** E' possibile aggiungere validatori per la validazione dei dati.
  - **Conversioni personalizzate:** E' possibile registrare convertitori personalizzati per i tipi di dati.
  - **Formati di data:** E' possibile configurare i formati di data desiderati.
  - **Disabilitazione di campi:** E' possibile disabilitare la conversione per determinati campi.

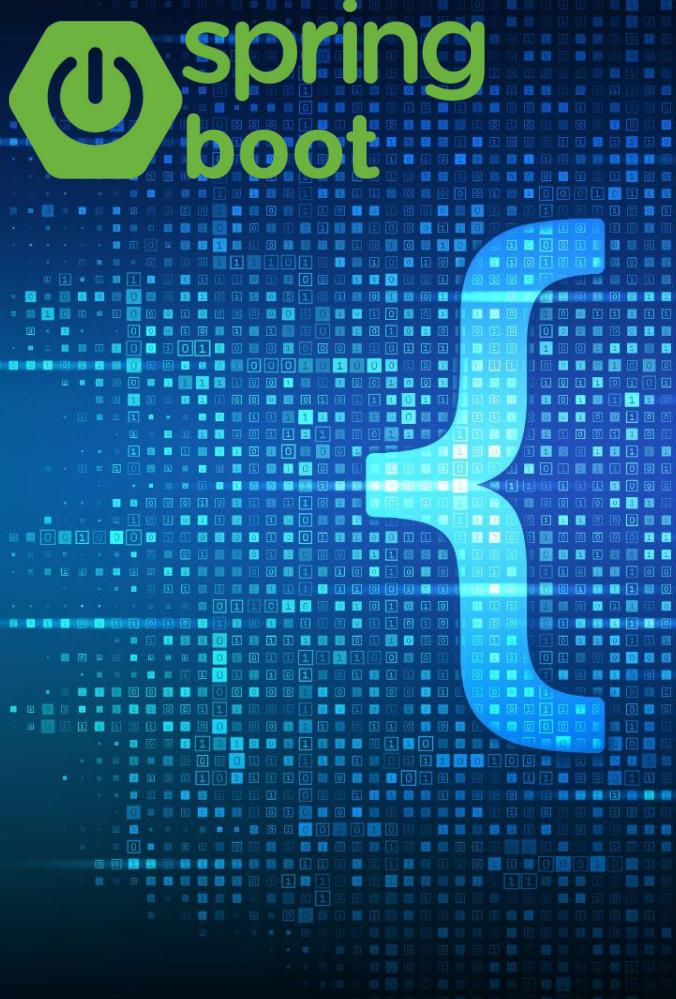
# Spring Boot 3

Introduzione alla sicurezza in  
Spring Boot 3



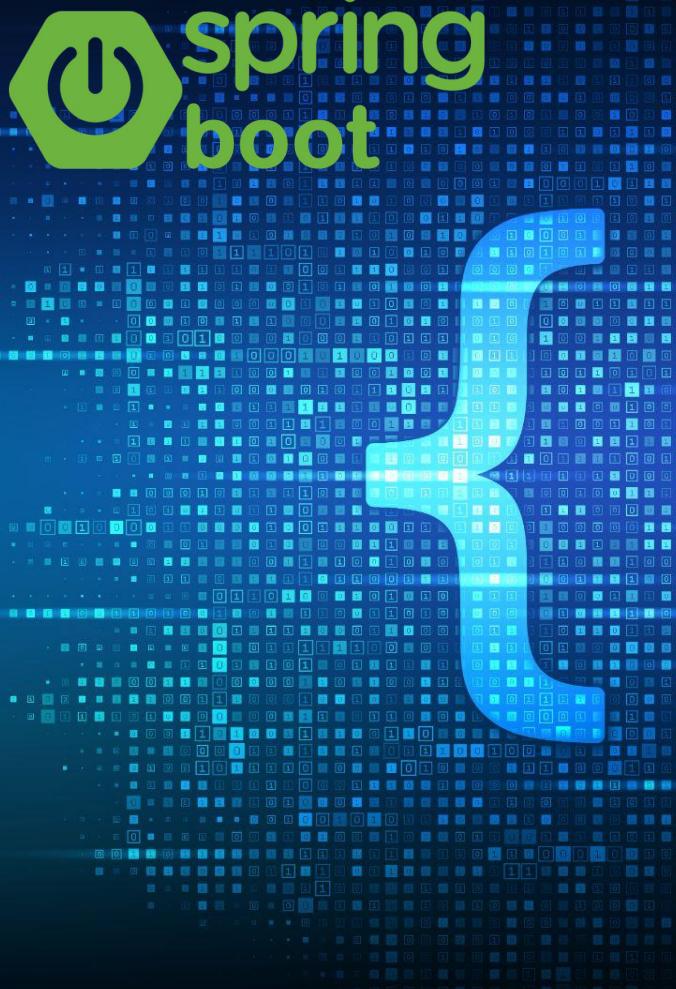
## Introduzione alla sicurezza

- La sicurezza delle web app in Spring Boot è un aspetto cruciale per **proteggere le risorse e garantire che solo gli utenti autorizzati possano accedere alle risorse disponibili.**
- Spring Boot offre un'ampia gamma di funzionalità e strumenti per implementare la sicurezza delle web app.
- Una panoramica dei principali concetti e funzionalità di sicurezza offerti da Spring Boot:
- **Autenticazione:** L'autenticazione è il processo di **verifica dell'identità dell'utente**. Spring Boot supporta diversi meccanismi di autenticazione, come l'autenticazione basata su form, l'autenticazione basata su token, l'autenticazione basata su certificati, l'autenticazione OAuth2 e altro ancora. E' possibile configurare l'autenticazione nel progetto Spring Boot utilizzando le classi di configurazione e le annotazioni appropriate.



## Introduzione alla sicurezza

- **Autorizzazione:** L'autorizzazione è il processo di concedere o negare l'accesso alle risorse in base ai ruoli e ai privilegi dell'utente autenticato. In Spring Boot, è possibile utilizzare le annotazioni come `@PreAuthorize` o `@Secured` per specificare quali ruoli o autorizzazioni sono richiesti per accedere a un endpoint specifico. E' possibile anche personalizzare la logica di autorizzazione utilizzando l'interfaccia `AccessDecisionVoter`.
- **Protezione CSRF:** Il Cross-Site Request Forgery (CSRF) è un attacco in cui un utente viene ingannato per eseguire azioni non desiderate su un'applicazione web senza il suo consenso. Spring Boot fornisce la protezione CSRF di default attraverso l'uso di token CSRF generati automaticamente. Questo protegge le tue API contro attacchi CSRF.



## Introduzione alla sicurezza

- **Gestione delle sessioni:** La gestione delle sessioni è un aspetto importante della sicurezza delle applicazioni web. Spring Boot fornisce diverse opzioni per la gestione delle sessioni, inclusa la gestione delle sessioni lato server e l'utilizzo di token JWT (JSON Web Token) per la gestione delle sessioni stateless.
- **Protezione delle risorse statiche:** Spring Boot consente di proteggere le risorse statiche come file CSS, immagini e pagine HTML da accessi non autorizzati. Puoi configurare facilmente le regole di accesso alle risorse statiche utilizzando la classe `WebSecurityConfigurerAdapter` e le configurazioni appropriate.
- **Log di sicurezza:** Spring Boot fornisce funzionalità per il logging di eventi di sicurezza, consentendo di tenere traccia delle attività di autenticazione, autorizzazione e altri eventi correlati alla sicurezza dei web services.



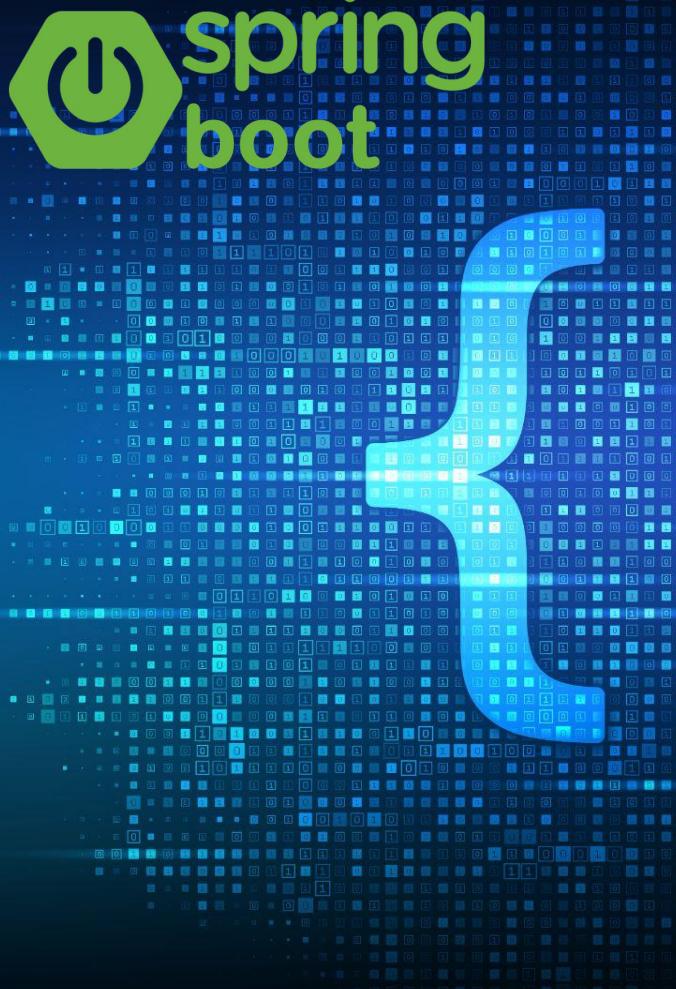
## Introduzione alla sicurezza

- **Test di sicurezza:** Spring Boot offre anche strumenti e librerie per testare la sicurezza delle applicazioni, come ad esempio `MockMvc` per simulare le richieste HTTP e verificare che le regole di sicurezza siano correttamente implementate.



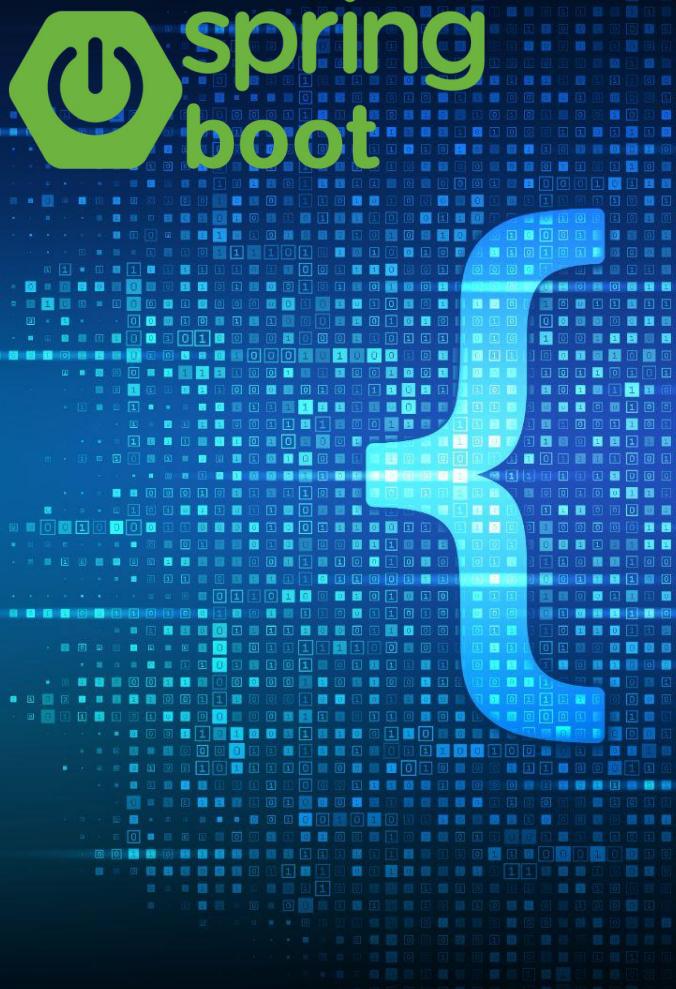
## Introduzione al Spring Security

- Spring Security è un **framework di sicurezza** per applicazioni web basate su tecnologie Java.
- Questo framework **fornisce strumenti e funzionalità** per gestire l'autenticazione, l'autorizzazione e altri aspetti correlati alla sicurezza delle applicazioni.
- Lo Spring Security si basa su un **modello di sicurezza basato sui filtri**, in cui ogni richiesta viene analizzata da una serie di filtri che **gestiscono l'autenticazione, l'autorizzazione e altre operazioni di sicurezza**.
- Questi filtri possono essere **configurati per gestire diversi aspetti di sicurezza**, come la gestione delle credenziali degli utenti, la protezione delle risorse, la gestione delle sessioni e altre funzioni correlate alla sicurezza.



## Introduzione al Spring Security

- Lo Spring Security fornisce anche un sistema di gestione delle autorizzazioni basato sui ruoli degli utenti, consentendo di definire regole di accesso alle risorse dell'applicazione in base al ruolo dell'utente.
- Questo permette di definire facilmente quali azioni possono essere eseguite da un determinato utente in base ai suoi privilegi.
- Inoltre, lo Spring Security offre funzionalità avanzate come la protezione da attacchi di sicurezza comuni come XSS (Cross-Site Scripting), CSRF (Cross-Site Request Forgery) e SQL injection, il supporto per l'integrazione con sistemi di autenticazione esterni come LDAP o database, e la possibilità di personalizzare e estendere le funzionalità di sicurezza in base alle esigenze specifiche dell'applicazione.



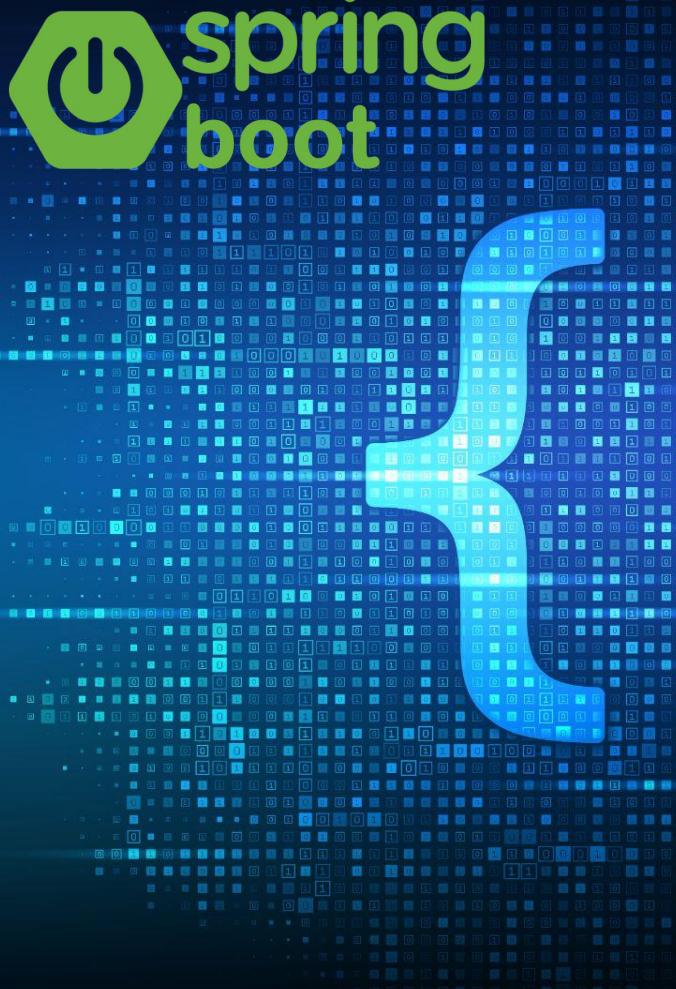
## Configurare la sicurezza

- Per configurare la sicurezza dei web services in Spring Boot, è necessario seguire i seguenti passaggi:
- **Aggiungere dipendenze:** Nel file `pom.xml`, è necessario aggiungere le dipendenze per Spring Security e le altre dipendenze correlate.
- **Configurare Spring Security:** Nel codice di configurazione di Spring Boot, è necessario creare una classe di configurazione. Questa classe fornirà i metodi per configurare l'autenticazione e l'autorizzazione.
- **Configurare l'autenticazione:** è possibile specificare come viene effettuata l'autenticazione degli utenti. Ad esempio, è possibile utilizzare un database per archiviare le credenziali degli utenti o fornire un sistema di autenticazione personalizzato.



## Configurare la sicurezza

- **Configurare l'autorizzazione:** è possibile specificare come vengono gestiti i permessi degli utenti. Ad esempio, è possibile configurare i ruoli degli utenti e definire quali pagine o servizi possono accedere.
- **Proteggere le risorse:** È possibile proteggere le risorse specificando i ruoli necessari per accedere a determinate API o pagine. Ad esempio, è possibile utilizzare le annotazioni `@PreAuthorize` o `@Secured` per specificare i permessi richiesti per le endpoint dei web service.
- **Gestire sessioni e token:** Spring Security offre diverse opzioni per gestire la gestione delle sessioni e l'uso di token di autenticazione.

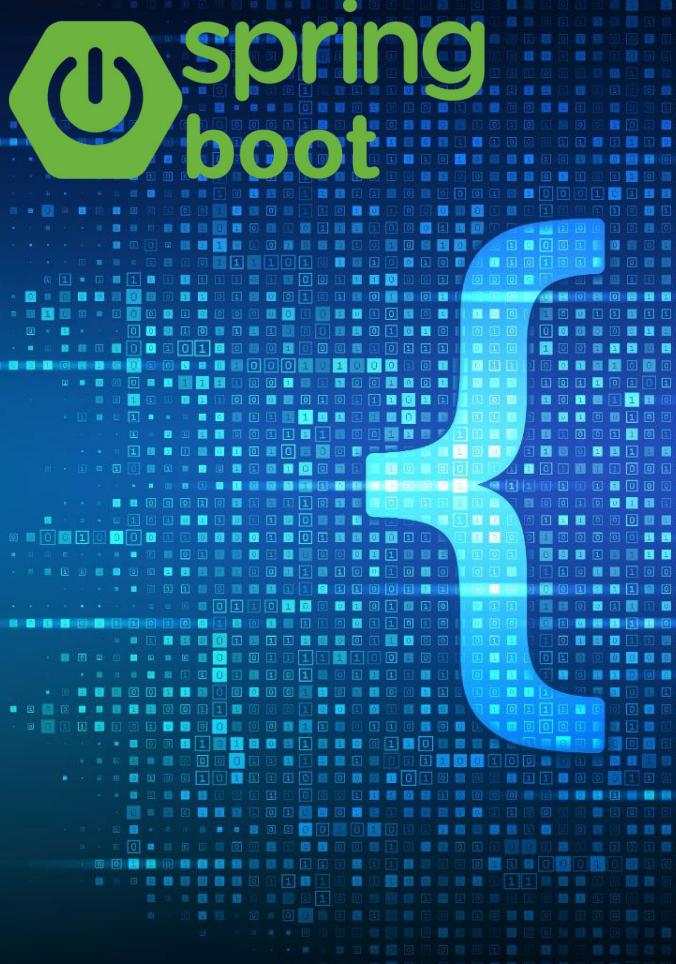


## Configurare la sicurezza

- **Gestire le eccezioni:** È possibile gestire le eccezioni di sicurezza personalizzando la classe `AuthenticationEntryPoint` o altre classi di gestione delle eccezioni fornite da Spring Security. In questo modo è possibile controllare come vengono gestiti gli errori di autenticazione o autorizzazione.
- **La Basic Authentication** è un meccanismo di autenticazione molto comune e semplice da implementare. Nella Basic Authentication, le credenziali dell'utente (username e password) vengono inviate come parte dell'intestazione della richiesta HTTP. Spring Boot fornisce il supporto per la Basic Authentication tramite le sue funzionalità di sicurezza.

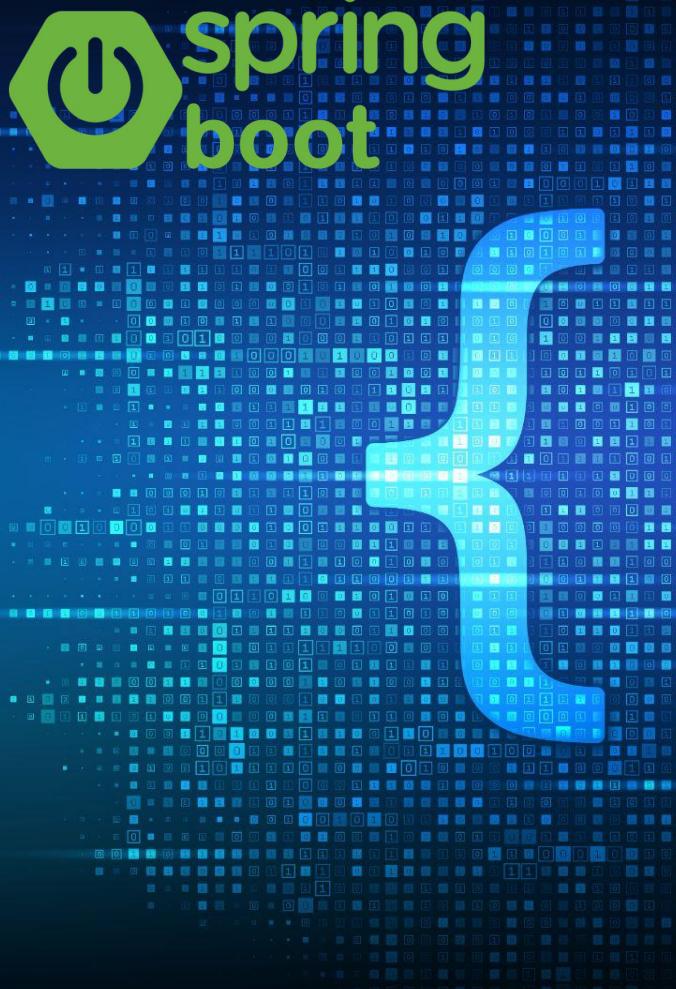
# Spring Boot 3

## Introduzione al Thymeleaf in Spring Boot 3



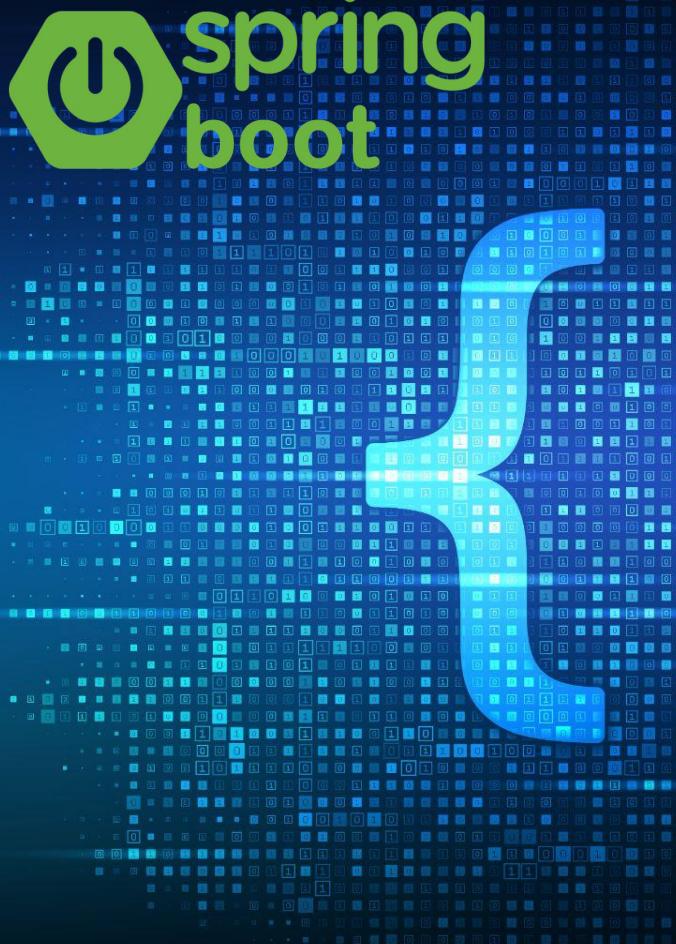
## Che cosa è il Thymeleaf

- Thymeleaf è un motore di template Java-based open source sviluppato principalmente per lo sviluppo di applicazioni web. È ampiamente utilizzato nel framework Spring Boot per la generazione di pagine web dinamiche.
- Una caratteristica distintiva di Thymeleaf è la sua natura basata su HTML. I template utilizzati in Thymeleaf sono semplici file HTML che possono essere visualizzati e modificati anche nel browser web. Questo rende più facile per i progettisti e gli sviluppatori collaborare sullo sviluppo delle pagine.
- Thymeleaf utilizza un'approccio di rendering lato server, il che significa che le pagine vengono generate dal server e inviate al browser dell'utente finale. Questo consente a Thymeleaf di fornire funzionalità avanzate come la gestione delle variabili, l'iterazione e la condizionalità direttamente nel template.



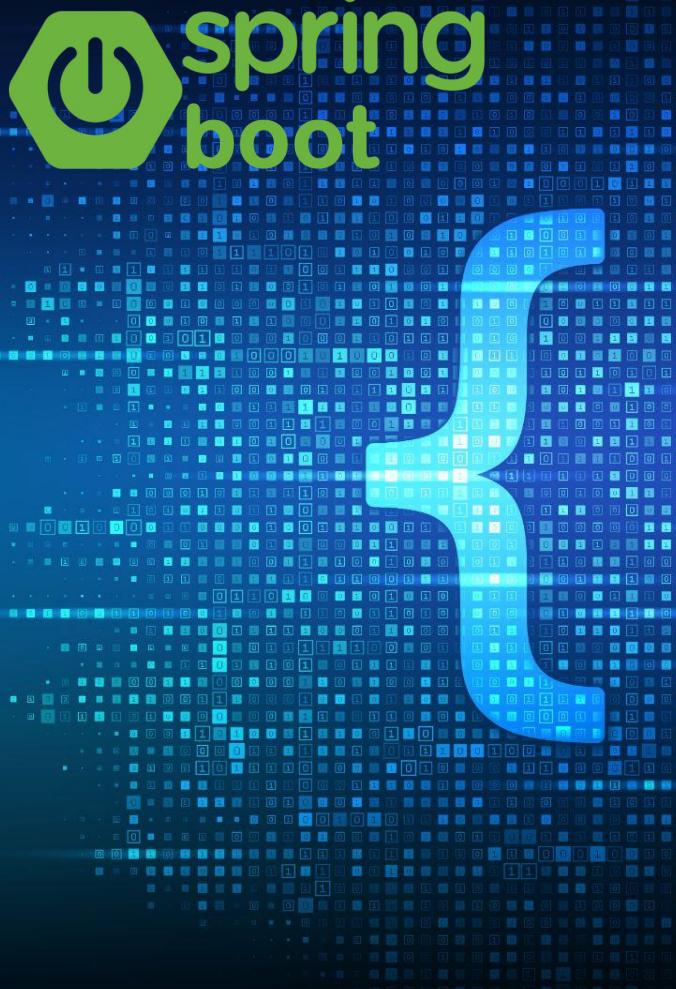
## Usare il Thymeleaf

- Spring Boot fornisce un'ampia integrazione con Thymeleaf, semplificando l'utilizzo di Thymeleaf nelle applicazioni Spring.
- In una tipica applicazione Spring Boot che utilizza Thymeleaf, i template possono essere posizionati nella directory "`src/main/resources/templates`" e saranno automaticamente rilevati e utilizzati dal motore di template Thymeleaf.
- Per utilizzare Thymeleaf in un'applicazione Spring Boot, è necessario configurare una dipendenza per Thymeleaf nel file "`pom.xml`" o "`build.gradle`" del progetto.
- Inoltre, è possibile configurare le impostazioni di Thymeleaf nel file "`application.properties`" o "`application.yml`" per fornire ulteriori configurazioni come il prefisso e il suffisso dei template.



## Che cosa è il Thymeleaf

- Thymeleaf offre molte funzionalità utili, come la gestione di attributi, il collegamento di eventi, la formattazione dei dati, l'accesso alle risorse e l'internazionalizzazione.
- I tag principali utilizzati in Thymeleaf sono utilizzati per inserire espressioni Thymeleaf nel tuo HTML o XML per la gestione dinamica dei dati.
- Di seguito sono elencati alcuni dei tag principali utilizzati in Thymeleaf:
  - Thymeleaf Standard Expression =
  - `<p th:text="${espressione}">` - Mostra il valore dell'espressione nel testo del paragrafo.
  - `<input th:value="${valore}">` - Imposta il valore dell'input con l'espressione.



## Che cosa è il Thymeleaf

- Iterazione e condizioni =  
`<th:each="element : ${lista}">` - Esegui un ciclo su una lista.
- `<div th:if="${condizione}">` - Condizione per la visualizzazione di un elemento.
- `<div th:unless="${condizione}">` - Condizione inversa.
- `<div th:switch="${condizione}">` - Dichiara un'espressione switch e contiene più elementi th:case tra cui scegliere in base al valore switch.
- `<p th:case=="ADMIN">` - Definisce un blocco case all'interno di un elemento th:switch,
- Inclusione di modelli:
- `<div th:include="fragments/fragmento">` - Include un frammento di template.
- `<div th:replace="fragments/fragmento">` - Sostituisce un elemento con un frammento di template.
- `<div th:insert="fragments/fragmento">` - Inserisce un frammento di template.



## Che cosa è il Thymeleaf

- Link  
`<a th:href="${link}">` - Imposta il valore dell'attributo href per il tag (`<a>`).
- `` - specifica la sorgente di un'immagine o di altri elementi multimediali.
- Attributi condizionali:
  - `<a th:href="${link}" th:if="${mostraLink}">` - Aggiunge un attributo solo se la condizione è vera.
  - `<a th:href="${link}" th:unless="${mostraLink}">` - Aggiunge un attributo solo se la condizione è falsa.
- Gestione degli errori di validazione:
  - `<span th:errors="${campo}">` - Mostra gli errori di validazione per un campo specifico.
  - `<div th:if="#{fields.hasErrors()}">` - Mostra un blocco solo se ci sono errori di validazione.

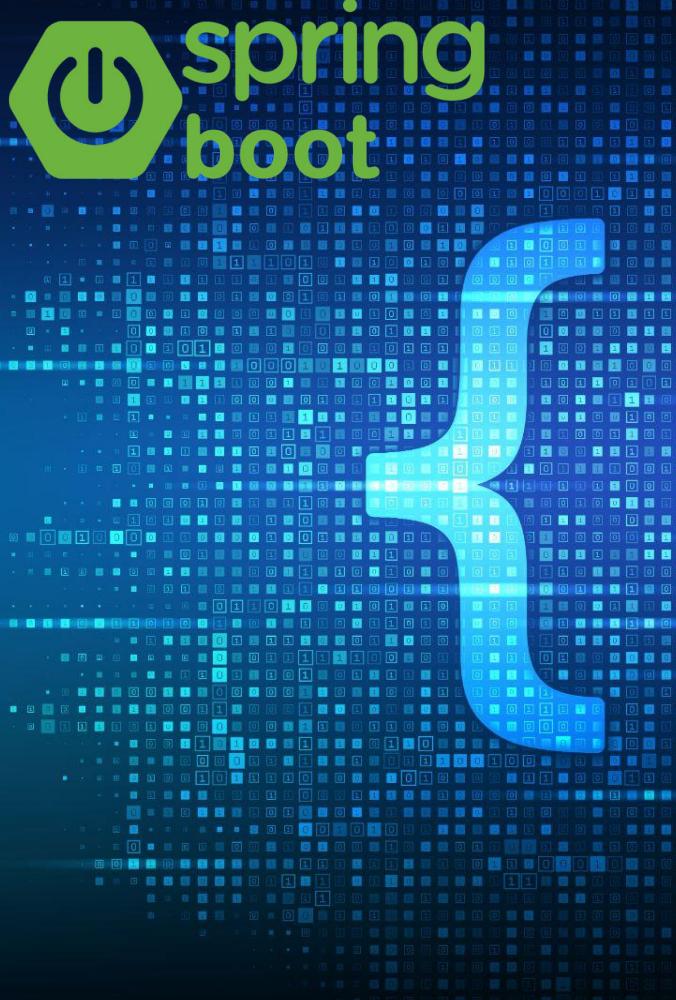


## Che cosa è il Thymeleaf

- Accesso alle variabili di contesto:  
`<p th:object="${oggetto}">` - Imposta un oggetto come oggetto di contesto.
- `<span th:with="varName=${oggetto.proprietà}">` - Crea una variabile nel contesto.
- Lingua e internazionalizzazione:  
`<span th:text="#{chiave_messaggio}">` - Utilizza messaggi internazionalizzati.
-

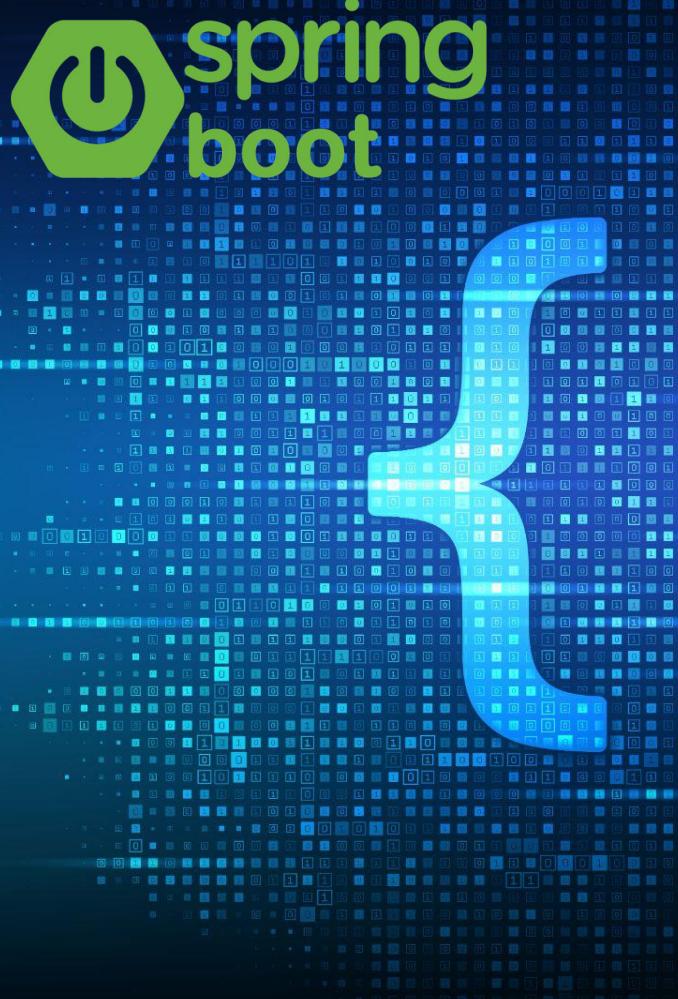
# Spring Boot 3

## Introduzione ai Web Services



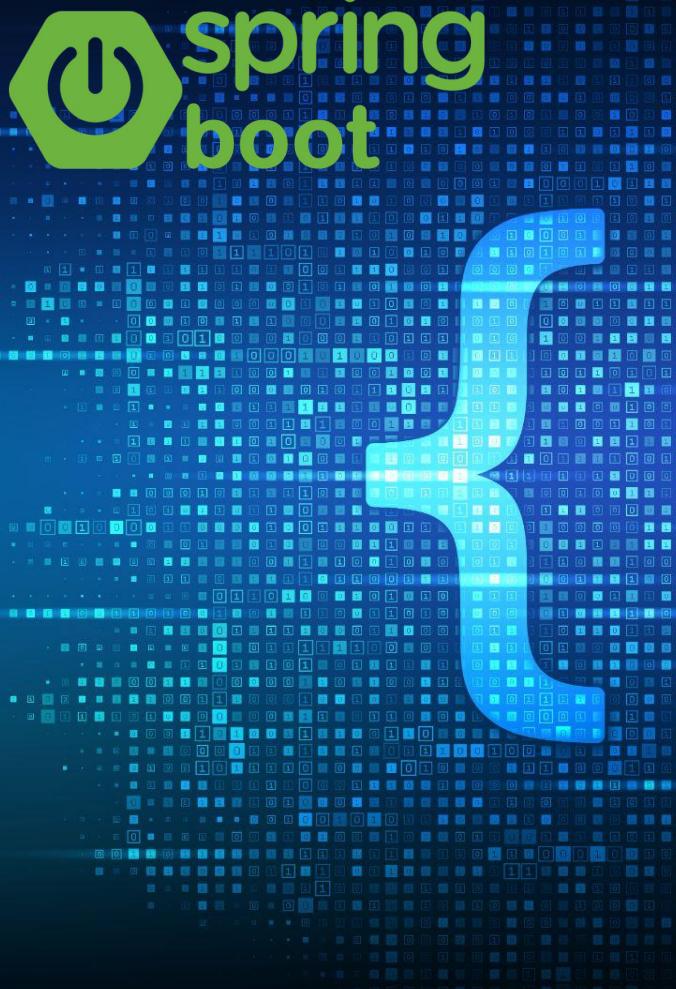
## Cosa sono i web services

- I Web Services sono una tecnologia che consente a diverse applicazioni software di comunicare tra loro attraverso Internet.
- I web services permettono a diverse applicazioni di interagire e lavorare insieme in modo seamless, anche se sono sviluppate con tecnologie e linguaggi di programmazione differenti. Questa interazione avviene attraverso richieste e risposte tra un client e un server web.
- I Web Services sono basati su un insieme di standard e protocolli come XML (eXtensible Markup Language), SOAP (Simple Object Access Protocol), WSDL (Web Services Description Language) e il REST che definiscono come le applicazioni possono comunicare tra loro.



## Cosa sono i web services

- Un web service è composto da **tre componenti principali**: il provider, il registro e il consumatore.
- Il provider è l'entità che mette a disposizione un **web service**, offrendo funzionalità o servizi che possono essere richiamati da altre applicazioni.
- Il registro è un **repository di metadati** che contiene informazioni sulle funzionalità e i servizi offerti dai vari **web services disponibili**. Il registro può essere utilizzato dai consumatori per trovare e scoprire i web services.
- Il consumatore è l'applicazione che utilizza il **web service** per accedere alle funzionalità offerte dal provider. Il consumatore invia una richiesta al provider, che elabora la richiesta e restituisce una risposta.



## Cosa sono i web services

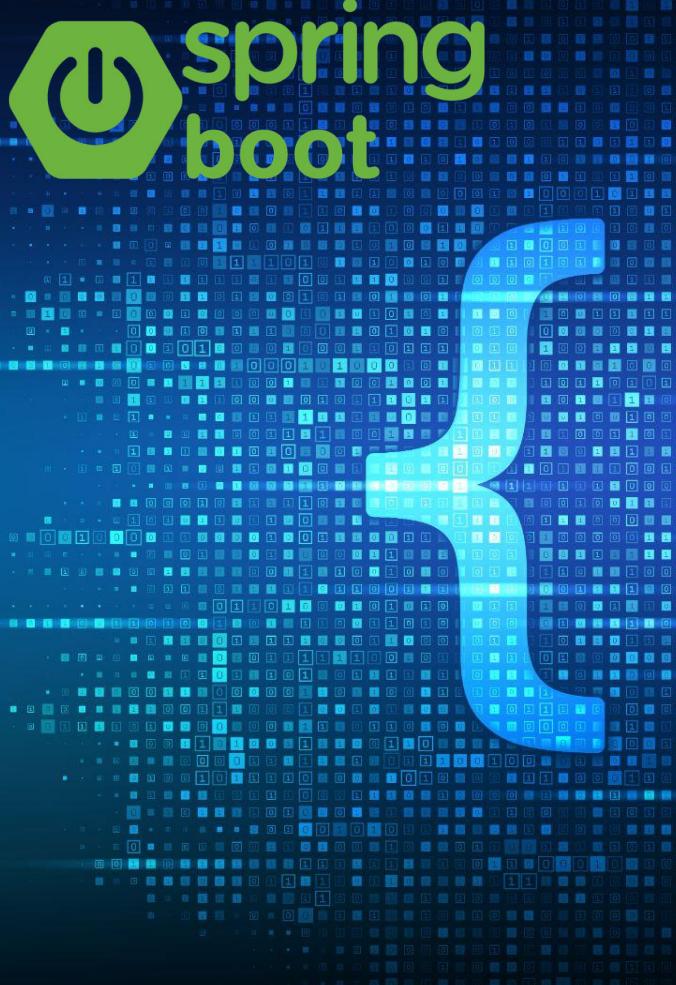
- Uno dei principali standard utilizzati per la comunicazione tra applicazioni è il protocollo **HTTP** (Hypertext Transfer Protocol).
- La comunicazione tra le applicazioni avviene **attraverso scambi di messaggi HTTP che contengono informazioni codificate**.
- Uno dei vantaggi principali dei Web Services è la loro **interoperabilità e l'indipendenza dalle piattaforme su cui le applicazioni sono eseguite**.
- Ciò significa che l'applicazione del client può essere scritta in un linguaggio di programmazione diverso dall'applicazione del server che risponde alla richiesta, e le due applicazioni possono essere eseguite su sistemi operativi diversi.



## Cosa sono i web services

- I web service utilizzano principalmente **due tecnologie per la comunicazione e l'interscambio dati**:
- SOAP (Simple Object Access Protocol)**: SOAP è un protocollo di messaggistica basato su **XML**. I messaggi SOAP sono scambiati tra il client e il server in formato XML e possono essere trasportati su diversi protocolli di rete come HTTP, SMTP e altri. Un messaggio SOAP contiene informazioni sulle operazioni da eseguire e sui dati da inviare o ricevere. Di seguito è riportato un esempio di struttura di un messaggio SOAP:

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
    <!-- Intestazioni opzionali -->
  </soap:Header>
  <soap:Body>
    <!-- Corpo del messaggio -->
  </soap:Body>
</soap:Envelope>
```



## Cosa sono i web services

- REST (Representational State Transfer): REST è un'architettura basata su risorse che utilizza i verbi HTTP (GET, POST, PUT, DELETE) per operare su di esse. In un'architettura REST, le risorse sono identificate dal **URI** (Uniform Resource Identifier) e le operazioni vengono eseguite **invia**ndole richieste HTTP **ai corrispondenti URI**. I dati vengono solitamente trasmessi in formato **JSON o XML**. Di seguito è riportato un esempio di richiesta e risposta REST:

```
GET /api/products/123 HTTP/1.1
Host: example.com

HTTP/1.1 200 OK
Content-Type: application/json

{
  "id": 123,
  "name": "Product Name",
  "price": 9.99
}
```



## Cosa sono è il REST

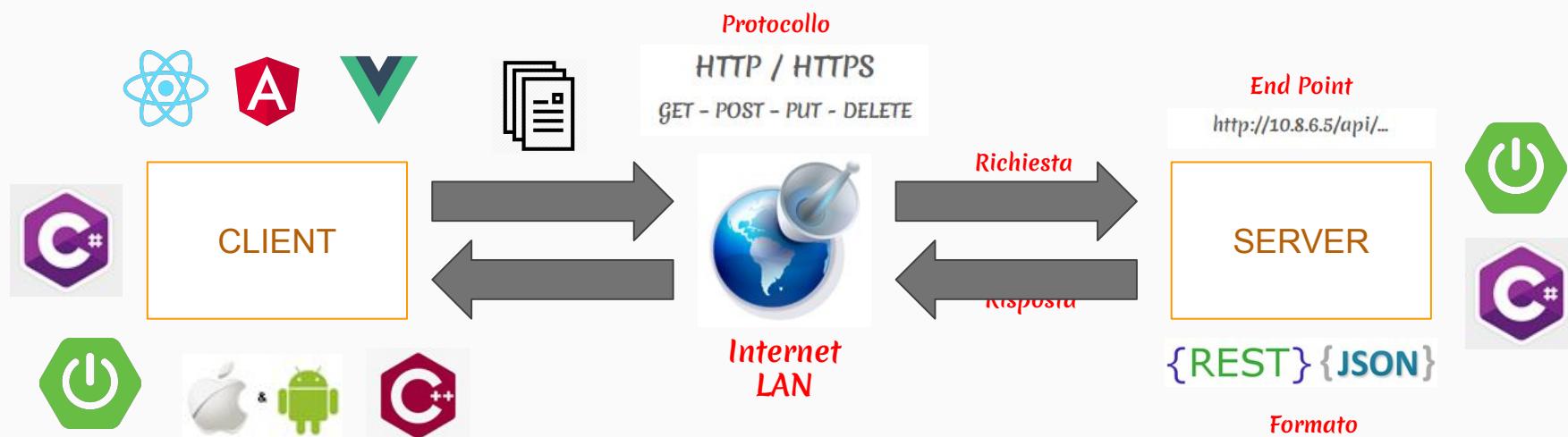
- REST (Representational State Transfer) è un'architettura software basata su un insieme di principi e vincoli che definiscono come i sistemi distribuiti dovrebbero essere progettati e come possono comunicare tra di loro sul web.
- È uno stile architettonico ampiamente adottato per sviluppare servizi web scalabili, leggeri e modulari.
- I principi fondamentali di REST includono:
- Risorse identificabili: ogni risorsa (come un oggetto o un insieme di dati) deve essere identificata univocamente da un **URI** (Uniform Resource Identifier).
- Manipolazione delle risorse tramite rappresentazioni: le risorse possono essere manipolate attraverso le operazioni standard del protocollo HTTP come **GET**, **POST**, **PUT** e **DELETE** utilizzando una rappresentazione dei dati (ad esempio **JSON** o **XML**).



## Cosa sono i web services

- **Comunicazione stateless:** ogni richiesta tra client e server deve contenere tutte le informazioni necessarie per comprendere e soddisfare la richiesta senza bisogno di alcuno stato di sessione salvato dal server.
- **Interfaccia uniforme:** REST definisce un insieme di operazioni standard per manipolare le risorse come GET per leggere una risorsa, POST per creare una nuova risorsa, PUT per aggiornare una risorsa esistente e DELETE per rimuovere una risorsa.
- **Sistema a cache:** REST supporta la memorizzazione nella cache delle risorse per migliorare le prestazioni e ridurre il carico sul server.

# Il funzionamento dei Web Service REST





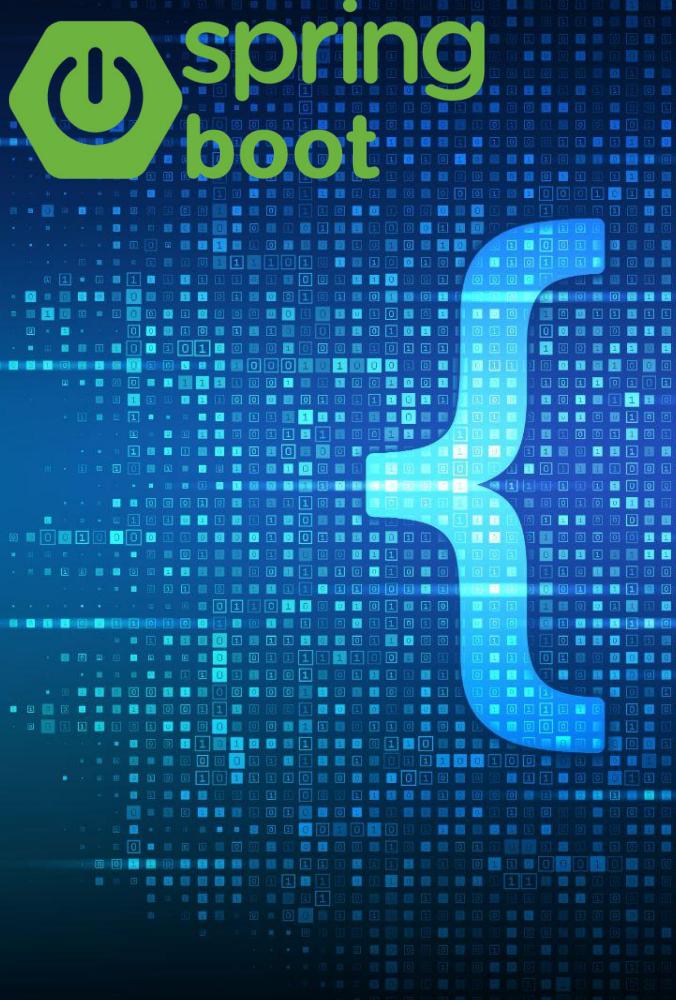
## Cosa sono i web services

- I web service possono essere utilizzati per una vasta gamma di applicazioni, ad esempio:
- **Integrazione di sistemi:** I web service consentono a diverse applicazioni aziendali di condividere dati e funzionalità, facilitando l'integrazione dei sistemi informativi.
- **Sviluppo di applicazioni mobili:** Le app mobili possono utilizzare i web service per accedere a dati e funzionalità remote, consentendo l'interazione con server e sistemi backend.
- **Integrazione di API di terze parti:** Le API di terze parti possono essere esposte come web service, consentendo l'accesso a funzionalità e dati forniti da altri servizi online.



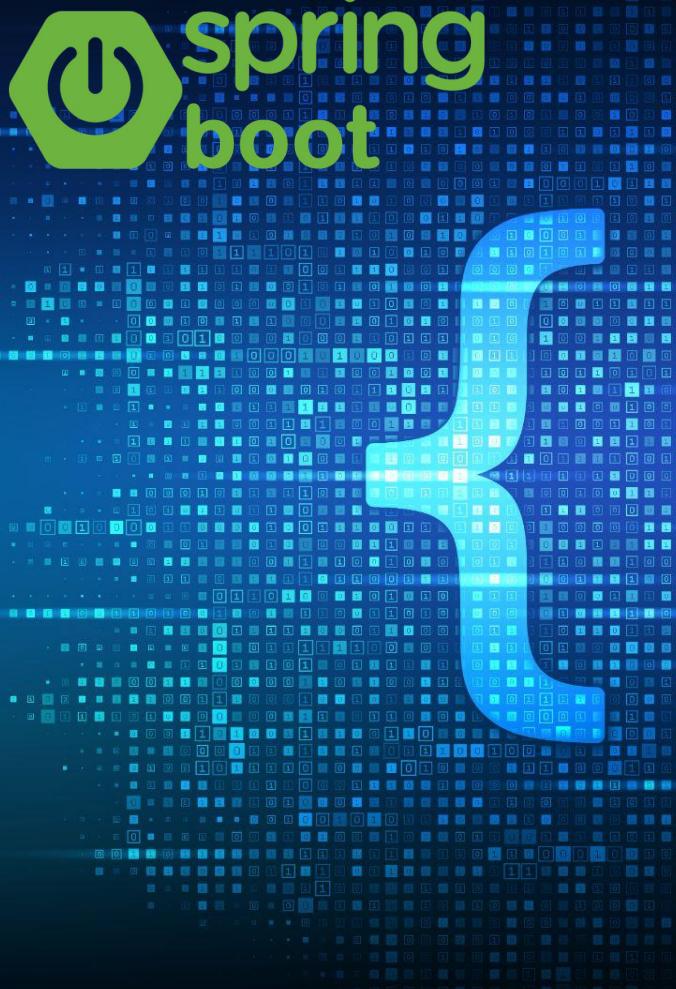
## Cosa sono i web services

- **Sistemi di pagamento:** I web service possono essere utilizzati per integrare sistemi di pagamento online, consentendo transazioni sicure tra le applicazioni e i sistemi di pagamento.
- **Sistemi di messaggistica:** I web service possono essere utilizzati per inviare e ricevere messaggi tra diverse applicazioni o componenti di un sistema distribuito.
- In conclusione, i web service sono una **tecnologia fondamentale per la comunicazione tra applicazioni distribuite**.
- Consentono l'integrazione e lo scambio di dati tra sistemi eterogenei utilizzando protocolli standard come SOAP e REST, facilitando lo sviluppo di applicazioni complesse e interconnesse.



## Introduzione ai test in Spring Boot

- Gli unit test in Spring Boot sono test che si concentrano sull'esecuzione e sulla verifica del corretto funzionamento di singole unità di codice, come classi, metodi o funzioni, in isolamento dagli altri componenti dell'applicazione.
- Gli unit test sono fondamentali per garantire che le singole unità di codice siano corrette e si comportino come previsto.
- I test aiutano a identificare eventuali bug o problemi di funzionalità nel codice, consentendo agli sviluppatori di individuare e correggere tali problemi prima del rilascio dell'applicazione.



## Introduzione ai test in Spring Boot

- Nel dettaglio gli unit test in Spring Boot sono composte:
- **Framework di testing:** Per scrivere unit test in Spring Boot, viene spesso utilizzato il framework di testing **JUnit**, che è ampiamente utilizzato nella comunità di sviluppo Java per testare le unità di codice. JUnit fornisce un'ampia gamma di annotazioni e metodi di asserzione per facilitare la scrittura e l'esecuzione dei test.
- **Annotazione @SpringBootTest:** Per avviare un contesto di applicazione Spring Boot completo durante i test, è possibile utilizzare l'annotazione **@SpringBootTest**. Questa annotazione carica l'intera applicazione e tutte le sue dipendenze, consentendo di eseguire test di integrazione più complessi che coinvolgono più componenti.



## Introduzione ai test in Spring Boot

- **Annotazione `@Test`:** Per definire un metodo come test, viene utilizzata l'annotazione `@Test` fornita da JUnit. Questa annotazione indica al framework di testing che il metodo è un test da eseguire.
- **Configurazione dei dati di test:** È comune avere dati di test specifici per i test unitari e di integrazione. Spring Boot fornisce diverse opzioni per configurare i dati di test, come ad esempio l'utilizzo di un database in memoria o la creazione di dati mock. È possibile utilizzare annotazioni come `@Before`, `@BeforeEach` o `@BeforeAll` per eseguire il setup delle risorse necessarie per il test, ad esempio creare istanze delle classi da testare o configurare l'ambiente di test.



## Introduzione ai test in Spring Boot

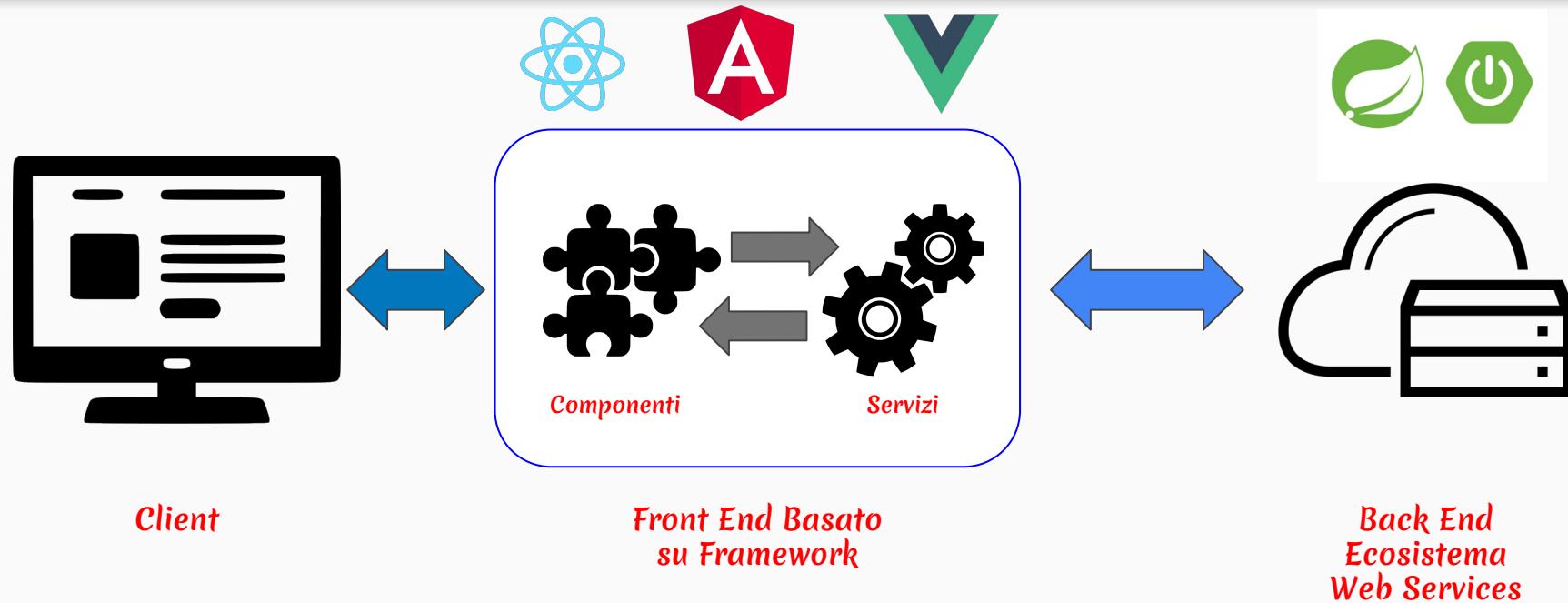
- **Esecuzione dei test:** Durante l'esecuzione di un test, vengono chiamati i metodi da testare e vengono eseguite le asserzioni per verificare i risultati attesi. Le asserzioni possono essere eseguite utilizzando metodi forniti da JUnit, come `assertEquals`, `assertTrue`, `assertNotNull`, ecc., per confrontare i valori attesi con quelli ottenuti durante l'esecuzione del test.
- **Pulizia delle risorse:** Dopo l'esecuzione di un test, è importante liberare le risorse utilizzate durante il test, come la chiusura di connessioni al database o la liberazione della memoria. Per fare ciò, è possibile utilizzare annotazioni come `@After`, `@AfterEach` o `@AfterAll` per eseguire le operazioni di pulizia necessarie.

# Spring Boot 3



## Introduzione ai Micro Servizi

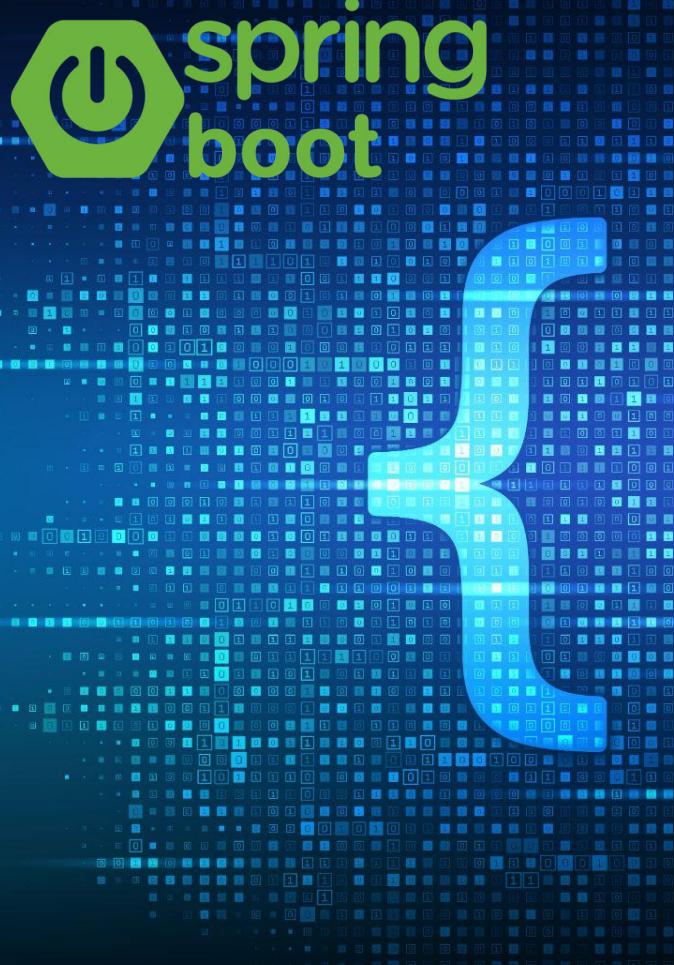
# Schema delle Moderne Web App





## Introduzione all'architettura a micro servizi

- L'architettura a microservizi è un approccio allo sviluppo di software in cui **un'applicazione** viene suddivisa in componenti autonomi, noti come **microservizi**.
- Questi microservizi sono **unità indipendenti** con funzionalità specifiche e possono essere sviluppati, testati e distribuiti **in modo autonomo e indipendente** uno dagli altri.
- Con questa architettura anziché avere **un'unica applicazione monolitica** che gestisce tutte le funzionalità, ogni microservizio **sarà responsabile** di **una parte specifica** dell'applicazione.
- Ad esempio, si potrebbe avere un microservizio per la gestione degli utenti, un altro per la gestione dei pagamenti e un altro ancora per l'elaborazione delle richieste di ricerca.



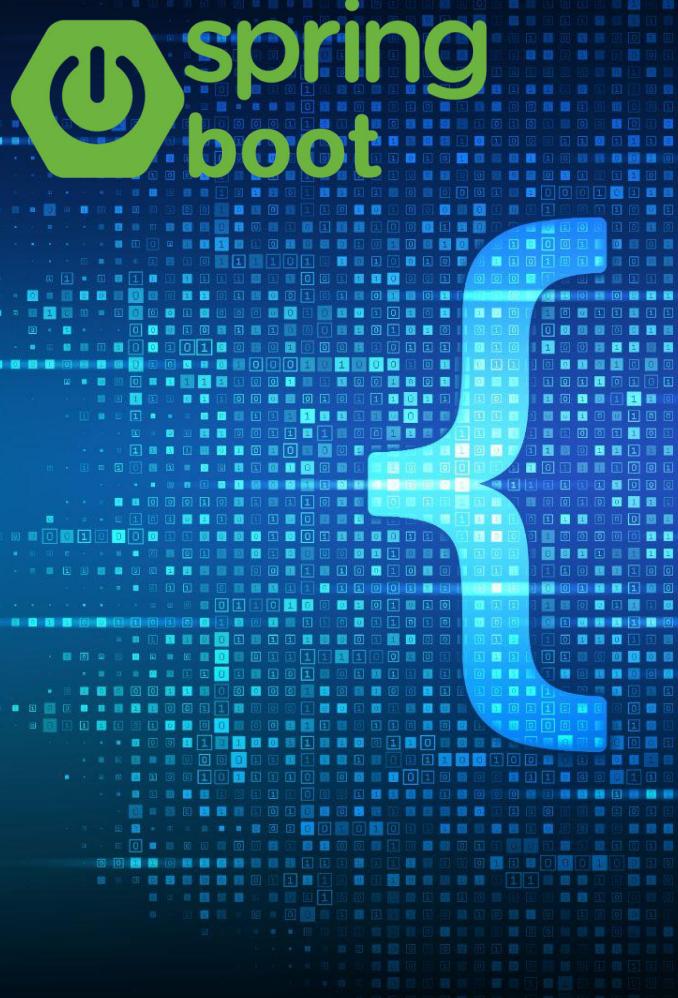
## Introduzione all'architettura a micro servizi

- L'architettura a microservizi favorisce la scalabilità, la manutenibilità e la flessibilità del software. Ecco alcune caratteristiche chiave dell'architettura a microservizi:
- **Servizi Indipendenti:** In un'architettura a microservizi, ogni componente dell'applicazione è un servizio autonomo, noto come microservizio. Ogni microservizio è responsabile esclusivo di una specifica funzionalità dell'applicazione. Questo favorisce la separazione dei compiti e la modularità.
- **Comunicazione via API:** I microservizi comunicano tra loro attraverso API (Application Programming Interface) ben definite. Le API consentono ai servizi di scambiare dati e richieste, consentendo la cooperazione tra di loro.
- **Distribuzione Indipendente:** I microservizi possono essere sviluppati, testati e distribuiti in modo indipendente. Questo significa che gli aggiornamenti o le correzioni di bug per un microservizio non richiedono la distribuzione dell'intera applicazione.



## Introduzione all'architettura a micro servizi

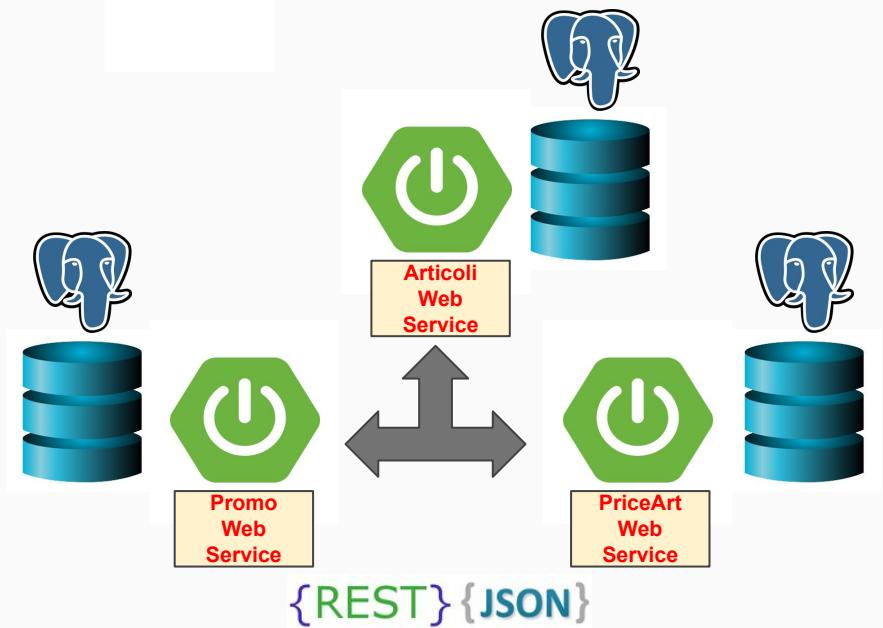
- **Scalabilità Granulare:** La scalabilità è più granulare in un'architettura a microservizi. È possibile scalare solo i servizi che richiedono più risorse senza dover scalare l'intera applicazione.
- **Tecnologie Eterogenee:** Poiché i microservizi sono indipendenti e interagiscono tramiti specifici protocolli, è possibile utilizzare diverse tecnologie e stack tecnologici per il loro sviluppo, a seconda delle esigenze specifiche del servizio.
- **Resilienza:** Gli errori in uno specifico microservizio non dovrebbero influenzare l'intera applicazione. Gli altri microservizi dovrebbero essere in grado di funzionare in modo resiliente anche in presenza di errori.
- **Gestione delle Dati:** La gestione dei dati in un'architettura a microservizi può essere complessa. Spesso, ciascun microservizio ha il proprio database o sistema di memorizzazione dati o controlla una parte specifica della base dati centralizzata.



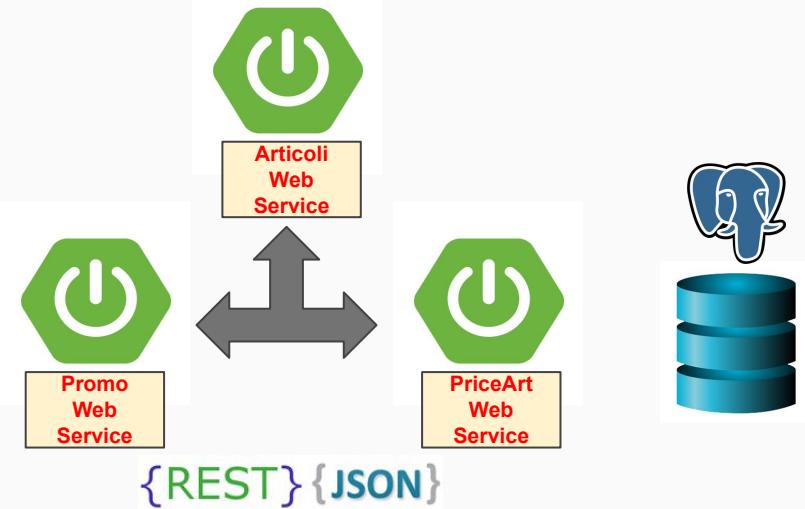
## Introduzione all'architettura a micro servizi

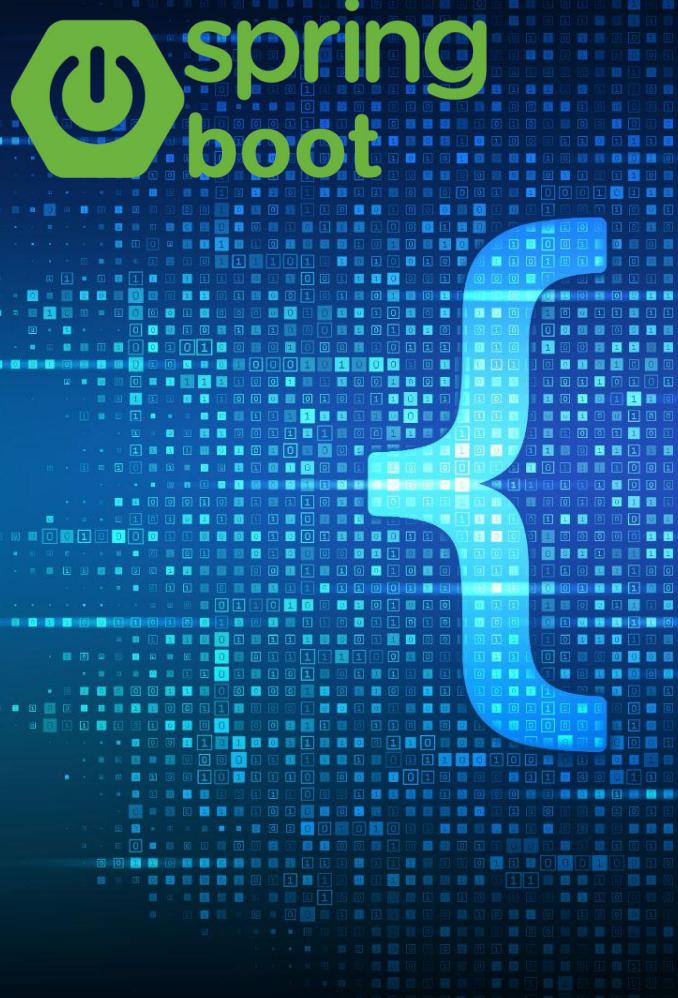
- **Complessità della Gestione:** Sebbene l'architettura a microservizi offra molte vantaggi, può comportare una maggiore complessità nella gestione, nell'implementazione della sicurezza e nell'orchestrazione dei servizi.
- **DevOps e Automazione:** L'automazione è fondamentale per la gestione dei microservizi. L'approccio DevOps è spesso utilizzato per automatizzare la distribuzione, il monitoraggio e la gestione dei servizi.
- **Cultura Organizzativa:** L'architettura a microservizi richiede una buona organizzazione e una gestione adeguata, poiché comporta la gestione di numerosi servizi indipendenti. È importante definire i confini dei microservizi (dominio) in modo chiaro e garantire che ci sia una comunicazione adeguata tra di loro.

# La Base Dati Individuale



# La Base Dati Centrale





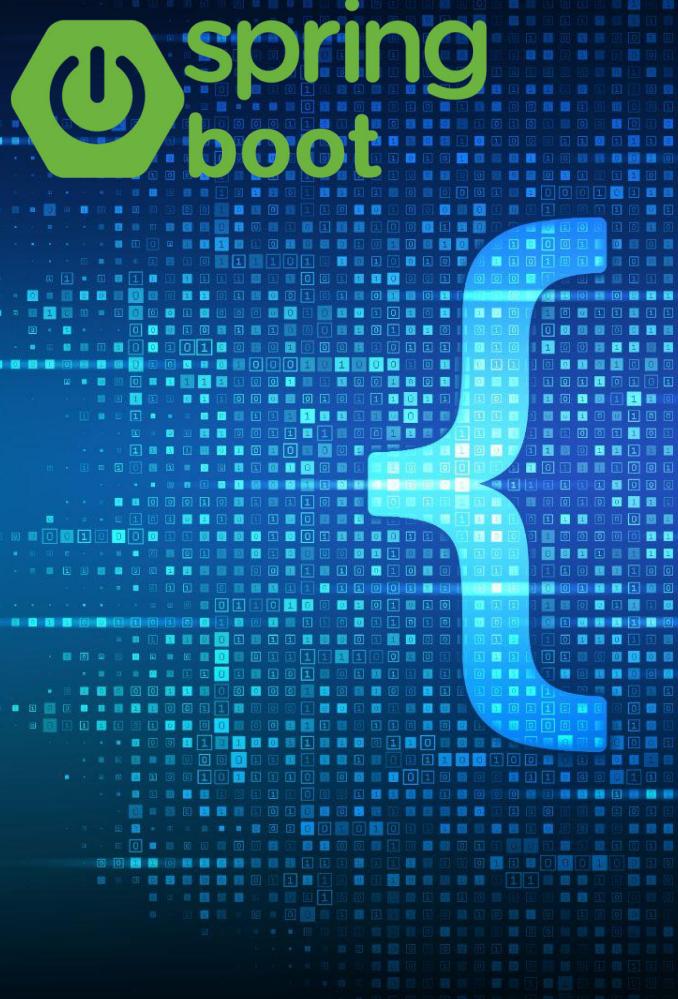
## Introduzione al Test Driven Development (TDD)

- Il Test Driven Development (TDD) è una metodologia di sviluppo del software che si basa sulla scrittura dei test automatici prima dell'implementazione effettiva del codice. Il ciclo di sviluppo del TDD è composto da tre fasi principali: "Red", "Green" e "Refactor".
- Fase "Red"
- Durante questa fase, si scrive un test automatizzato che descrive il comportamento desiderato della funzionalità che si intende implementare. Il test dovrebbe fallire inizialmente, poiché la funzionalità non è ancora stata implementata.
- La scrittura del test si concentra sugli aspetti chiave che si vogliono verificare e sulle asserzioni che si desidera eseguire per confermare il corretto funzionamento del codice.
- Questa fase si basa sul concetto "Write the test first", ovvero la scrittura del test prima dell'implementazione effettiva.



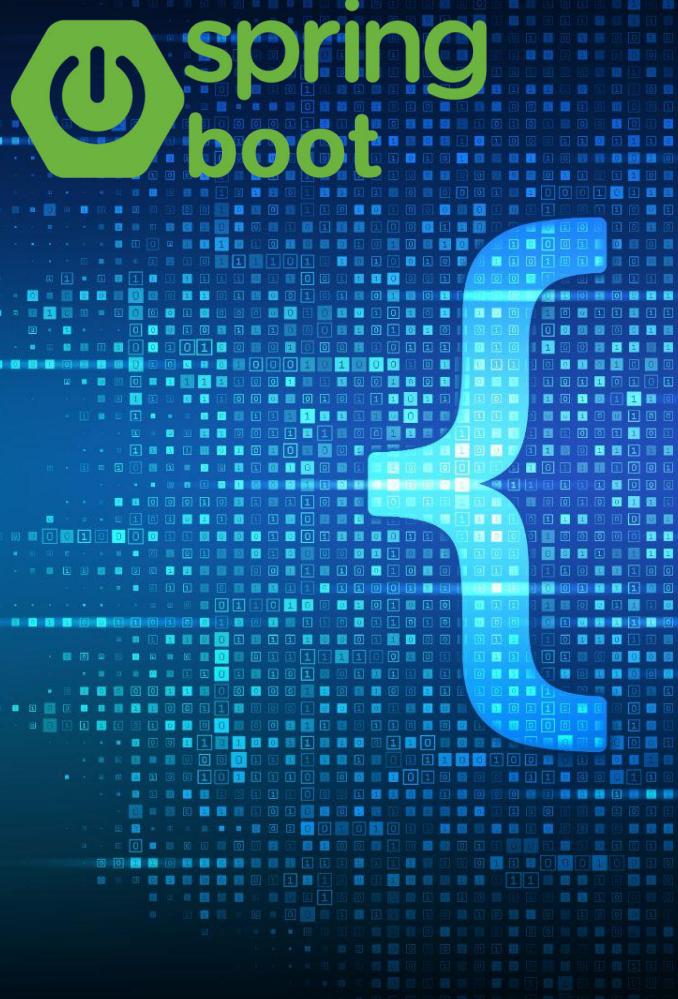
## Introduzione al Test Driven Development (TDD)

- Fase "Green":
- Durante questa fase, si implementa il codice necessario per far passare il test precedentemente scritto. L'obiettivo è scrivere il minimo codice necessario per soddisfare il test e farlo passare.
- L'implementazione può essere suddivisa in piccoli incrementi, con l'obiettivo di far passare un test alla volta.
- Durante questa fase, l'attenzione è rivolta solo al fatto che il test passi correttamente, senza preoccuparsi dell'ottimizzazione o dell'implementazione completa della funzionalità.



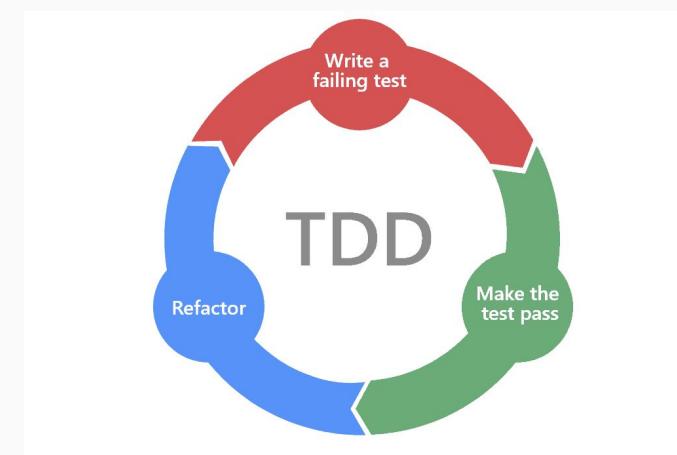
## Introduzione al Test Driven Development (TDD)

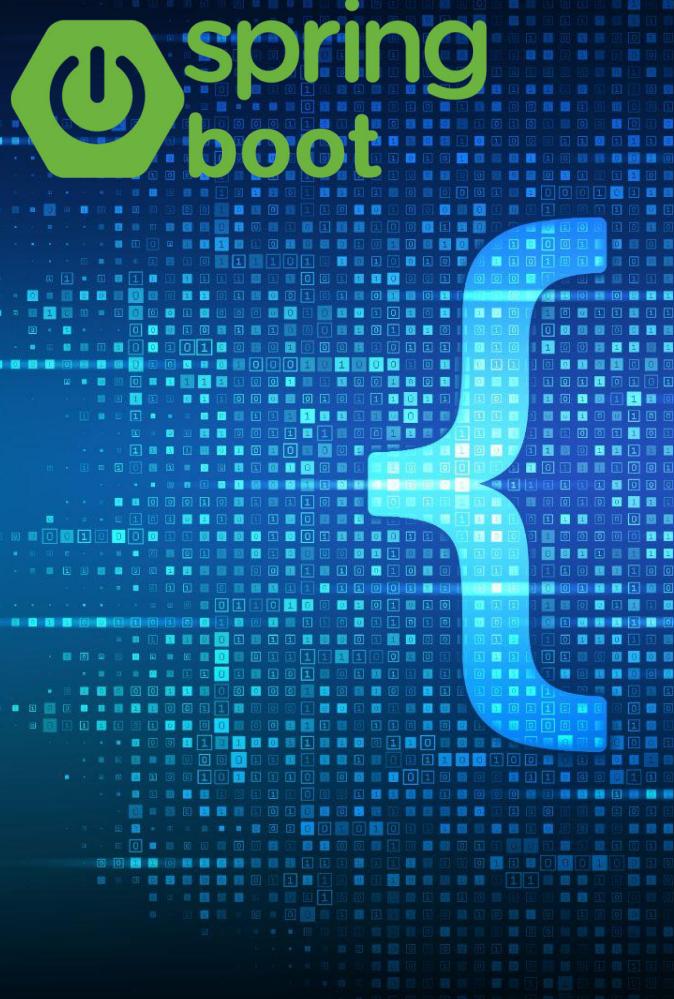
- Fase "Refactor":
- Durante questa fase, si rivede il codice scritto in precedenza e lo si raffina. L'obiettivo è migliorare la struttura del codice, l'organizzazione, la leggibilità e la manutenibilità, senza modificare il comportamento funzionale del codice.
- Durante il refactoring, si possono identificare opportunità per rimuovere duplicazioni, semplificare il codice, applicare principi di progettazione e rendere il codice più robusto e pulito.
- È importante eseguire nuovamente i test per garantire che le modifiche di refactoring non abbiano introdotto nuovi errori o rotto la funzionalità esistente.



## Introduzione al Test Driven Development (TDD)

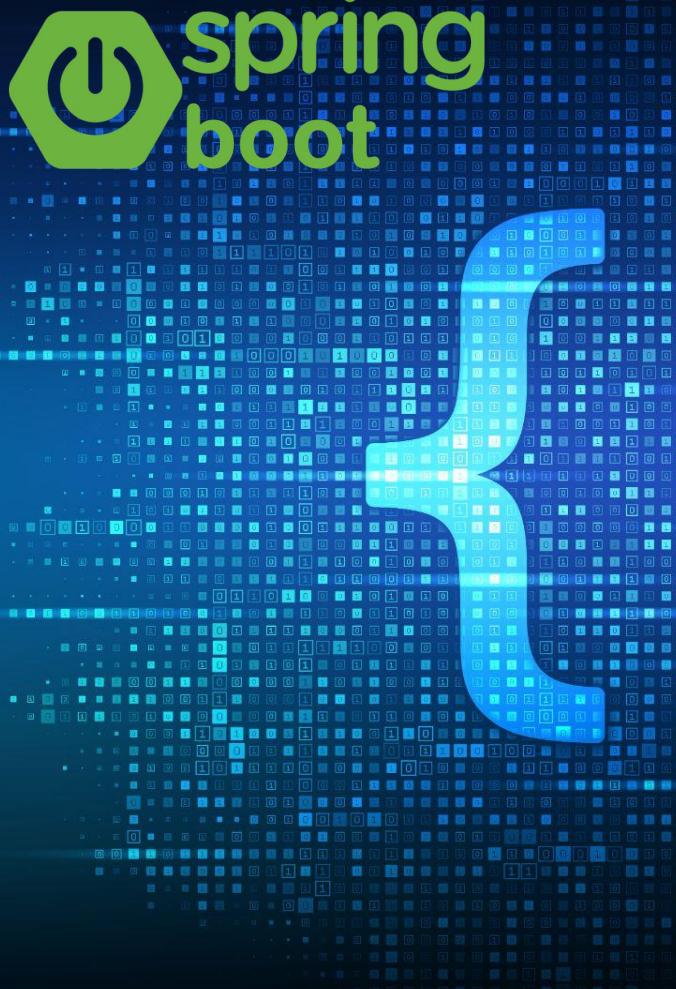
- Questo ciclo "Red-Green-Refactor" viene ripetuto iterativamente per ogni funzionalità o unità di codice da implementare.
- L'obiettivo del TDD è quello di creare un insieme completo di test automatizzati che coprano tutti gli aspetti critici dell'applicazione e forniscano un feedback rapido sull'integrità del codice.





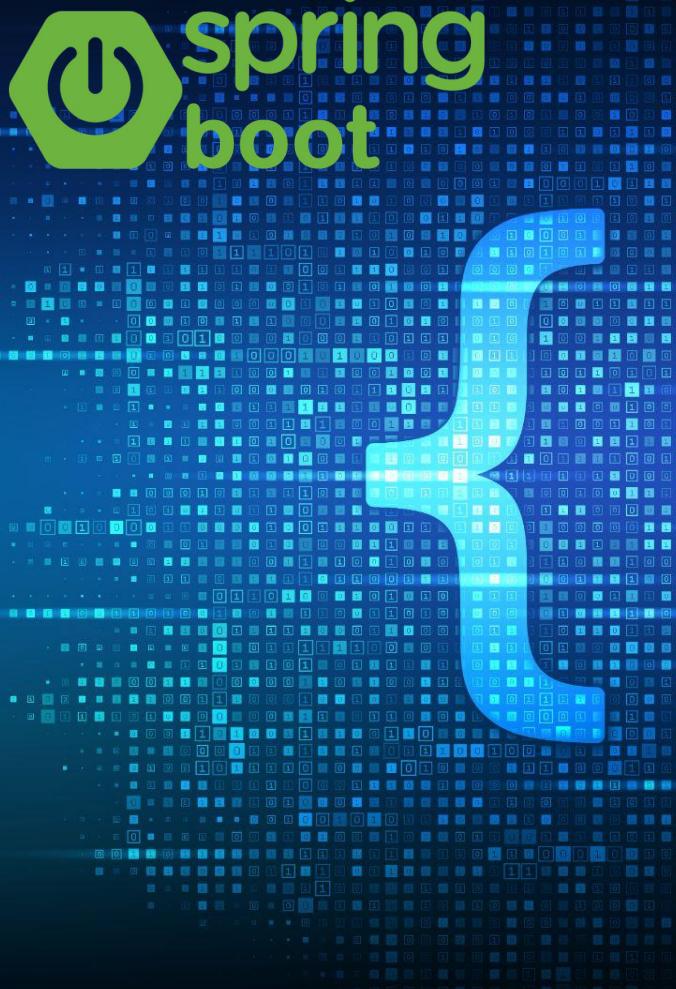
## Introduzione al Test Driven Development

- Benefici del Test Driven Development:
- **Miglioramento della qualità del codice:** I test automatizzati consentono di individuare e correggere errori in modo tempestivo.
- **Progettazione modulare:** Lo sviluppo guidato dai test incoraggia la progettazione modulare e la scrittura di codice più pulito e ben strutturato.
- **Maggiore affidabilità:** I test continui consentono di individuare regressioni o errori non intenzionali durante lo sviluppo.
- **Refactoring sicuro:** I test forniscono una sicurezza di base per eseguire modifiche di refactoring senza introdurre nuovi bug.
- **Documentazione vivente:** I test automatizzati fungono da documentazione vivente del comportamento atteso del codice.



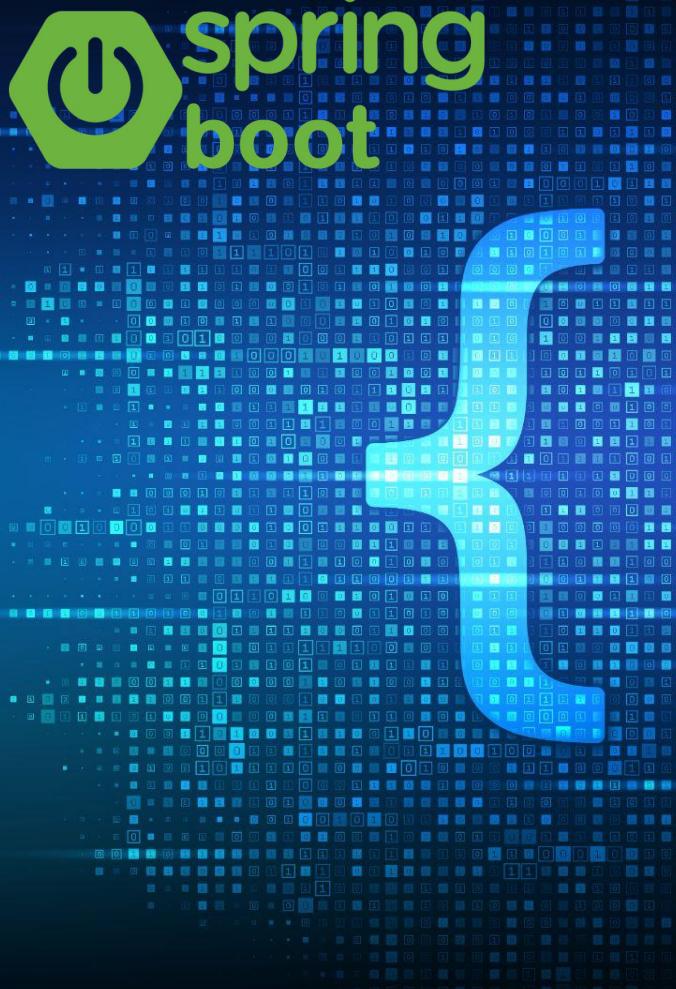
## Introduzione al ResponseEntity

- Il `ResponseEntity` è una classe di Spring Framework che rappresenta l'entità di risposta in un'applicazione web.
- Questa classe viene utilizzata per **incapsulare la risposta HTTP** restituita da un server.
- Il `ResponseEntity` fornisce metodi per **impostare il corpo della risposta, l'header e il codice di stato HTTP**.
- Questa flessibilità permette di **personalizzare completamente la risposta** che viene inviata al client.
- Inoltre, il `ResponseEntity` può essere utilizzato per **restituire oggetti complessi come risposta**, come ad esempio oggetti JSON o XML.



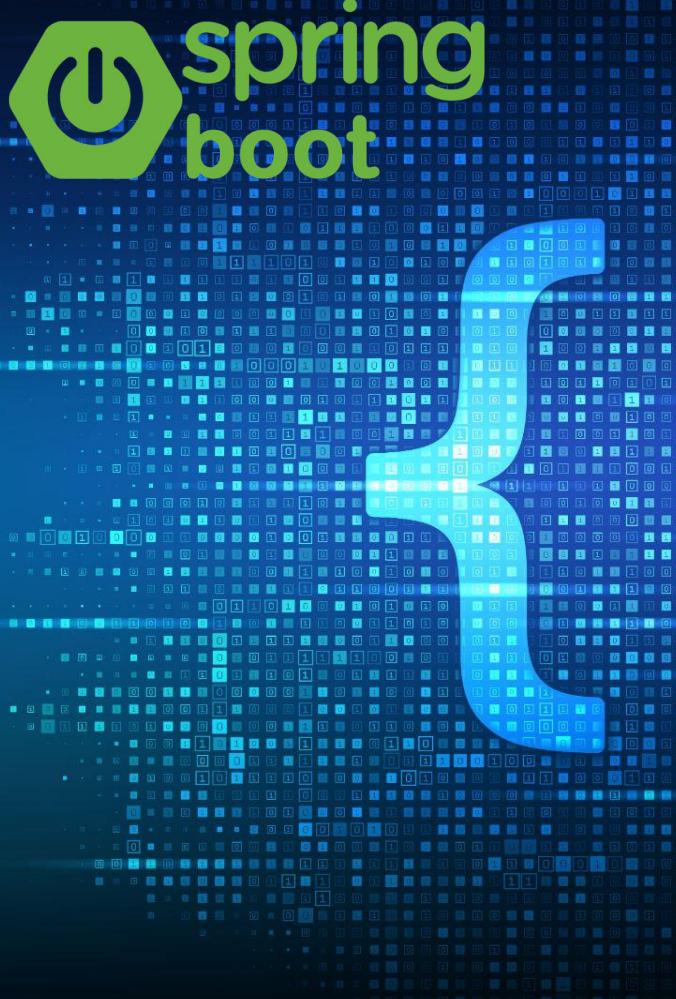
## Introduzione al ResponseEntity

- Caratteristiche della classe ResponseEntity:
- **Risposta HTTP personalizzata:** ResponseEntity consente di creare una risposta HTTP personalizzata da restituire da un controller Spring. Puoi specificare il corpo della risposta, gli intestazioni HTTP e il codice di stato.
- **Gestione dei codici di stato:** Puoi specificare il codice di stato HTTP che desideri restituire, ad esempio 200 (OK), 201 (Created), 404 (Not Found), ecc. Questo è utile per comunicare il risultato dell'operazione al client.
- **Corpo della risposta:** Puoi impostare il corpo della risposta in vari formati, come JSON, XML, testo o qualsiasi altro formato necessario. Spring fornisce supporto per la serializzazione automatica degli oggetti in JSON o XML utilizzando librerie come Jackson o JAXB.



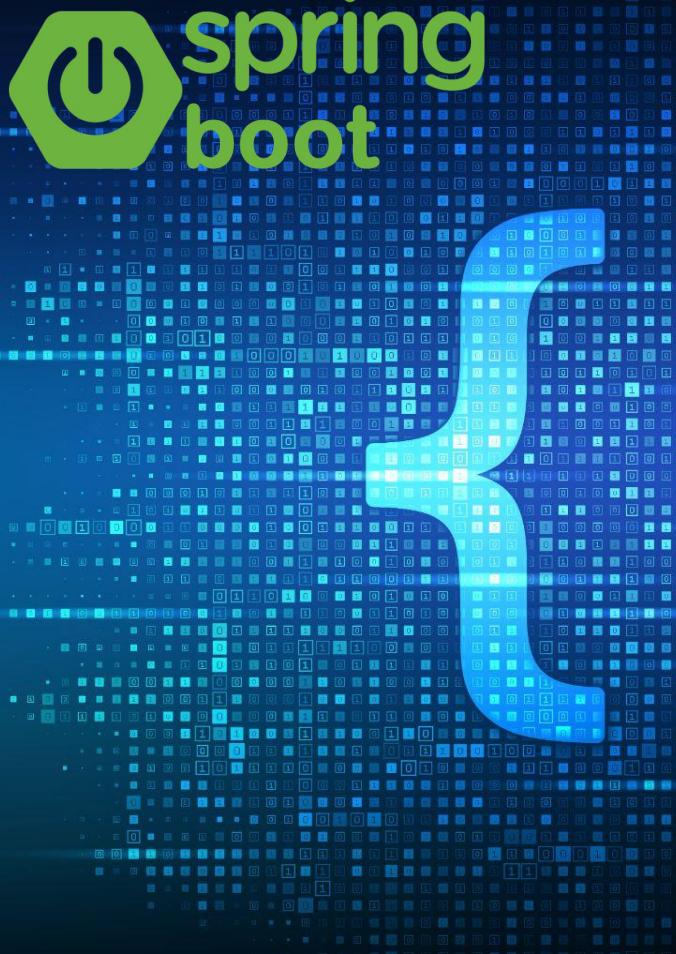
## Introduzione al ResponseEntity

- **Intestazioni HTTP personalizzate:** È possibile aggiungere intestazioni HTTP personalizzate alla risposta, ad esempio l'intestazione "Content-Type" per indicare il tipo di contenuto restituito.
- **Flessibilità:** ResponseEntity è flessibile e può essere utilizzata con diversi tipi di dati. Puoi restituire oggetti complessi o risposte vuote a seconda delle tue esigenze.



## Introduzione al @ControllerAdvice

- La notazione `@ControllerAdvice` è un'annotazione in Java che viene utilizzata per creare una classe in grado di gestire le eccezioni a livello globale all'interno di un'applicazione web basata su Spring MVC.
- Quando viene utilizzata questa annotazione su una classe, viene identificata come un "consiglio" per i controller e viene utilizzata per gestire le eccezioni che possono essere sollevate durante l'esecuzione delle richieste HTTP.
- La classe contrassegnata da `@ControllerAdvice` può contenere metodi annotati con `@ExceptionHandler`, che vengono utilizzati per gestire specifici tipi di eccezioni.

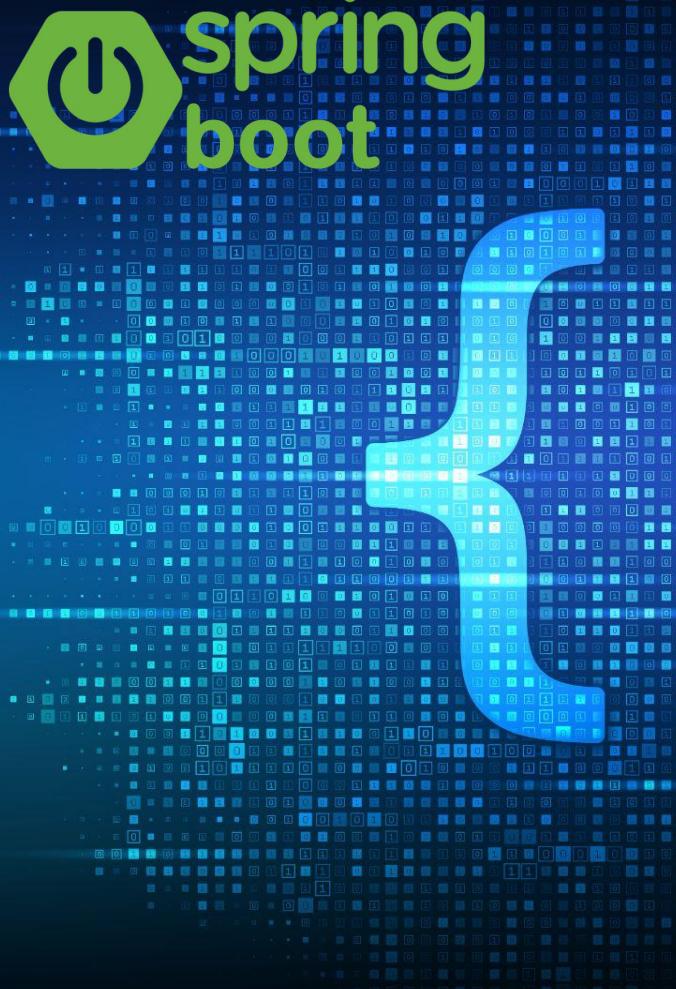


## Introduzione al @ControllerAdvice

- I metodi annotati con `@ExceptionHandler` all'interno di una classe `@ControllerAdvice` vengono chiamati automaticamente quando viene sollevata un'eccezione all'interno dei controller.
- Possono essere utilizzati per personalizzare la risposta HTTP alle eccezioni, ad esempio restituendo un messaggio di errore specifico o reindirizzando l'utente a una pagina di errore personalizzata.
- Inoltre, l'annotazione `@ControllerAdvice` può essere utilizzata anche per definire altre funzionalità globali a livello di applicazione, come la gestione delle risposte per tutti i controller o l'intercettazione e la modifica delle richieste in ingresso.

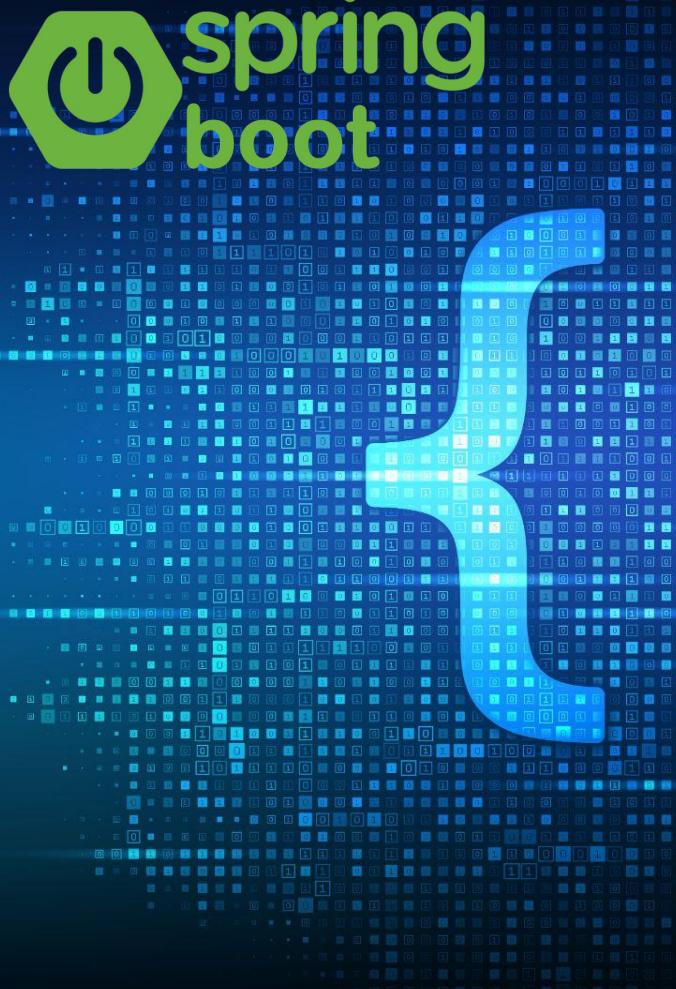
# Spring Boot 3

Approfondimenti Spring Data  
JPA e Hibernate



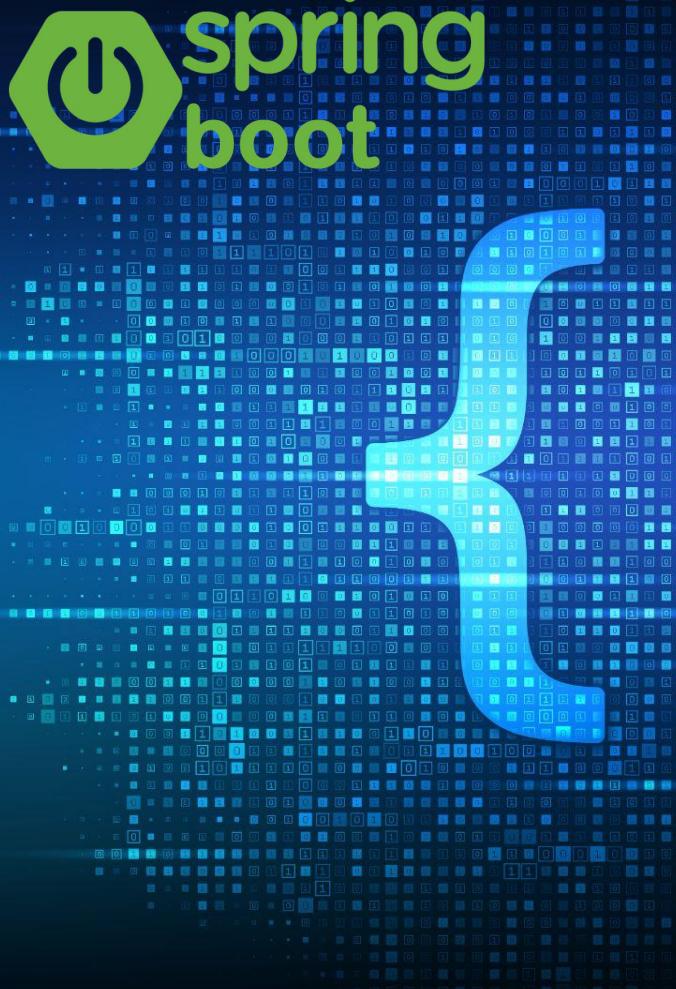
## Introduzione alla Abstract Entity

- In Spring Boot e Hibernate, una **Abstract Entity** è una classe astratta che rappresenta una entità persistente nel database.
- Questa classe astratta fornisce una serie di attributi e metodi comuni che possono essere ereditati da altre classi rappresentanti entità specifiche.
- L'Abstract Entity di solito include attributi come un **identificatore univoco** (ad esempio, un campo ID), metodi per ottenere e impostare i valori degli attributi e metodi per la gestione delle operazioni di persistenza nel database (ad esempio, creazione, aggiornamento, eliminazione).
- Le classi che ereditano dall'Abstract Entity possono specificare ulteriori attributi e metodi unici per rappresentare le proprie entità specifiche. L'ereditarietà dell'Abstract Entity aiuta a ridurre il codice duplicato e a fornire una struttura comune per la gestione delle entità nel sistema.



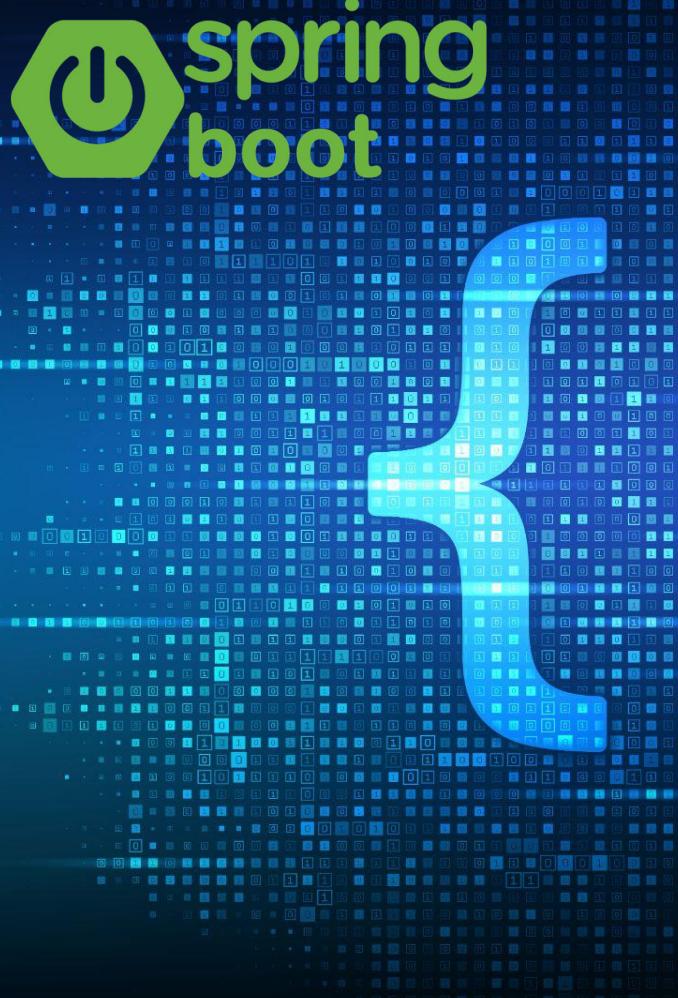
## Introduzione alla Abstract Entity

- Come Funziona la "Abstract Entity" in Spring Boot e Hibernate:
- **Classe Base:** Un'Abstract Entity è una classe Java astratta che viene annotata con `@MappedSuperclass`. Questa annotazione indica a Hibernate che questa classe non deve essere mappata direttamente a una tabella del database, ma invece serve come classe base per altre classi di entità.
- **Attributi Comuni:** L'Abstract Entity può contenere attributi che sono comuni a molte classi di entità dell'applicazione. Ad esempio, se hai diverse classi di entità che devono avere un campo "data di creazione" e un campo "data di modifica", puoi definire questi attributi nell'Abstract Entity in modo da evitare la duplicazione del codice.



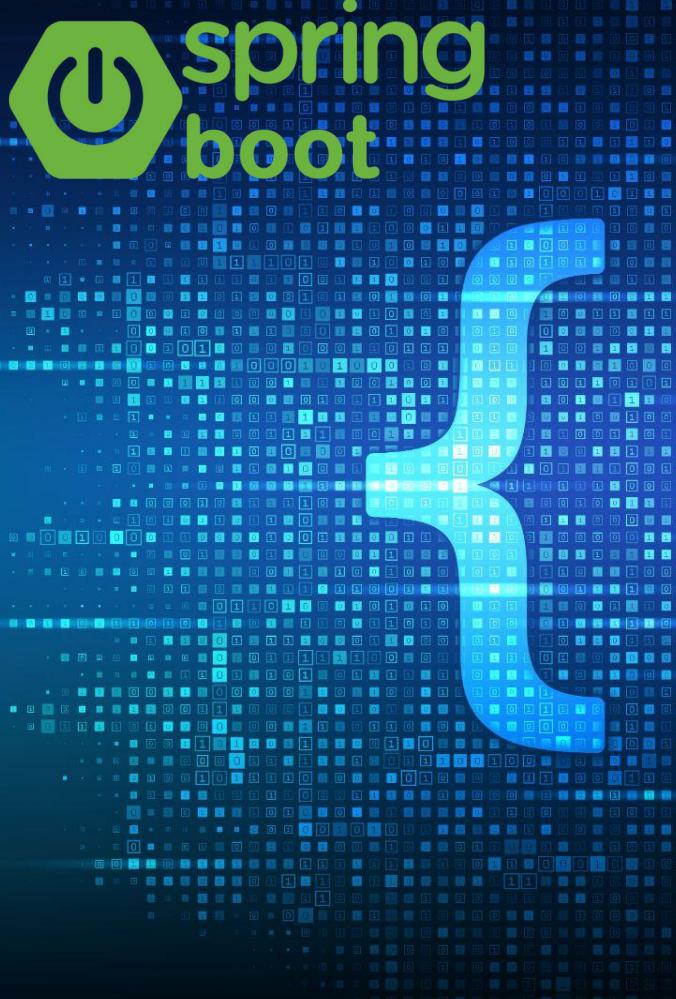
## Introduzione alla Abstract Entity

- **Comportamenti Comuni:** In aggiunta agli attributi, puoi anche definire **metodi comuni** che sono utili per tutte le classi di entità che ereditano da questa classe astratta. Ad esempio, potresti definire metodi di utilità per la manipolazione dei dati condivisi.
- **Mapping delle Entità:** Le classi di entità che ereditano dall'Abstract Entity utilizzano l'ereditarietà Java per acquisire attributi, metodi e configurazioni dalla classe base. Inoltre, puoi aggiungere attributi specifici o annotazioni di mappatura per ciascuna classe di entità.



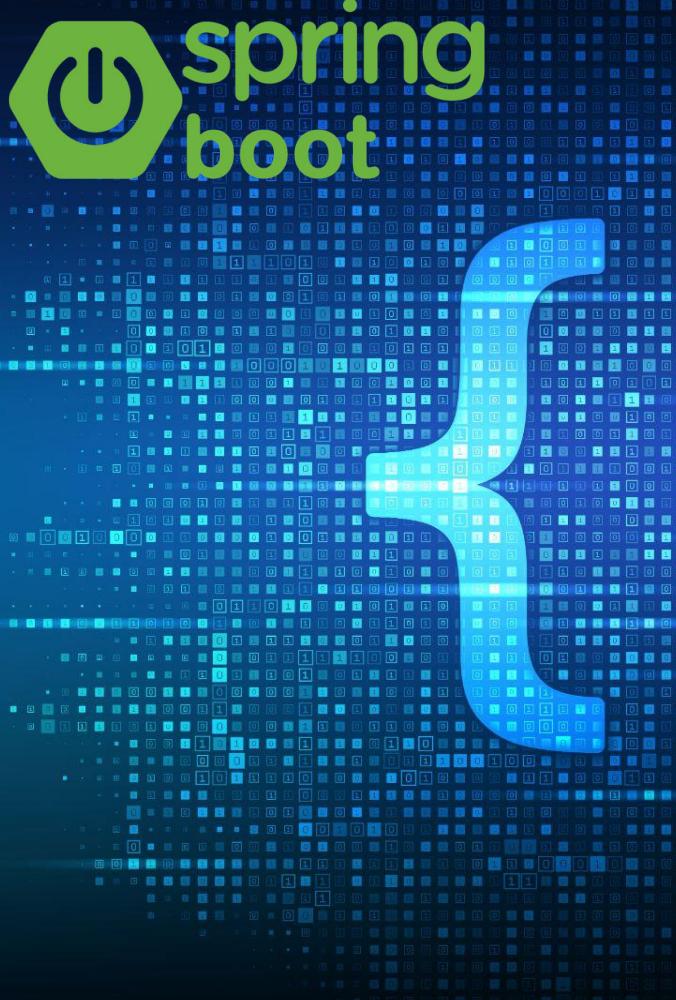
## Introduzione alle transazioni nei DB

- Le transazioni nei database sono un insieme di operazioni che vengono eseguite come un'unità indivisibile.
- Queste operazioni possono includere l'aggiunta, la modifica o l'eliminazione delle informazioni memorizzate nel database.
- Il concetto di transazione è utilizzato per garantire che tutte le operazioni siano eseguite in modo corretto e coerente.
- Una transazione deve soddisfare quattro proprietà fondamentali, comunemente note come proprietà ACID:
- **Atomicità:** una transazione è un'unità atomica, il che significa che tutte le operazioni all'interno della transazione devono essere completate con successo o annullate. Se una o più operazioni non possono essere eseguite correttamente, tutte le operazioni precedenti vengono ripristinate allo stato originale (rollback).



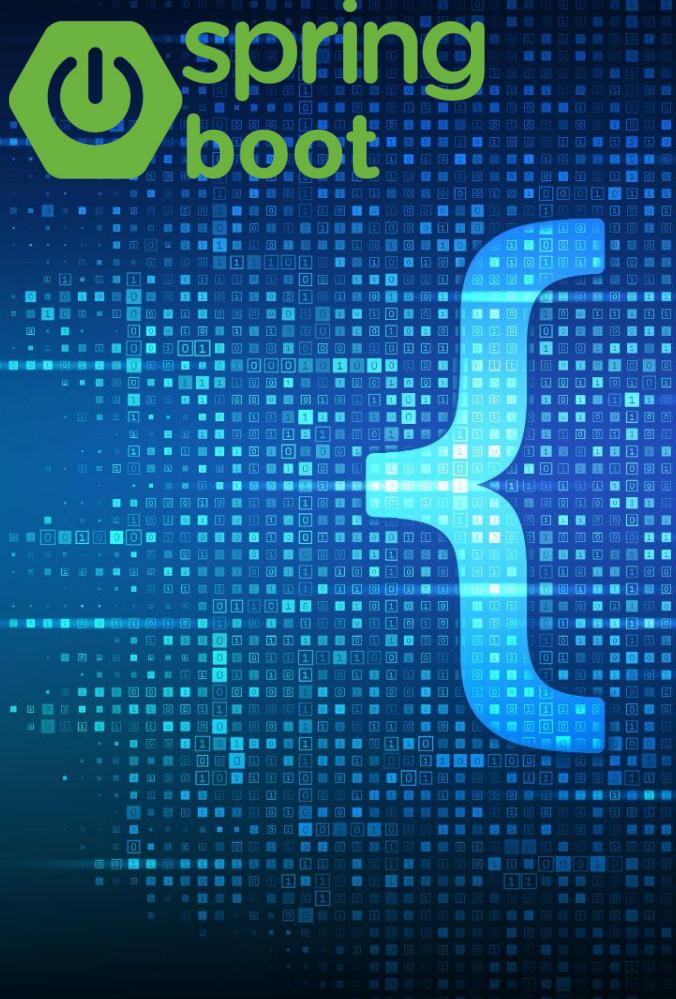
## Introduzione alle transazioni nei DB

- **Coerenza:** una transazione deve garantire la coerenza del database. Ciò significa che il database deve passare da uno stato consistente a un altro stato consistente. Ad esempio, se una transazione aggiorna una determinata quantità in un record, il nuovo valore deve essere coerente con il resto delle informazioni nel database.
- **Isolamento:** ogni transazione deve essere isolata dagli effetti di altre transazioni concorrenti. Ciò significa che una transazione in corso non deve vedere i cambiamenti apportati da altre transazioni finché non viene completata. Questo garantisce che ogni transazione lavori con una visione consistente del database.
- **Durabilità:** una transazione completata con successo deve essere permanente e persistente nel database anche in caso di guasto del sistema. Ciò richiede che le informazioni modificate siano archiviate in modo permanente nel database.



## Introduzione alle transazioni nei DB

- Commit e Rollback: Un commit è l'azione di applicare permanentemente le modifiche apportate da una transazione al database. Un rollback è l'azione di annullare permanentemente tutte le modifiche apportate da una transazione e riportare il database allo stato precedente alla transazione.
- Punti di Salvataggio (Savepoints): I punti di salvataggio consentono di suddividere una transazione in sotto-transazioni più piccole. È possibile eseguire il commit o il rollback di singole sotto-transazioni senza coinvolgere l'intera transazione.
- Gestione delle Transazioni: Le transazioni possono essere gestite manualmente attraverso il codice (**gestione esplicita**) o automaticamente dal sistema di gestione del database o da framework come Spring Framework (**gestione implicita**).



## Introduzione alle transazioni nei DB

- In un contesto di sviluppo di applicazioni, le transazioni sono spesso utilizzate per garantire l'integrità dei dati e la coerenza delle operazioni.
- Ad esempio, quando si effettuano trasferimenti di denaro in un'applicazione bancaria, le operazioni di prelievo da un conto e di accredito su un altro conto **devono essere eseguite in una singola transazione** per evitare situazioni incoerenti come il denaro "sospeso nel limbo".
- Affinché una transazione possa essere eseguita, vengono utilizzati i cosiddetti **log di transazione**. Un log di transazione **registra tutte le operazioni eseguite all'interno di una transazione**, consentendo il ripristino dello stato del database in caso di guasto del sistema.
- Le transazioni nei database sono fondamentali per garantire l'integrità e la coerenza dei dati. Sono ampiamente utilizzate in applicazioni aziendali, servizi bancari, sistemi di prenotazione, sistemi di gestione delle risorse umane e in molti altri contesti in cui è importante mantenere la precisione e l'affidabilità delle informazioni memorizzate.



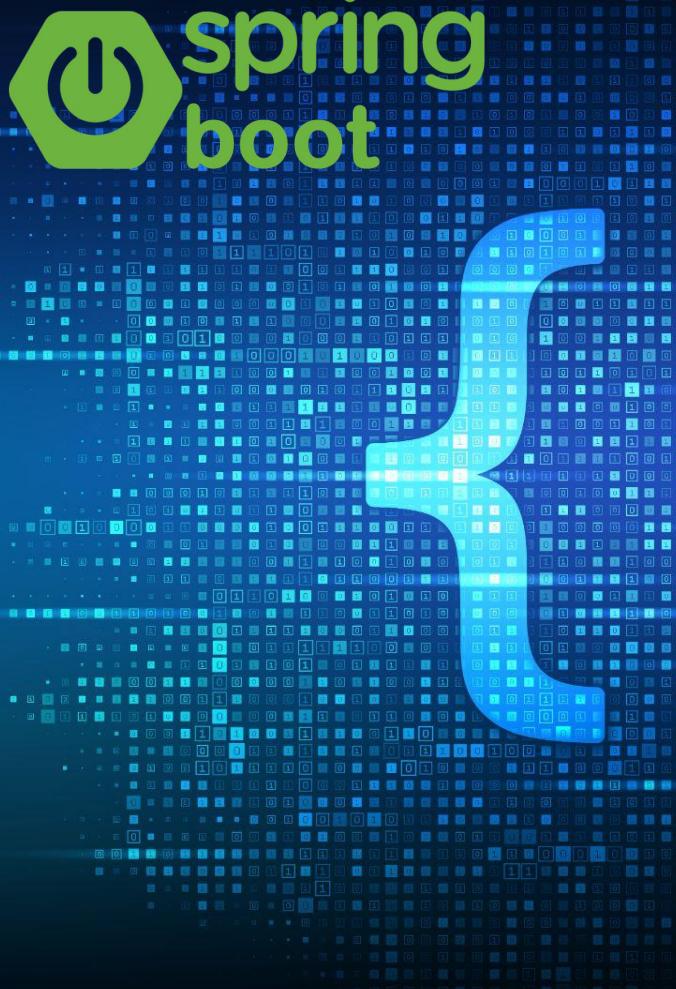
## Le transazioni in Spring Boot

- Nello Spring Boot, la gestione delle transazioni è una funzionalità potente e flessibile fornita dal modulo **Spring Data JPA**, che si basa a sua volta sulla **gestione delle transazioni di Spring Framework**.
- La gestione delle transazioni nello Spring Boot è basata su due concetti principali:
- **Transazioni Declarative:** Spring Boot offre un approccio dichiarativo per la gestione delle transazioni utilizzando annotazioni come **@Transactional**. Queste annotazioni vengono applicate alle classi o ai metodi delle classi per indicare che il metodo deve essere eseguito all'interno di una transazione.
- **PlatformTransactionManager:** Spring Boot utilizza il concetto di **PlatformTransactionManager**, che è responsabile della **gestione delle transazioni effettive**. Il PlatformTransactionManager fornisce un'astrazione per diverse piattaforme di transazioni, consentendo a Spring di lavorare con diverse tecnologie di gestione delle transazioni, come JPA, JDBC, Hibernate, ecc.



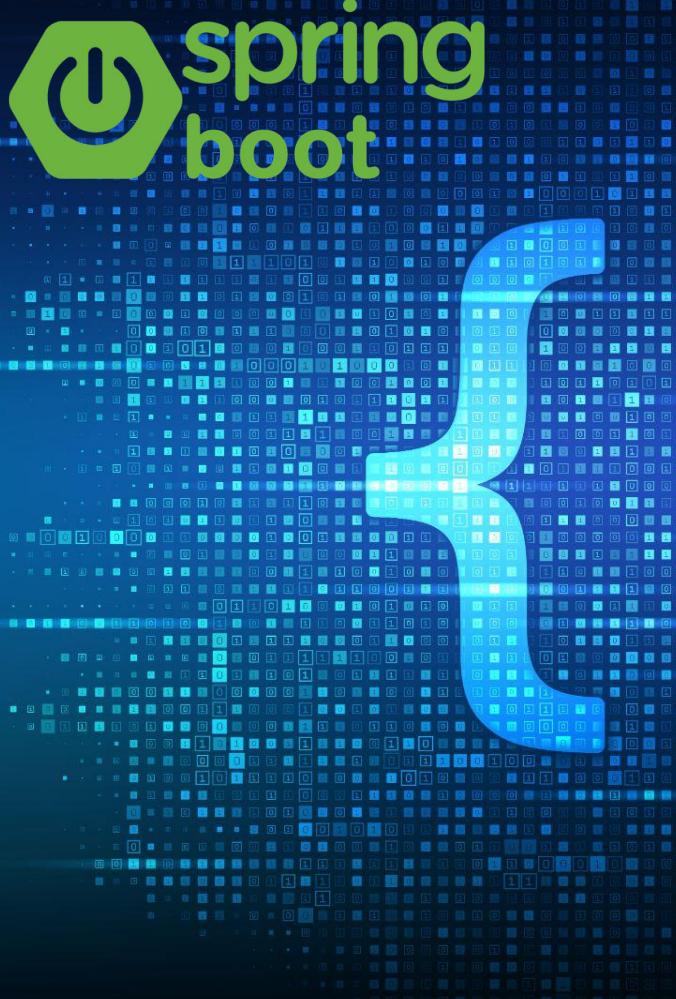
## Le transazioni in Spring Boot

- La notazione `@Transactional` in Spring Boot è un'annotazione che viene utilizzata per gestire la transazionalità dei metodi all'interno della applicazione.
- Quando viene applicata a un metodo o a una classe, `@Transactional` indica a Spring che il metodo (o tutti i metodi della classe) deve essere eseguito all'interno di una transazione.
- Se la transazione ha successo, i cambiamenti apportati al database durante l'esecuzione del metodo verranno committati; se si verifica un errore, la transazione verrà annullata e i cambiamenti verranno roll back.



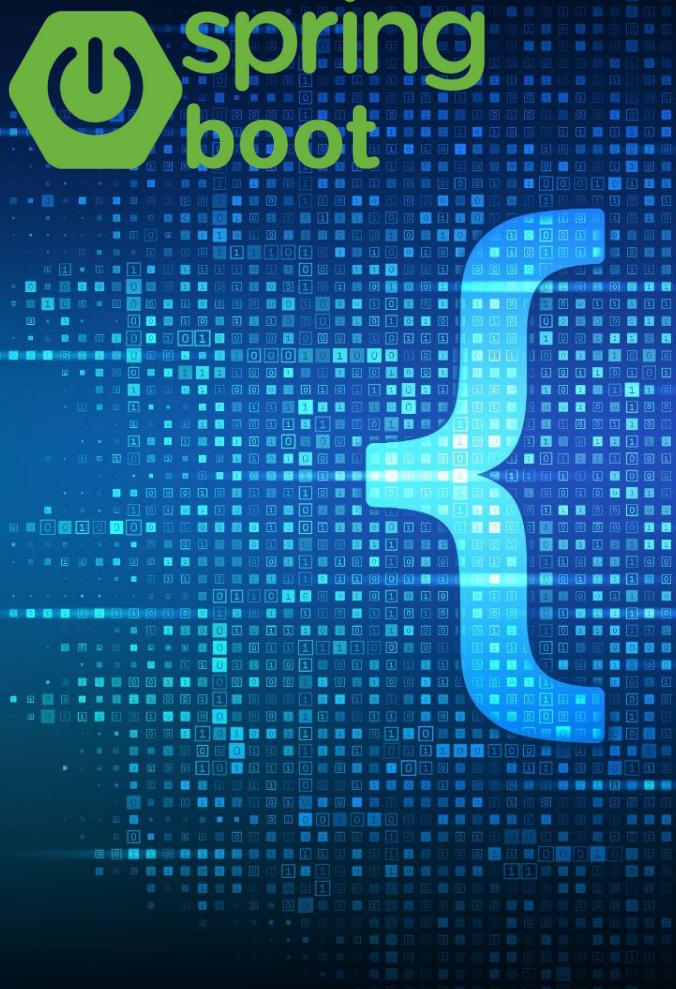
## Le transazioni in Spring Boot

- Le caratteristiche delle transazioni con `@Transactional`:
- Gestione automatica delle transazioni: Utilizzando `@Transactional`, non è necessario gestire manualmente l'inizio e la fine delle transazioni. Spring gestisce automaticamente l'inizio della transazione prima dell'esecuzione del metodo annotato e il commit o il rollback della transazione dopo che il metodo è terminato.
- Scoping delle transazioni: La notazione `@Transactional` può essere applicata a livello di classe o a livello di metodo. Quando applicata a livello di classe, tutte le chiamate ai metodi di quell'oggetto saranno eseguite all'interno di una transazione. Quando applicata a livello di metodo, solo quel metodo sarà eseguito all'interno di una transazione.



## Le transazioni in Spring Boot

- Isolamento delle transazioni: E' possibile specificare il livello di isolamento delle transazioni utilizzando il parametro `isolation`. L'isolamento determina il grado di visibilità delle modifiche apportate da una transazione rispetto ad altre transazioni. I valori possibili includono `READ_COMMITTED`, `READ_UNCOMMITTED`, `REPEATABLE_READ`, `SERIALIZABLE`, ecc.
- Rollback delle transazioni: Di default, `@Transactional` effettuerà un rollback della transazione se il metodo annotato lancia un'eccezione controllata (cioè una sottoclasse di `java.lang.Exception`). È possibile specificare eccezioni personalizzate per il rollback tramite il parametro `rollbackFor` o escludere alcune eccezioni dal rollback tramite il parametro `noRollbackFor`.



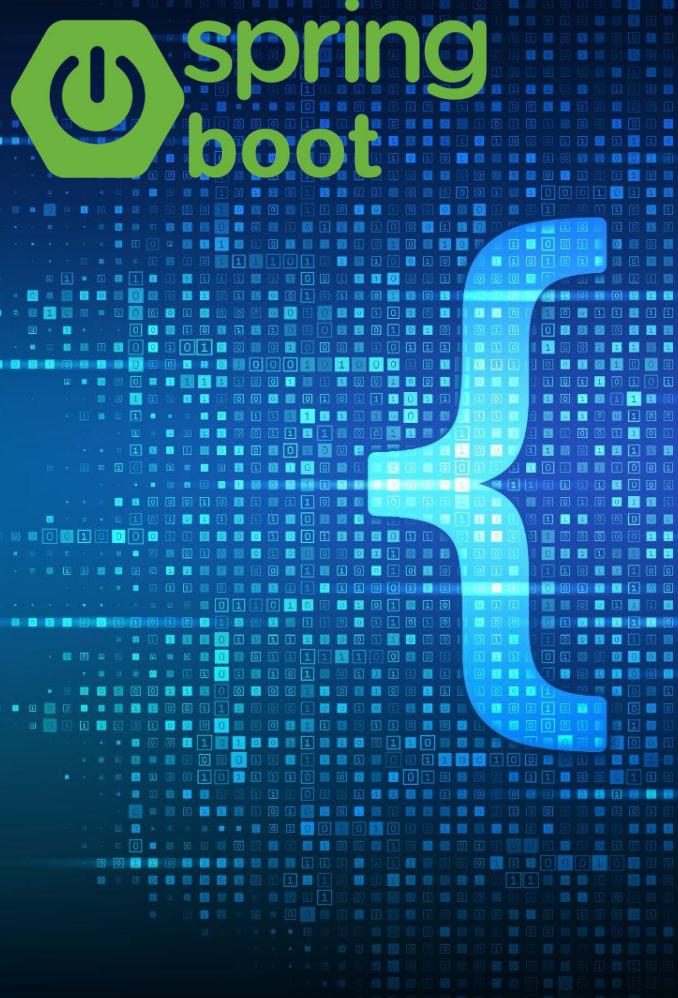
## Le transazioni in Spring Boot

- Propagation delle transazioni: E' possibile specificare come le transazioni si propagheranno tra i metodi utilizzando il parametro `propagation`. I valori possibili includono `REQUIRED`, `REQUIRES_NEW`, `SUPPORTS`, `NOT_SUPPORTED`, `MANDATORY`, `NEVER`, ecc.
- Gestione dei lock: E' possibile gestire i lock del database utilizzando il parametro `readOnly`, che indica se il metodo sarà in sola lettura e non eseguirà operazioni di scrittura sul database.
- L'annotazione `@Transactional(readOnly = true)` indica a Spring che il metodo o il metodo della classe contrassegnato come tale deve essere eseguito all'interno di una transazione in sola lettura.
- Ciò significa che durante l'esecuzione del metodo annotato, le operazioni di scrittura sul database, come l'inserimento, l'aggiornamento o l'eliminazione dei dati, non sono consentite. Il metodo può solo effettuare operazioni di lettura, come il recupero dei dati dal database.



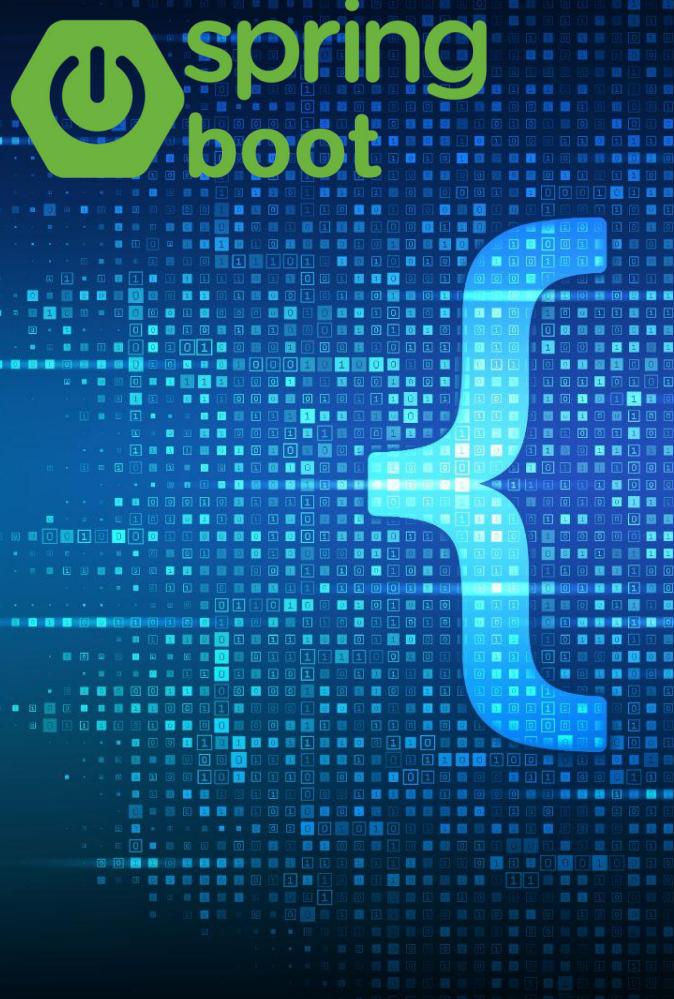
## Le transazioni in Spring Boot

- Impostare il metodo in sola lettura (read-only) può migliorare le prestazioni, in quanto Spring può ottimizzare la gestione delle transazioni, sapendo che non ci saranno operazioni di scrittura da eseguire.
- L'utilizzo dell'annotazione `@Transactional(readOnly = true)` è anche un'indicazione per gli sviluppatori che il metodo ha solo scopi di lettura e non modifica i dati. Ciò rende il codice più leggibile e comprensibile.
- È importante notare che l'annotazione `@Transactional(readOnly = true)` non impedisce completamente le operazioni di scrittura sul database. Se il codice del metodo tenta di eseguire un'operazione di scrittura, ad esempio modificando lo stato di un oggetto gestito dall'EntityManager di JPA, verrà lanciata un'eccezione `javax.persistence.TransactionRequiredException`.



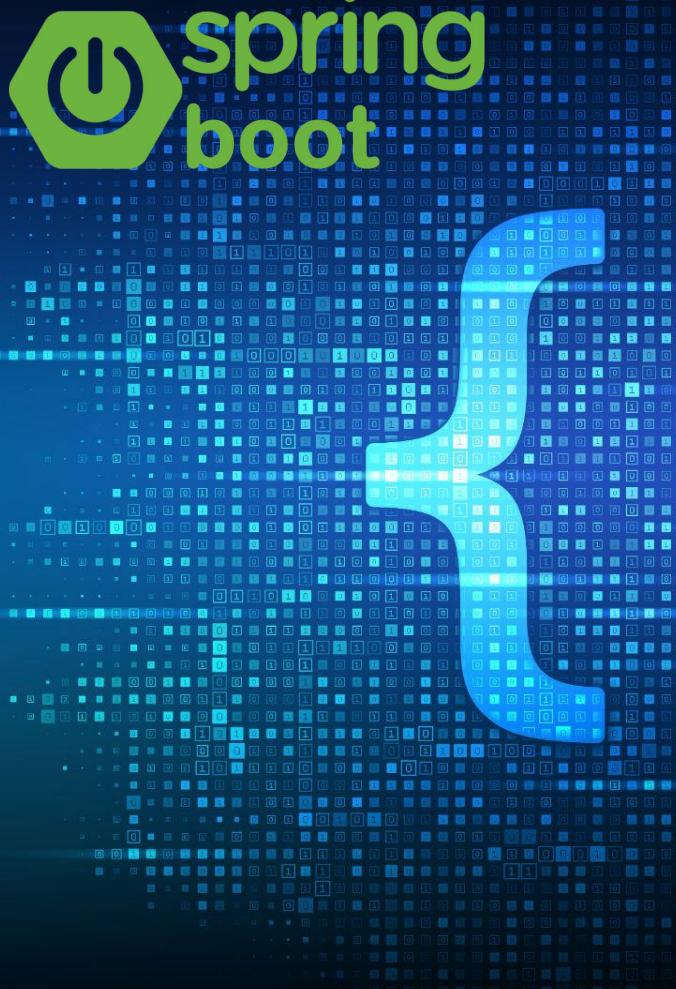
## Introduzione alla notazione @Version

- La notazione `@Version` in Spring Boot è utilizzata per indicare un campo di una classe come versione delle entità gestite dal framework `Spring Data JPA`.
- Questa notazione viene tipicamente applicata a un **campo intero o long** nella classe dell'entità, rappresentando la **versione dell'oggetto**.
- Quando viene utilizzata la notazione `@Version`, Spring Data JPA gestisce automaticamente l'incremento della versione quando un'entità viene modificata e salvata nel database.
- Questo è utile per controllare le “**collisioni di concorrenza**” quando più processi o thread cercano di modificare la stessa entità contemporaneamente.
- Quando un'entità viene salvata nel database, il valore del campo di versione viene incrementato.



## Introduzione alla notazione @Version

- Se, nel frattempo, un altro processo o thread ha modificato la stessa entità, la versione salvata nel database non corrisponderà più alla versione dell'oggetto che si sta cercando di salvare.
- In questo caso, Spring Data JPA lancerà un'eccezione di tipo **OptimisticLockingFailureException** per gestire la collisione di concorrenza.
- Per utilizzare correttamente la notazione **@Version**, è importante garantire che sia attivato il supporto alla gestione della versione nel repository JPA.
- Ciò può essere fatto aggiungendo l'annotazione **@EnableJpaRepositories** con l'opzione **enableDefaultTransactions = false** nella classe di configurazione dell'applicazione.



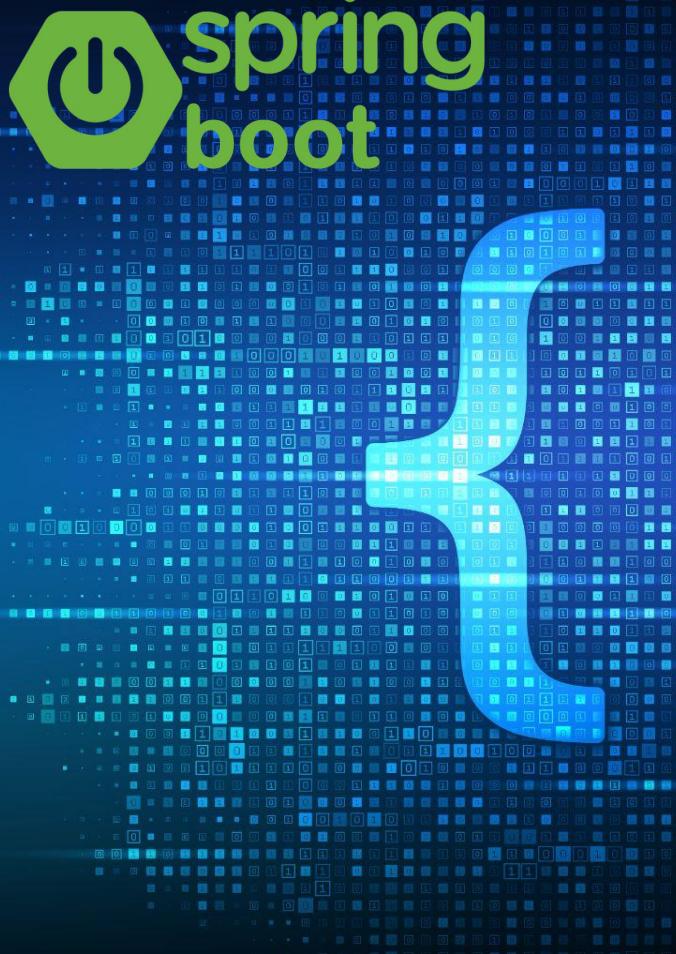
## Differenza tra save e saveAndFlush

- In Spring Data JPA e Hibernate, `save` e `saveAndFlush` sono due metodi utilizzati per persistere (salvare) un'entità in un database. Tuttavia, hanno un comportamento leggermente diverso:
- `save`: Il metodo `save` viene utilizzato per salvare un'entità nel contesto di persistenza senza forzare la sincronizzazione immediata con il database. In altre parole, l'entità viene memorizzata nella cache di persistenza (prima di essere inviata effettivamente al database) e quindi aspetta il momento migliore per essere sincronizzata con il database. La sincronizzazione avviene automaticamente quando viene eseguito il commit della transazione corrente o quando viene chiamato `flush()` esplicitamente.



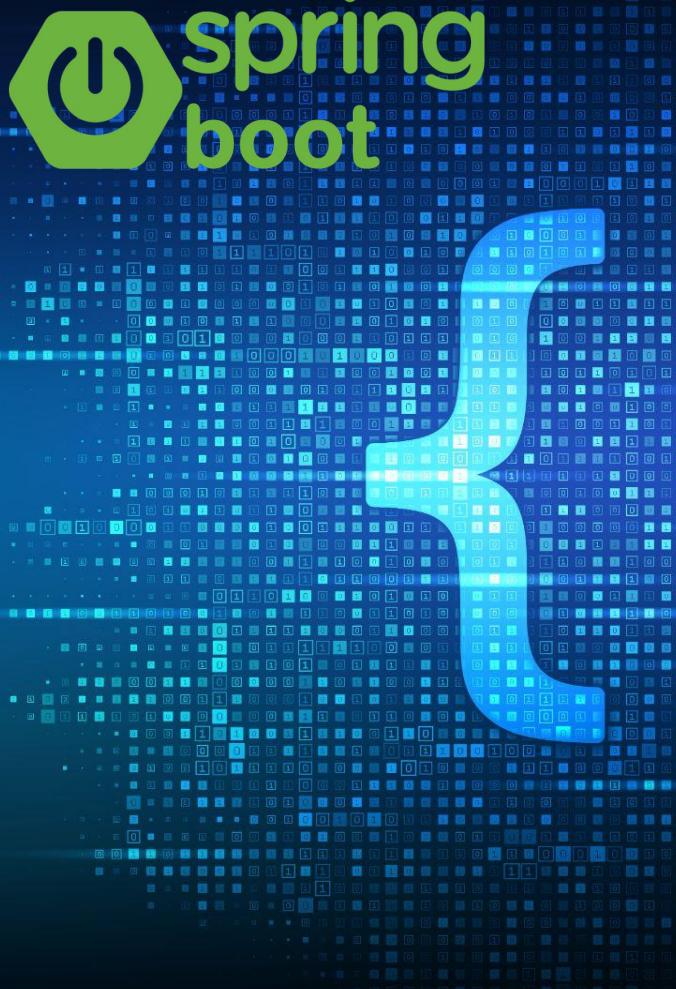
## Differenza tra save e saveAndFlush

- **saveAndFlush:** Il metodo `saveAndFlush`, d'altra parte, effettua immediatamente la sincronizzazione tra l'entità e il database. Questo significa che dopo aver chiamato `saveAndFlush`, l'entità viene salvata nel database in modo persistente e immediato, e qualsiasi eccezione o errore di persistenza verrà generato in questo momento se ci sono problemi.
- In breve, la differenza principale tra i due metodi è il momento in cui avviene la sincronizzazione tra l'entità e il database.
- **save** ritarda la sincronizzazione, mentre `saveAndFlush` la esegue immediatamente. La scelta tra i due dipenderà dalle tue esigenze specifiche. Se desideri massimizzare le prestazioni e consentire una sincronizzazione asincrona, `save` è la scelta migliore. Se invece vuoi verificare immediatamente se ci sono errori di persistenza o garantire che l'entità sia salvata in modo coerente nel database, allora `saveAndFlush` è l'opzione da utilizzare.



## Opzioni @Transactional - isolation

- Le opzioni di isolation della notazione `@Transactional` rappresentano il livello di isolamento che può essere applicato a un metodo o a un'intera transazione nel contesto di un'applicazione basata su Spring Framework.
- Le opzioni di isolation sono utilizzate per gestire la concorrenza e l'accesso concorrente ai dati all'interno di una transazione. Esse determinano come le transazioni interagiscono tra loro e come le modifiche effettuate da una transazione saranno visibili per altre transazioni.
- Le opzioni di isolation disponibili sono:
- **DEFAULT**: Questa opzione utilizza l'isolamento predefinito del database, che di solito dipende dal sistema DBMS utilizzato. Ad esempio, nel PG l'isolamento predefinito è **Read Committed**.



## Opzioni @Transactional - isolation

- **READ\_UNCOMMITTED:** Questa opzione consente a una transazione di leggere e scrivere dati non ancora confermati da altre transazioni. L'accesso ai dati non è isolato, il che significa che le modifiche effettuate da altre transazioni possono essere lette anche se non sono ancora state confermate.
- **READ\_COMMITTED:** Questa opzione consente a una transazione di leggere solo i dati confermati da altre transazioni. Le modifiche effettuate da altre transazioni diventano visibili solo dopo essere state confermate.
- **REPEATABLE\_READ:** Questa opzione garantisce che una transazione leggendo i dati non venga influenzata da modifiche effettuate da altre transazioni fino al termine della transazione stessa. In altre parole, una transazione ripeterà la stessa lettura di un dato durante tutta la sua esecuzione, anche se altri hanno modificato il dato.



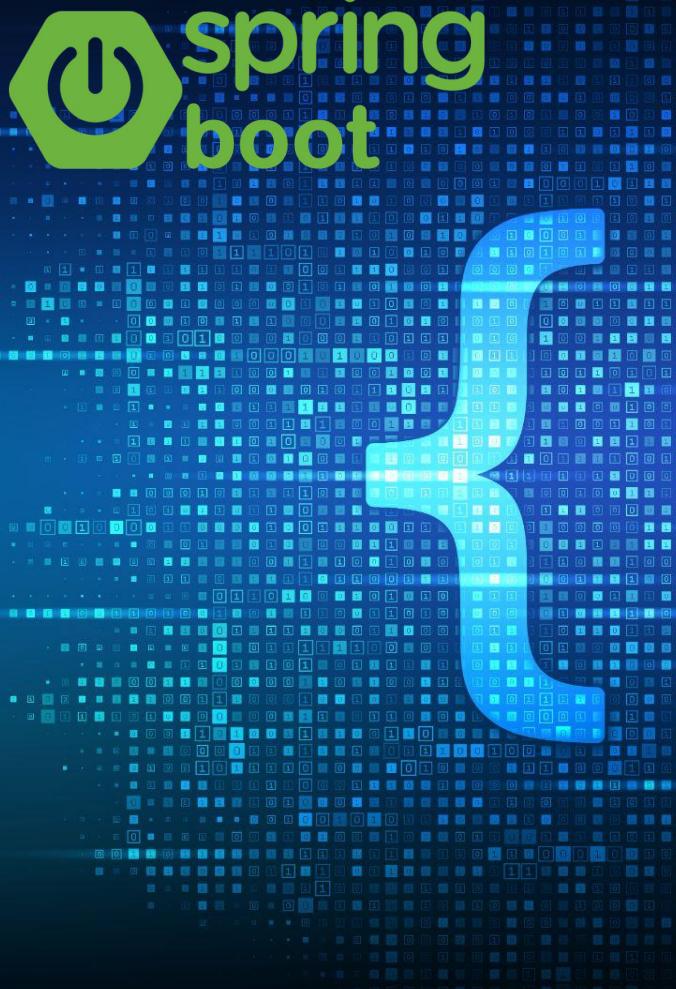
## Opzioni @Transactional - isolation

- **SERIALIZABLE:** Questa opzione fornisce il massimo livello di isolamento. Garantisce che più transazioni vengano eseguite come se fossero eseguite in modo seriale, ovvero una dopo l'altra. Questo livello di isolamento offre la massima protezione contro problemi di concorrenza ma può anche comportare un calo delle prestazioni a causa del blocco degli accessi concorrenti.
- L'opzione di isolamento delle transazioni può essere specificata come segue utilizzando il codice  
`@Transactional(isolation = Isolation.READ_COMMITTED)`



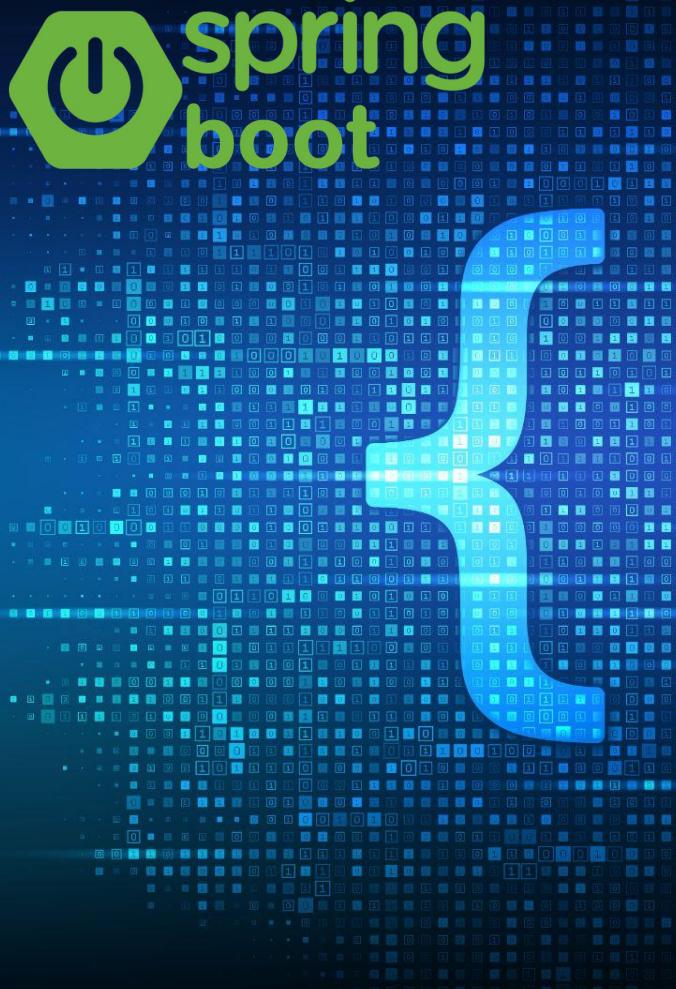
## Opzioni @Transactional - propagation

- Le opzioni di propagation della notazione `@Transactional` sono utilizzate per definire il comportamento di una transazione quando viene invocato un metodo annotato `@Transactional` all'interno di un'altra transazione già esistente.
- Le opzioni di propagation possono essere specificate come parametro nella notazione `@Transactional`. Di seguito sono elencate le opzioni di propagation e la loro funzione:
- **REQUIRED**: Questa è l'opzione di propagazione di default. Se un metodo annotato con `@Transactional` viene invocato all'interno di una transazione già esistente, il metodo utilizzerà la transazione esistente. Altrimenti, se nessuna transazione esiste, il metodo inizierà una nuova transazione.



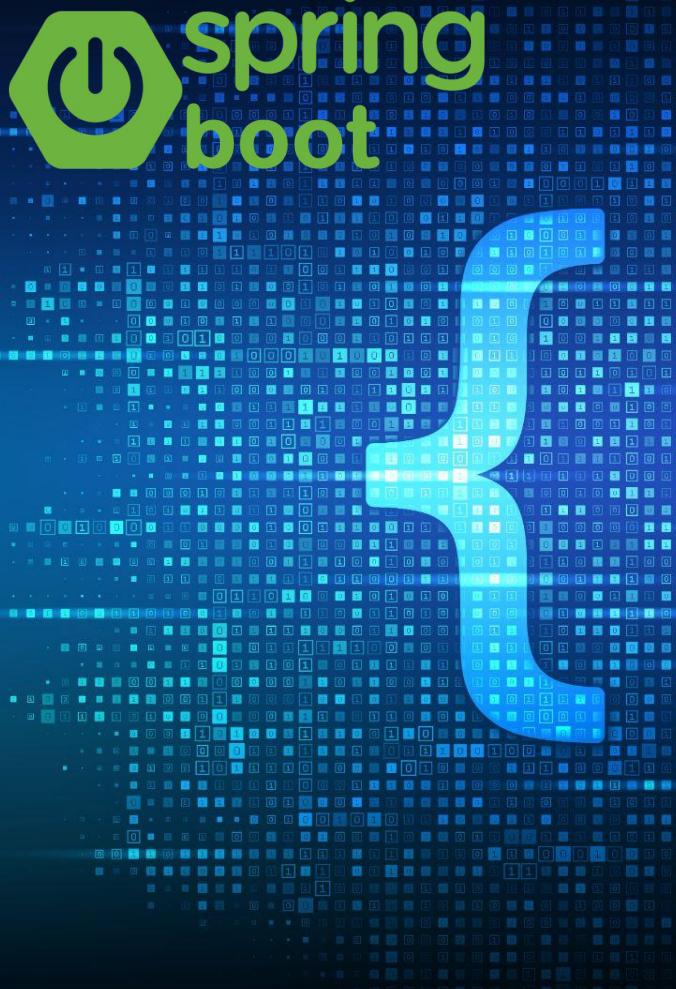
## Opzioni @Transactional - propagation

- **REQUIRES\_NEW:** Questa opzione **inizia sempre una nuova transazione** indipendentemente dall'esistenza di una transazione precedente. Se un metodo annotato con @Transactional viene invocato all'interno di una transazione già esistente, la transazione precedente viene sospesa temporaneamente e viene avviata una nuova transazione. Una volta che la nuova transazione viene completata, la transazione precedente viene ripresa.
- **SUPPORTS:** Questa opzione non richiede una transazione per l'esecuzione del metodo annotato con @Transactional. Se un metodo viene invocato all'interno di una transazione esistente, il metodo utilizzerà quella transazione. Altrimenti, se nessuna transazione esiste, **il metodo verrà eseguito senza transazione.**



## Opzioni @Transactional - propagation

- **NOT\_SUPPORTED:** Questa opzione esegue il metodo annotato con `@Transactional` senza transazione, anche se una transazione è attualmente in corso. Se un metodo viene invocato all'interno di una transazione esistente, la transazione viene temporaneamente sospesa e **il metodo viene eseguito senza transazione**. Una volta completata l'esecuzione del metodo, **la transazione viene ripresa**.
- **MANDATORY:** Questa opzione **richiede che esista una transazione**. Se un metodo annotato con `@Transactional` viene invocato senza una transazione esistente, **viene lanciata un'eccezione**.
- **NEVER:** Questa opzione **richiede che non esista una transazione**. Se un metodo annotato con `@Transactional` viene invocato all'interno di una transazione esistente, **viene lanciata un'eccezione**.



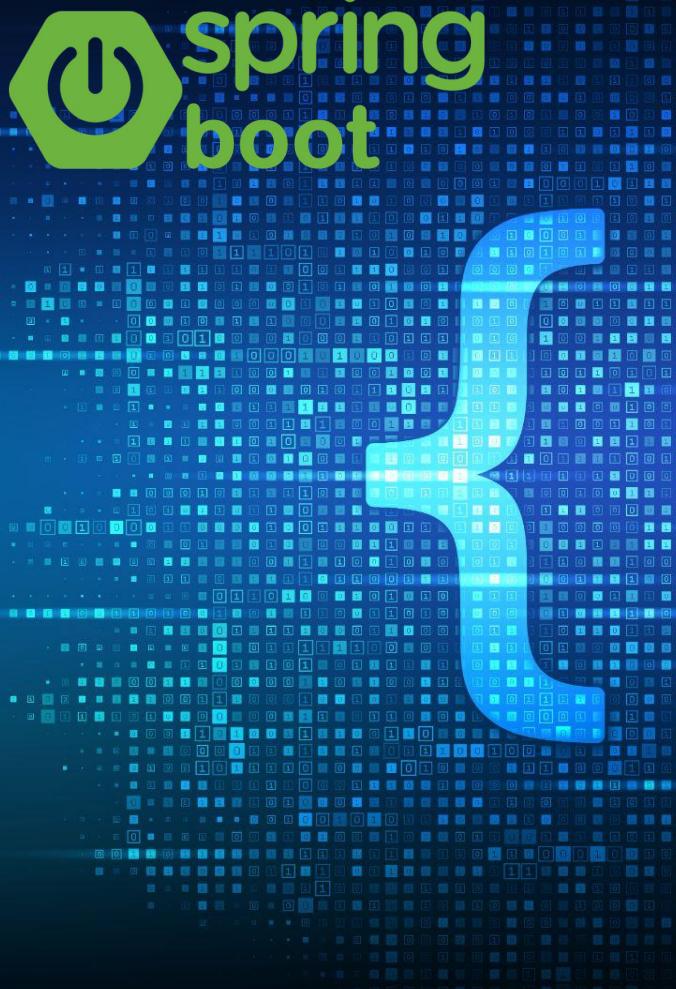
## Opzioni @Transactional - propagation

- **NESTED:** Questa opzione consente di creare una transazione nidificata all'interno di una transazione esistente. Se un metodo annotato con `@Transactional` viene invocato all'interno di una transazione già esistente, il metodo creerà una transazione nidificata all'interno di quella transazione. Se non esiste una transazione esterna, viene creata una nuova transazione. La transazione nidificata può essere sottoposta a `committ` o `rollback` indipendentemente dalla transazione esterna.

# Developing Full Stack Applications

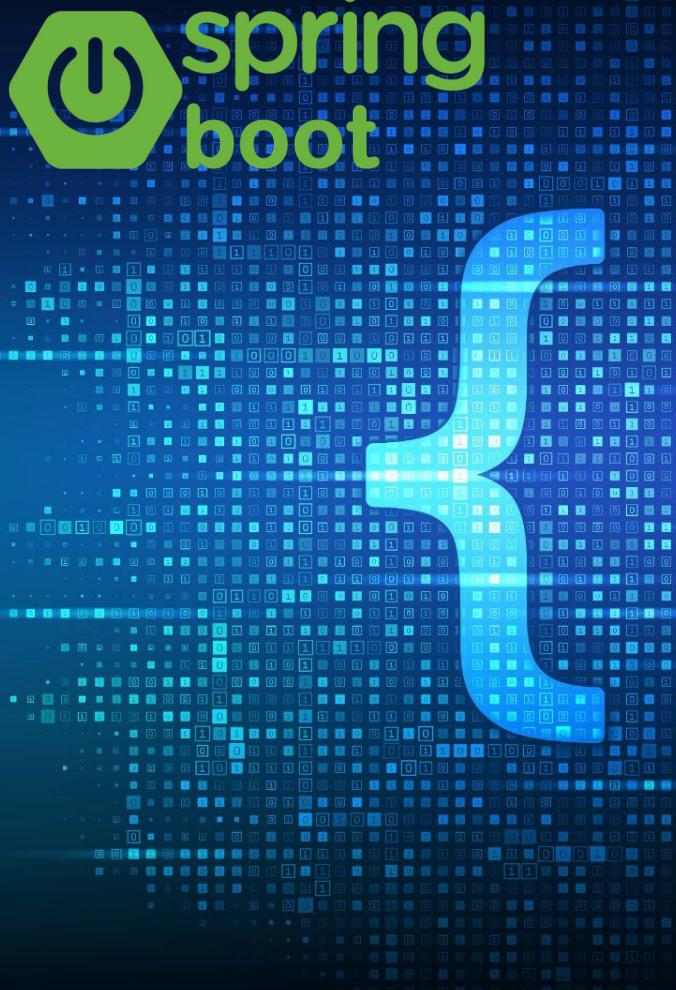
A background photograph of a person's hands typing on a white keyboard. Overlaid on the image is a large, semi-transparent digital lock icon. The lock is blue and white, featuring a circular pattern of dots and lines. The overall aesthetic is tech-oriented and cybersecurity-focused.

Uso del Json Web Token  
(JWT) come sistema di  
sicurezza



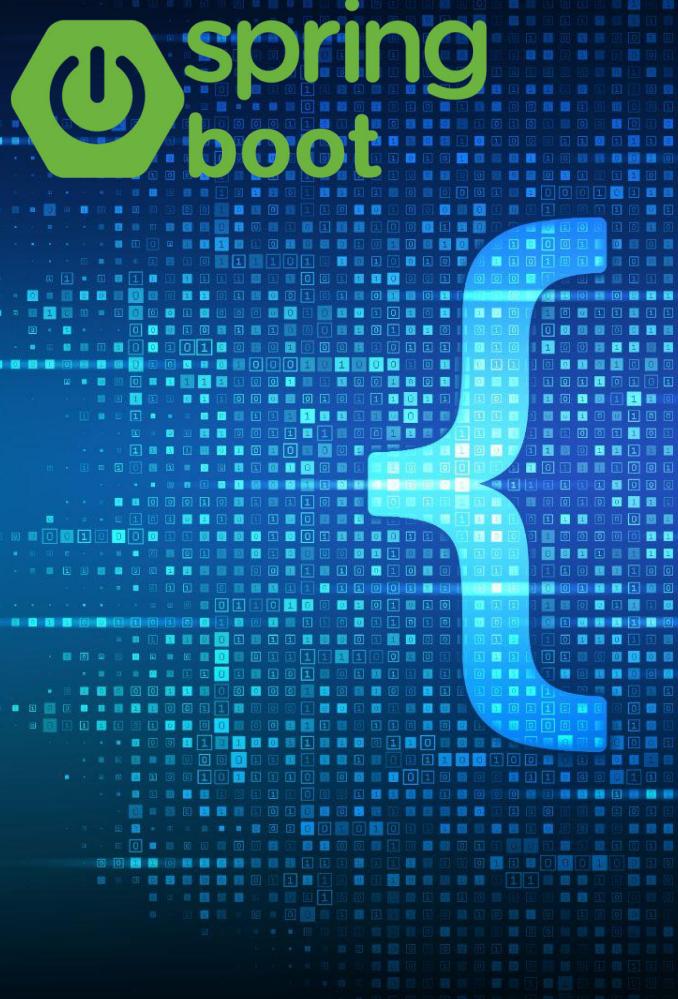
## Svantaggi della Basic Auth

- La Basic Authentication, sebbene semplice da implementare, presenta alcuni **svantaggi** che dovrebbero essere presi in considerazione:
- **Sicurezza debole:** La Basic Authentication invia le credenziali dell'utente (username e password) come testo non crittografato nell'intestazione della richiesta HTTP. Ciò rende vulnerabili le credenziali agli attacchi di tipo "man-in-the-middle" o al furto di credenziali. Senza un canale sicuro, come HTTPS, le credenziali sono esposte al rischio di intercettazione.
- **Mancanza di scalabilità:** La Basic Authentication richiede che le credenziali dell'utente siano inviate con ogni richiesta, il che può aumentare il carico di rete e rallentare le prestazioni, soprattutto **se si dispone di un gran numero di richieste autenticate**. Questo può rappresentare un ostacolo alla scalabilità dell'applicazione.



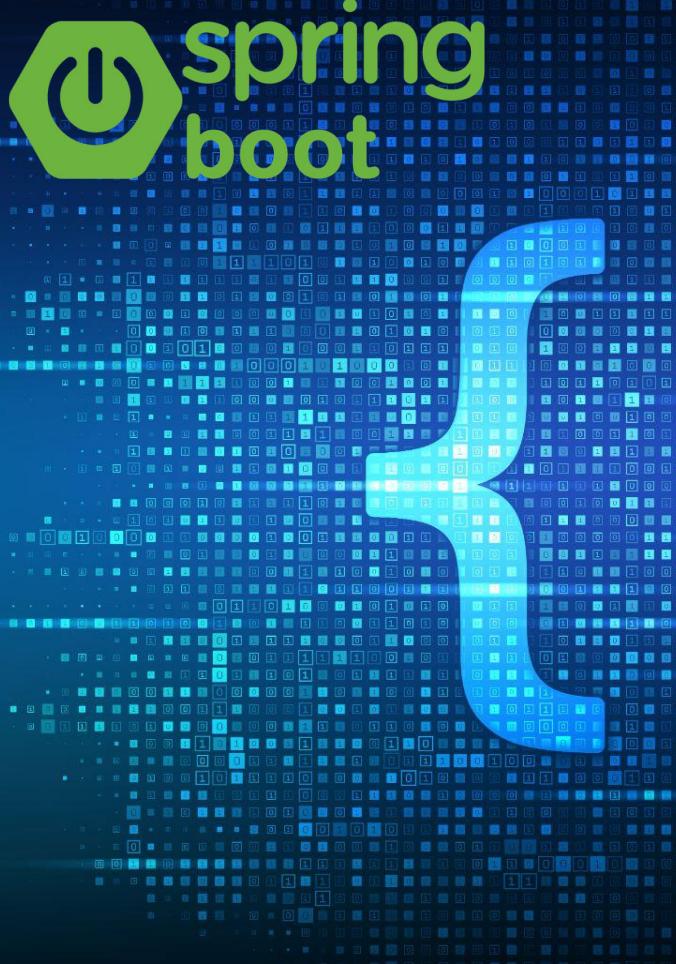
## Svantaggi della Basic Auth

- Mancanza di flessibilità per implementare logiche personalizzate: La Basic Authentication offre un'implementazione di autenticazione molto semplice, ma limitata. Non offre molte opzioni per personalizzare le regole di autenticazione o implementare logiche personalizzate come la gestione delle autorizzazioni o l'integrazione con sistemi di autenticazione esterni.
- Nessun supporto per le esigenze di autenticazione avanzate: La Basic Authentication non supporta funzionalità avanzate di autenticazione come la gestione dei token di accesso, il refresh dei token, l'autenticazione a due fattori o l'integrazione con fornitori di autenticazione esterni come OAuth o SAML. Queste funzionalità possono essere necessarie in scenari più complessi.
- Quando si valuta l'utilizzo della Basic Authentication, è importante considerare questi svantaggi e valutare se soddisfa i requisiti di sicurezza e funzionalità del sistema. In molti casi, può essere preferibile utilizzare meccanismi di autenticazione più sicuri e avanzati, come OAuth, JWT o l'autenticazione basata su token.



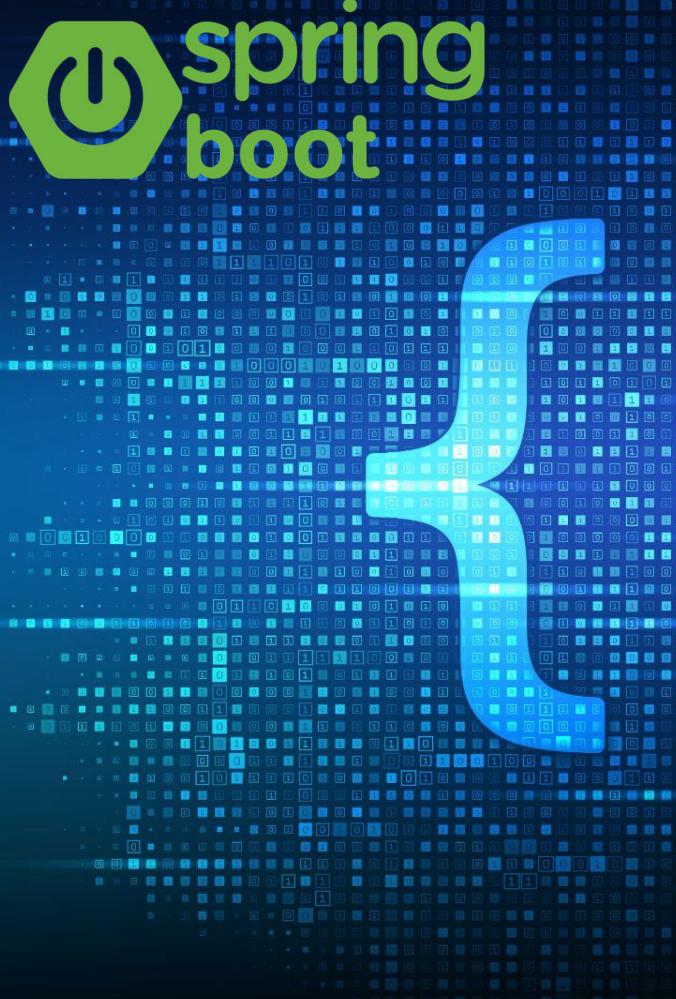
## Introduzione al Json Web Token

- Il JSON Web Token (JWT) è uno standard aperto ([RFC 7519](#)) per la creazione di token di accesso sicuri e autocontenuti.
- È un formato di token particolarmente utilizzato nell'autenticazione e nell'autorizzazione di servizi web e applicazioni.
- Un JSON Web Token è costituito da tre parti separate, separate da punti (`header.payload.signature`).
- Header: contiene informazioni relative al tipo di token e all'algoritmo di crittografia utilizzato per firmare e verificare il token. Di solito, il tipo di token è "JWT" e l'algoritmo di firma può essere ad esempio HMAC SHA256 oppure RSA.
- Payload: Il payload, anche chiamato claim, è la sezione centrale del token che contiene le informazioni utili che si desidera trasmettere. Queste informazioni possono includere l'identità dell'utente, i diritti di accesso, le autorizzazioni o qualsiasi altra informazione aggiuntiva necessaria.



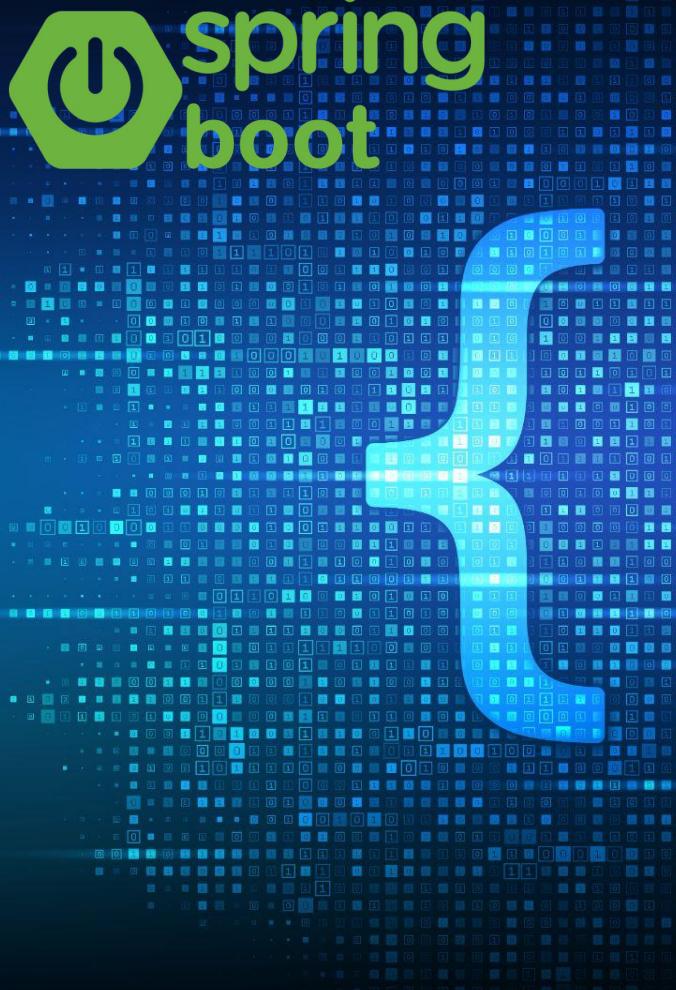
## Introduzione al Json Web Token

- **Signature:** La firma del token viene utilizzata per verificare l'integrità del token e per verificare che il mittente sia affidabile. La firma viene creata utilizzando l'header codificato, il payload codificato, una chiave segreta o una coppia di chiavi pubblica/privata e l'algoritmo di firma specificato nell'header. La firma può essere crittografata utilizzando algoritmi come HMAC o RSA.
- L'intero token JWT può essere codificato in base64 per facilitare la trasmissione come stringa sicura. Quindi, il token JWT risultante può essere utilizzato per l'autenticazione e l'autorizzazione in applicazioni web, API RESTful e servizi distribuiti.
- Un aspetto importante dei token JWT è che possono essere verificati e decodificati dal server senza dover mantenere uno stato di sessione lato server. Ogni richiesta che include un token JWT consente al server di verificare l'autenticità e l'integrità del token utilizzando la firma. Questo rende i token JWT particolarmente adatti per l'autenticazione stateless.



## Introduzione al Json Web Token

- È importante sottolineare che i token JWT non devono contenere informazioni sensibili o riservate poiché sono decodificabili da chiunque abbia accesso al token. Pertanto, è necessario evitare di includere password o altre informazioni confidenziali all'interno del payload di un token JWT.
- In sintesi, il JSON Web Token (JWT) è uno standard per la creazione di token di accesso sicuri e autocontenuti. I token JWT consentono il trasferimento sicuro di informazioni tra parti affidabili, fornendo un meccanismo per l'autenticazione e l'autorizzazione in applicazioni web e servizi distribuiti.



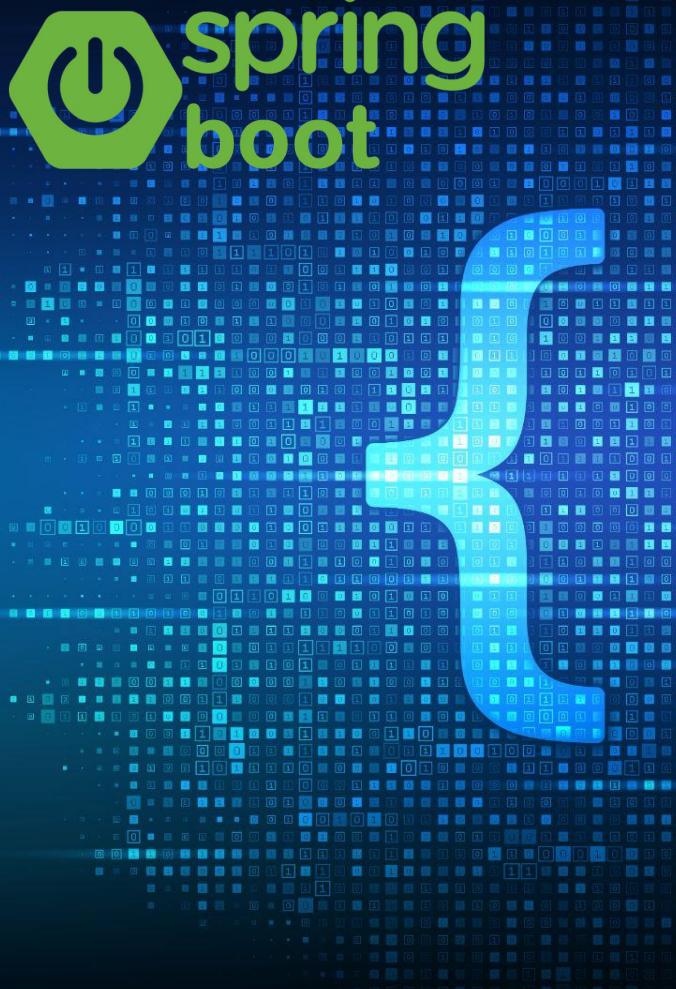
## Vantaggi del Json Web Token

- **Autenticazione stateless:** Con JWT, l'autenticazione può essere resa stateless, ovvero il server non deve memorizzare alcuna informazione di sessione per gestire l'autenticazione dell'utente. Questo significa che l'applicazione può scalare facilmente, in quanto il server non deve mantenere uno stato di sessione per ogni utente. Ogni richiesta viene autenticata utilizzando il token JWT che contiene tutte le informazioni necessarie per verificare l'identità dell'utente.
- **Riduzione del traffico di rete:** Poiché il token JWT viene inviato come parte delle richieste HTTP, non è necessario inviare le credenziali dell'utente (come username e password) ad ogni richiesta. Ciò riduce il traffico di rete, migliorando le prestazioni dell'applicazione.
- **Sicurezza:** I token JWT sono firmati digitalmente utilizzando un algoritmo di firma, come HMAC o RSA. Ciò consente di verificare l'integrità del token e garantire che non sia stato manipolato. Inoltre, i token possono essere crittografati per proteggere le informazioni sensibili contenute nel payload del token.



## Vantaggi del Json Web Token

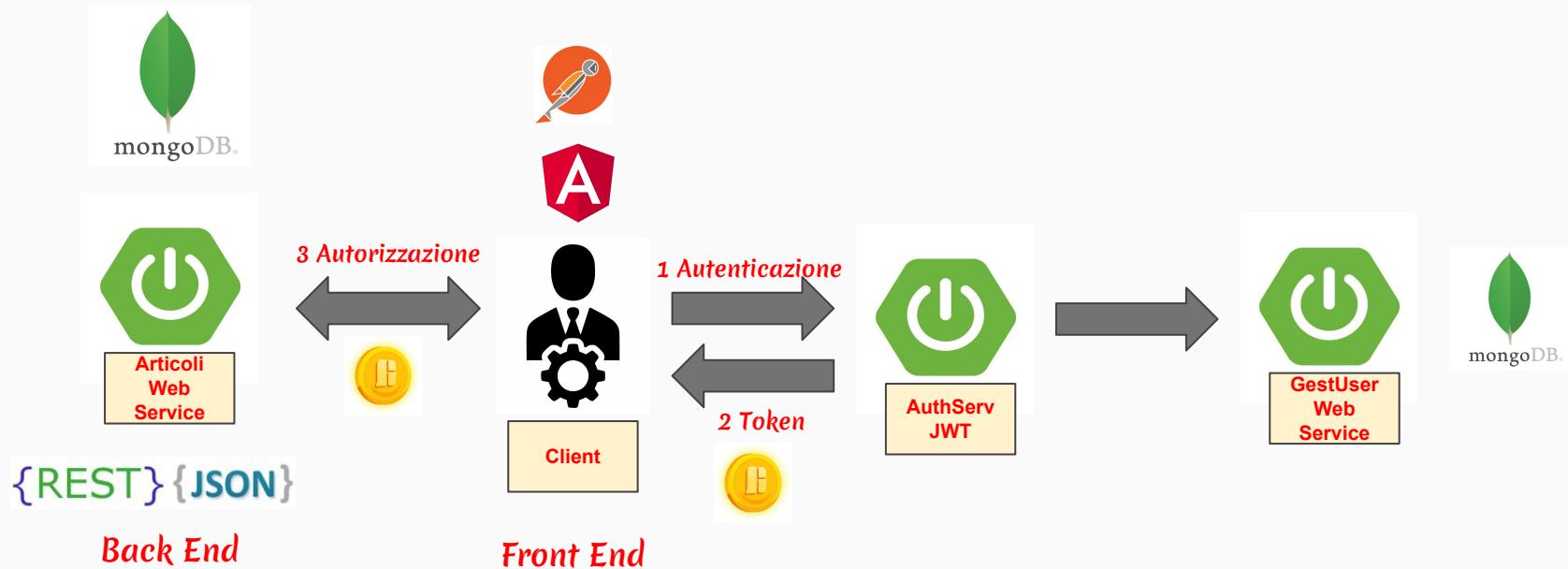
- **Scalabilità:** Poiché JWT non richiede uno stato di sessione memorizzato lato server, l'applicazione può essere facilmente scalata in orizzontale per gestire un grande numero di richieste concorrenti. Ogni istanza del server può verificare autonomamente la validità del token senza dover comunicare con un server centralizzato.
- **Personalizzazione delle informazioni nel token:** Il payload di un token JWT può contenere informazioni personalizzate, come i diritti di accesso, le autorizzazioni o altre informazioni specifiche dell'utente. Queste informazioni possono essere utilizzate per prendere decisioni di autorizzazione o per personalizzare il comportamento dell'applicazione in base al ruolo o alle autorizzazioni dell'utente.
- **Supporto per l'autenticazione multipli:** JWT consente di supportare l'autenticazione multipli in un'applicazione. Ad esempio, un'applicazione può utilizzare token JWT per l'autenticazione degli utenti, ma può anche utilizzare token diversi per l'autenticazione di servizi o di altri tipi di client.



## Vantaggi del Json Web Token

- Complessivamente, l'utilizzo di JWT in un'applicazione Spring Boot offre una soluzione sicura, scalabile e stateless per l'autenticazione e l'autorizzazione. Tuttavia, è importante adottare le pratiche di sicurezza consigliate, come la corretta gestione delle chiavi di firma e la crittografia adeguata, per garantire la sicurezza dei token JWT utilizzati nell'applicazione.

# Autenticazione basata sul JWT



# Developing Full Stack Applications

Creazione PriceArt Web  
Service