

# Spring Boot

---

## Passaggi Preliminari

### 1. Creazione del progetto:

- Premi **F1**.
- Seleziona **Initializer Maven Project**.
- Configura il progetto con i seguenti parametri:
  - **Versione:** 3.4.3
  - **Linguaggio:** Java
  - **Group:** edu.unict.wsos
  - **Artifact:** esame
  - **Tipo:** jar
  - **Java Version:** 17
- Aggiungi le seguenti dipendenze:
  - **Spring Web**
  - **Thymeleaf**
  - **MySQL Driver**
  - **Spring Data JPA**

### 2. Salvataggio:

- Salva e apri la cartella del progetto nella tua home.

---

## Configurazione del Progetto

### 1. Compilazione:

- Esegui `mvn spring-boot:run` nella cartella creata.
- Java: Clean Workspace Cache.

### 2. Configurazione delle proprietà di applicazione:

- Apri il file `src/main/resources/application.properties` e aggiungi la seguente configurazione:

```
spring.datasource.url=jdbc:mysql://localhost:3306/exam
spring.datasource.username=user
spring.datasource.password=password
spring.jpa.hibernate.ddl-auto=update
```

# Struttura del Progetto

All'interno di `src/main/java/edu/unict/dmi/wsos/aeroporti`, crea le seguenti cartelle:

- **Model** (step 1)
- **Data** (step 2)
- **Controller** (step 3)

## Ordine di implementazione:

### 1. Model

Crea una classe Java che rappresenta la tabella nel database.

La classe deve includere:

- `@Entity`
- I campi del database
- L'ID deve avere `@Id` e `@GeneratedValue(strategy = GenerationType.IDENTITY)`
- Crea getter e setter
- Costruttori con tutti i parametri e costruttore vuoto tramite l'azione "Azione origine di Visual Studio Code"

## Esempio:

```
@Entity
public class Compagnia {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    Long id;
    String descrizione;
    String link;

    public Compagnia() {
    }

    public Compagnia(Long id, String descrizione, String link) {
        this.id = id;
        this.descrizione = descrizione;
        this.link = link;
    }

    // Getter e Setter
    ...
}
```

## 2. Repository

Crea un'interfaccia repository che estende `JpaRepository<NomeModel, Long>`.

**Esempio:**

```
public interface CompagniaRepository extends JpaRepository<Compagnia, Long>
{
}
```

## 3. Controller

Crea una classe `Controller` con l'annotazione `@Controller`, un riferimento **private final** alla repository e un costruttore che lo inizializzi.

**Esempio:**

```
@Controller
public class CompagniaController {
    private final CompagniaRepository repo;

    public CompagniaController(CompagniaRepository repo) {
        this.repo = repo;
    }
}
```

## 4. Thymeleaf: Creazione dei template

All'interno della cartella `/src/main/resources/templates`, crea un file `compagnie/list.html` tramite il comando `html:5` e modifica:

```
<html lang="en"></html>
```

in:

```
<html lang="it" xmlns:th="https://www.thymeleaf.org"></html>
```

---

# Sviluppo delle funzionalità CRUD

## Read

Modifica il file `list.html` creando una tabella che contenga tutti i campi del model.

La riga relativa ai dati deve avere l'attributo `th:each="compagnia : ${compagnie}"`.

**Nota bene:** Il campo tra parentesi graffe deve essere uguale a quello che scriviamo nel controller nel metodo `addAttribute()`. Ogni dato della tabella sarà dato dalla notazione `esame.attributo`, facendo attenzione ad usare la stessa notazione usata nel model.

## Esempio:

```
<!DOCTYPE html>
<html lang="it" xmlns="https://www.thymeleaf.org">
  <body>
    <h1>
      <center>Compagnie Aeree</center>
    </h1>

    <table border="1px">
      <tr>
        <th>Id</th>
        <th>Descrizione</th>
        <th>Link</th>
        <th>Modifica</th>
        <th>Elimina</th>
      </tr>
      <tr th:each="compagnia : ${compagnie}">
        <td th:text="${compagnia.id}"></td>
        <td th:text="${compagnia.descrizione}"></td>
        <td th:text="${compagnia.link}"></td>
        <td>
          <a th:href="@{/compagnie/{id}/edit(id=${compagnia.id})}">Modifica </a>
        </td>
        <td>
          <a th:href="@{/compagnie/{id}/delete(id=${compagnia.id})}">
            >Elimina
          </a>
        </td>
      </tr>
    </table>

    <h3>Inserisci una nuova compagnia</h3>
    <a th:href="@{/compagnie/new}">Inserisci una nuova compagnia aerea</a>
  </body>
</html>
```

Modifica il `controller` aggiungendo un metodo `@GetMapping` che prende come parametro il `Model`.  
Verrà richiamato `model.addAttribute()` con i seguenti parametri:

- **Nome della variabile** che usiamo nel template Thymeleaf nella notazione `${}` presente nel `th:each`.
- `repo.findAll()` che permette di selezionare tutti i dati presenti nel database MySQL.

**Esempio:**

```
@GetMapping("/compagnie")
public String getCompagnie(Model model) {
    model.addAttribute("compagnie", repo.findAll());
    return "compagnie.list";
}
```

---

## Create

Creiamo il file `compagnie/edit.html` creando un form con metodo `POST`.  
Il form avrà tanti campi quanti sono i dati da inserire nella tabella del database.

**Nota bene:** L'attributo `name` di ogni tag `input` deve corrispondere esattamente con il nome delle variabili utilizzate nel model.

**Esempio:**

```
<body>
  <h1>
    <center>Modifica/crea Compagnia Aerea</center>
  </h1>

  <form th:action="@{/compagnie/new}" method="post" th:object="${compagnie}">
    <input type="hidden" name="id" th:field="${compagnie.id}" />
    <span>Inserisci descrizione</span>
    <input type="text" name="descrizione" th:field="${compagnie.descrizione}" />
    <span>Inserisci link</span>
    <input type="url" name="link" th:field="${compagnie.link}" />
    <button>Invia</button>
  </form>
</body>
```

Aggiungi al `controller` un metodo `@PostMapping` ed un metodo `@GetMapping`.  
Il primo prende come parametro la classe creata allo step 1. Il secondo viene richiamato dall'utente e crea un nuovo oggetto vuoto da inizializzare.  
Verrà richiamato `repo.save()` per salvare l'oggetto nel database.  
Successivamente, reindirizziamo l'utente alla home.

### Esempio:

```
@GetMapping("/compagnie/new")
public String createCompagnia(Model model) {
    model.addAttribute("compagnie", new Compagnia());
    return "compagnia/edit";
}

@PostMapping("/compagnie/new")
public String addCompagnia(Compagnia compagnia) {
    repo.save(compagnia);
    return "redirect:/compagnie";
}
```

---

## Delete

Aggiungi al **controller** un metodo **@GetMapping** che prende come parametro l'id dell'oggetto da eliminare (passato come **hidden** nel form della tabella).

Verrà richiamato **repo.deleteById()** per eliminare l'oggetto dal database.

Successivamente, reindirizziamo l'utente alla home.

Nota Bene: L'utilizzo di **@PathVariable** è fondamentale

### Esempio:

```
@GetMapping("/compagnie/{id}/delete")
public String deleteCompagnia(@PathVariable Long id) {
    repo.deleteById(id);
    return "redirect:/compagnie";
}
```

---

## Update

Modifica il **controller** aggiungendo un metodo **@GetMapping** che prende come parametri:

- Il **Model**
- L'**ID** dell'oggetto da modificare (passato tramite url).

Il compito del metodo sarà:

Richiamare **model.addAttribute()** passando la variabile per Thymeleaf (assicurati che il nome sia coerente tra controller e template) ed utilizzando **repo.getReferenceById()** per ottenere i dati dal database e precompilare il form.

**Esempio:**

```
@GetMapping("/compagnie/{id}/edit")
public String editCompagnia(@PathVariable Long id, Model model) {
    model.addAttribute("compagnie", repo.getReferenceById(id));
    return "compagnia/edit";
}
```