



# Qualidade - Cobertura de Código & Testes



Alberto Yano

# Introdução

O que vamos ver neste e-book?

Em um ambiente corporativo é exigido no desenvolvimento de software qualidade de código, documentação, testabilidade, cobertura de testes, segurança, manutenibilidade, modularização, dentre outros.

Assim, é comumente utilizado o software Sonar para a verificação de alguns desses itens.

Dessa forma, para agilizar o processo de desenvolvimento é recomendável utilizar o Sonar para verificação do código antes de subí-lo para um repositório.

Então vamos ver como isso funciona!?

# Introdução

## Escopo

Utilizaremos o Docker e o Docker Compose para rodar o Sonar configurando-o para utilizar o mesmo profile usado em uma empresa.

Criaremos um projeto no Sonar com uma pequena API REST Java utilizando Spring Boot.

E por fim, faremos um teste unitário e veremos o que o Sonar nos informa do projeto.





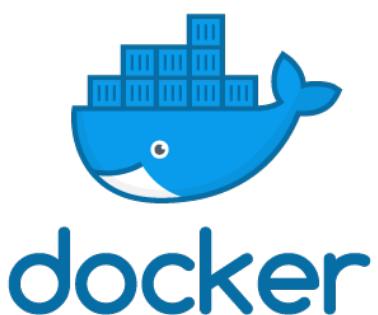
# Docker e Docker Compose

# Docker e Docker Compose

## O que é Docker?

**Docker** é uma plataforma *open source* que permite a criação, execução e o gerenciamento de **aplicações em containers**. Os Containers são unidades leves, portáteis e isoladas que empacotam uma aplicação com todas as suas **dependências, bibliotecas e configurações**, garantindo que ela funcione de maneira consistente em qualquer ambiente.

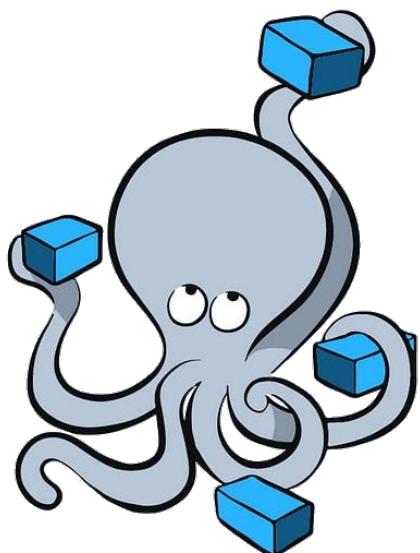
O **Docker** simplifica o processo de desenvolvimento e implantação, evitando o clássico problema do "na minha máquina funciona".



# Docker e Docker Compose

## O que é Docker Compose?

**Docker Compose** é uma ferramenta complementar ao Docker que permite definir e gerenciar **múltiplos containers** de forma coordenada, utilizando um arquivo de configuração no formato YAML (docker-compose.yml).



**docker**  
**Compose**

# Docker e Docker Compose

## Instalação

Não é o objetivo deste e-book ensinar a instalação do Docker e do Docker Compose, mas eles serão necessários para a utilização do Sonar.

Então seguem os links para a instalação:

(Windows)

<https://docs.docker.com/desktop/setup/install/windows-install/>

(Linux)

<https://docs.docker.com/desktop/setup/install/linux/>

(Mac)

<https://docs.docker.com/desktop/setup/install/mac-install/>

02

# Sonar ou SonarQube

---

# Sonar

## O que é o Sonar ou SonarQube?

O **SonarQube** é uma plataforma open source projetada para a **inspeção contínua da qualidade de código** em diversos tipos de linguagens de programação, como Java, JavaScript, C#, Python, entre outras. Ele automatiza a análise estática do código-fonte, identificando problemas relacionados à **manutenibilidade, segurança, confiabilidade e complexidade**.



# Sonar

## Objetivo do Sonar

Seu principal objetivo é fornecer um **feedback rápido e contínuo** aos desenvolvedores, ajudando a prevenir a introdução de bugs, vulnerabilidades e más práticas de codificação ao longo do ciclo de desenvolvimento.

Além disso, o SonarQube promove a **melhoria contínua** ao permitir que equipes definam **padrões de qualidade** e acompanhem a sua evolução por meio de **painéis e relatórios detalhados**.





# Instalando o Sonar

# Sonar

## Instalação do Sonar

A forma mais fácil para a **instalação do Sonar** é baixar o arquivo docker-compose.yml no seguinte link:

<https://github.com/SonarSource/docker-sonarqube/tree/master/example-compose-files/sq-with-postgres>

E no diretório onde está o arquivo executar a seguinte linha de comando:

```
● ● ● prompt
1 docker-compose up -d
2 [+] Running 9/9
3   ✓ Network sonar_ipv4                                Created
4   ✓ Volume "sonar_sonarqube_temp"                      Created
5   ✓ Volume "sonar_postgresql"                           Created
6   ✓ Volume "sonar_postgresql_data"                     Created
7   ✓ Volume "sonar_sonarqube_data"                      Created
8   ✓ Volume "sonar_sonarqube_extensions"                Created
9   ✓ Volume "sonar_sonarqube_logs"                      Created
10  ✓ Container postgresql                               Healthy
11  ✓ Container sonarqube                            Started
```

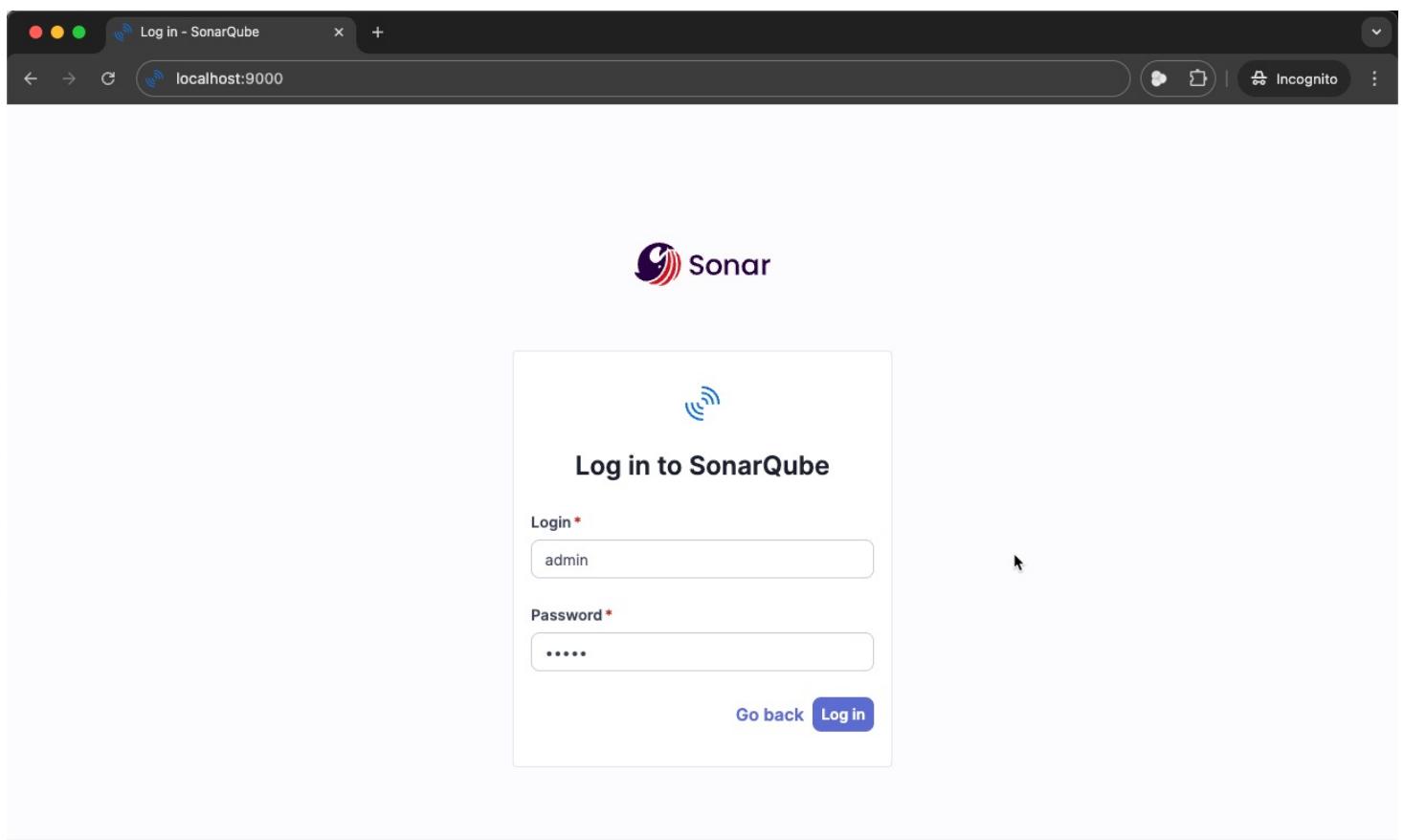
# Sonar

## Acessando o Sonar

Utilize a seguinte URL no seu navegador:

<http://localhost:9000/>

Utilize o usuário e senha “admin”



# Sonar

## Acessando o Sonar

Será solicitada a troca da senha, que precisará atender às exigências de uma senha forte.

The screenshot shows a web browser window titled "Update password - SonarQube". The URL in the address bar is "localhost:9000/account/reset\_password". The page content is as follows:

**Update your password**

⚠ This account should not use the default password.

**Enter a new password**

**Old Password \***  
.....

**Password \***  
.....

Your password must include at least:

- ✓ 12 characters
- ✓ 1 upper case letter
- ✓ 1 lower case letter
- ✓ 1 number
- ✓ 1 special character

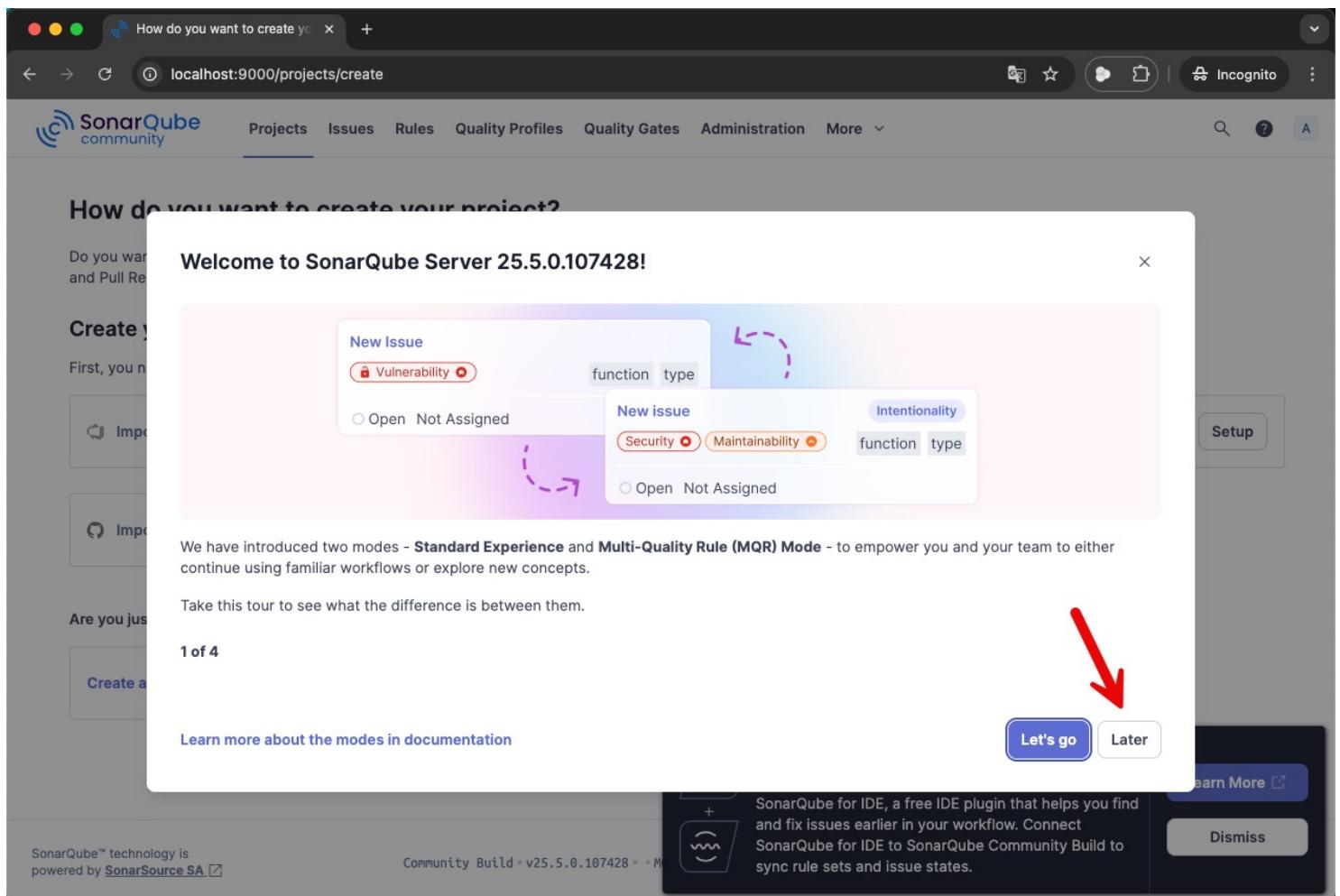
**Confirm Password \***  
.....

**Update**

# Sonar

## Acessando o Sonar

Clique em “Later”



# Sonar

## Acessando o Sonar

Clique em “*Dismiss*”

The screenshot shows the SonarQube Community Build interface at [localhost:9000/projects/create](http://localhost:9000/projects/create). The main page asks "How do you want to create your project?" with options for Azure DevOps, GitHub, and Bitbucket Cloud. A tooltip for the Bitbucket Cloud import says "You can replay the tour from the help section". On the right, a sidebar lists links like Documentation, Web API, and Tours. A prominent dark banner at the bottom right contains text about SonarQube for IDE and a "Dismiss" button, which is highlighted with a large red arrow.

How do you want to create your project?

Do you want to benefit from all of SonarQube Community Build's features (like repository import and Pull Request decoration)?

Create your project from your favorite DevOps platform.

First, you need to set up a DevOps platform configuration.

Import from Azure DevOps    Setup

Import from GitHub    Setup

Import from Bitbucket Cloud    Setup

You can replay the tour from the help section

Import

Got it

Documentation

Web API

Web API v2

Get Help

Stay Connected

Product News

Product Roadmap

X @SonarQube

Tours

Standard Experience and MQR mode tour

Are you just testing or have an advanced use-case?

Create a local project

SonarQube™ technology is powered by [SonarSource SA](#)

Community Build v25.5.0.107428 M

Get the most out of SonarQube Community Build!

Take advantage of the whole ecosystem by using SonarQube for IDE, a free IDE plugin that helps you find and fix issues earlier in your workflow. Connect SonarQube for IDE to SonarQube Community Build to sync rule sets and issue states.

Learn More

Dismiss

# Sonar

## Acessando o Sonar

Pronto seu Sonar estará instalado ...!!!

The screenshot shows a web browser window with the following details:

- Address Bar:** localhost:9000/projects/create
- Page Title:** How do you want to create yo
- Header:** SonarQube community
- Navigation:** Projects, Issues, Rules, Quality Profiles, Quality Gates, Administration, More
- Content:**
  - Section:** How do you want to create your project?
  - Text:** Do you want to benefit from all of SonarQube Community Build's features (like repository import and Pull Request decoration)?
  - Section:** Create your project from your favorite DevOps platform.
  - Text:** First, you need to set up a DevOps platform configuration.
  - Buttons:** Import from Azure DevOps (Setup), Import from Bitbucket Cloud (Setup), Import from Bitbucket Server (Setup), Import from GitHub (Setup), Import from GitLab (Setup).
  - Section:** Are you just testing or have an advanced use-case?
  - Button:** Create a local project



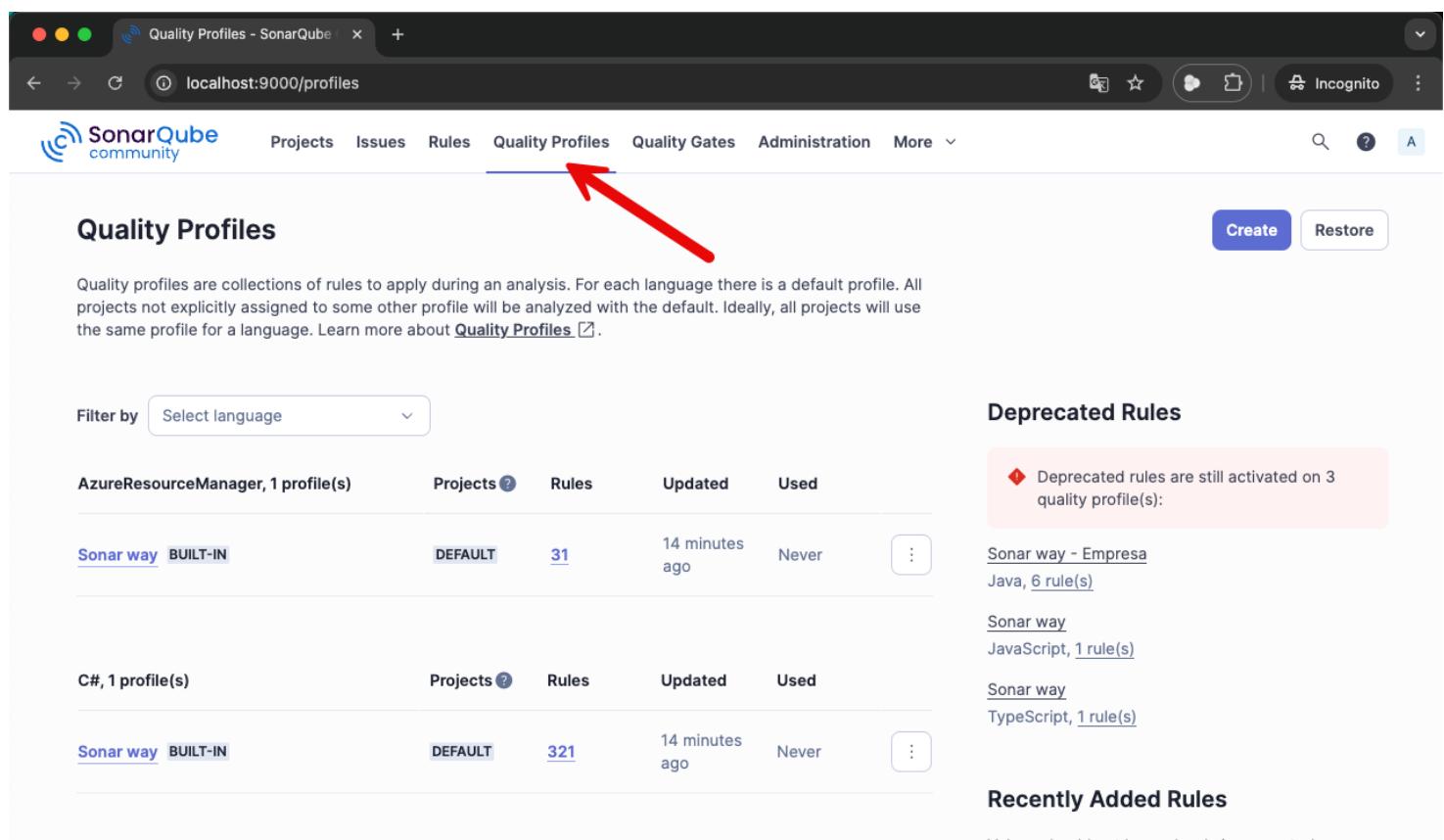
# Configurando o Sonar

# Sonar

## Configurando o Sonar

Muito provavelmente sua empresa tem um sonar configurado com as políticas de verificações, então precisaremos importá-las.

Acesse o Sonar da sua empresa e siga os seguintes passos:



The screenshot shows the SonarQube interface for managing quality profiles. The top navigation bar includes tabs for Projects, Issues, Rules, Quality Profiles (which is highlighted with a red arrow), Quality Gates, Administration, and More. Below the navigation is a search bar and a toolbar with Create and Restore buttons. The main content area is titled "Quality Profiles" and contains a brief description of what quality profiles are. It features a table of profiles, a "Deprecated Rules" section, and a "Recently Added Rules" section. The table lists profiles like "AzureResourceManager, 1 profile(s)" and "Sonar way BUILT-IN". The "Deprecated Rules" section highlights three rules still activated on three profiles, and the "Recently Added Rules" section lists rules added in the last 14 minutes.

Profile Name	Type	Projects	Rules	Updated	Used	Actions
AzureResourceManager, 1 profile(s)		Projects	Rules	Updated	Used	⋮
Sonar way BUILT-IN	BUILT-IN	DEFAULT	31	14 minutes ago	Never	⋮
C#, 1 profile(s)		Projects	Rules	Updated	Used	⋮
Sonar way BUILT-IN	BUILT-IN	DEFAULT	321	14 minutes ago	Never	⋮

**Deprecated Rules**

- Deprecation status: ⚠️ Deprecated rules are still activated on 3 quality profile(s):
  - Sonar way - Empresa Java, 6 rule(s)
  - Sonar way JavaScript, 1 rule(s)
  - Sonar way TypeScript, 1 rule(s)

**Recently Added Rules**

Values should not be uselessly incremented

# Sonar

## Configurando o Sonar

Neste exemplo, exportaremos o *profile* Java da Empresa.

Salve em um diretório do seu computador.

The screenshot shows the SonarQube Quality Profiles page at [localhost:9000/profiles](http://localhost:9000/profiles). The 'Java, 2 profile(s)' section is selected. Two profiles are listed:

- Sonar way** (BUILT-IN) - DEFAULT Rules: 527 Updated: 18 minutes ago Used: Never
- Sonar way - Empresa** - 0 Rules: 356 (6 critical) Updated: 4 minutes ago Used: Never

A context menu is open over the 'Sonar way - Empresa' profile, with the following options:

- Activate More Rules
- Backup (highlighted with red arrow 3)
- Compare
- Extend
- Copy
- Rename
- Set as Default
- Delete

Red arrows numbered 1, 2, and 3 point to the profile name, the three-dot menu icon, and the 'Backup' option respectively.

# Sonar

## Configurando o Sonar

Volte ao Sonar do seu computador e siga os seguintes passos selecionando o arquivo salvo.

The screenshot shows the SonarQube web interface at `localhost:9000/profiles`. The main navigation bar includes Projects, Issues, Rules, Quality Profiles (which is the active tab), Quality Gates, Administration, and More. Below the navigation, there's a search bar and a toolbar with Create and Restore buttons. A red arrow labeled '1' points to the 'Restore' button. Red arrows labeled '2' and '3' point to the 'Choose file' button and the 'Restore' button respectively in a modal dialog titled 'Restore Profile'. The dialog also contains a 'Cancel' button. The background shows a list of quality profiles: AzureResourceManager (1 profile), Sonar way (BUILT-IN), C# (1 profile), CSS (1 profile), and another Sonar way (BUILT-IN). The bottom right corner features a 'Recently Added Rules' section with several listed rules.

# Sonar

## Configurando o Sonar

Depois deixe o *profile* como sendo o *default*, para que ao enviar os dados para o Sonar ele utilize o *profile* da empresa.

The screenshot shows the SonarQube Quality Profiles page at [localhost:9000/profiles](http://localhost:9000/profiles). The page lists profiles for Java, JavaScript, and Kotlin. A context menu is open over the 'Sonar way - Empresa' profile in the Java section. The menu options include: Activate More Rules, Back up, Compare, Extend, Copy, Rename, Set as Default (which is highlighted), and Delete. Red numbered arrows point to each step: 1 points to the profile name 'Sonar way - Empresa'; 2 points to the three-dot menu icon; 3 points to the 'Set as Default' option in the context menu.

Language	Profile Name	Type	Default	Rules	Updated	Used
Java	Sonar way	BUILT-IN	DEFAULT	527	25 minutes ago	Never
Java	Sonar way - Empresa	Custom		356 (6)	40 seconds ago	Never
JavaScript	Sonar way	BUILT-IN	DEFAULT	315 (1)	25 minutes ago	Never
Kotlin	Sonar way	BUILT-IN	DEFAULT	131	25 minutes ago	Never



# Criando um Projeto no Sonar

# Sonar

## Criando um Projeto no Sonar

Clique em “*Projects*” e depois em “*Create a local project*”.

The screenshot shows the SonarQube interface for creating a new project. A red arrow labeled '1' points to the 'Projects' menu item in the top navigation bar. Another red arrow labeled '2' points to the 'Create a local project' button in the main content area.

How do you want to create your project?

Do you want to benefit from all of SonarQube Community Build's features (like repository import and Pull Request decoration)?

Create your project from your favorite DevOps platform.

First, you need to set up a DevOps platform configuration.

Import from Azure DevOps	Setup
Import from Bitbucket Cloud	Setup
Import from Bitbucket Server	Setup
Import from GitHub	Setup
Import from GitLab	Setup

Are you just testing or have an advanced use-case?

Create a local project

# Sonar

## Criando um Projeto no Sonar

Coloque o “*Project Display name*” e o “*Project key*”. Preencha com todas as letras minúsculas e sem espaço.

The screenshot shows a web browser window for 'Create a local project' at the URL [localhost:9000/projects/create?mode=manual](http://localhost:9000/projects/create?mode=manual). The page title is 'Create a local project'. The main content area is titled 'Create a local project'. It contains three input fields: 'Project display name \*' with value 'teste-api', 'Project key \*' with value 'teste-api', and 'Main branch name \*' with value 'main'. Below the main branch name field is a note: 'The name of your project's default branch [Learn More](#)'. At the bottom are 'Cancel' and 'Next' buttons.

1 of 2 X

### Create a local project

Project display name \* ⓘ  
teste-api

Project key \* ⓘ  
teste-api

Main branch name \* ⓘ  
main

The name of your project's default branch [Learn More](#)

Cancel Next

# Sonar

## Criando um Projeto no Sonar

Selecione “*Use the global setting*” e clique em “*Create project*”.

The screenshot shows a web browser window for 'Create a local project' at 'localhost:9000/projects/create?mode=manual&setncd=true'. The page title is 'Set up project for Clean as You Code'. It explains that the new code definition sets which part of your code will be considered new code, helping to focus on recent changes. A red arrow points to the radio button for 'Use the global setting'.

**Set up project for Clean as You Code**

The new code definition sets which part of your code will be considered new code. This helps you focus attention on the most recent changes to your project, enabling you to follow the Clean as You Code methodology. Learn more: [Defining New Code](#).

Choose the baseline for new code for this project

Use the global setting ←

**Previous version**

Any code that has changed since the previous version is considered new code.  
Recommended for projects following regular versions or releases.

Define a specific setting for this project

Previous version  
Any code that has changed since the previous version is considered new code.  
Recommended for projects following regular versions or releases.

Number of days  
Any code that has changed in the last x days is considered new code. If no action is taken on a new issue after x days, this issue will become part of the overall code.  
Recommended for projects following continuous delivery.

# Sonar

## Criando um Projeto no Sonar

Coloque o “*Token name*” igual ao “*Project key*” tudo em minúsculo e sem espaço.

Por fim, selecione “*No expiration*” e clique em “*Generate*”.

The screenshot shows the SonarQube interface for generating a project token. The URL in the browser is `localhost:9000/tutorials?id=teste-api&selectedTutorial=local`. The page title is "SonarQube community". The main content area is titled "Analyze your project" and includes a note: "We initialized your project on SonarQube Community Build, now it's up to you to launch analyses!". Below this, there is a section for "Provide a token". It has two options: "Generate a project token" (selected) and "Use existing token". A red arrow labeled "1" points to the "Token name" input field, which contains "teste-api". Another red arrow labeled "2" points to the "Expires in" dropdown menu, which is set to "No expiration". A third red arrow labeled "3" points to the "Generate" button. A callout box below the token input field provides instructions: "Please note that this token will only allow you to analyze the current project. If you want to use the same token to analyze multiple projects, you need to generate a global token in your [user account](#). See the [documentation](#) for more information." At the bottom, a note states: "The token is used to identify you when an analysis is performed. If it has been compromised, you can revoke it at any point in time in your [user account](#)".

# Sonar

## Criando um Projeto no Sonar

Clique em “*Continue*”.

The screenshot shows the SonarQube interface for the 'Teste API' project. The URL in the browser is `localhost:9000/tutorials?id=teste-api&selectedTutorial=local`. The main heading is 'Analyze your project'. Below it, a message says 'We initialized your project on SonarQube Community Build, now it's up to you to launch analyses!'. A section titled '1 Provide a token' contains a token value: `teste-api: sqp_ab0bf705c384fd672afb2d14cee076ce812e8de3`. A note below states: 'The token is used to identify you when an analysis is performed. If it has been compromised, you can revoke it at any point in time in your [user account](#)'. A 'Continue' button is present, with a large red arrow pointing to it from the left. Another section below is titled '2 Run analysis on your project'.

# Sonar

## Criando um Projeto no Sonar

Clique em “*Maven*” e copie o conteúdo.

The screenshot shows the SonarQube interface for a project named 'teste-api'. The top navigation bar includes links for Projects, Issues, Rules, Quality Profiles, Quality Gates, Administration, and More. Below the navigation, there are tabs for Overview, Issues, Security Hotspots, Measures, Code, and Activity. On the right, there are Project Settings and Project Information tabs. The main content area is titled 'Analyze your project' and contains instructions: 'We initialized your project on SonarQube Community Build, now it's up to you to launch analyses!'. It shows a step 1 'Provide a token' with a green checkmark and token 'teste-api: sqp\_d903c4b0b6a46e75fb6a5c9ce25353ff1c0939ea'. Step 2 'Run analysis on your project' is highlighted with a red arrow pointing to the 'Maven' tab in a dropdown menu. The dropdown also includes options for Gradle, JS/TS & Web, .NET, and Other (for Go, Python, PHP, ...). Below this, there is a section titled 'Execute the Scanner for Maven' with a command-line example:

```
mvn clean verify sonar:sonar \
-Dsonar.projectKey=teste-api \
-Dsonar.projectName='teste-api' \
-Dsonar.host.url=http://localhost:9000 \
-Dsonar.token=sqp_d903c4b0b6a46e75fb6a5c9ce25353ff1c0939ea
```

A red arrow points from the 'Copy' button to the command line. At the bottom, a note says: 'Please visit the [official documentation of the Scanner for Maven](#) for more details.'

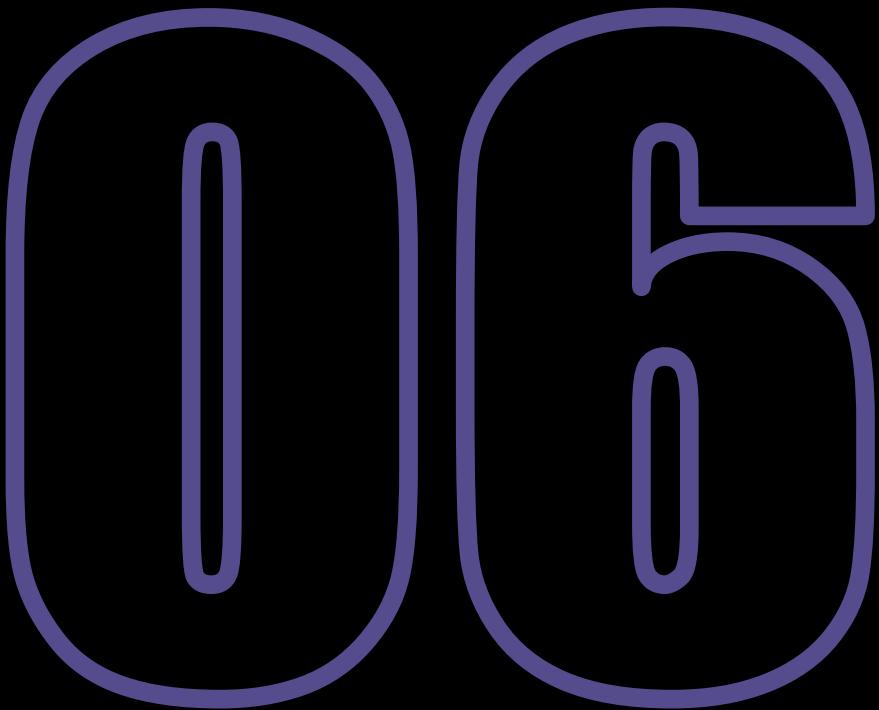
# Sonar

## Criando um Projeto no Sonar

Cole o conteúdo em um editor de texto e retire o “*mvn*” os “\“ e deixe tudo em uma única linha.

Ficará da seguinte forma:

```
clean verify sonar:sonar -Dsonar.projectKey=teste-api -  
Dsonar.projectName='teste-api' -  
Dsonar.host.url=http://localhost:9000 -  
Dsonar.token=sqp_d903c4b0b6a46e75fb6a5c9ce2535  
3ff1c0939ea
```



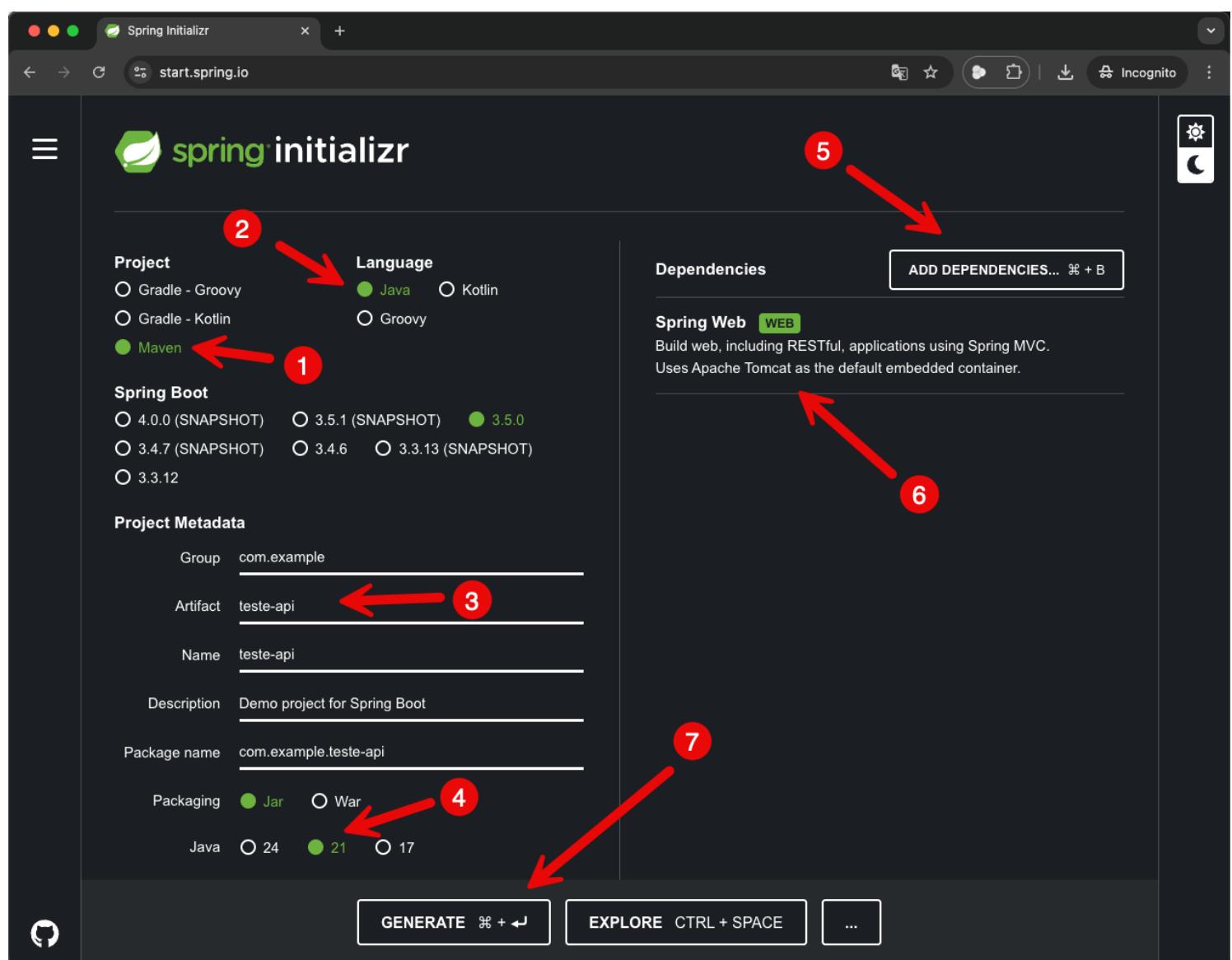
# Spring Boot Aplicação REST API

---

# Spring Boot

## Criando uma Aplicação REST API

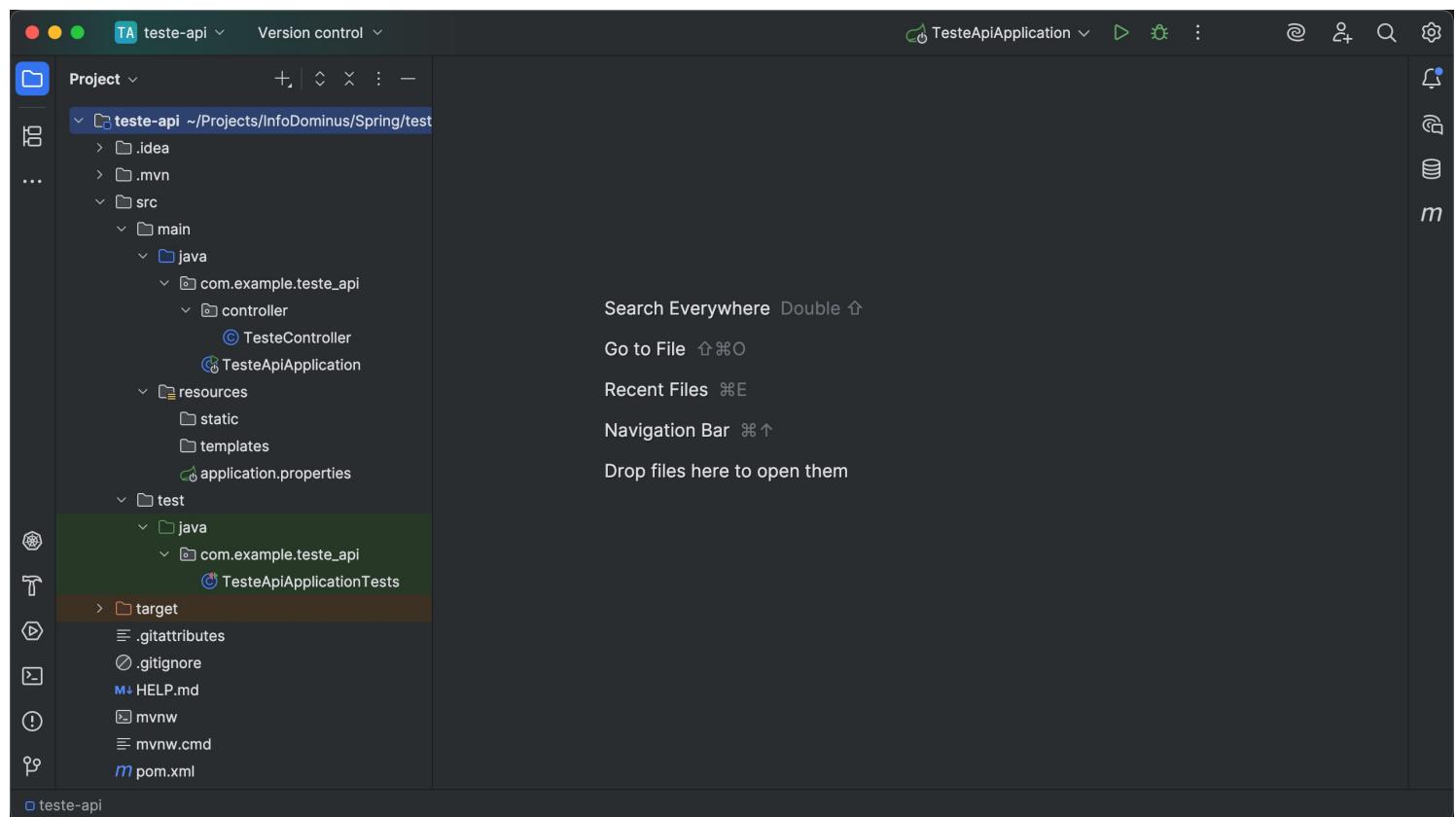
Acesse o seguinte link: <https://start.spring.io/> e selecione preenchendo da forma abaixo:



# Spring Boot

## Criando uma Aplicação REST API

Descompacte o arquivo baixado e abra em sua IDE preferida.

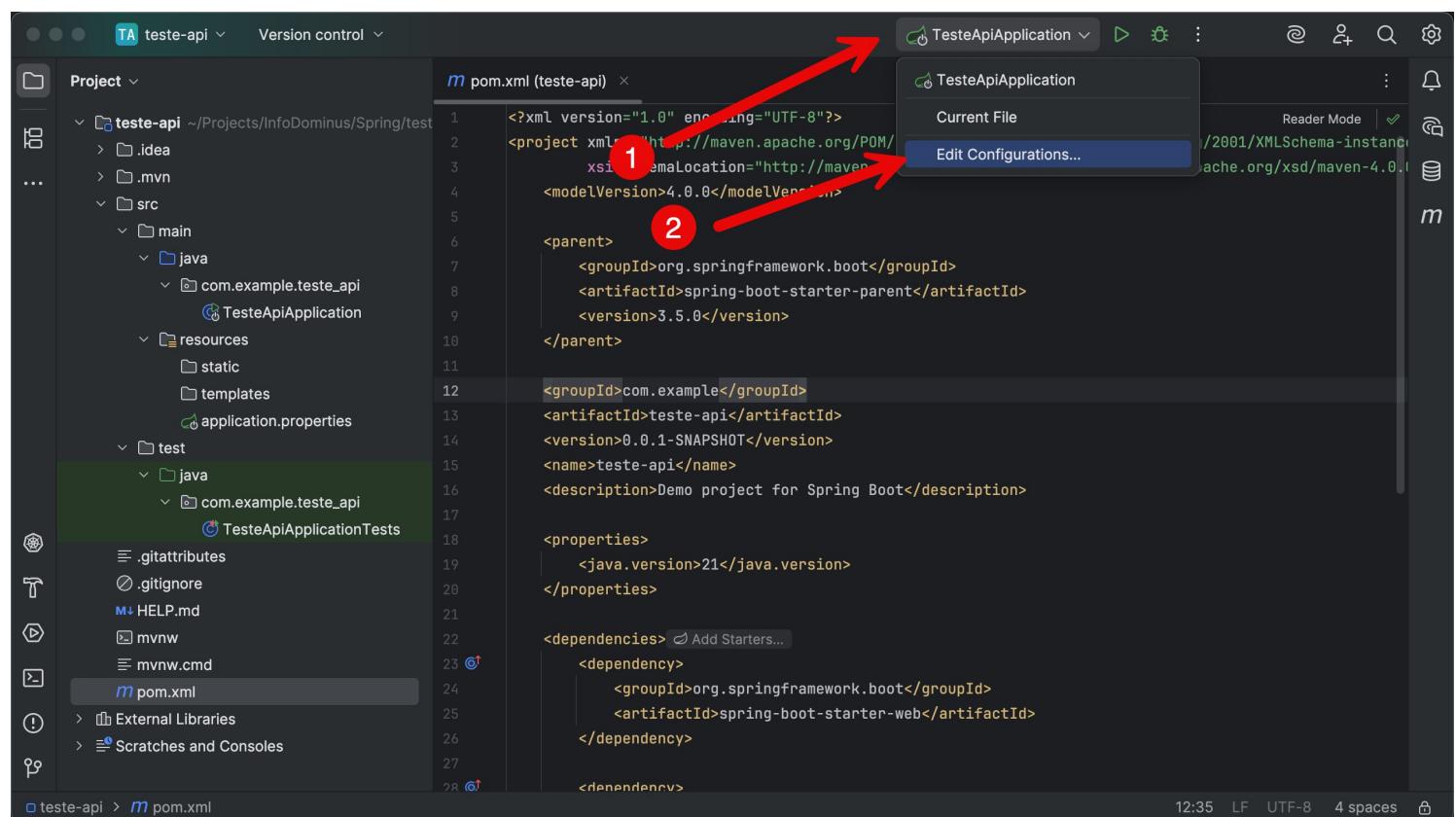


# Spring Boot

## Criando uma Aplicação REST API

Crie em sua IDE uma execução Maven com o comando da página 30 acrescentando ao final o seguinte código: -Dsonar.scm.disabled=true

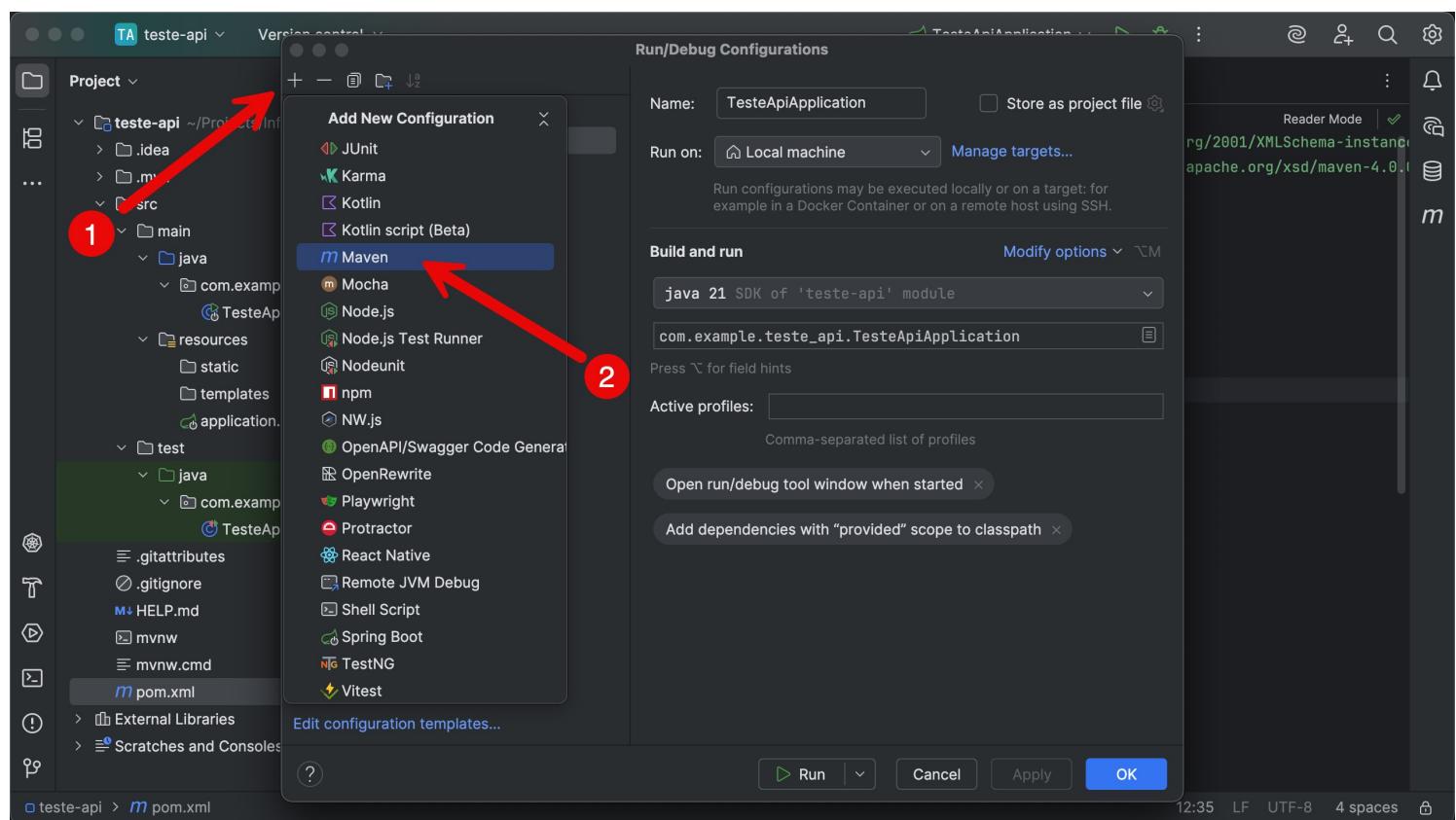
Usaremos aqui o IntelliJ:



# Spring Boot

## Criando uma Aplicação REST API

No IntelliJ, clique no “+” e selecione *Maven* para criar uma execução *Maven* para rodar o projeto com teste para o Sonar.

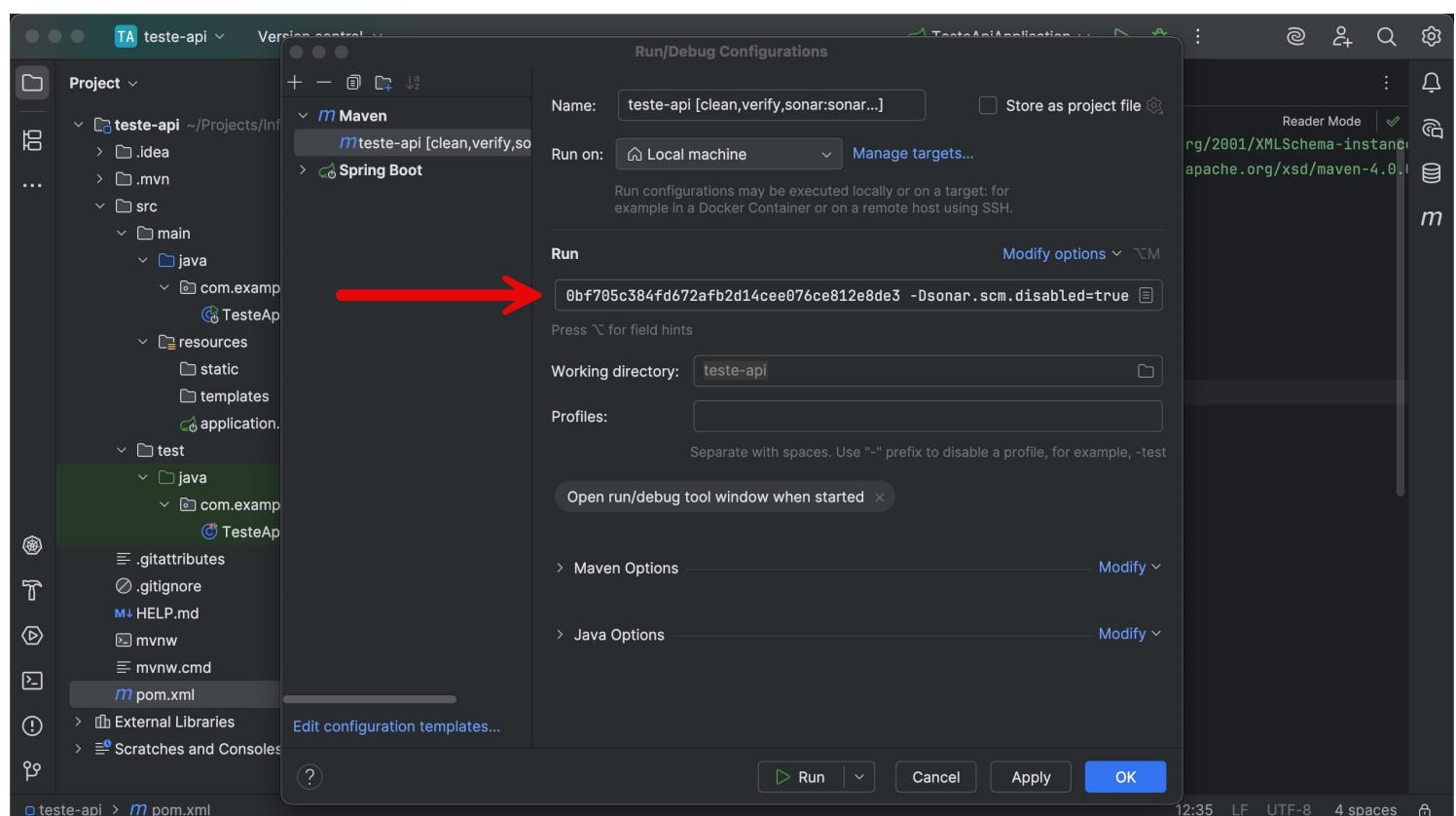


# Spring Boot

## Criando uma Aplicação REST API

Cole o comando completo no espaço  
“Run” e clique em “OK”:

```
clean verify sonar:sonar -Dsonar.projectKey=teste-api -  
Dsonar.projectName='teste-api' -  
Dsonar.host.url=http://localhost:9000 -  
Dsonar.token=sqp_d903c4b0b6a46e75fb6a5c9ce25353ff1  
c0939ea -Dsonar.scm.disabled=true
```



# Spring Boot

## Criando uma Aplicação REST API

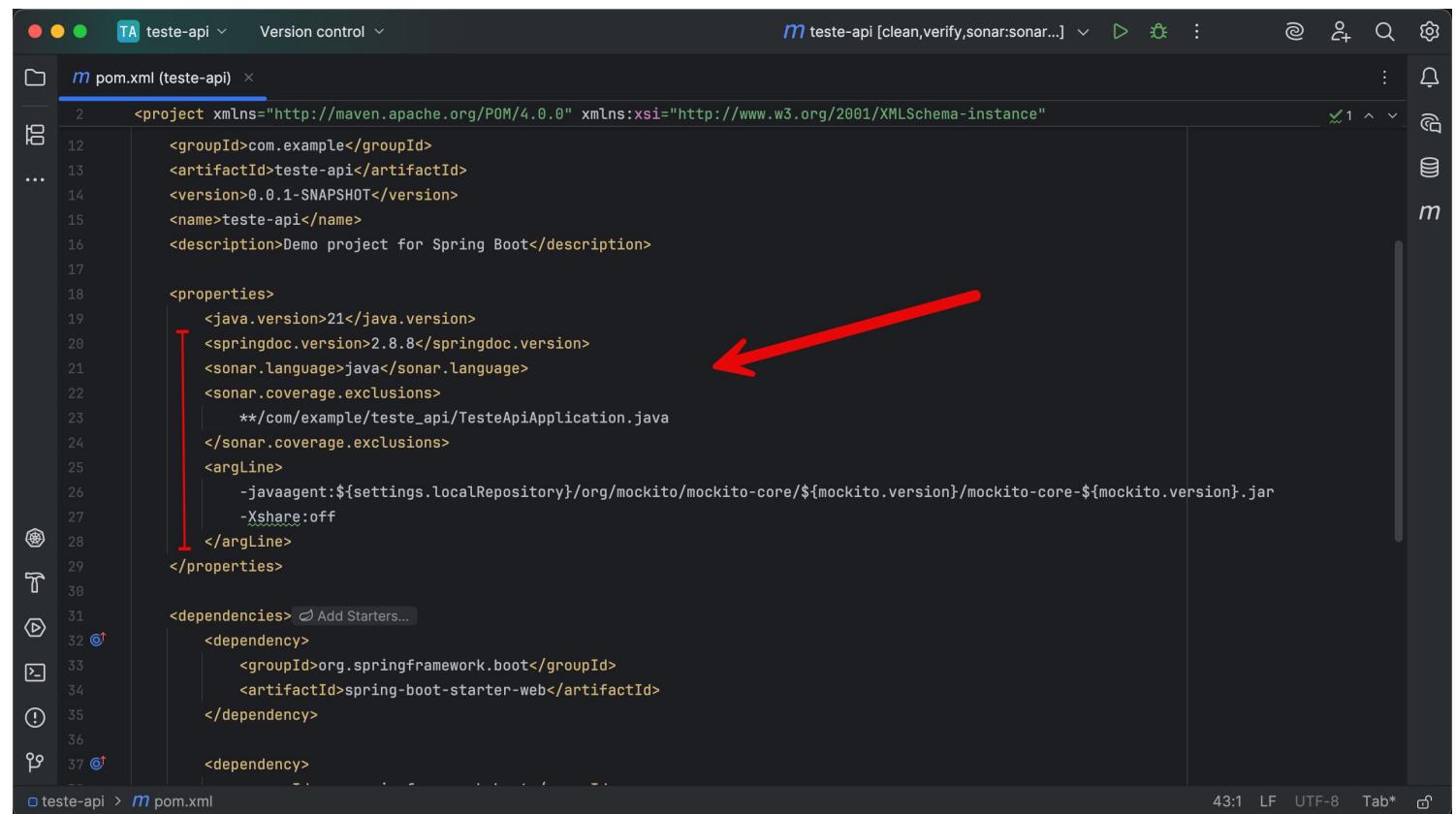
No POM.XML vamos acrescentar algumas tags dentro de <properties> dessa forma coloque as seguintes tags:

```
<springdoc.version>2.8.8</springdoc.version>
<sonar.language>java</sonar.language>
<sonar.coverage.exclusions>
  **/com/example/teste_api/TesteApiApplication.java
</sonar.coverage.exclusions>
<argLine>
-
  javaagent:${settings.localRepository}/org/mockito/mockit
  o-core/${mockito.version}/mockito-core-
  ${mockito.version}.jar
  -Xshare:off
</argLine>
```

# Spring Boot

## Criando uma Aplicação REST API

O <properties> do POM.XML ficará da seguinte forma:



```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <groupId>com.example</groupId>
    <artifactId>teste-api</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>teste-api</name>
    <description>Demo project for Spring Boot</description>

    <properties>
        <java.version>21</java.version>
        <springdoc.version>2.8.8</springdoc.version>
        <sonar.language>java</sonar.language>
        <sonar.coverage.exclusions>
            **/com/example/teste_api/TesteApiController.java
        </sonar.coverage.exclusions>
        <argLine>
            -javaagent:${settings.localRepository}/org/mockito/mockito-core/${mockito.version}/mockito-core-${mockito.version}.jar
            -Xshare:off
        </argLine>
    </properties>

    <dependencies> ⚡ Add Starters...
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
    <dependencies>
```

# Spring Boot

## Criando uma Aplicação REST API

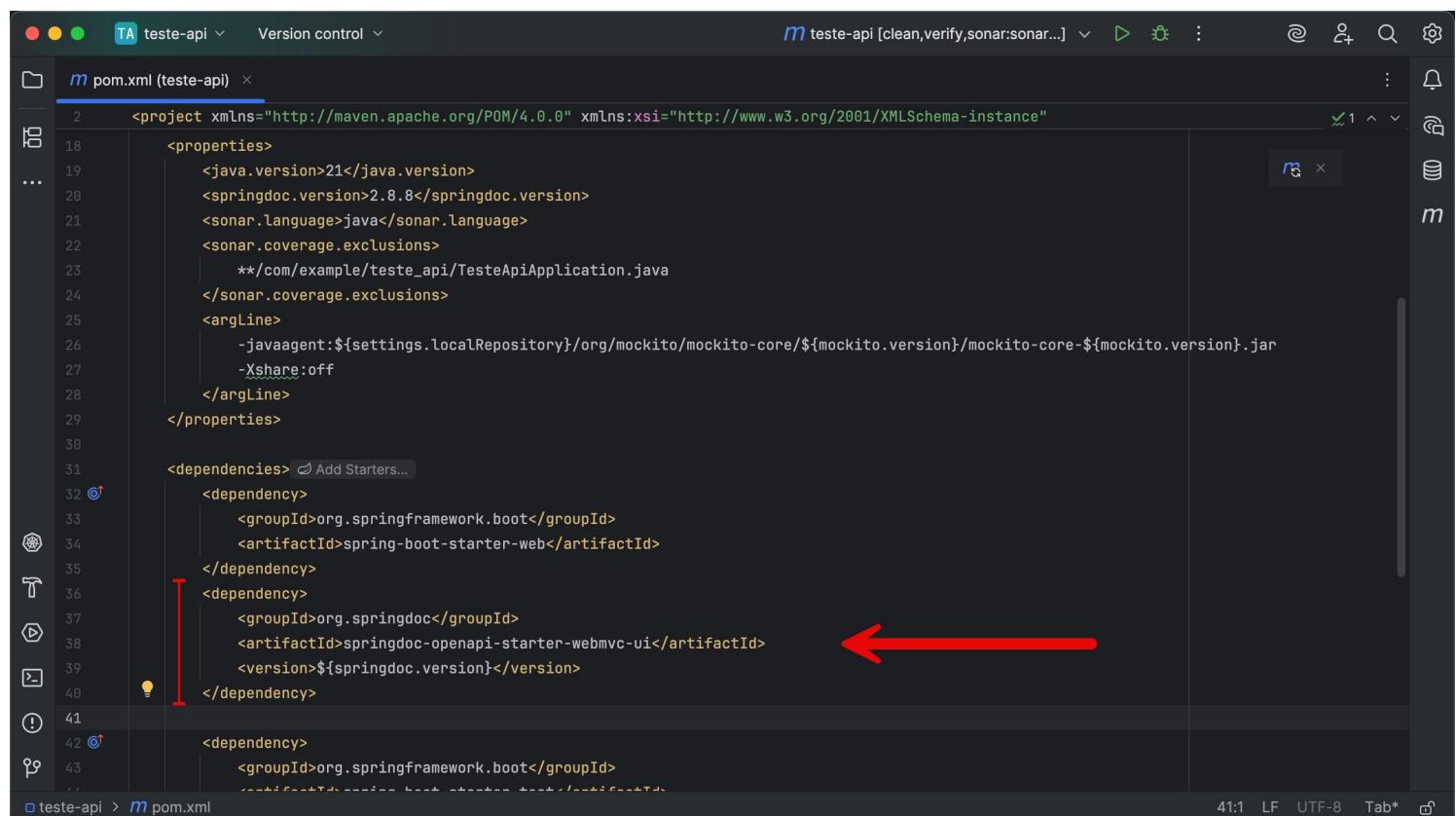
No POM.XML vamos acrescentar uma dependência para a documentação e acesso à API:

```
<dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-starter-webmvc-
ui</artifactId>
    <version>${springdoc.version}</version>
</dependency>
```

# Spring Boot

## Criando uma Aplicação REST API

O <dependencies> do POM.XML ficará da seguinte forma:



```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <properties>
    <java.version>21</java.version>
    <springdoc.version>2.8.8</springdoc.version>
    <sonar.language>java</sonar.language>
    <sonar.coverage.exclusions>
      **/com/example/teste_api/TesteApiApplication.java
    </sonar.coverage.exclusions>
    <argLine>
      -javaagent:${settings.localRepository}/org/mockito/mockito-core/${mockito.version}/mockito-core-${mockito.version}.jar
      -Xshare:off
    </argLine>
  </properties>

  <dependencies> ⚡ Add Starters...
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springdoc</groupId>
      <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
      <version>${springdoc.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-devtools</artifactId>
    </dependency>
  </dependencies>
</project>
```

# Spring Boot

## Criando uma Aplicação REST API

No POM.XML vamos acrescentar ainda alguns plugins:

```
<plugin>
    <groupId>org.sonarsource.scanner.maven</groupId>
    <artifactId>sonar-maven-plugin</artifactId>
    <version>5.1.0.4751</version>
</plugin>
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
</plugin>
<plugin>
```

# Spring Boot

## Criando uma Aplicação REST API

O <plugins> do POM.XML ficará da seguinte forma:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
      <plugin>
        <groupId>org.sonarsource.scanner.maven</groupId>
        <artifactId>sonar-maven-plugin</artifactId>
        <version>5.1.0.4751</version>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
      </plugin>
      <plugin>
        <groupId>org.jacoco</groupId>
        <artifactId>jacoco-maven-plugin</artifactId>
        <version>0.8.13</version>
        <configuration>
          <output>file</output>
          <append>true</append>
        </configuration>
        <executions>
          <execution>
            <goals>
              <goal>prepare-agent</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
</project>
```

# Spring Boot

## Criando uma Aplicação REST API

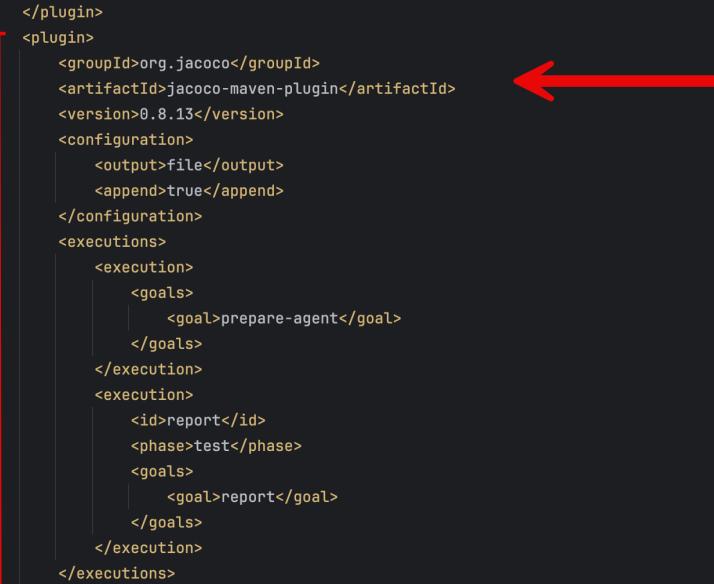
No POM.XML vamos acrescentar um *plugin* para cobertura de código:

```
<plugin>
    <groupId>org.jacoco</groupId>
    <artifactId>jacoco-maven-plugin</artifactId>
    <version>0.8.13</version>
    <configuration>
        <output>file</output>
        <append>true</append>
    </configuration>
    <executions>
        <execution>
            <goals>
                <goal>prepare-agent</goal>
            </goals>
        </execution>
        <execution>
            <id>report</id>
            <phase>test</phase>
            <goals>
                <goal>report</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

# Spring Boot

## Criando uma Aplicação REST API

O <plugins> do POM.XML ficará da seguinte forma:

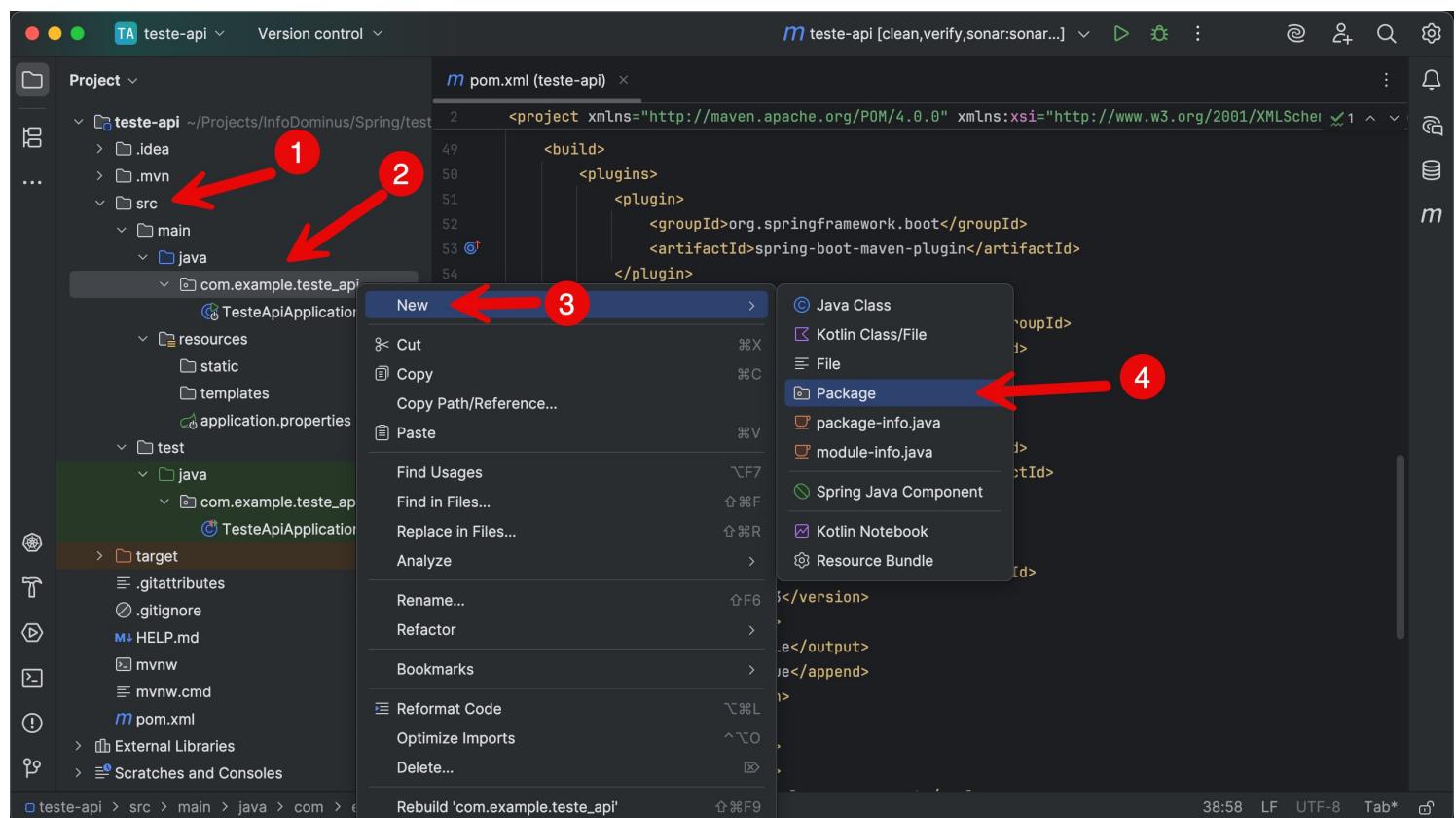


```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <build>
    <plugins>
      <!-- maven-compiler-plugin -->
      <plugin>
        <groupId>org.jacoco</groupId>
        <artifactId>jacoco-maven-plugin</artifactId>
        <version>0.8.13</version>
        <configuration>
          <output>file</output>
          <append>true</append>
        </configuration>
        <executions>
          <execution>
            <goals>
              <goal>prepare-agent</goal>
            </goals>
          </execution>
          <execution>
            <id>report</id>
            <phase>test</phase>
            <goals>
              <goal>report</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
</project>
```

# Spring Boot

## Criando uma Aplicação REST API

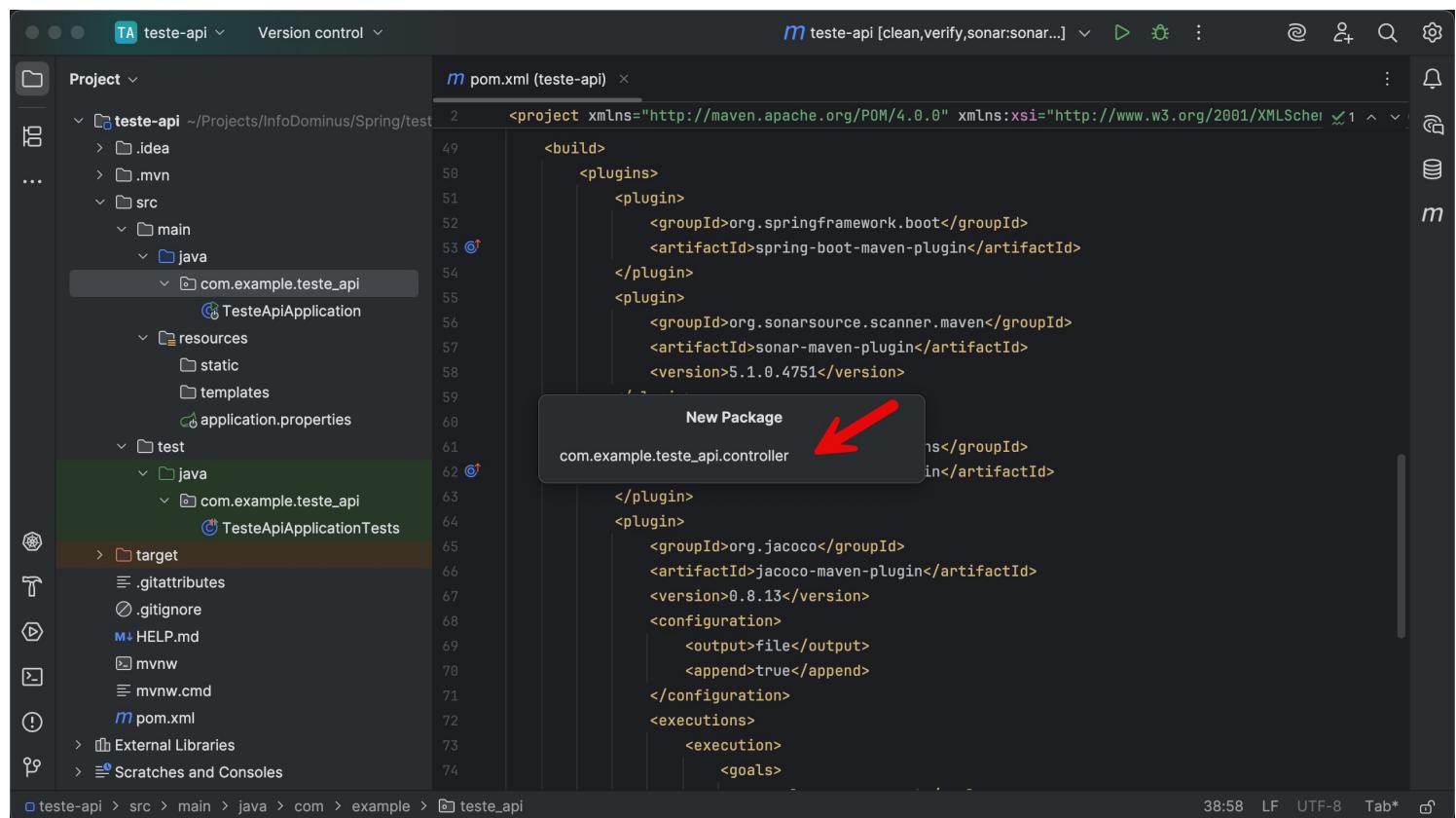
Vamos criar uma package “*controller*” onde ficará a nossa classe que receberá as requisições:



# Spring Boot

## Criando uma Aplicação REST API

Colocar no nome da package “*controller*”.



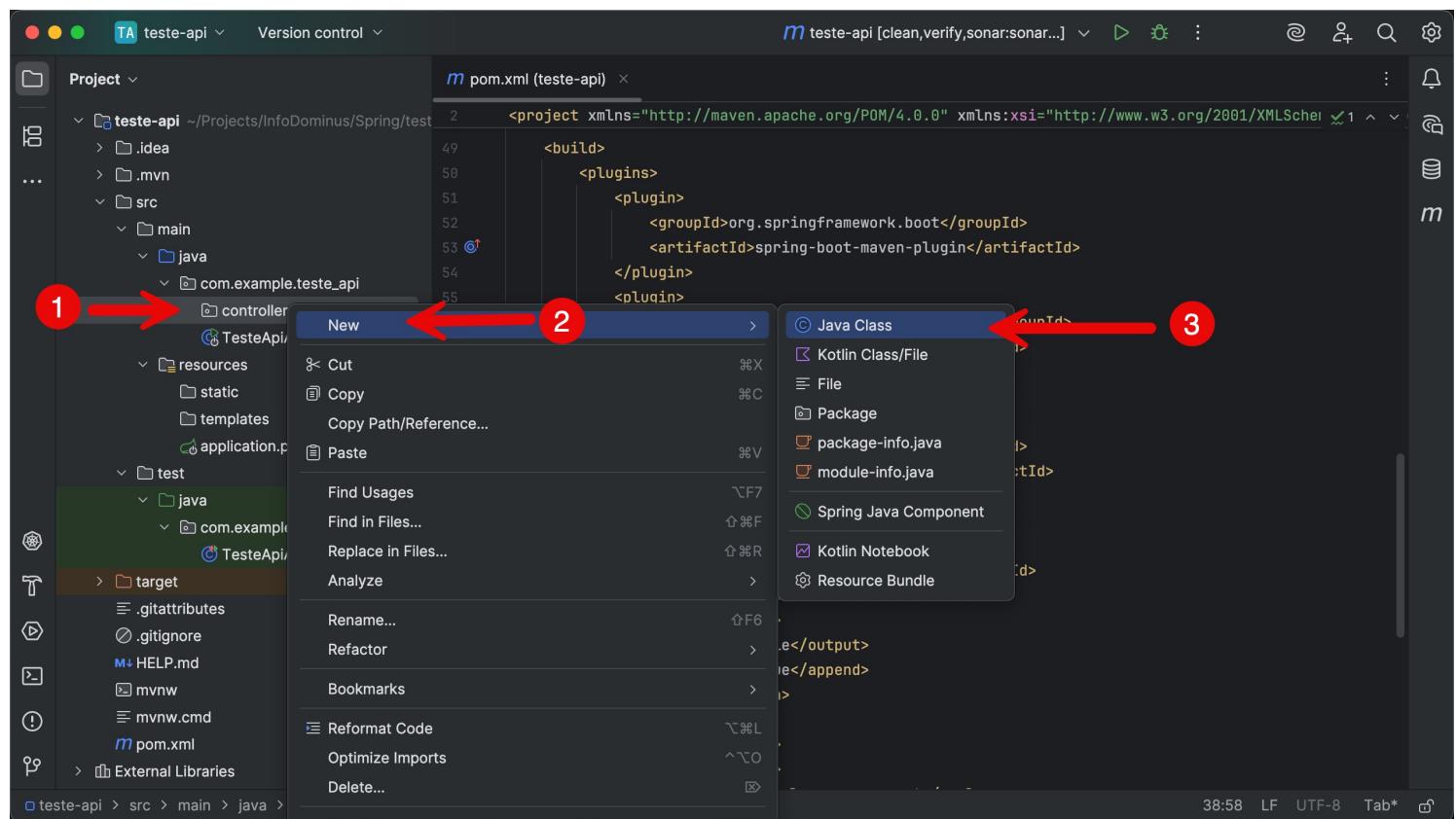
The screenshot shows the IntelliJ IDEA interface with the project 'teste-api' open. The left sidebar displays the project structure, including the 'src' directory with 'main' and 'test' sub-directories, each containing 'java' and 'resources' sub-directories. The 'pom.xml' file is open in the main editor window. A 'New Package' dialog box is overlaid on the code, showing the package name 'com.example.teste\_api.controller'. A red arrow points from the 'New Package' dialog to the 'controller' part of the package name in the dialog. The code in the editor includes Maven build configurations for Spring Boot and SonarQube.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
      <plugin>
        <groupId>org.sonarsource.scanner.maven</groupId>
        <artifactId>sonar-maven-plugin</artifactId>
        <version>5.1.0.4751</version>
      </plugin>
      <plugin>
        <groupId>org.jacoco</groupId>
        <artifactId>jacoco-maven-plugin</artifactId>
        <version>0.8.13</version>
        <configuration>
          <output>file</output>
          <append>true</append>
        </configuration>
        <executions>
          <execution>
            <goals>
```

# Spring Boot

## Criando uma Aplicação REST API

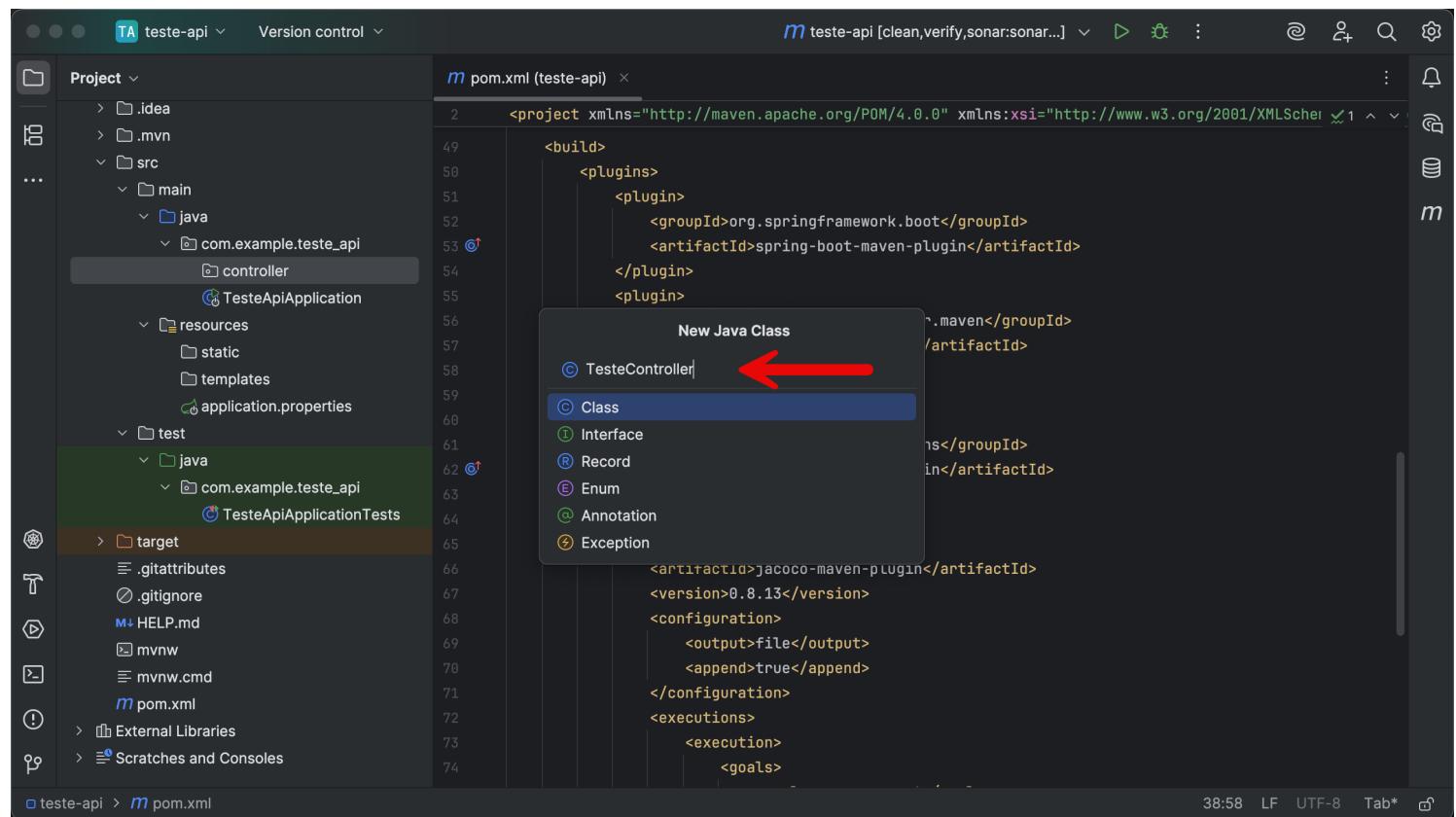
Vamos criar a nossa classe *Controller* dentro da package “controller”.



# Spring Boot

## Criando uma Aplicação REST API

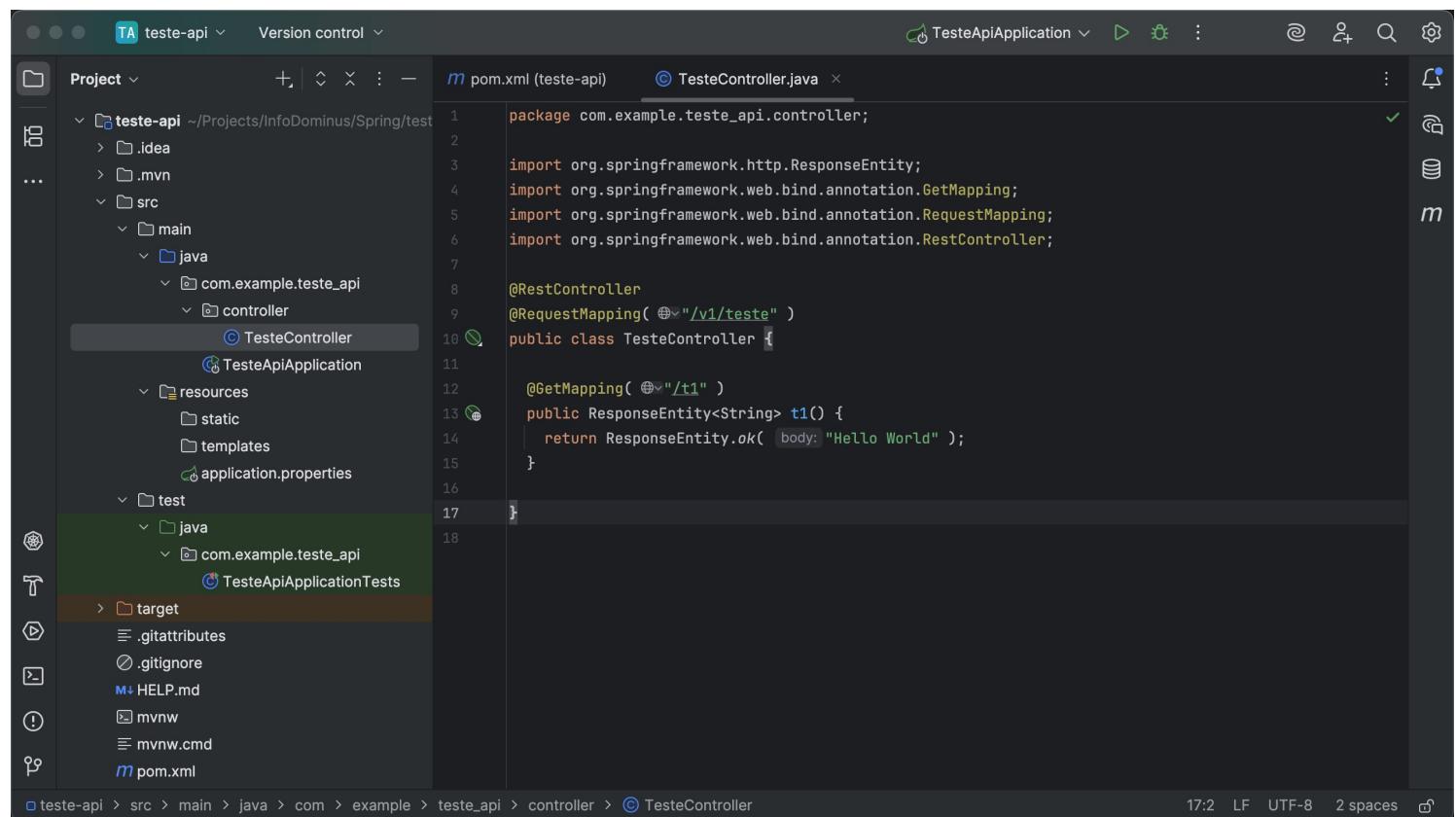
Colocar o nome da Classe como *TesteController*.



# Spring Boot

## Criando uma Aplicação REST API

Vamos anotar a classe com as tags: `@RestController` e `@RequestMapping` e criar um endpoint t1 que retorna o Texto “Hello World”.



The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the project structure under "teste-api". It includes a .idea folder, .mvn folder, src directory (containing main and test), resources (static and templates), application.properties, and target folder.
- Code Editor:** The file `TesteController.java` is open. The code defines a `TesteController` class with a `@GetMapping("/t1")` method that returns a ResponseEntity with the body "Hello World".
- Pom.xml:** The pom.xml file is also visible in the editor.
- Status Bar:** At the bottom right, it shows the time as 17:2, LF, UTF-8, 2 spaces, and a file icon.

# Spring Boot

## Criando uma Aplicação REST API

Com isso já podemos rodar o projeto e acessar o *Swagger*.

The screenshot shows the IntelliJ IDEA IDE interface. The left sidebar displays the project structure for a 'teste-api' project, which includes a 'src' folder containing 'main' and 'test' packages, and a 'target' folder. The 'TesteController.java' file is open in the main editor area. The code defines a controller with a single endpoint that returns 'Hello World'. Annotations 1 and 2 are overlaid on the interface: annotation 1 points to the 'TesteApiController' entry in the navigation bar, and annotation 2 points to the 'TesteApiControllerTests' entry in the navigation bar.

```
package com.example.teste_api.controller;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
@RestController  
@RequestMapping( "/v1/teste" )  
public class TesteController {  
  
    @GetMapping( "/t1" )  
    public ResponseEntity<String> t1() {  
        return ResponseEntity.ok( "Hello World" );  
    }  
}
```

# Spring Boot

## Criando uma Aplicação REST API

No browser acesse a URL:

<http://localhost:8080/swagger-ui/index.html>

The screenshot shows a web browser window titled "Swagger UI". The address bar contains the URL "localhost:8080/swagger-ui/index.html#/". A red arrow with the number "1" points to the address bar. The main content area displays the "OpenAPI definition" page, which is version v0 OAS 3.1. It includes a "Servers" dropdown set to "http://localhost:8080 - Generated server url". Below this, there is a section for "teste-controller" with a "GET /v1/teste/t1" endpoint. A red arrow with the number "2" points to this endpoint.

# Spring Boot

## Criando uma Aplicação REST API

Clique em “*Try it out*” e depois em “*Execute*” para realizar a chamada.

The screenshot shows the Swagger UI interface for a Spring Boot application. The URL in the browser is `localhost:8080/swagger-ui/index.html#/teste-controller/t1`. The page displays an OpenAPI definition (v0 OAS 3.1) for a `teste-controller`. A specific endpoint `/v1/teste/t1` is selected, showing its details. The endpoint is a `GET` method with no parameters. Below the method details, there are two buttons: `Try it out` and `Execute`. Red arrows with numbers 1 and 2 point to these buttons respectively, indicating the steps to perform the API call.

# Spring Boot

## Criando uma Aplicação REST API

O Resultado aparecerá embaixo, e então nossa API REST estará funcionando.

The screenshot shows the Swagger UI interface for a Spring Boot application. The URL in the browser is `localhost:8080/swagger-ui/index.html#/teste-controller/t1`. The request method is `GET` to the endpoint `/v1/teste/t1`. The response body is displayed as `Hello World`, with a red arrow pointing to it. The response headers are shown below:

```
connection: keep-alive
content-length: 11
content-type: text/plain;charset=UTF-8
date: Thu,29 May 2025 02:35:50 GMT
keep-alive: timeout=60
```



# Spring Boot

## Teste Unitário

# Spring Boot

## Criando o Teste Unitário

Para simplificar vamos utilizar a classe de teste criada automaticamente pelo *Spring Initializr* devendo só complementar com o código abaixo (OBS: Não esqueça dos imports).

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the project structure under "teste-api". The "test/java/com.example.teste\_api/TesteApiControllerTests.java" file is selected, indicated by a red circle labeled "1".
- Code Editor:** The "TesteApiControllerTests.java" tab is active, showing the generated test code:

```
package com.example.teste_api;

import com.example.teste_api.controller.TesteController;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
import org.mockito.InjectMocks;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;

@SpringBootTest
class TesteApiControllerTests {

    @InjectMocks
    private TesteController controller;

    @Test
    void contextLoads() {
        ResponseEntity<String> responseEntity = controller.t1();
        Assertions.assertEquals(HttpStatus.OK, responseEntity.getStatusCode());
        Assertions.assertEquals("Hello World", responseEntity.getBody());
    }
}
```
- Annotations:** Three red arrows with circles numbered 2, 3, and 4 point to specific parts of the code:
  - Arrow 2 points to the line `@InjectMocks`.
  - Arrow 3 points to the line `private TesteController controller;`.
  - Arrow 4 points to the line `void contextLoads()`.

# Spring Boot

## Criando o Teste Unitário

Agora vamos selecionar o runner do *Maven* que criamos anteriormente e executar. A execução deve ocorrer sem erros.

The screenshot shows the IntelliJ IDEA interface with the following highlights:

- A red arrow points from the top center to the 'Run' button in the toolbar.
- A red circle with the number '2' is placed on the 'Run' button.
- A red arrow points from the bottom right to the status bar message 'Process finished with exit code 0'.
- A red circle with the number '3' is placed next to the status bar message.

The code editor displays the `TesteApiControllerTests.java` file:

```
package com.example.teste_api;
import ...
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.ResultActions;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;

@SpringBootTest
public class TesteApiControllerTests {
    private MockMvc mockMvc;

    @MockBean
    private TesteController testeController;

    @Test
    void contextLoads() {
        mockMvc.perform(get("/"))
            .andExpect(status().isOk())
            .andExpect(content().string("Hello World"));
    }
}
```

# Spring Boot

## Criando o Teste Unitário

Acessando o Sonar na URL:

<http://localhost:9000/>

Veremos que tivemos 100% de cobertura.

The screenshot shows the SonarQube dashboard for the 'teste-api' project. The main header includes the SonarQube logo, navigation links for Projects, Issues, Rules, Quality Profiles, Quality Gates, Administration, and More, along with search and filter options. Below the header, the project navigation bar shows 'teste-api / main'. The main content area is titled 'main' and displays analysis details: '105 Lines of Code' and 'Version 0.0.1-SNAPSHOT'. A green 'Passed' status is shown under 'Quality Gate'. The 'Overall Code' tab is selected, showing metrics for Security (0 Open issues), Reliability (0 Open issues), and Maintainability (0 Open issues). In the 'Coverage' section, it shows '100%' coverage, with a note 'On 2 lines to cover.' A red arrow points to the '100%' coverage value. Other sections include 'Accepted issues' (0) and 'Duplications' (0.0%). The bottom of the page includes footer links for Quality and Test.

# Spring Boot

## Criando o Teste Unitário

Ao clicar no “100%” mostrará as classes e suas coberturas.

The screenshot shows the SonarQube interface for a project named 'teste-api'. The 'Measures' tab is selected. On the left, there's a sidebar with sections like Project Overview, Security, Reliability, Maintainability, and Security Review. The main area displays coverage details for a file named 'TesteController.java'. The coverage is shown as 100%, with 0 uncovered lines and 100 uncovered conditions. There are buttons for 'Select files' and 'Navigate'.

	Coverage	Uncovered Lines	Uncovered Conditions
src/main/java/com/example/teste_api/controller/TesteController.java	100%	0	100



# Esclarecimentos Finais

# Esclarecimentos Finais

Espero que este e-Book lhe ajude a melhorar e aperfeiçoar o seu código de forma mais rápida e eficiente.

Contudo alertamos que pode ocorrer de tentar colocar as dicas desse e-Book em um projeto já existente e não funcionar a cobertura de Código.

E isso ocorre, por exemplo, devido aos conflitos de bibliotecas e plugins. A dica é tentar retirar tudo do seu antigo projeto e colocar somente o que foi mostrado aqui, e ir incrementando aos poucos para tentar descobrir qual biblioteca ou plugin está conflitando e que não esteja deixando realizar os testes e a cobertura de código.

# Esclarecimentos Finais

O projeto mínimo mostrado aqui no e-book ficará disponível no github no seguinte link:

<https://github.com/alberto-yano/dio-lab-ebook>

E o vídeo de apresentação no link:

<https://youtu.be/86Zb8l8n6Mc>

Seguem os meus contatos para o caso de dúvidas e sugestões:

<https://www.linkedin.com/in/albertoyano/>

<https://github.com/alberto-yano>

[https://gitlab.com/alberto\\_yano](https://gitlab.com/alberto_yano)

