

Come usare VSCode per Spring e non solo!

PERCHÉ NON
SUDDIVIDERE
L'EDITOR IN
BASE ALL'
AMBIENTE
CHE
VOGLIAMO
USARE?

Possiamo usare due parametri di avvio:
(Compatibili con molte altre app Electron)

`--extensions-dir "<DIR>"`

`--user-data-dir "<DIR>"`

Es: creiamo un collegamento a:

`"C:\Program Files\Microsoft VS Code\Code.exe"`

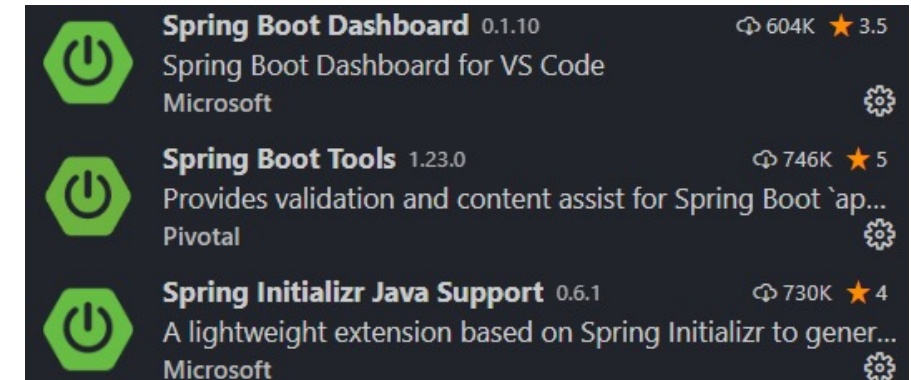
`--extensions-dir "C:\VSCodeEnv\Spring\ext"`

`--user-data-dir "C:\VSCodeEnv\Spring\usr"`

(Se code non si trova in programmi, controllare
`"%localappdata%\Programs\Microsoft VS Code"`)

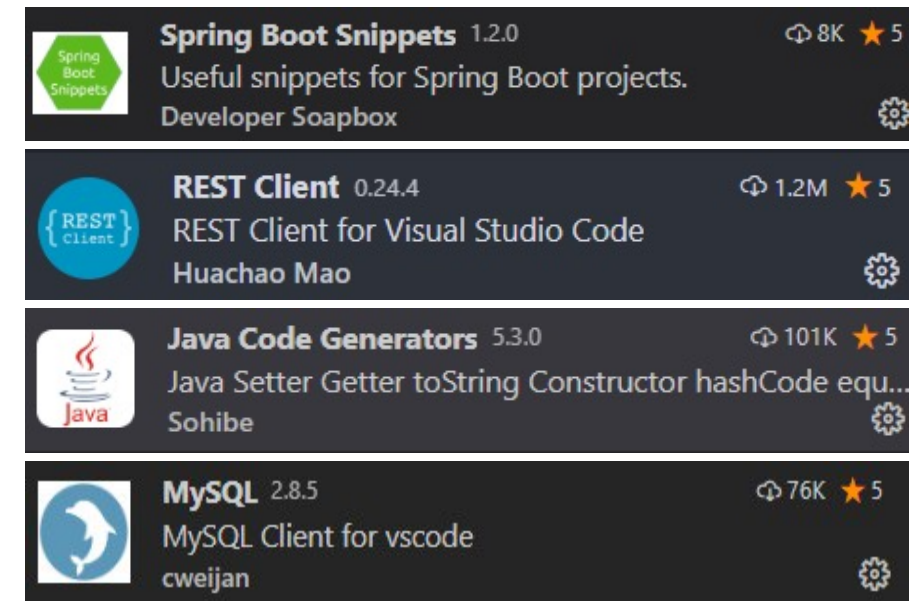
Se `code.exe` è nel **PATH**, possiamo omettere il
percorso intero e usare "`code`")

Estensioni consigliate per Spring Boot (A parte quelle di default per Java)



Altre estensioni utili, ma non fondamentali

- **Spring Boot Snippets**
- **REST Client** per non usare Postman
- **Java Code Generators** per generare (tramite tasto destro, Java Generator:GUI) getters, setters e constructors
- **MySQL** per gestire le connessioni a DB MySQL e MariaDB



Inizializzare un progetto

Dopo aver installato le estensioni consigliate, aprire la command palette (View, Command Palette.."), digitare:

INITIALIZR

E scegliere Create a Maven Project. Seguire tutti i passi necessari (Aggiungere le dipendenze consigliate dal prof) e scegliere la cartella di destinazione del nuovo progetto. Cliccare su open per aprire subito il progetto.

Se si dimentica qualche starter o si vuole aggiungere successivamente, si può digitare nuovamente "initializr" e scegliere "Spring Initializr: Add Starters"

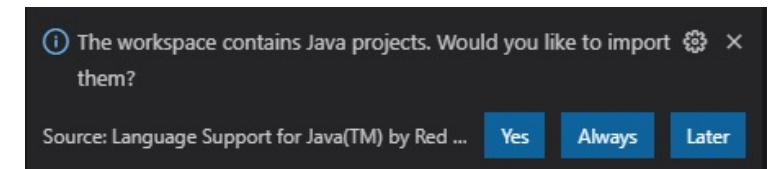
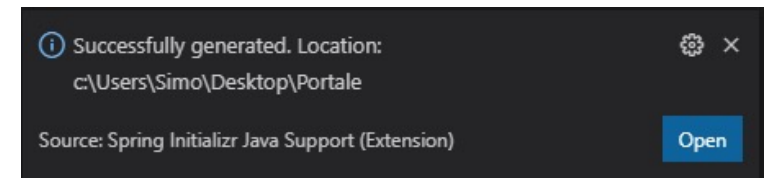
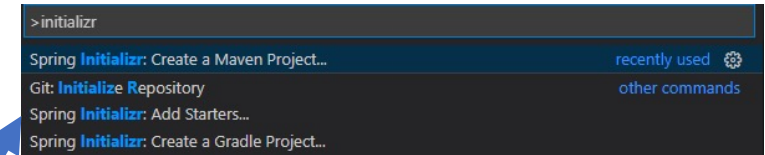
Se dovesse spuntare questo popup dopo aver aperto il progetto premere SI.

Altri due popup potrebbero comparire.

Uno vi chiederà se volete usare Maven. Accettate.

L'altro potrebbe comparire se si apre per la prima volta un file .java. L'editor vi sta consigliando delle estensioni utili. Premere INSTALLA.

Per altri dubbi: <https://code.visualstudio.com/docs/java/java-spring-boot>

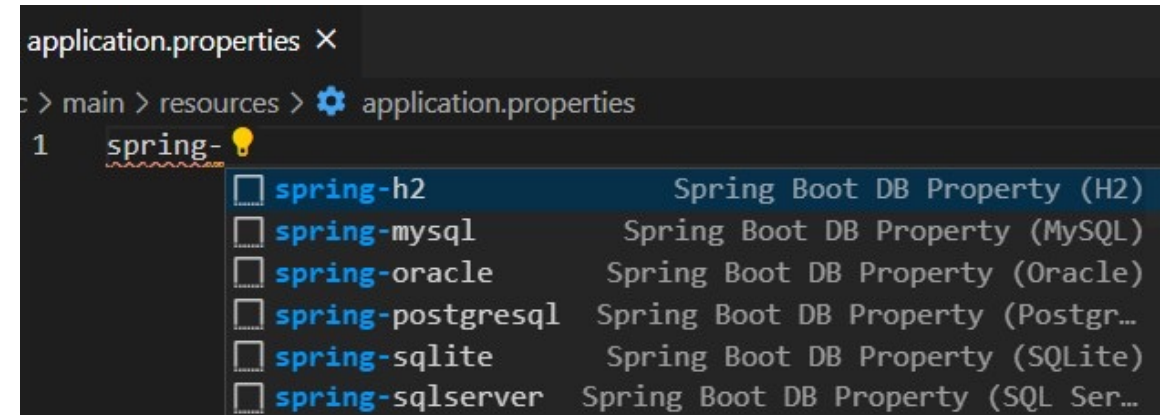


Snippets 1/3

"Spring boot snippets" aiuta nella creazione del file application.properties.

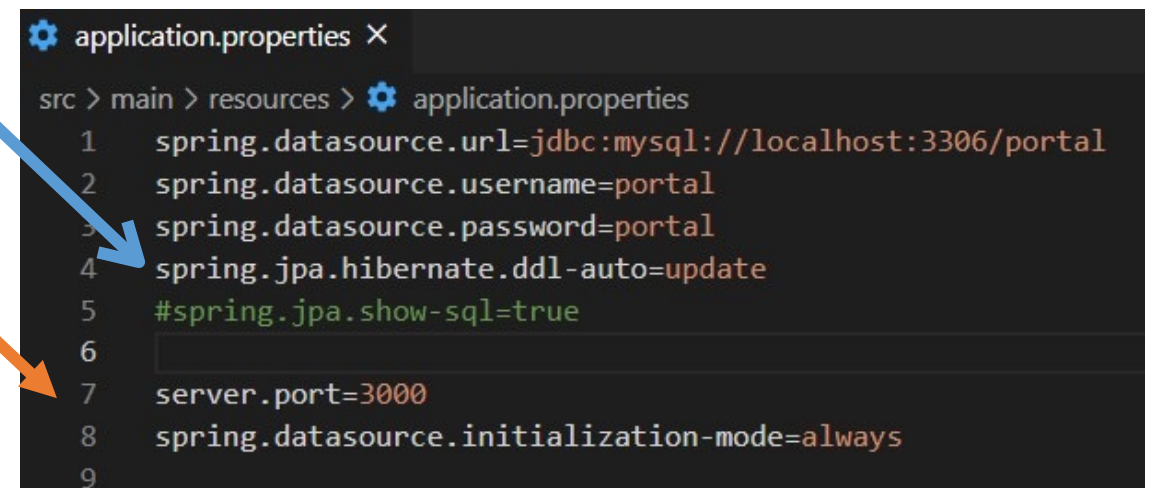
Scrivendo "spring-" compariranno alcuni suggerimenti.

Visto che il tipo di DB che abbiamo scelto di usare è MySQL, usiamo spring-mysql.



```
application.properties X
src > main > resources > application.properties
1 spring-
   spring-h2      Spring Boot DB Property (H2)
   spring-mysql   Spring Boot DB Property (MySQL)
   spring-oracle  Spring Boot DB Property (Oracle)
   spring-postgresql Spring Boot DB Property (Postgr...
   spring-sqlite  Spring Boot DB Property (SQLite)
   spring-sqlserver Spring Boot DB Property (SQL Ser...
```

Il codice pre-generato non è però sufficiente. E' necessario decommentare (Rimuovere il #) dalla riga 4, e aggiungere le due righe (7 e 8) manualmente. A riga 7 si può personalizzare la porta di ascolto dell'applicazione (E conseguentemente, del REST server)



```
application.properties X
src > main > resources > application.properties
1 spring.datasource.url=jdbc:mysql://localhost:3306/portal
2 spring.datasource.username=portal
3 spring.datasource.password=portal
4 spring.jpa.hibernate.ddl-auto=update
5 #spring.jpa.show-sql=true
6
7 server.port=3000
8 spring.datasource.initialization-mode=always
9
```

Snippets 2/3

"Spring boot snippets" offre un enorme aiuto fornendo alcuni "snippets", interi pezzi di codice generati automaticamente scrivendo delle parole chiave.

Creiamo un file .java e rendiamolo vuoto.

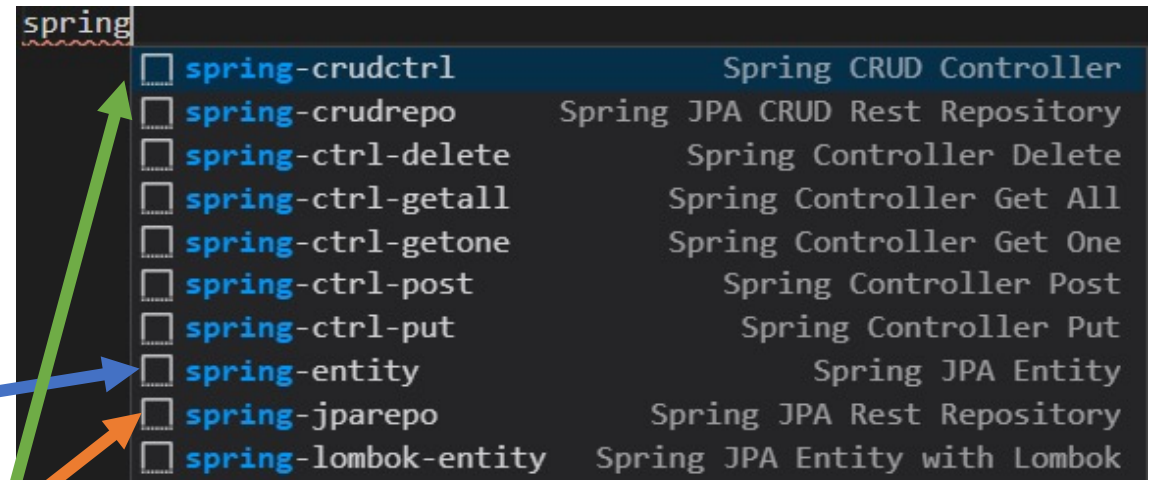
Scriviamo "spring-".

Spring-entity genera una classe mock-up per un'entità.

Consiglio l'uso di spring-jparepo invece che di spring-crudrepo. Jparepo è un'interfaccia che estende crudrepo aggiungendo il supporto al Sorting e all'uso di Example. (Vedi wiki)

Spring-crudctrl crea una classe controller con alcuni metodi all'interno. Gli altri spring-ctrl sono mock-up di singoli metodi

Una volta scelto lo snippet usare il tasto **TAB** per compilare i campi dello snippet (Quelli illuminati).
Il campo dopo "package" è buggato, lasciare il valore di default e modificarlo dopo.



spring	
<input type="checkbox"/> spring-crudctrl	Spring CRUD Controller
<input type="checkbox"/> spring-crudrepo	Spring JPA CRUD Rest Repository
<input type="checkbox"/> spring-ctrl-delete	Spring Controller Delete
<input type="checkbox"/> spring-ctrl-getall	Spring Controller Get All
<input type="checkbox"/> spring-ctrl-getone	Spring Controller Get One
<input type="checkbox"/> spring-ctrl-post	Spring Controller Post
<input type="checkbox"/> spring-ctrl-put	Spring Controller Put
<input type="checkbox"/> spring-entity	Spring JPA Entity
<input type="checkbox"/> spring-jparepo	Spring JPA Rest Repository
<input type="checkbox"/> spring-lombok-entity	Spring JPA Entity with Lombok

Snippets 3/3

"Spring boot snippets" purtroppo non è provvisto di "spring-service" ed è orientato solo all'uso di repository e controller.

Come risolvere? Possiamo:

1. Creare uno snippet spring-service e modificare manualmente il controller dopo averlo generato
2. **Non** usare l'estensione e usare snippet fatti da noi.

Qui trovate gli snippets necessari per spring:

<https://pastebin.com/svyGXXhN> (**java**.json solo spring-service) (Opzione 1)

Oppure (Opzione 2)

<https://pastebin.com/cBNRAtyc> (**java**.json completo)

<https://pastebin.com/ugYMwsan> (**spring-boot-properties**.json)

Dove aggiungerli? (Su windows)

Se avete seguito la slide 2: `<DIR>\User\snippets`

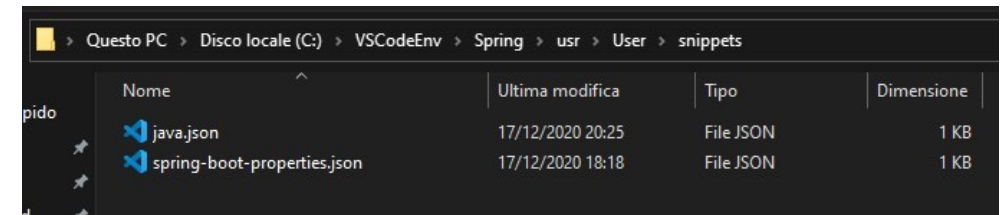
Se **NON** avete seguito la slide 2: `%APPDATA%\Code\User\snippets`

In alternativa, è possibile aggiungerli manualmente, andando, da VS Code, su:

View -> Command Palette -> "preferences: Configure User Snippets" ->

"**java**/**spring-boot-properties**" (A seconda del file) e incollando il codice corretto

Tutto ciò che è stato detto



Per info su come creare/è stato creato lo snippet:

<https://code.visualstudio.com/docs/editor/userdefinedsnippets>



Idee per MySQL

-
1. Installare solo MySQL/MariaDB (Rinunciando a PhpMyAdmin)
 2. Installare uno stack LAMP manualmente (Su linux o su WSL)
 3. Installare uno xAMP one-click

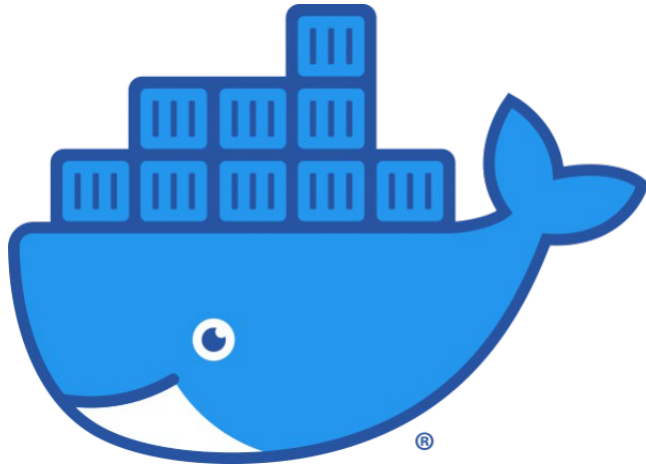
(Attenzione: l'account usato per loggare automaticamente in PhpMyAdmin, potrebbe non è root, ma **pma**. Inoltre tra gli utenti già definiti potrebbe essere presente l'account senza nome che potrebbe avere priorità rispetto ad altri utenti. **Si consiglia di creare un utente con tutti i privilegi ed eliminare quello senza nomi**).

Esempi:

- EasyPHP
- XAMPP

Oppure...

Docker!



```
You can now connect to this MySQL Server with BCyiSRcU8uY2
mysql -uadmin -pBCyiSRcU8uY2 -h<host> -P<port>
Please remember to change the above password as soon as possible!
```

Su windows è fortemente consigliato/necessario usare WSL2.

Una volta installato docker (O docker desktop su windows/Mac),

Digitare da linea di comando:

- docker volume create **mysqlData**
- docker run --name "TSDW_DB" -d -p "**3306:3306**" -p "**3380:80**" -v **mysqlData**:\var\lib\mysql matrayner/lamp:latest

Legenda: in **blu** il nome del volume, in **verde** le porte host, in **arancio** le porte del container

Abbiamo installato: localhost:**3380**/phpmyadmin

N.B. "*root*" non ha password ma **non è utilizzabile da PhpMyAdmin** per motivi di sicurezza.

In fase di run, in console (Se si usa **-d** tra i parametri, si vedrà solo dall'app desktop) nei log spunterà un account "*admin*" e una password casuale. admin è utilizzabile anche da remoto.

Usare l'estensione 'MySQL' su VSCode

Si può scegliere di creare a priori l'utente che userà il progetto, ad esempio usando PhpMyAdmin, oppure da linea di comando con:

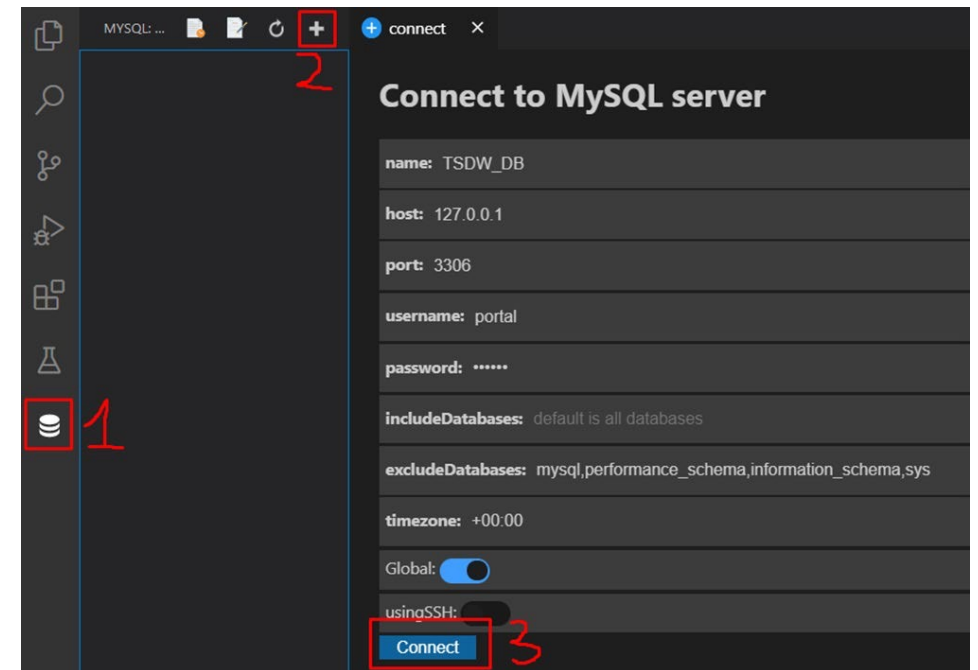
```
CREATE USER 'portal'@'%' IDENTIFIED BY 'portal';  
CREATE DATABASE IF NOT EXISTS 'portal';  
GRANT ALL PRIVILEGES ON 'portal'.* TO 'portal'@'%';
```

A questo punto, possiamo usare questo nuovo account.

In alternativa, si può scegliere di usare qualsiasi altro account, come l'account admin o root*, esponendo però il DB a "rischi".

Seguire l'immagine per definire una nuova connessione.

**(In alcune installazioni l'account di root è utilizzabile solo tramite password e/o localhost)*



Risoluzione step by step su VSCode

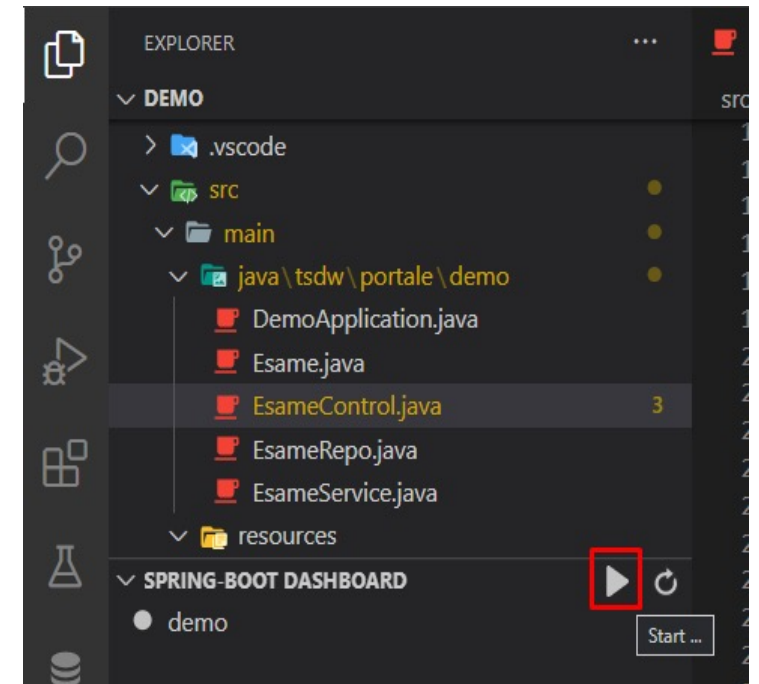
Da qui in poi vedremo **come realizzare un progetto completo** su Visual Studio Code 2021.

- 1) Creare un progetto vuoto seguendo gli step sopra.
- 2) Inizializzare le **entità** utilizzando lo snippet **«spring-entity»**.
Modificare «*entityClassName*» con il nome dell'entità desiderata ed «*entityIdType*» con il tipo dell'ID (Long).
- 3) Estendere la classe in base agli **attributi necessari**, aggiungendo dipendenze e relazioni.
- 4) Se ci troviamo nella relazione **uno-a-molti**:

```
@OneToMany(mappedBy="entity", cascade=CascadeType.REMOVE , fetch= FetchType.LAZY)
private Collection<foreign_entity> foreign = new HashSet<>();
```

Se, invece, ci troviamo nella relazione **molti-a-uno**:

```
@ManyToOne
@OnDelete(action = OnDeleteAction.CASCADE)
private Foreign foreign;
```



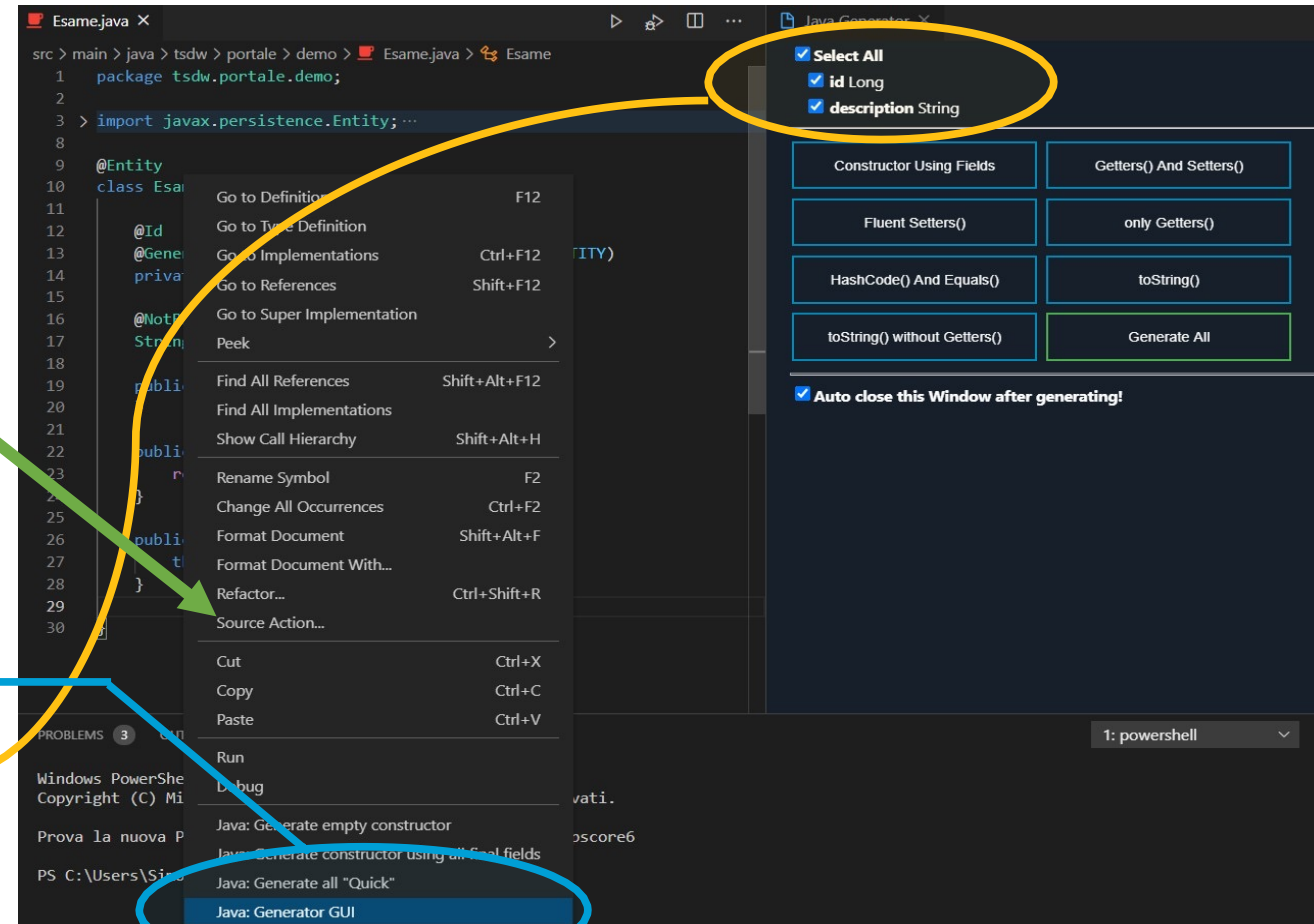
Il tag “@OnDelete” è necessario a far sì che, a seguito dell'**eliminazione della classe dipendente**, vengano eliminate a cascata anche le rispettive dipendenze, mantenendo integrità nel DB.

Usare l'estensione 'Java Code Generators' su VSCode

VSCode offre già un supporto alla generazione di codice. Cliccando su Source Action è possibile generare automaticamente costruttori, getters e setters.

Se invece preferiamo un'interfaccia grafica, possiamo installare e usare Java Code Generators.

Basterà cliccare sulla riga in cui vogliono inserire i metodi, tasto destro, Java Generator GUI, spuntare gli attributi di cui si vogliono generare i metodi, cliccare il tasto corrispondente ai metodi desiderati, ad esempio, Getters() and Setters()



Repositories e Services

Dopo aver correttamente realizzato le entità, è necessario **istanziare le repositories** mediante lo snippet **«spring-repo»**. Creato lo scheletro, è necessario modificare il **tipo dell'entità** e l'**id** a cui la repository fa riferimento.

```
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import tsdw.demo.Entities.Entity;

@Repository
public interface MedicoRepository extends JpaRepository<Entity, Long> {}
```

Istanziamo i services mediante lo snippet **«spring-service»** modificando il **tipo della classe** e il **tipo dell'id**.

```
@Service
public class MedicoService {

    @Autowired
    private MedicoRepository repository;

    public Medico addMedico(Medico e) {
        return repository.save(e);
    }

    public Optional<Medico> getMedico(Long id) {
        return repository.findById(id);
    }

    public List<Medico> getAllMedicos() {
        List<Medico> output = new ArrayList<Medico>();
        repository.findAll().forEach(output::add);
        return output;
    }

    public Medico updateMedico(Medico e) {
        return repository.save(e);
    }

    public void deleteMedico(Medico e) {
        repository.delete(e);
    }

    public void deleteMedico(Long id) {
        repository.deleteById(id);
    }
}
```


Controllers

Infine, per gestire opportunamente le richieste HTTP, è necessario **istanziare i controllers**.

Per fare ciò occorre:

- 1) Istanziare il file utilizzando lo snippet «*spring-ctrl*» modificando i parametri.
- 2) Modificare il «*resourceRootPath*», il nome dell'entità e il nome del servizio.

```
EntityController
@RequestMapping("/resourceRootPath")
public class EntityNameController {

    @Autowired
    private EntityNameService service;

    @GetMapping("/resourcePath")
    public ResponseEntity<List<EntityName>> getAllEntityNames() {
        List<EntityName> entityList = service.getAllEntityNames();
        return ResponseEntity.ok(entityList);
    }

    @GetMapping("/resourcePath/{id}")
    public ResponseEntity<EntityName> getEntityName(@PathVariable long id) {
        Optional<EntityName> entity = service.getEntityName(id);
        if(entity.isPresent())
            return ResponseEntity.ok(entity.get());
        else
            return ResponseEntity.notFound().build();
    }
}
```

3) Il controller dell'entità con relazione **uno-a-molti** non va più modificato. Invece, il controller relativo alla relazione **molti-a-uno** va modificato:

```
@PostMapping("/{foreign_id}/entity")
public ResponseEntity<Vaccino> addEntity(@PathVariable(name="foreign_id") Long foreign_id, @RequestBody Entity e) throws URISyntaxException {
    Optional<Foreign> foreign = foreignService.getForeign(foreign_id);
    if (e.getId() != null) {
        return ResponseEntity.badRequest().build();
    }
    e.setForeign(medico.get());
    Entity entity = service.addEntity(e);
    return ResponseEntity.created(new URI("/api/entity" + entity.getId())).body(entity);
}
```


Come eseguire e testare il WS creato

Una volta che tutti i componenti sono stati creati (Entity, Repository e Controller (Service a scelta)), possiamo avviare il WS dalla dashboard, cliccando l'icona "play".

Se tutto andrà bene, l'icona vicino il nome del progetto ("demo" nella figura) resterà verde.

Adesso possiamo usare postman per provare le interfacce API create. Se invece non vogliamo spostarci da VSCode, possiamo usare l'estensione REST Client.

Creiamo un file con estensione .http, e scriviamo qui la richiesta. Per richieste di tipo POST è necessario aggiungere:

- **content-type: application/json**
- **una riga vuota (riga 3)**

