

SVILUPPO WEB

- **Pattern MVC**
- Web Front End
- Web Back End
- Maven
- Framework, IoC e DI
- Introduzione a Spring

Dott. Antonio Giovanni Lezzi

PAGINE WEB

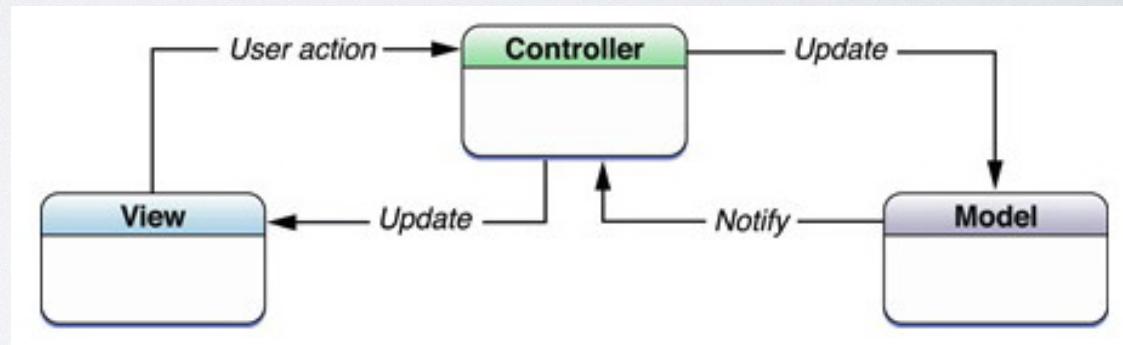
- Il World Wide Web è una delle maggiori fonti di informazione a cui oggi possiamo attingere: avendo a disposizione un collegamento a Internet ed un browser Web, un software ormai comune su qualsiasi computer, con la possibilità di consultare un patrimonio di centinaia di milioni di pagine riguardanti praticamente ogni argomento.
- Spesso queste pagine non sono documenti statici, ma vengono creati dinamicamente quando noi li richiediamo e le informazioni che contengono vengono estratte da un database.

PARADIGMA MVC

- Entriamo nel vivo della trattazione andando ad introdurre il **Model-View-Controller** design pattern (MVC) che rappresenta il design pattern più utilizzato nello sviluppo di applicazioni Web/ Mobile ecc.
- Il design pattern MVC in quanto rappresenta uno dei concetti fondamentali della programmazione ad oggetti e permette di strutturare l'applicazione in maniera molto efficiente.
- Nel design pattern MVC ogni singolo oggetto può ricoprire solo uno dei seguenti ruoli: modello, vista o controllore.
- Rappresentano una sorta di classificazione dell'oggetto che stiamo utilizzando che risultano dunque essere elementi logicamente separati ai quali però è consentita, ovviamente, una stretta comunicazione.

PARADIGMA MVC

- Il pattern non solo stabilisce che ruolo deve avere un determinato oggetto all'interno dell'applicazione, ma anche il modo in cui gli oggetti comunicano tra di loro.
- Andiamo adesso a descrivere, in maniera più approfondita, i tre tipi di oggetti che il pattern definisce.



MODELLO

- L'oggetto che ricopre il ruolo di modello **è un oggetto contenente i dati specifici di un'applicazione e si occupa di definire tutte le varie procedure che effettuano la manipolazione dei dati stessi.**
- Data la sua natura, un modello non dovrebbe avere nessun tipo di connessione diretta con un oggetto di tipo vista (che vedremo tra poco), in quanto il modello ha il compito di possedere solamente dei dati che però non devono essere legati ad un particolar tipo di visualizzazione.

VISTA

- Il ruolo principale di un oggetto vista è quello di **presentare all'utente i dati contenuti all'interno di un modello.**
- Esiste subito una netta differenza concettuale tra modello e vista: il primo è un oggetto non concreto, mentre il secondo è un oggetto concreto e con il quale l'utente può interagire andando, per esempio, a modificare i dati contenuti.
- Quello che offre la vista è dunque una realizzazione di un “oggetto” non concreto e mette a disposizione un’interfaccia per la modifica dei dati contenuti nel modello.
- Dato che anche un oggetto di tipo vista non dovrebbe avere un riferimento esplicito ad un oggetto di tipo modello è necessario introdurre il terzo concetto di questo pattern: l’oggetto **controllore**.

CONTROLLORE

- L'oggetto che ricopre il ruolo di **controllore svolge la funzione di intermediario tra oggetti di tipo vista ed oggetti di tipo modello.**
- Il numero di relazioni tra un singolo controllore ed oggetti di tipo modelli e vista è arbitraria: può essere una relazione uno ad uno come una relazione molti a molti.
- Il controllore, data la sua natura di mediatore, si occuperà di inizializzare la vista con i dati contenenti nel modello, informare la vista che i dati all'interno del modello hanno subito modifiche e rendere le modifiche effettuate dall'utente (tramite la vista) permanenti all'interno del modello.

SVILUPPO WEB

- Pattern MVC
- **Web Front End**
- Web Back End
- Maven
- Framework, IoC e DI
- Introduzione a Spring

Dott. Antonio Giovanni Lezzi

STRUTTURA DEL WEB

Le applicazioni Web sono composte da 3 elementi:

- **URL**: i nomi delle risorse nel Web
- **HTTP**: il modo in cui vengono trasferite le risorse nel Web
- **HTML**: il linguaggio delle risorse più importanti del Web

URL

- Gli Uniform Resource Locator ci permettono di individuare in maniera univoca una risorsa nel Web. Corrispondono all'indirizzo che mettiamo nelle buste delle lettere: senza indirizzo non sappiamo dove inviare fisicamente la lettera. Un Url è composto dalle seguenti parti:
- **protocollo**: normalmente http, altri valori sono ftp, telnet, mailto;
- **user e password (opzionali)**: sono le credenziali per accedere al Web Server;
- **l'indirizzo del Web Server**: può essere specificato tramite un I.P.address (157.138.20.15) o un nome (www.antonio.it);
- **una porta (opzionale)**: indica il numero di porta TCP alla quale connettersi;
- **il path della risorsa** (file normalmente) cercata;
- **i parametri della richiesta (opzionali)**: importanti per le applicazioni Web permettono di scambiare utili parametri tra il browser e l'applicazione tramite ad esempio i FORM HTML.

HTTP

- L'Hyper Text Transfer Protocol stabilisce delle regole per il trasferimento delle informazioni, il protocollo HTTP prevede le seguenti fasi:
 1. il browser (Internet Explorer/Netscape) apre una connessione col Web server specificato nel URL;
 2. il browser invia la richiesta al Web server attraverso la connessione;
 3. il Web server invia la risposta al browser attraverso la connessione;
 4. il server chiude la connessione.
- Le richieste inviate al server normalmente sono richieste di lettura di file (file HTML, testo, immagini etc.). Le richieste di lettura di file avvengono tramite il richieste HTTP di tipo GET. Talvolta le richieste fatte al Web server riguardano l'invio di dati dal Browser al Server. Queste richieste avvengono tramite richieste HTTP di tipo POST.

HTML

- L'HyperText Markup Language è il linguaggio con il quale scrivere i documenti scambiati nel Web.
- Il file HTML specifica al browser l'aspetto grafico e alcuni aspetti semantici della pagina da visualizzare. Il linguaggio HTML ha una sintassi molto semplice che prevede i seguenti TAG principali:
 - **<html>** è il tag di apertura che indica l'inizio del contenuto HTML del file;
 - **<body>** è il tag di inizio della parte "visibile" del file HTML;
 - **** è il tag principale per specificare gli aspetti semantici di un file html, cioè i collegamenti allo stesso o altri file. Questi collegamenti vengono specificati tramite il loro "nome" ovvero l'Url;
 - **<form action="URL">** è il tag che indica l'inizio di un modulo dove l'utente può inserire o selezionare dei dati da inviare dal browser al Web server.
- I tag HTML possono avere un corrispondente tag di chiusura composta nel seguente modo: </tag>. Ad esempio i tag di chiusura per i tag elencati prima sono: **</html>, </body>, , </form>**.

HTML FORM

- I form sono usati principalmente per trasferire dati dal client al server. All'interno della coppia di tag di apertura e chiusura del form possono comparire una varietà di altri tag quali:
- <INPUT NAME=nome TYPE=tipo VALUE=valore> che serve per introdurre un testo, può avere uno dei seguenti tipi:
- **TEXT**: per introdurre un testo libero di una riga;
- **SUBMIT**: visualizza un pulsante per la sottomissione;
- **PASSWORD**: per introdurre un testo nascosto;
- **FILE**: per introdurre un file.
- **<TEXTAREA NAME=nome ROWS=n>** per introdurre più linee di testo.

APPLICAZIONI NEL BROWSER

- Analizzando il funzionamento del protocollo HTTP possiamo vedere alcune sue particolarità che condizionano il funzionamento delle applicazioni Web.
- Prima di tutto ogni richiesta di risorsa richiede una nuova connessione dal browser al Web server; questo comporta un rallentamento dovuto al fatto che la gestione della connessione TCP ne incrementa le prestazioni con l'uso della connessione stessa.
- Sempre perché dopo l'invio della risorsa il server chiude la connessione, non c'è nessuna interazione tra browser e server fino alla prossima richiesta da parte del client (PULL).
- Inoltre il protocollo HTTP è senza stato e questo ci complica le cose quando dobbiamo mantenere delle informazioni riguardo le richieste precedenti.

APPLICAZIONI NEL BROWSER

- Altro problema è rappresentato dal fatto che il Web server ritorna file HTML (gli altri tipi o sono più stupidi o non sono supportati da tutti i browser) e i tipi di funzionalità è quella permessa dal linguaggio HTML all'inizio ideato solo per facilitare la navigazione, ora invece si tende ad usare il browser per implementare le funzionalità di un generico client.
- Le funzionalità aggiunte al linguaggio HTML è Javascript un linguaggio di scripting in grado di programmare il funzionamento del browser. Javascript permette di aumentare l'interattività del browser senza bisogno di interagire col server

ESEMPIO HTML

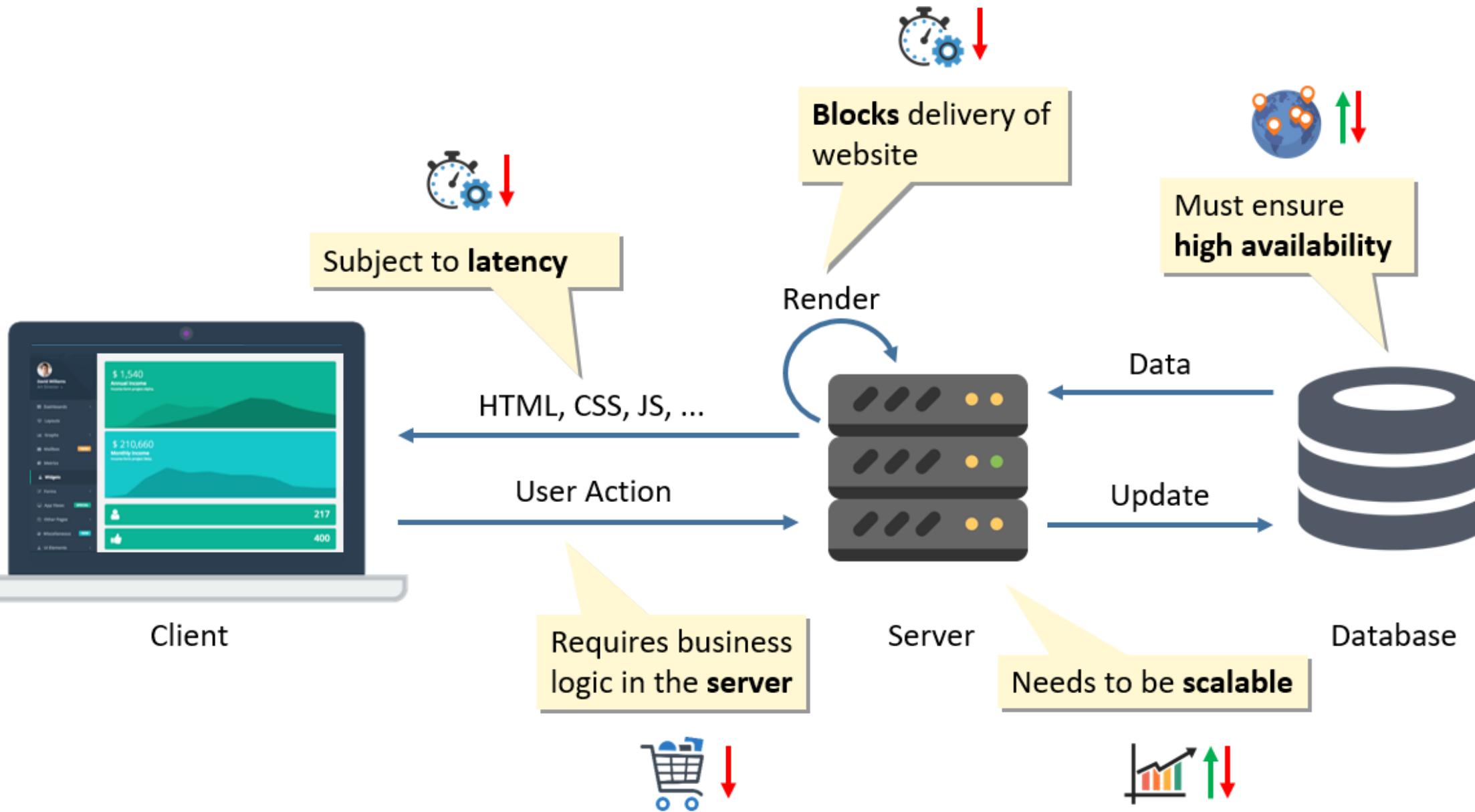
```
<html>
<body>
<form method="GET" action="testform.jsp">
NOME:
<input type="text" name="nome" value="">
<BR>
COGNOME:
<input type="text" name="cognome" value="">
<BR>
<textarea name="linee" rows=5>
Linea1
Linea2
Linea3
</textarea>
<BR>
<input type="submit" value="ok">
</form>
</body>
</html>
```

The image shows a screenshot of a web browser displaying a form. The form contains the following elements:

- A text input field labeled "NOME:" with an empty value.
- A text input field labeled "COGNOME:" with an empty value.
- A text area labeled "linee" with rows=5, containing the text "Linea1", "Linea2", and "Linea3".
- An "ok" button located below the text area.

SVILUPPO WEB

- Pattern MVC
- Web Front End
- **Web Back End**
- Maven
- Framework, IoC e DI
- Introduzione a Spring



SERVER HTTP

- Il server HTTP o server Web (ad esempio Apache o IIS), è un programma sempre in esecuzione che aspetta richieste dai client e ritorna le risorse specificate nelle richieste. Ci sono due tipi fondamentali di risorse disponibili nel Web server:
 1. **risorse statiche:** le risorse esistono prima della richiesta e/o il loro contenuto è fissato;
 2. **risorse dinamiche:** le risorse vengono create al momento della richiesta e in maniera automatica con programma.
- In generale le risorse che un Web server gestisce sono principalmente pagine html, file di testo, formati per lo scambio di documento di testo (Adobe Portable Document Format, PostScript,), immagini (GIF, JPEG, PNG), animazioni e altri contenuti multimediali. Tutti queste risorse possono essere statiche o dinamiche, anche se generalmente vengono generate pagine html
- Per la creazione delle pagine html ma non solo, sono stati predisposti dei meccanismi da agganciare ai Web server in modo da aggiungere la generazione delle pagine dinamiche (PHP, Perl etc, le servlet e le JSP)

INTRODUZIONE JAVA EE

- La tecnologia Java Enterprise Edition (J2EE) è diventata negli anni sinonimo di sviluppo di applicazioni aziendali robuste, sicure ed efficienti
- Il successo è dovuto al linguaggio object oriented Java e a come attraverso quest'ultimo è stato creato. È un continuo lavoro delle più importanti aziende di information technology: Sun, IBM, Oracle, BEA, ecc
- La tecnologia JEE può facilitare la creazione di modelli B2B (Business to Business) e B2C (Business to Consumer) e quindi permettere all'azienda lo sviluppo di nuovi servizi.
- Attraverso il modello di sviluppo proposto, rende facile l'accesso ai dati e la sua rappresentazione in diverse forme (un browser web, un applet, un dispositivo mobile, un sistema esterno, ecc).

INTRODUZIONE JAVA EE

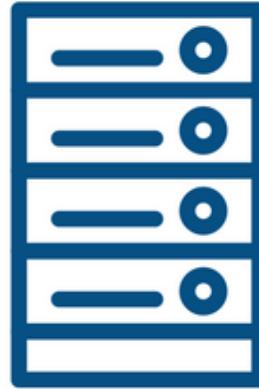
- Dal punto di vista tecnologico tutto ciò è realizzato con una struttura tecnologica a livelli, dove ogni livello implementa uno specifico servizio, a partire dal quale può essere implementato il processo aziendale, curando quindi la sua evoluzione senza preoccuparsi delle operazioni di base.
- J2EE è un raccoglitore di tecnologie che facilitano lo sviluppo di software web based distribuito
- Per vedere solo una parte di queste tecnologie non è sufficiente un corso di qualche mese

IL RUOLO DELL'APPLICATION SERVER

- Un Application Server è uno strato software residente su una macchina server che fornisce una serie di servizi, fornisce tutti i servizi che la tecnologia Java Enterprise espone
- Diciamo subito che di application server ce ne sono tantissimi. Come Bea Weblogic, IBM Websphere, Sun Application Server, Pramati, ecc ed alcuni prodotti opensource molto noti come JBoss
- In generale la principale differenza tra i diversi application server risiede in come alcune particolari operazioni (deploy, clustering, ecc) sono realizzate, in quanto, come abbiamo detto, la specifica Java Enterprise deve essere rispettata dando così vita ad uno sviluppo che è server-independent (a beneficio di chi si occupa di sviluppo)

IL RUOLO DELL'APPLICATION SERVER

- Un application server si installa e si avvia come un web server. Infatti un web server è uno dei tanti servizi che l'application server ha in sé. Inoltre esso presenta un pannello di amministrazione accessibile da postazione remota attraverso il quale possono essere definiti i diversi servizi.
- Possono essere installate le applicazioni che andremo a sviluppare, possono essere definiti servizi accessori come le transazioni o l'accesso a sorgenti di dati, i web services o le code di messaggi.
- La connessione ai servizi avviene mediante i protocolli http, rmi, iiop, in base ai tipi di servizi richiesti
- Per come lo vedremo noi, un application server assolverà ai tipici compiti di web server, per la gestione delle interfacce di presentazione dinamiche, a quelli di gestione delle componenti che contengono la logica di business (quindi la parte applicativa) e delle componenti che realizzano l'accesso ai dati



Front End

- Markup and web languages such as HTML, CSS and Javascript
- Asynchronous requests and Ajax
- Specialized web editing software
- Image editing
- Accessibility
- Cross-browser issues
- Search engine optimisation

Back End

- Programming and scripting such as Python, Ruby and/or Perl
- Server architecture
- Database administration
- Scalability
- Security
- Data transformation
- Backup

SVILUPPO WEB

- Pattern MVC
- Web Front End
- Web Back End
- **Maven**
- Framework, IoC e DI
- Introduzione a Spring

maven
Commands

MAVEN

- Apache Maven è un potente strumento per la gestione dei progetti per la piattaforma Java, in termini di compilazione del codice, distribuzione, documentazione e collaborazione del team di sviluppo.
- I principali componenti di Maven sono:
 - **pom.xml:** file xml di configurazione, contiene tutte le informazioni sul progetto (dipendenze, test, documentazione, etc...).
 - **Goal:** singola funzione che può essere eseguita sul progetto, l'equivalente Maven dei task Ant.
 - **Plug-in:** goal riutilizzabili in tutti i progetti.
 - **Repository:** directory strutturata destinata alla gestione delle librerie. Può essere locale o remoto.

POM, FILE DI CONFIGURAZIONE

- Project Object Model e ogni singolo progetto è descritto attraverso il file di configurazione pom.xml, senza il quale Maven non può fare niente.
- Il POM è un file XML e guida l'esecuzione in Maven definendo in modo chiaro l'identità e la struttura di un progetto in ogni suo aspetto contiene:
 - *informazioni generali del progetto*
 - *dipendenze*
 - *processo di compilazione*
 - *fasi secondarie come la generazione di documentazione.*

POM, FILE DI CONFIGURAZIONE

- Un POM ai minimi termini è composto da un tag root <project> che conterrà i tag:
 - <modelVersion> che dichiara a quale versione di POM questo progetto è conforme
 - <groupId> ID del gruppo del progetto
 - <artifactId> ID dell'artefatto (del progetto)
 - <version> cioè la versione del progetto
 - <packaging> che è il tipo di archivio che vogliamo esportare (jar, war o ear)

POM, FILE DI CONFIGURAZIONE

- I parametri **groupId**, **artifactId**, **version** e **packaging** identifieranno univocamente un progetto. Se packaging non è specificato nel POM, assumerà “jar” a valore di default.
- Ecco un esempio di **pom.xml** minimale con packaging non specificato:

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1</version>
</project>
```

POM, FILE DI CONFIGURAZIONE

- Con Maven la gestione e download automatico delle librerie necessarie al progetto è una componente fondamentale.
- Maven di default si collegherà al repository remoto e scaricherà lui tutti i pacchetti di cui necessita, comprese le dipendenze degli stessi.
- Per inserire questa dipendenza è sufficiente aggiungere nel file `pom.xml` un tag `<dependencies>` e dentro questo mettiamo quanto segue:

```
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.13</version>
</dependency>
```

SVILUPPO WEB

- Pattern MVC
- Web Front End
- Web Back End
- Maven
- **Framework, IoC e DI**
- Introduzione a Spring



FRAMEWORK

- Un web framework viene definito come **un ambiente contenente un insieme di librerie e altri strumenti che possono esser configurate** all'interno del quale sviluppare applicazioni native e siti web.
- I framework sono progettati per supportare lo sviluppatore nella creazione di progetti e, più nello specifico, per **facilitare il processo di scrittura del codice**.
- Frequentemente in passato gli sviluppatori realizzavano i loro prodotti partendo da zero. Oggi è considerato un comportamento da evitare, perché partire dalle basi significa imbattersi spesso in problemi di routine che, per essere risolti, richiedono tempo.
- La loro funzione principale è aiutare gli sviluppatori ad accelerare il processo di creazione di applicazioni native o sito web.

FRAMEWORK

- Innanzitutto i framework **riducono lo stress che comporta realizzare un progetto complesso** in quanto sono loro a gestire il controllo dei flussi.
- Hanno un comportamento di default, che si suppone sia progettato **per aiutare gli sviluppatori a risolvere i problemi, è facile capire che queste piattaforme seguono regole precise.**
- Si ha la necessità di **aggiungere funzionalità** specifiche alla piattaforma di base? È facile **estenderla o specializzarla**: basta aggiungere il code necessario.
- **L'unica regola** da tenere presente quando si modifica il proprio framework è questa: **aggiungete il vostro codice ma non cambiate quello esistente.**

FRAMEWORK

- L'obiettivo principale dei framework è quello di ridurre il carico di lavoro degli sviluppatori, essi promuovono il riutilizzo del codice.
- Quando si deve creare un oggetto specifico, sarà sufficiente scrivere il codice applicativo una volta sola, anche se l'oggetto in questione verrà utilizzato più di una volta all'interno del progetto.
- La modularità è una diretta conseguenza del modello dell'architettura.
- La maggioranza dei framework, infatti, utilizza il modello MVC (model-view-controller) o varianti di esso e quindi la struttura logica rimane separata dall'interfaccia utente.

IOC E DEPENDENCY INJECTION

- L'**inversione del controllo** (*inversion of control*, oppure **IoC**) è un pattern per cui un componente di livello applicativo riceve il controllo da un componente appartenente a un libreria riusabile.
- Questo schema ribalta quello tradizionale della programmazione procedurale, dove il codice applicativo svolge i propri compiti richiamando (e quindi *passando il controllo a*) procedure di libreria.

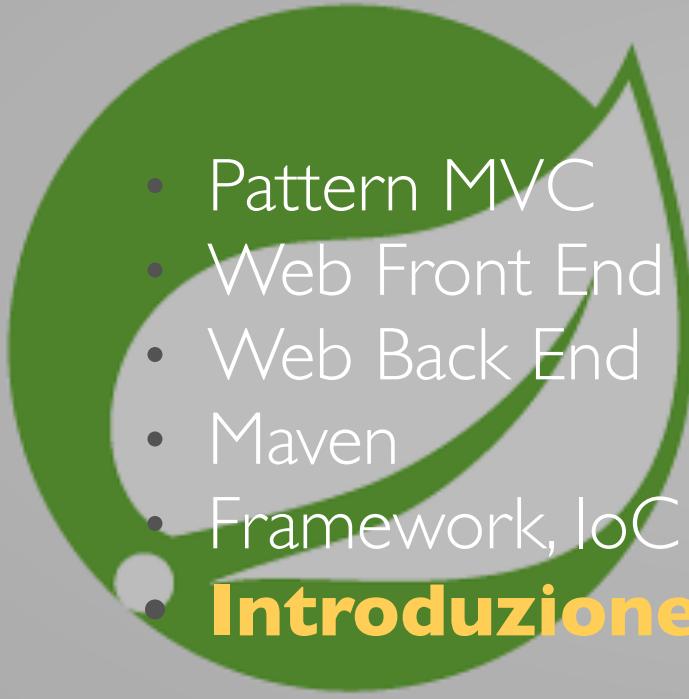
IOC E DEPENDENCY INJECTION

- La Dependency injection **è una delle tecniche con le quali si può attuare l'inversione del controllo.**
- Essa prende il controllo su tutti gli aspetti di creazione degli oggetti e delle loro dipendenze.
- Dependency injection (DI) **è un design pattern** della Programmazione orientata agli oggetti il cui scopo è quello di semplificare lo sviluppo e migliorare la testabilità di software di grandi dimensioni.
- Per utilizzare tale design pattern **è sufficiente dichiarare le dipendenze di cui un componente necessita** (dette anche interface contracts).

IOC E DEPENDENCY INJECTION

- **Quando il componente verrà istanziato, un iniettore si prenderà carico di risolvere le dipendenze** (attuando dunque l'inversione del controllo).
- Se è la prima volta che si tenta di risolvere una dipendenza l'injector istanzierà il componente dipendente, lo salverà in un contenitore di istanze e lo restituirà.
- Se non è la prima volta, allora restituirà la copia salvata nel contenitore. Una volta risolte tutte le dipendenze, il controllo può tornare al componente applicativo.

SVILUPPO WEB

- 
- Pattern MVC
 - Web Front End
 - Web Back End
 - Maven
 - Framework, IoC e DI
 - **Introduzione a Spring**

spring

SPRING

- La sua filosofia, detta object oriented, mira a semplificare al massimo il processo di programmazione: ogni parte di codice relativa ad una specifica funzione è un “blocco”, da aggiungere o rimuovere a piacere senza andare a compromettere le funzionalità del programma già sviluppato.
- La semplicità di questo framework è una delle forme caratterizzanti di Spring Boot: questo ambiente di sviluppo è ideato specificatamente per rendere la programmazione facile e comprensibile anche per i neofiti, pur offrendo straordinarie possibilità agli sviluppatori più navigati.
- Il framework Spring usa molto diffusamente la Dependency Injection con il risultato, tra le altre cose, di eliminare dal codice applicativo ogni logica di inizializzazione.

SPRING

- È possibile iniziare a costruire una web app sfruttando Gradle o Maven, due sistemi di automatizzazione della programmazione perfettamente compatibili con i sistemi Spring.
- Requisiti minimi per implementare una applicazione, usando Spring Boot:
 - JDK 1.8 o superiore
 - Gradle 4+ or Maven 3.2+
- Per poter creare un template minimale di un progetto, esistono diverse modalità per farlo, si può accedere alla pagina
 - <https://start.spring.io>



Spring Initializr

Bootstrap your application

Project

Maven Project

Gradle Project

Language

Java

Kotlin

Groovy

Spring Boot

2.2.0 RC1

2.2.0 (SNAPSHOT)

2.1.10 (SNAPSHOT)

2.1.9

Project Metadata

Group

com.example

Artifact

demo

› Options

Dependencies



Search dependencies to add

Web, Security, JPA, Actuator, Devtools...

Selected dependencies

No dependency selected

SPRING WEB APPLICATION

- Nell'approccio di Spring alla costruzione di siti Web, le richieste HTTP sono gestite da un controller.
- È possibile identificare facilmente queste richieste mediante l'annotazione **@Controller**.
- Nel GreetingController gestisce le richieste GET per / saluto restituendo il nome di una vista, in questo caso “saluto”.
- Una vista è responsabile del rendering del contenuto HTML

SPRING WEB APPLICATION

```
// src/main/java/hello/GreetingController.java
package hello;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;

@Controller
public class GreetingController {
    @GetMapping("/greeting")
    public String greeting(@RequestParam(name="name", required=false,
defaultValue="World") String name, Model model) {
        model.addAttribute("name", name);
        return "greeting";
    }
}
```

SPRING WEB APPLICATION

- Questo controller è conciso e semplice, analizziamo passo dopo passo.
- L'annotazione **@GetMapping** garantisce che le richieste HTTP GET a / greeting siano associate al metodo greeting().
- **@RequestParam** associa il valore della query Nome parametro stringa al parametro nome del metodo greeting(). Questo parametro String della query non è obbligatorio; se è assente nella richiesta, viene utilizzato il valore predefinito di "Mondo". Il valore del parametro name viene aggiunto a un oggetto Model, rendendolo infine accessibile al modello di visualizzazione.
- L'implementazione del metodo body si basa su una tecnologia di visualizzazione, in questo caso Thymeleaf, per eseguire il rendering lato HTML dell'HTML.
- Thymeleaf analizza il modello greeting.html di seguito e valuta l'espressione th:text per rendere il valore del parametro \${name} impostato nel controller.

SPRING WEB APPLICATION

```
<!--src/main/resources/templates/greeting.html-->
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Getting Started: Serving Web Content</title>
    <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />
</head>
<body>
    <p th:text=""Hello, ' + ${name} + '!"' />
</body>
</html>
```

SPRING WEB APPLICATION

- Sebbene sia possibile impacchettare questo servizio come un file WAR tradizionale per la distribuzione su un server delle applicazioni esterno, l'approccio più semplice dimostrato di seguito crea un'applicazione autonoma. Impacchettate tutto in un unico file JAR eseguibile, guidato da un buon vecchio metodo main() Java.
- Lungo il percorso, utilizzi il supporto di Spring per incorporare il contenitore servlet Tomcat come runtime HTTP, anziché distribuirlo a un'istanza esterna.

```
src/main/java/hello/Application.java
package hello;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

SPRING WEB APPLICATION

- **@SpringBootApplication** è un'annotazione di praticità che aggiunge tutto quanto segue:
 - **@Configuration**: etichetta la classe come fonte di definizioni bean per il contesto dell'applicazione.
 - **@EnableAutoConfiguration**: indica a Spring Boot di iniziare ad aggiungere bean in base alle impostazioni del percorso di classe, altri bean e varie impostazioni delle proprietà. Ad esempio, se spring-webmvc si trova sul percorso di classe, questa annotazione contrassegna l'applicazione come un'applicazione Web e attiva comportamenti chiave, come l'impostazione di DispatcherServlet.
 - **@ComponentScan**: dice a Spring di cercare altri componenti, configurazioni e servizi nel pacchetto Hello, permettendogli di trovare i controller.
- Il metodo main() utilizza il metodo SpringApplication.run() di Spring Boot per avviare un'applicazione.
Da notare che non c'era una sola riga di XML e non esiste nemmeno un file web.xml.
- Questa applicazione Web è pura al 100% Java e non è stato necessario gestire la configurazione di alcun file o infrastruttura.

SPRING WEB APPLICATION

- Le risorse statiche, come HTML o JavaScript o CSS, possono essere facilmente servite dall'applicazione Spring Boot semplicemente rilasciandole nel posto giusto nel codice sorgente. Per impostazione predefinita, Spring Boot fornisce contenuto statico da risorse nel percorso di classe in "/static" (o "/ public").
- La risorsa index.html è speciale perché viene utilizzata come "pagina di benvenuto" se esiste, il che significa che verrà servita come risorsa di root, ovvero su `http://localhost:8080 /` nel nostro esempio. Quindi crea questo file:

```
src/main/resources/static/index.html
<!DOCTYPE HTML>
<html>
<head>
    <title>Getting Started: Serving Web Content</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
    <p>Get your greeting <a href="/greeting">here</a></p>
</body>
</html>
```

- E quando si avvia l'applicazione vera visualizzato il file HTML all'indirizzo `http://localhost:8080/`.

The background of the slide features a futuristic server room. The floor has glowing blue and purple light trails forming a heart shape. The room is filled with tall server racks, each with multiple glowing blue and green lights. The ceiling is dark with some glowing blue and purple elements. The overall atmosphere is high-tech and futuristic.

MYSQL FOR DEVELOPER

Dott. Antonio Giovanni Lezzi

SVILUPPO WEB

- **Validazione**
- Web Service
- ORM e JPA
- Hibernate

SPRING VALIDAZIONE

- Se l'applicazione prevede la convalida di alcuni valori come ad esempio il nome e dell'età di un utente, si deve procedere con la creazione di una classe per consentire la creazione di una persona. (src/main/java/hello/PersonForm.java)

```
package hello;

import javax.validation.constraints.Min;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;

public class PersonForm {
    @NotNull
    @Size(min=2, max=30)
    private String name;

    @NotNull
    @Min(18)
    private Integer age;

    public String getName() { return this.name; }
    public void setName(String name) { this.name = name; }
    public Integer getAge() { return age; }
    public void setAge(Integer age) { this.age = age; }
    public String toString() { return "Person(Name: " + this.name + ", Age: " + this.age + ")"; }
}
```

SPRING VALIDAZIONE

- La classe PersonForm ha due attributi: nome ed età. È contrassegnato con diverse annotazioni di convalida standard:
- **@Size (min = 2, max = 30)** consentirà solo nomi di lunghezza compresa tra 2 e 30 caratteri
- **@NotNull** non consentirà un valore null, che è ciò che Spring MVC assegna ad un campo
- **@Min(18)** non lo consentirà se l'età è inferiore a 18 anni
- Inoltre sono stati generati anche getter/setter per nome ed età e un metodo `toString()`.

SPRING VALIDAZIONE

- Definita la classe di supporto del modello si crea un controller web

```
src/main/java/hello/WebController.java
package hello;

import javax.validation.Valid;
import org.springframework.stereotype.Controller;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.servlet.config.annotation.ViewControllerRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Controller
public class WebController implements WebMvcConfigurer {
    @Override
    public void addViewControllers(ViewControllerRegistry registry) {
        registry.addViewController("/results").setViewName("results");
    }

    @GetMapping("/")
    public String showForm(PersonForm personForm) { return "form"; }

    @PostMapping("/")
    public String checkPersonInfo(@Valid PersonForm personForm, BindingResult bindingResult) {
        if (bindingResult.hasErrors())
            return "form";
        return "redirect:/results";
    }
}
```

SPRING VALIDAZIONE

- Il controller mostrato ha un metodo GET e POST, entrambi associati a /.
 1. Il metodo showForm restituisce il nome della view da renderizzare. Include un PersonForm nella sua firma del metodo in modo che il modello possa associare gli attributi del modulo a un PersonForm.
 2. Il metodo checkPersonFormInfo accetta due argomenti:
 - Un oggetto personForm contrassegnato con @Valid per raccogliere gli attributi compilati.
 - Un oggetto bindingResult che consente di verificare e recuperare errori di convalida.
- È possibile recuperare tutti gli attributi dal modulo associato all'oggetto PersonForm e nel caso si verificano errori, si richiede di visualizzare il form con i dati inseriti e visualizzati tutti gli attributi di errore.
- Se tutti gli attributi della persona sono validi, allora si reindirizza il browser alla view dei risultati finali.

SPRING VALIDAZIONE

```
src/main/resources/templates/form.html
<html>
  <body>
    <form action="#" th:action="@{/}" th:object="${personForm}" method="post">
      <table>
        <tr>
          <td>Name:</td>
          <td><input type="text" th:field="*{name}" /></td>
          <td th:if="#{fields.hasErrors('name')}" th:errors="*{name}">Name Error</td>
        </tr>
        <tr>
          <td>Age:</td>
          <td><input type="text" th:field="*{age}" /></td>
          <td th:if="#{fields.hasErrors('age')}" th:errors="*{age}">Age Error</td>
        </tr>
        <tr>
          <td><button type="submit">Submit</button></td>
        </tr>
      </table>
    </form>
  </body>
</html>
```

SPRING VALIDAZIONE

- La pagina contiene un form semplice con ogni campo in uno slot separato di una tabella.
- Il form è impostato per avere come action di inoltrare i dati verso l'endpoint /.
- Nel caso in cui questa pagina sia stata invocata dal metodo GET, nell'ipotesi si volesse vedere il dettaglio o aggiornare il dato nel database si passa al metodo l'oggetto personForm (come indicato nel metodo GET nel controller web).
- Esistono due campi nel bean PersonForm contrassegnati da th: field = "{name}" e th: field = "{age}". Accanto a ciascun campo è presente un elemento secondario utilizzato per mostrare eventuali errori di convalida.

SPRING VALIDAZIONE

- Infine si ha un pulsante per poter inviare i dati alla pagina/endpoint indicata dell'action.
- Se l'utente immette un nome o un'età che viola i vincoli @Valid, tornerà a questa pagina con il messaggio di errore visualizzato.
- Se vengono immessi un nome e un'età validi, l'utente viene indirizzato alla pagina Web successiva.

`src/main/resources/templates/results.html`

```
<html>
  <body>
    Congratulations! You are old enough to sign up for this site.
  </body>
</html>
```

SVILUPPO WEB



SPRING WEB SERVICE

- Qualora si volessero gestire dei servizi web, che siano in grado di disporre dei dati come interfaccia al database server; si deve creare una classe che gestisca tali richieste.
- Prima di tutto si pensano quali possono essere gli endpoint con i parametri di input e il risultato Json da generare (si consiglia di fornire una documentazione dettagliata)
- Il servizio gestirà le richieste GET per / greeting, facoltativamente con un parametro name nella stringa di query. La richiesta GET dovrebbe restituire una risposta di 200 OK con JSON nel corpo che rappresenta un saluto. Dovrebbe assomigliare a qualcosa di simile a questo:

```
{  
  "id": 1,  
  "content": "Hello, World!"  
}
```

- Il campo ID è un identificatore univoco per il saluto e il contenuto è la rappresentazione testuale del saluto.

SPRING WEB SERVICE

- Per modellare la rappresentazione di saluto, si crea una classe di rappresentazione delle risorse. Si crea un POJO con costruttori e accessori per i dati di id e contenuto:

```
src/main/java/hello/Greeting.java  
package hello;
```

```
public class Greeting {  
    private final long id;  
    private final String content;  
  
    public Greeting(long id, String content) {  
        this.id = id;  
        this.content = content;  
    }  
  
    public long getId() { return id; }  
    public String getContent() { return content; }  
}
```

- Spring utilizza la libreria Jackson JSON per eseguire il marshalling automatico delle istanze di tipo Saluto in JSON e successivamente si crea il controller delle risorse che genererà questi saluti.

SPRING WEB SERVICE

- Nell'approccio di Spring alla creazione di servizi web RESTful, le richieste HTTP sono gestite da un controller. Questi componenti sono facilmente identificabili dall'annotazione @RestController e il GreetingController in basso gestisce le richieste GET per /greeting restituendo una nuova istanza della classe Greeting:

src/main/java/hello/GreetingController.java
package hello;

```
import java.util.concurrent.atomic.AtomicLong;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class GreetingController {
    private static final String template = "Hello, %s!";
    private final AtomicLong counter = new AtomicLong();

    @RequestMapping("/greeting")
    public Greeting greeting(@RequestParam(value="name", defaultValue="World") String name) {
        return new Greeting(counter.incrementAndGet(),
                            String.format(template, name));
    }
}
```

SPRING WEB SERVICE

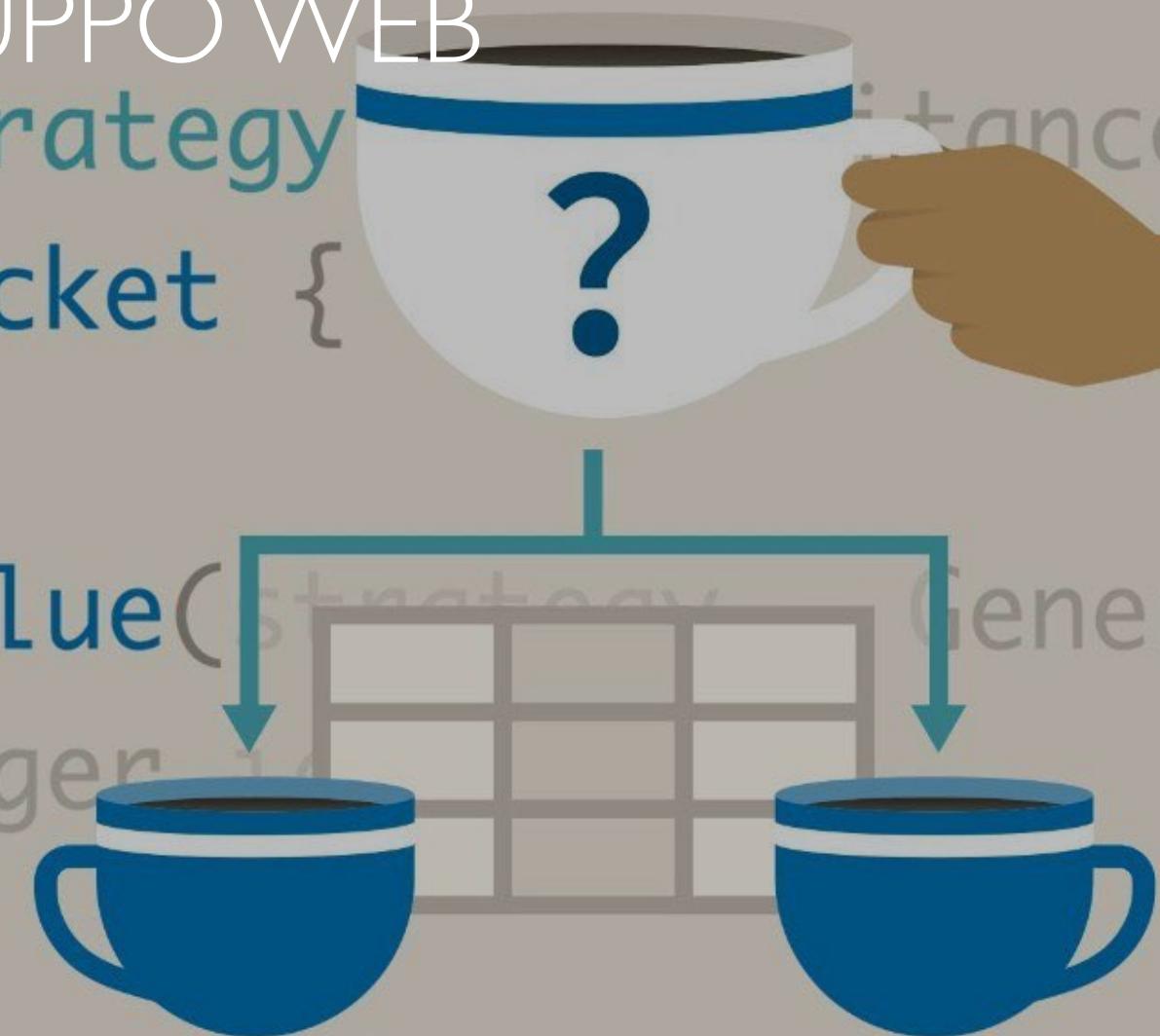
- Questo controller è conciso e semplice. Analizziamolo passo dopo passo.
- L'annotazione **@RequestMapping** garantisce che le richieste HTTP a /greeting siano associate al metodo greeting().
- L'esempio non specifica il metodo Rest da usare GET o PUT, POST e così via, perché @RequestMapping mappa tutte le operazioni HTTP per impostazione predefinita. Usa **@RequestMapping(method = GET)** per restringere questo mapping.
- **@RequestParam** associa il valore del nome del parametro della stringa di query al parametro name del metodo greeting(). Se il parametro name è assente nella richiesta, viene utilizzato il valore predefinito di "Mondo".
- L'implementazione del corpo del metodo crea e restituisce un nuovo oggetto Saluto con attributi id e contenuto in base al valore successivo dal contatore e formatta il nome dato utilizzando il modello di saluto.

SPRING WEB SERVICE

- Una differenza fondamentale tra un controller MVC tradizionale e il controller del servizio Web RESTful sopra è il modo in cui viene creato il corpo della risposta HTTP.
- Anziché basarsi su una tecnologia di visualizzazione per eseguire il rendering sul lato server dei dati di saluto in HTML, questo controller di servizi Web RESTful semplicemente popola e restituisce un oggetto di saluto. I dati dell'oggetto verranno scritti direttamente nella risposta HTTP come JSON.
- Questo codice utilizza la nuova annotazione **@RestController** di Spring 4, che contrassegna la classe come controller in cui ogni metodo restituisce un oggetto dominio anziché una vista. È una scorciatoia per **@Controller** e **@ResponseBody** messi insieme.
- L'oggetto Greeting deve essere convertito in JSON. Grazie al supporto del convertitore di messaggi HTTP di Spring, non è necessario eseguire questa conversione manualmente. Poiché Jackson 2 si trova sul percorso di classe, Spring MappingJackson2HttpMessageConverter viene automaticamente scelto per convertire l'istanza di saluto in JSON.

```
@Entity  
@Inheritance(strategy=  
public class Ticket {
```

- Validazione
- Web Service
- **ORM e JPA**
- Hibernate



ORM E JPA

- Object-Relational Mapping (ORM) è una tecnica di programmazione che favorisce l'integrazione di sistemi software aderenti al paradigma della programmazione orientata agli oggetti con sistemi RDBMS.
- Un prodotto ORM fornisce, mediante un'interfaccia orientata agli oggetti, tutti i servizi inerenti alla persistenza dei dati, astraendo nel contempo le caratteristiche implementative dello specifico RDBMS utilizzato.

ORM E JPA

- JPA è Java Persistence API, una specifica che gestisce la mappatura degli oggetti Java e le loro relazioni con un database relazionale.
- Questo è chiamato mappatore di oggetti relazionale (ORM) ed è un'alternativa per (o integrare a) il JDBC di livello più basso.
- È molto utile quando si persegue un approccio orientato a Java e quando i grafici di oggetti complessi devono essere mantenuti.

HIBERNATE

SVILUPPO WEB

- Validazione
- Web Service
- ORM e JPA
- **Hibernate**

SPRING DATABASE

- Le procedure fondamentali per poter utilizzare dati da un database sono:
- Apri un terminale (prompt dei comandi) e aprire un client MySQL con un determinato utente.
- Supponiamo ci si collega a MySQL come root e consente l'accesso all'utente da tutti gli host.
- Per creare un nuovo database, eseguire i seguenti comandi al prompt mysql:

```
mysql> create database db_example; -- Creates the new database
mysql> create user 'springuser'@'%' identified by 'ThePassword'; -- Creates the user
mysql> grant all on db_example.* to 'springuser'@'%'; -- Gives all privileges to the new user on the newly created database
```

SPRING DATABASE: CONFIGURAZIONE

- Spring Boot fornisce le impostazioni predefinite su H2. Quando si desidera utilizzare qualsiasi altro database, è necessario definire gli attributi di connessione nel file application.properties.
- Si procede con la creazione di un file di risorse chiamato src/main/resources/application.properties,:
`spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://${MYSQL_HOST}:localhost:3306/db_example
spring.datasource.username=springuser
spring.datasource.password=ThePassword`

- L'attributo spring.jpa.hibernate.ddl-auto può essere none, update, create, or create-drop.

1. **none**: l'impostazione predefinita per MySQL. Non viene apportata alcuna modifica alla struttura del database.
2. **update**: Hibernate modifica il database in base alle strutture entità specificate.
3. **create**: crea il database ogni volta ma non lo si cancella alla chiusura.
4. **create-drop**: crea il database e lo rilascia alla chiusura di SessionFactory.

SPRING DATABASE: CONFIGURAZIONE

- È necessario iniziare con la creazione o l'aggiornamento del database, poiché non si dispone ancora della struttura del database.
- Dopo la prima esecuzione, è possibile passare a aggiornamento o nessuno.
- Utilizzare aggiorna quando si desidera apportare alcune modifiche alla struttura del database.
- L'impostazione predefinita per H2 e altri database incorporati è create-drop. Per altri database, come MySQL, il valore predefinito è none.
- È buona prassi di sicurezza, dopo che il database è in stato di produzione, impostarlo su none, revocare tutti i privilegi dell'utente MySQL connesso all'applicazione Spring e fornire all'utente MySQL solo SELECT, UPDATE, INSERT e DELETE.

SPRING DATABASE: ENTITY

- Il modello **@Entity** consente di creare un modello che si relazione alla tabella del database avente lo stesso nome della classe, come mostrato nel codice seguente: (in src/main/java/hello/User.java):

```
package com.example.accessingdatamysql;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity // This tells Hibernate to make a table out of this class
public class User {
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private Integer id;
    private String name;
    private String email;

    public Integer getId() { return id; }
    public void setId(Integer id) { this.id = id; }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }
}
```

- Hibernate automaticamente traduce l'entità in tabella.

SPRING DATABASE: REPOSITORY

- Si deve creare il repository per mantenere i record degli utenti (in src/main/java/com/example/accessingdatamysql/UserRepository.java):

```
package com.example.accessingdatamysql;

import org.springframework.data.repository.CrudRepository;
import com.example.accessingdatamysql.User;

// This will be AUTO IMPLEMENTED by Spring into a Bean called userRepository
// CRUD refers Create, Read, Update, Delete
public interface UserRepository extends CrudRepository<User, Integer> {
}
```

- Spring automaticamente implementa questo repository in un bean che ha lo stesso nome (chiamato userRepository).

SPRING DATABASE: CONTROLLER

- Si procede con la creazione di un controller per gestire le richieste HHTP nell'applicazione (in src/main/java/hello/MainController.java):

```
package com.example.accessingdatamysql;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller // This means that this class is a Controller
@RequestMapping(path="/demo") // This means URL's start with /demo (after Application path)
public class MainController {
    @Autowired // This means to get the bean called userRepository
        // Which is auto-generated by Spring, we will use it to handle the data
    private UserRepository userRepository;
```

SPRING DATABASE: CONTROLLER

```
@PostMapping(path="/add") // Map ONLY POST Requests
public @ResponseBody String addNewUser (@RequestParam String name, @RequestParam String email) {
    // @ResponseBody means the returned String is the response, not a view name
    // @RequestParam means it is a parameter from the GET or POST request
    User n = new User();
    n.setName(name);
    n.setEmail(email);
    userRepository.save(n);
    return "Saved";
}

@GetMapping(path="/all")
public @ResponseBody Iterable<User> getAllUsers() {
    // This returns a JSON or XML with the users
    return userRepository.findAll();
}
```

- Nel controller si mostrano la gestione delle richieste **POST** e **GET** per i due endpoint con **@RequestMapping** che mappa tutte le operazioni HTTP.

SPRING DATABASE: MODELLO

- In questa classe consente la memorizzazione degli oggetti Customer tramite le annotazioni JPA (file src/main/java/com/example/accessingdatajpa/Customer.java):

```
package com.example.accessingdatajpa;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Customer {
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private Long id;
    private String firstName;
    private String lastName;
```

SPRING DATABASE: MODELLO

```
protected Customer() {}
public Customer(String firstName, String lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
}

@Override
public String toString() { return String.format("Customer[id=%d,
firstName='%s', lastName='%s']", id, firstName, lastName); }

public Long getId() { return id; }
public String getFirstName() { return firstName; }
public String getLastName() { return lastName; }
}
```

SPRING DATABASE: MODELLO

- Il costruttore di default esiste solo per esser usato da JPA, il programmatore non interagisce direttamente infatti è definito come protetto.
- L'altro costruttore crea le istanze di Customer per esser salvate nel database.
- La classe Customer è annotto con l'annotation `@Entity`, consente di fornire a JPA che è una entità (non esiste l'annotation `@Table` perché si assume che questa entità è mappata su una tabella nominata `Customer`).
- La classe Customer ha una entità id annotata con `@id`, per fare in modo che JPA possa identificare che ha un elemento di ID, mentre `@GeneratedValue` indica che ID deve esser generato automaticamente.
- Le altre due proprietà `firstName` e `lastName` sono stati lasciati senza annotazioni, si assume che gli attributi vengano mappate con le colonne avente lo stesso nome.

SPRING DATABASE: DAO

- Spring Data JPA focalizza l'uso di JPA a memorizzare i dati in un database relazionale.
- Create Simple Queries
- Spring Data JPA focuses on using JPA to store data in a relational database. La sua caratteristica più interessante è la possibilità di creare implementazioni di repository automaticamente, in fase di esecuzione, da un'interfaccia di repository.

- file (src/main/java/com/example/accessingdatajpa/CustomerRepository.java):

```
package com.example.accessingdatajpa;

import java.util.List;
import org.springframework.data.repository.CrudRepository;

public interface CustomerRepository extends CrudRepository<Customer, Long> {
    List<Customer> findByLastName(String lastName);
    Customer findById(long id);
}
```

SPRING DATABASE: DAO

- CustomerRepository estende l'interfaccia di CrudRepository. Il tipo di entità e ID con cui funziona, Customer e Long, sono specificati nei parametri generici su CrudRepository.
- Estendendo CrudRepository, CustomerRepository eredita diversi metodi per lavorare con la persistenza del cliente, inclusi i metodi per salvare, eliminare e trovare entità del cliente.
- Spring Data JPA consente inoltre di definire altri metodi di query dichiarandone la firma. Ad esempio, CustomerRepository include il metodo findByLastName().
- In una tipica applicazione Java, ci si potrebbe aspettare di scrivere una classe che implementa CustomerRepository. Tuttavia, questo è ciò che rende Spring Data JPA così potente: non è necessario scrivere un'implementazione dell'interfaccia del repository. Spring Data JPA crea un'implementazione quando si esegue l'applicazione.

SPRING DATABASE: CREAZIONE RECORD

- Si procede con la modifica della classe che Initializr ha creato. Per ottenere l'output (sulla console, in questo esempio), è necessario impostare un logger. Quindi è necessario impostare alcuni dati e utilizzarli per generare output. Il seguente elenco mostra la classe AccessingDataJpaApplication (in [src/main/java/com/example/accessingdatajpa/AccessingDataJpaApplication.java](#)):

```
package com.example.accessingdatajpa;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

@SpringBootApplication
public class AccessingDataJpaApplication {
    private static final Logger log = LoggerFactory.getLogger(AccessingDataJpaApplication.class);
    public static void main(String[] args) {
        SpringApplication.run(AccessingDataJpaApplication.class);
    }
}
```

SPRING DATABASE: CREAZIONE RECORD

```
@Bean
public CommandLineRunner demo(CustomerRepository repository) {
    return (args) -> {
        // save a few customers
        repository.save(new Customer("Jack", "Bauer"));
        repository.save(new Customer("Chloe", "O'Brian"));
        repository.save(new Customer("Kim", "Bauer"));
        repository.save(new Customer("David", "Palmer"));
        repository.save(new Customer("Michelle", "Dessler"));

        // fetch all customers
        log.info("Customers found with findAll():");
        log.info("-----");
        for (Customer customer : repository.findAll()) {
            log.info(customer.toString());
        }
        log.info("");

        // fetch an individual customer by ID
        Customer customer = repository.findById(1L);
        log.info("Customer found with findById(1L):");
        log.info("-----");
        log.info(customer.toString());
    };
}
```

SPRING DATABASE: CREAZIONE RECORD

```
log.info("");

// fetch customers by last name
log.info("Customer found with findByLastName('Bauer'):");
log.info("-----");
repository.findByLastName("Bauer").forEach(bauer -> {
    log.info(bauer.toString());
});
// for (Customer bauer : repository.findByLastName("Bauer")) {
//     log.info(bauer.toString());
// }
log.info("");
};

}
```

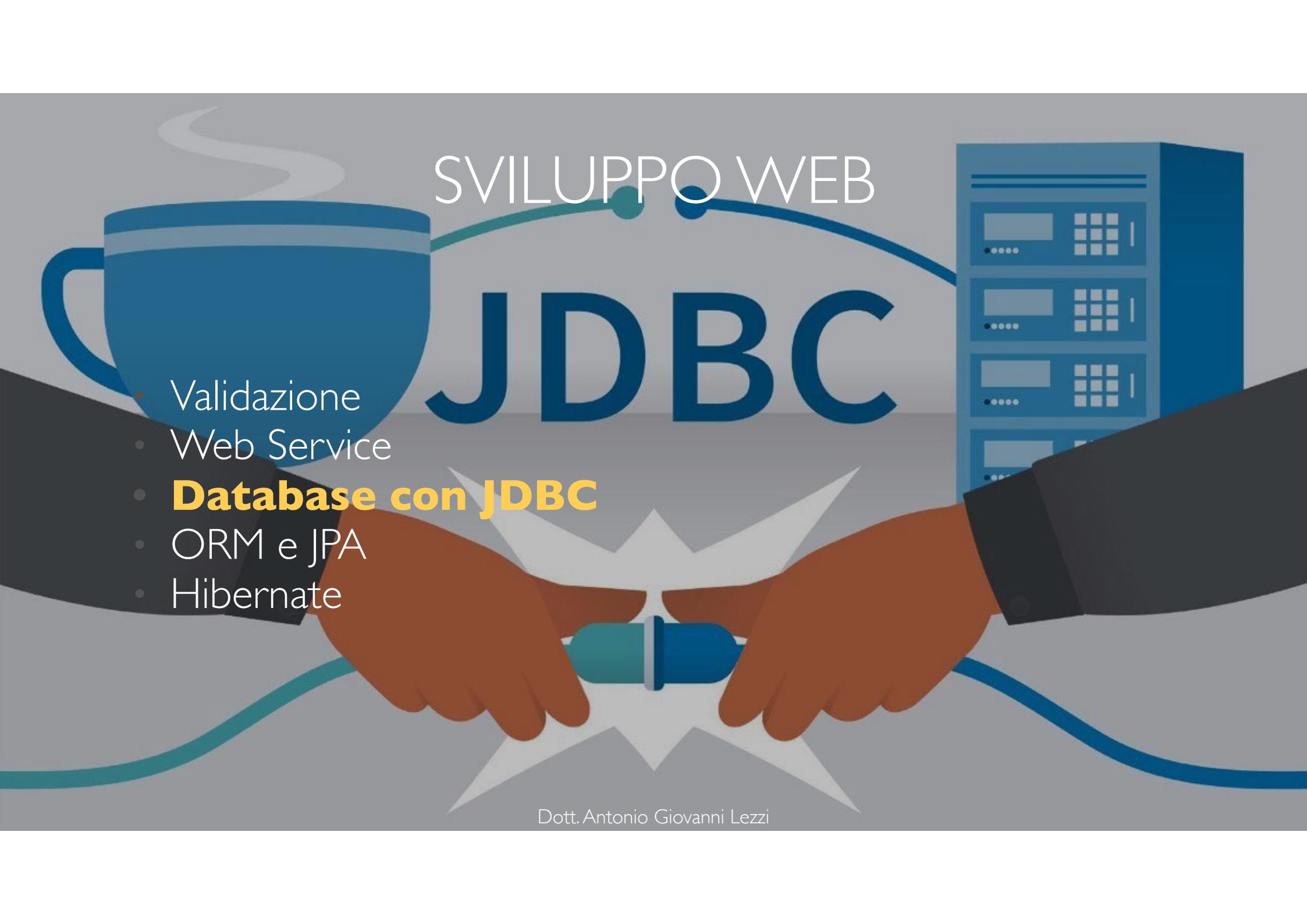
SPRING DATABASE: CREAZIONE RECORD

- La classe AccessingDataJpaApplication include un metodo main() che sottopone il CustomerRepository ad alcuni test. Innanzitutto, recupera CustomerRepository dal contesto dell'applicazione Spring. Quindi salva una manciata di oggetti Customer, dimostrando il metodo save() e impostando alcuni dati da usare.
- Successivamente, chiama findAll() per recuperare tutti gli oggetti Customer dal database. Quindi chiama findOne() per recuperare un singolo cliente dal suo ID. Infine, chiama findByLastName() per trovare tutti i clienti il cui cognome è "Bauer".
- Per impostazione predefinita, Spring Boot abilita il supporto del repository JPA e cerca nel pacchetto (e nei relativi pacchetti secondari) in cui si trova @SpringBootApplication.
- Se la configurazione ha definizioni dell'interfaccia del repository JPA situate in un pacchetto che non è visibile, puoi indicare pacchetti alternativi usando @EnableJpaRepositories e il suo parametro basePackageClasses = MyRepository.class sicuro per i tipi.

The background of the slide features a futuristic server room. The floor has glowing blue and purple light trails forming a heart shape. The room is filled with tall server racks, each with multiple glowing blue and green lights. The ceiling is dark with some glowing blue and purple elements. The overall atmosphere is high-tech and futuristic.

MYSQL FOR DEVELOPER

Dott. Antonio Giovanni Lezzi



SVILUPPO WEB

JDBC

- Validazione
- Web Service
- **Database con JDBC**
- ORM e JPA
- Hibernate

Dott. Antonio Giovanni Lezzi

SPRING DATABASE JDBC

- La semplice Business Logic di accesso ai dati con cui si vuole lavorare, richiede di gestire il nome e il cognome dei clienti. Per rappresentare questi dati a livello di applicazione, si deve creare una classe Customer.

```
src/main/java/hello/Customer.java
package hello;
public class Customer {
    private long id;
    private String firstName, lastName;
    public Customer(long id, String firstName, String lastName) {
        this.id = id;
        this.firstName = firstName;
        this.lastName = lastName;
    }
    @Override
    public String toString() {
        return String.format("Customer[id=%d, firstName='%s', lastName='%s']", id, firstName, lastName);
    }
    // getters & setters omitted for brevity
}
```

SPRING DATABASE JDBC

- Spring fornisce una classe modello chiamata JdbcTemplate che semplifica il lavoro con database relazionali SQL e JDBC.
- La maggior parte del codice JDBC riguarda l'acquisizione delle risorse, nella gestione delle connessioni, nella gestione delle eccezioni e nel controllo generale degli errori che non è completamente correlato a ciò che il codice intende raggiungere come scopo.
- JdbcTemplate si occupa di svolgere tutte queste mansioni, quindi quello che si deve sviluppare è concentrarsi sul compito da svolgere.

SPRING DATABASE JDBC

```
src/main/java/hello/Application.java
package hello;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.jdbc.core.JdbcTemplate;

import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

@SpringBootApplication
public class Application implements CommandLineRunner {
    private static final Logger log = LoggerFactory.getLogger(Application.class);
    public static void main(String args[]) {
        SpringApplication.run(Application.class, args);
    }
}

@Autowired
JdbcTemplate jdbcTemplate;
```

SPRING DATABASE JDBC

```
@Override
public void run(String... strings) throws Exception {
    log.info("Creating tables");
    jdbcTemplate.execute("DROP TABLE customers IF EXISTS");
    jdbcTemplate.execute("CREATE TABLE customers(" +
        "id SERIAL, first_name VARCHAR(255), last_name VARCHAR(255))");
    // Split up the array of whole names into an array of first/last names
    List<Object[]> splitUpNames = Arrays.asList("John Woo", "Jeff Dean", "Josh Bloch", "Josh Long").stream()
        .map(name -> name.split(" "))
        .collect(Collectors.toList());
    // Use a Java 8 stream to print out each tuple of the list
    splitUpNames.forEach(name -> log.info(String.format("Inserting customer record for %s %s", name[0], name[1])));
    // Uses JdbcTemplate's batchUpdate operation to bulk load data
    jdbcTemplate.batchUpdate("INSERT INTO customers(first_name, last_name) VALUES (?,?)", splitUpNames);
    log.info("Querying for customer records where first_name = 'Josh':");
    jdbcTemplate.query(
        "SELECT id, first_name, last_name FROM customers WHERE first_name = ?",
        new Object[] { "Josh" },
        (rs, rowNum) -> new Customer(rs.getLong("id"), rs.getString("first_name"), rs.getString("last_name"))
    ).forEach(customer -> log.info(customer.toString()));
}
```