

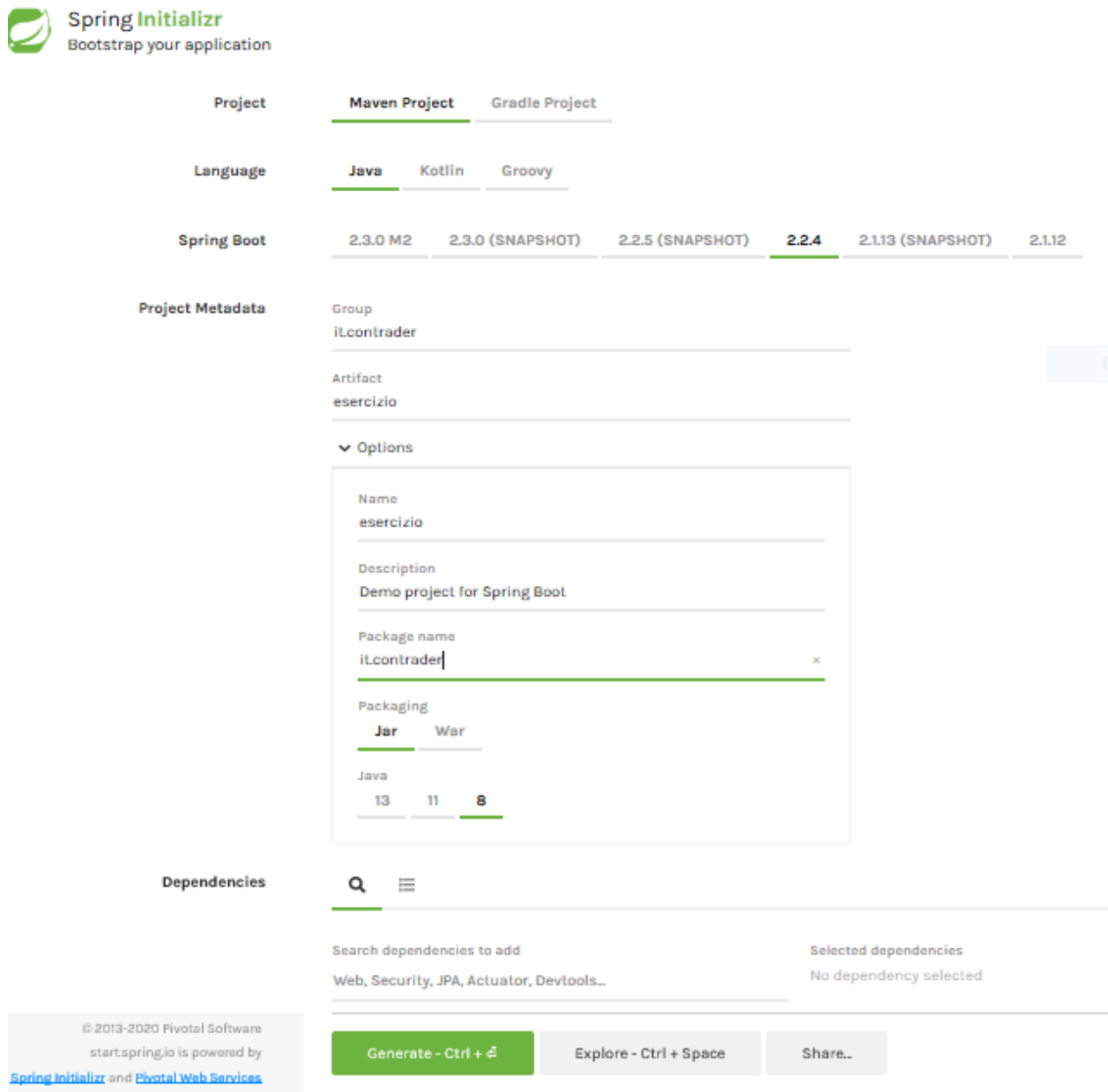


**CONTRADER**

# Guida

## Inizializzazione progetto su start.spring.io e testing con junit

1- Recarsi sul sito start.spring.io e creare un nuovo progetto



The screenshot shows the Spring Initializr web application interface. The header includes the Spring Initializr logo and the tagline "Bootstrap your application". The main content area is divided into several sections:

- Project:** Two tabs, "Maven Project" (selected) and "Gradle Project".
- Language:** Three tabs, "Java" (selected), "Kotlin", and "Groovy".
- Spring Boot:** A row of version links: "2.3.0 M2", "2.3.0 (SNAPSHOT)", "2.2.5 (SNAPSHOT)", "2.2.4" (selected), "2.1.13 (SNAPSHOT)", and "2.1.12".
- Project Metadata:**
  - Group:** A text input field containing "it.contrader".
  - Artifact:** A text input field containing "esercizio".
  - Options:** A dropdown menu showing a form with the following fields:
    - Name:** "esercizio"
    - Description:** "Demo project for Spring Boot"
    - Package name:** "it.contrader" (with a clear button 'x')
    - Packaging:** Two tabs, "Jar" (selected) and "War".
    - Java:** Three tabs, "13", "11", and "8" (selected).
- Dependencies:** A section with a search icon and a list of dependencies to add. The search bar contains "Web, Security, JPA, Actuator, Devtools...". Below it, there are two columns: "Search dependencies to add" and "Selected dependencies". The "Selected dependencies" column shows "No dependency selected".

At the bottom left, there is a footer with copyright information: "© 2013-2020 Pivotal Software", "start.spring.io is powered by", and links to "Spring Initializr" and "Pivotal Web Services". At the bottom right, there are three buttons: "Generate - Ctrl + G" (green), "Explore - Ctrl + Space", and "Share..".

1 Es. di creazione di un progetto

2- In Dependencies inserire le seguenti:

1-Spring boot devtools 2-Lombok 3-Spring configuration processor 4-Spring web 5-Rest repositories

6-JDBC api 7-Spring data JPA 8-Spring data JDBC (9-MySQL driver 10-Oracle driver, inserire solo quello utilizzato)

Aggiungere la dipendenza per il testing con junit, oppure passare al prossimo punto

Successivamente premere su Generate, verrà scaricato un file zip da cui estrarre il progetto e importarlo su eclipse

3- Aggiungere le entità con i relativi componenti e verificare il loro funzionamento

4- Seguire questo punto se non si ha aggiunto la dipendenza per il testing.

Nel file pom.xml trovare la dipendenza spring-boot-starter-test e fare in modo che la dipendenza sia scritta in questo modo:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
</dependency>
```

Salvare, poi tasto destro sul progetto e premere su Update Project in Maven

5- Su eclipse recarsi in src/test/java al cui interno è presente un file java:

```
1 package it.contrader;
2
3 import org.junit.jupiter.api.Test;
4 import org.junit.runner.RunWith;
5 import org.springframework.boot.test.context.SpringBootTest;
6 import org.springframework.test.context.junit4.SpringRunner;
7
8 @RunWith(SpringRunner.class)
9 @SpringBootTest
10 class EsercizioApplicationTests{
11
12     @Test
13     void contextLoads() {
14     }
15
16 }
17
```

2 Es. del file presente all'interno

Aggiungere @RunWith come in figura.

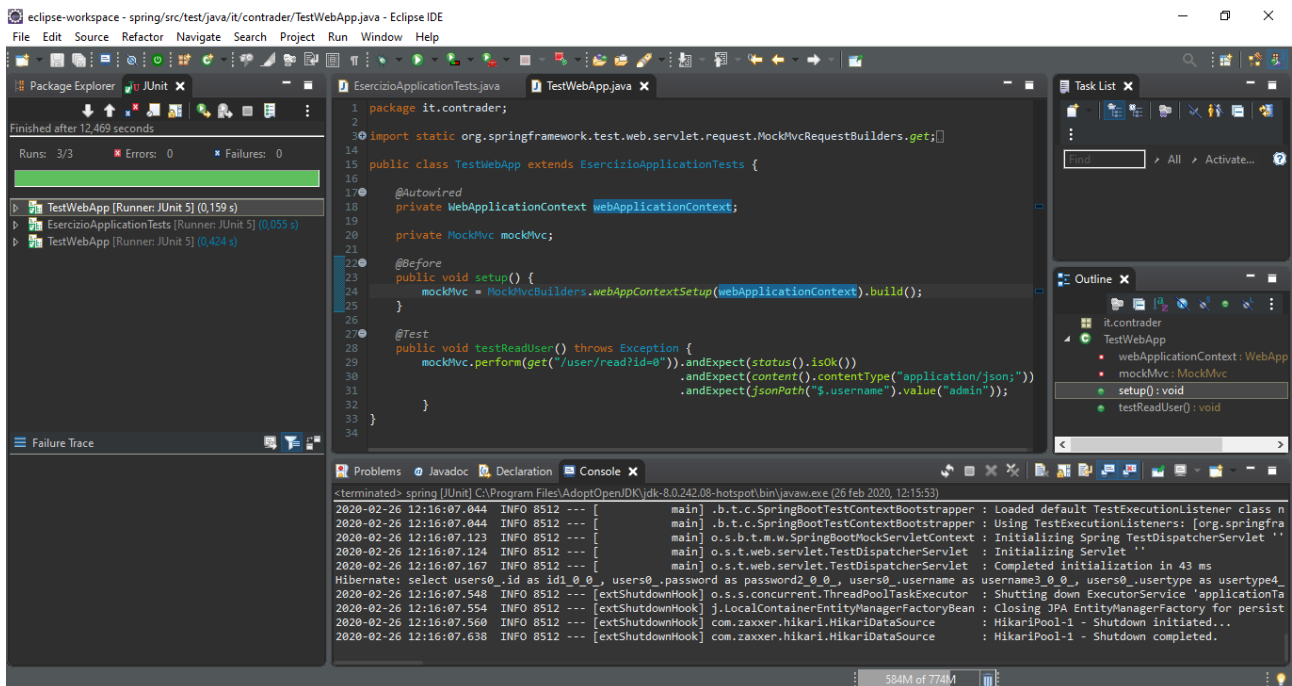
- 6- Creare una nuova classe java come in figura(Scrivere il test in base alle proprie esigenze):

```
1 package it.contrader;  
2  
3 import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;  
4  
14 public class TestWebApp extends EsercizioApplicationTests {  
15  
16  
17     @Autowired  
18     private WebApplicationContext webApplicationContext;  
19  
20     private MockMvc mockMvc;  
21  
22     @Before  
23     public void setup() {  
24         mockMvc = MockMvcBuilders.webAppContextSetup(webApplicationContext).build();  
25     }  
26  
27     @Test  
28     public void testReadUser() throws Exception {  
29         mockMvc.perform(get("/user/read?id=0"))  
30             .andExpect(status().isOk())  
31             .andExpect(content().contentType("application/json;"))  
32             .andExpect(jsonPath("$.username").value("admin"));  
33     }  
34 }
```

3 Es. della classe da creare

Con @test si indica il test da eseguire, in questa figura viene eseguito il test della read. Il metodo perform esegue la richiesta, in questo caso una get su user, e con `andExpect(status().isOk())` verifica che la richiesta vada a buon fine, mentre `contentType` verifica che nella body della risposta ci sia un json, l'ultimo verifica che all'interno del json il campo username sia uguale ad admin

- 7- Per eseguire il test premere il tasto destro sul progetto, Run As e cliccare su JUnit test. Si aprirà la finestra di junit che ci darà l'esito del test:



4 Esito positivo