Introduzione a Spring

Riferimento	Spring
Versione	2.0.0
Data	12/12/24
Destinatario	TMs
Presentato da	Raffaella Spagnuolo, Alessia Ture
Approvato da	Raffaella Spagnuolo, Alessia Ture

Revision History

Data	Versione	Descrizione	Autori
14/11/24	1.0.0	Prima Stesura	ATu
12/12/24	2.0.0	Revisione	RS

Project Managers

Nome	Acronimo	Contatto
Raffaella Spagnuolo	RS	r.spagnuolo6@studenti.unisa.it
Alessia Ture	ATu	a.ture@studenti.unisa.it

Sommario

1 Introduzione

1.1 Scopo del documento

Questo documento è stato creato per fornire una guida dettagliata ai concetti di Spring necessari per lo sviluppo del backend dell'applicazione *ZeroWaste Home*. Il documento è rivolto ai membri del team (TMs) e ha lo scopo di introdurre le funzionalità e le tecnologie chiave di Spring, aiutando il team a comprendere i componenti principali e le migliori pratiche per creare un backend solido e modulare. Includendo sia i concetti base che le configurazioni avanzate, il documento facilita la costruzione e la gestione di un backend che supporti le esigenze specifiche del progetto.

1.2 Obiettivi

L'obiettivo principale di questo documento è permettere al team di acquisire una comprensione chiara e pratica dei principali strumenti di Spring per la gestione del backend. La guida coprirà le basi di Spring, come la configurazione e l'uso delle annotazioni, fino all'implementazione della persistenza dei dati con Spring Data JPA e alla gestione delle richieste HTTP con Spring MVC.
In particolare, il documento mira a:

- Fornire una comprensione di come utilizzare Spring per la creazione e la gestione dei componenti backend.
- Facilitare l'utilizzo di Spring Boot per avviare e configurare l'applicazione in modo rapido ed efficiente.
- Introdurre le pratiche di gestione delle entità e dell'accesso ai dati tramite Spring Data JPA.
- Mostrare come gestire le richieste HTTP e le risposte JSON utilizzando Spring MVC.
- Promuovere la validazione e la sicurezza dei dati attraverso l'uso di annotazioni per la gestione delle dipendenze e la validazione degli input.

Con questa guida, il team potrà strutturare il backend dell'applicazione *ZeroWaste Home* in modo efficiente e seguendo le best practices di Spring, ottenendo un'architettura modulare e facilmente manutenibile.

2 Introduzione a Spring

2.1 Cos'è Spring?

Spring è un framework open-source per lo sviluppo di applicazioni Java, ampiamente utilizzato per costruire sistemi aziendali scalabili e modulari. Consente di gestire in modo efficiente componenti applicativi attraverso principi come l'Inversion of Control (IoC) e la Dependency Injection (DI). Questi principi aiutano a creare un'applicazione in cui i componenti sono facilmente configurabili e disaccoppiati, migliorando la manutenibilità e la testabilità del codice.

Nel contesto del progetto *ZeroWaste Home*, Spring fornirà la struttura portante del backend, gestendo sia le funzionalità core che le richieste client, così come l'accesso ai dati. Utilizzando Spring, il team potrà sviluppare un sistema backend robusto, che facilita la gestione dei dati, l'implementazione delle logiche di business e l'integrazione con il front-end.

2.2 Perché utilizzare Spring?

Spring è scelto per il progetto *ZeroWaste Home* per diversi motivi:

- **Modularità e Scalabilità**: consente di strutturare l'applicazione in moduli, semplificando l'implementazione e la crescita del sistema;
- **Gestione delle Dipendenze**: tramite Dependency Injection, rende il codice più flessibile e facilmente testabile;
- **Semplicità di Configurazione**: con Spring Boot, il setup di base dell'applicazione è semplificato, permettendo di concentrarsi sulla logica applicativa;
- **Supporto per il Pattern MVC**: facilita la gestione delle richieste HTTP e delle risposte verso il front-end:
- **Persistenza dei Dati**: con Spring Data JPA, semplifica l'accesso e la gestione dei dati, fondamentale per le funzionalità di *ZeroWaste Home* che richiedono l'archiviazione di prodotti, utenti, e logiche di scadenza alimentare.

3 Concetti Fondamentali in Spring

3.1 Inversion of Control (IoC)

L'Inversion of Control, o IoC, è un concetto che aiuta a rendere il codice più semplice e organizzato. In un'applicazione, spesso ci sono tanti "pezzi" che devono lavorare insieme, come moduli e componenti. Senza IoC, ogni pezzo deve "sapere" come trovare e collegarsi con gli altri pezzi necessari, il che può rendere il codice complicato e difficile da modificare.

Con IoC, è Spring stesso a prendersi carico di questo lavoro. Spring si occupa di collegare i vari pezzi dell'applicazione al momento giusto, in base alle configurazioni che forniamo. È come se fosse un organizzatore che distribuisce i compiti e si assicura che ogni parte abbia tutto ciò di cui ha bisogno senza che le singole parti debbano occuparsene.

Per esempio, se abbiamo un componente che deve accedere ai dati degli utenti, con IoC non è il componente stesso a "cercare" quei dati. Spring si assicura che i dati siano pronti e disponibili per quel componente, senza che quest'ultimo debba preoccuparsene.

Nel backend di *ZeroWaste Home*, DI facilita la creazione e gestione dei servizi. Ad esempio, se un servizio per il controllo delle scadenze necessita di un repository per accedere ai dati dei prodotti, Spring può iniettare automaticamente il repository richiesto nel servizio. Questo approccio riduce la dipendenza dei componenti e rende il codice più testabile, poiché le dipendenze possono essere sostituite con versioni mock durante i test.

```
// Senza IoC, l'istanza del servizio è creata manualmente
ProductService productService = new ProductService();

// Con IoC e Spring, l'istanza è gestita da Spring e iniettata automaticamente
@Component
public class ProductService {
    // Logica del servizio
}
```

Figura 1: Un esempio di IoC

3.2 Dependency Injection

La Dependency Injection (DI) è un metodo specifico con cui Spring applica il principio di Inversion of Control (IoC). Quando un componente dell'applicazione richiede un altro elemento, definito "dipendenza", Spring lo fornisce automaticamente senza necessità che il componente lo gestisca direttamente.

Per esempio, un servizio per l'invio di notifiche può richiedere un sistema di messaggistica per funzionare. Con DI, non è il servizio stesso a creare o configurare il sistema di messaggistica. Spring "inietta" questo sistema già pronto nel servizio. Questo approccio semplifica la gestione e il testing del codice, poiché le dipendenze (come il sistema di messaggistica) possono essere modificate senza alterare il servizio che le utilizza. In sostanza, DI permette a Spring di "fornire" a ogni componente le risorse necessarie, rendendo l'intera applicazione più efficiente e flessibile.

```
@Service
public class NotificationService {
    private final MessageService messageService;

    @Autowired
    public NotificationService(MessageService messageService) {
        this.messageService = messageService;
    }
    // Logica del servizio
}
```

Figura 2: La Dependency Injection (DI) è un modo per fornire automaticamente i componenti richiesti. Utilizzando @Autowired, Spring inietta automaticamente le dipendenze.

4 Componenti Principali di Spring

4.1 Configurazione con Annotazioni

Spring consente di configurare molti aspetti dell'applicazione tramite annotazioni, rendendo il codice più leggibile e riducendo la necessità di file di configurazione XML. Le annotazioni sono semplici "etichette" che si aggiungono direttamente nel codice per indicare a Spring come gestire i vari componenti. Usare le annotazioni facilita la gestione dei componenti e delle loro dipendenze in modo chiaro e conciso.

Le annotazioni principali utilizzate in Spring per configurare i componenti includono:

- **@Component**: Indica che una classe è un componente generico e che Spring dovrebbe gestirla automaticamente come un oggetto disponibile per l'applicazione;
- **@Service**: Contrassegna una classe come servizio, generalmente usato per la logica di business;
- **@Repository**: Usato per le classi che gestiscono l'accesso ai dati. Questa annotazione semplifica l'integrazione con Spring Data JPA;
- **@Autowired**: Indica a Spring di iniettare automaticamente la dipendenza specificata. Viene utilizzato sui campi o sui costruttori per definire le dipendenze.

```
@Repository
public interface ProductRepository extends JpaRepository<Product, Long> {
    // Interfaccia per la gestione dei dati di Product
}

@Service
public class ProductService {
    @Autowired
    private ProductRepository productRepository;

    // Metodi per la logica di business
}
```

Figura 3: Spring consente di configurare i componenti con annotazioni come @Component, @Service, @Repository, e @Autowired.

4.1.1 Creazione di Modelli di Dati Annotazioni Principali

Per gestire i dati in Spring, è comune creare modelli di dati che rappresentano le entità dell'applicazione. Questi modelli vengono annotati con specifiche etichette per indicare a Spring come trattare i dati.

Le annotazioni principali per la creazione dei modelli di dati sono:

- **@Entity**: Specifica che una classe è un'entità del database. Spring Data JPA la utilizza per creare e gestire automaticamente le tabelle nel database;
- **@Id**: Identifica il campo che rappresenta la chiave primaria dell'entità;
- **@GeneratedValue**: Usata insieme a @Id per specificare come la chiave primaria deve essere generata, solitamente in modo automatico;
- **@Table** (opzionale): Permette di personalizzare il nome della tabella o altre caratteristiche specifiche del database.

Queste annotazioni rendono semplice la gestione dei dati nell'applicazione *ZeroWaste Home*, permettendo di rappresentare gli elementi chiave (come prodotti e utenti) come entità facilmente integrabili nel database.

```java
@Entity
@Table(name = "products")
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "name", nullable = false)
    private String name;

    @Column(name = "expiry_date")
    private LocalDate expiryDate;

    // Getters e Setters
}
```

Figura 4: Esempio di Entity per un prodotto

5 Componenti Chiave di Spring per il Backend

5.1 Spring Boot per configurazione e avvio rapido

Spring Boot è un'estensione di Spring che semplifica la configurazione e l'avvio dell'applicazione. Fornisce un set di configurazioni predefinite e automatizza gran parte delle impostazioni necessarie per avviare rapidamente un'applicazione backend. Grazie a Spring Boot, è possibile creare un backend funzionale con poche configurazioni, e include strumenti integrati come server web (ad esempio, Tomcat), che rendono possibile avviare l'applicazione senza configurare server esterni. Spring Boot è ideale per il progetto *ZeroWaste Home*, poiché consente di ridurre i tempi di configurazione e accelerare il processo di sviluppo.

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

Figura 5: Esempio di classe principale con Spring Boot

5.2 Spring MVC per il controllo delle richieste e risposte http

Spring MVC (Model-View-Controller) è un componente che facilita la gestione delle richieste e risposte HTTP, rendendolo essenziale per le applicazioni web. Utilizzando controller definiti tramite annotazioni (come @RestController), Spring MVC consente di gestire in modo organizzato le richieste provenienti dal front-end. Ogni richiesta viene associata a un determinato metodo del controller, il quale elabora la richiesta e restituisce una risposta. Questo rende Spring MVC uno strumento efficace per il backend di *ZeroWaste Home*, poiché permette di controllare il flusso di dati tra il front-end e il back-end.

```java
@RestController
@RequestMapping("/api/products")
public class ProductController {
    @Autowired
    private ProductService productService;

    @GetMapping("/{id}")
    public ResponseEntity<Product> getProductById(@PathVariable Long id) {
        Product product = productService.getProductById(id);
        return ResponseEntity.ok(product);
    }

    @PostMapping
    public ResponseEntity<Product> addProduct(@RequestBody Product product) {
        Product savedProduct = productService.saveProduct(product);
        return ResponseEntity.status(HttpStatus.CREATED).body(savedProduct);
    }
}
```

Figura 6: Esempio di Controller REST

5.3 Spring Data JPA per la gestione della persistenza dei dati

Spring Data JPA è un modulo di Spring progettato per semplificare l'accesso ai dati e la gestione della persistenza. Permette di lavorare con database relazionali utilizzando Java Persistence API (JPA), riducendo la necessità di scrivere codice SQL manuale. Con Spring Data JPA, è possibile creare repository di dati che rappresentano le entità dell'applicazione, come utenti, prodotti e scadenze alimentari, e che consentono di salvare, aggiornare e recuperare dati in modo semplice. Nel contesto di *ZeroWaste Home*, Spring Data JPA permette di gestire la persistenza dei dati in modo efficiente, facilitando l'implementazione delle logiche di business legate ai prodotti e alla gestione delle scadenze.

6 Persistenza dei Dati con Spring Data JPA

6.1 Creazione e Gestione di Entity

In Spring Data JPA, le "entity" sono classi che rappresentano le tabelle di un database. Ogni entità mappa una tabella e ogni attributo della classe corrisponde a una colonna della tabella. Per creare un'entità, si definisce una classe Java e si utilizza l'annotazione @Entity per indicare a Spring che si tratta di una classe che rappresenta una tabella. Altri elementi importanti includono:

- **@Id**: Definisce la chiave primaria della tabella;
- **@GeneratedValue**: Specifica come viene generata la chiave primaria (ad esempio, in modo automatico);
- **@Column**: Utilizzata per personalizzare il nome o altre caratteristiche della colonna.

```java
@Entity
@Table(name = "users")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "username", nullable = false, unique = true)
    private String username;

    @Column(name = "email", nullable = false)
    private String email;

    // Getters e Setters
}
```

Figura 7:Esempio di Entity per un utente

6.2 Utilizzo di Repository per l'accesso ai dati

I repository in Spring Data JPA sono interfacce che forniscono metodi per interagire con il database, senza dover scrivere query SQL manuali. Spring gestisce automaticamente le operazioni comuni (come salvataggio, aggiornamento e cancellazione) utilizzando convenzioni di denominazione per i metodi, rendendo l'accesso ai dati rapido e intuitivo. Per creare un repository, si estende l'interfaccia JpaRepository, specificando il tipo di entità e il tipo della chiave primaria. Questo approccio semplifica l'accesso ai dati e rende il codice più leggibile.

```
public interface UserRepository extends JpaRepository<User, Long> {
    Optional<User> findByUsername(String username);
}
```

Figura 8: Esempio di Repository per l'entità User

7 Gestione delle Richieste http

7.1 Controller REST

I controller REST in Spring gestiscono le richieste HTTP dal front-end e inviano le risposte. Utilizzando l'annotazione @RestController, si definisce un controller REST che espone endpoint (URL) che possono essere chiamati dal front-end. Ogni metodo all'interno del controller è associato a una specifica operazione HTTP, come @GetMapping per le richieste GET e @PostMapping per le richieste POST. I controller REST facilitano la comunicazione tra il front-end e il back-end di **ZeroWaste Home**, gestendo le operazioni di recupero, creazione e modifica dei dati.

```
@RestController
@RequestMapping("/api/users")
public class UserController {
    @Autowired
    private UserService userService;

    @GetMapping("/{username}")
    public ResponseEntity<User> getUserByUsername(@PathVariable String username)
        User user = userService.findByUsername(username);
        return ResponseEntity.ok(user);
    }
}
```

Figura 9: Esempio di Controller per gli utenti

7.2 Gestione delle risposte JSON

Spring gestisce automaticamente la conversione delle risposte in formato JSON, un formato standard per la comunicazione tra client e server. Quando un metodo del controller restituisce un oggetto, Spring lo converte in JSON per poter essere letto e utilizzato dal front-end. Questo rende la

comunicazione tra le diverse parti del sistema efficiente e compatibile, poiché JSON è supportato da quasi tutte le tecnologie web.

7.3 Validazione degli Input

La validazione degli input è un passaggio importante per garantire che i dati inviati al server siano validi. In Spring, è possibile utilizzare l'annotazione @Valid per convalidare automaticamente i dati degli utenti (come i moduli di registrazione). Altre annotazioni, come @NotNull, @Size, e @Email, consentono di definire vincoli specifici per i campi dell'entità. Questo assicura che i dati siano validati prima di essere processati, migliorando la sicurezza e l'integrità del sistema di *ZeroWaste Home*.

```
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotNull(message = "Username is required")
    @Size(min = 3, max = 50)
    private String username;

    @NotNull(message = "Email is required")
    @Email(message = "Email should be valid")
    private String email;

    // Getters e Setters
}
```

Figura 10:Esempio di Validazione con @Valid