

# Selenium

- **Basic program**
- **WebDriverManager**

WebDriverManager is a popular third-party library used with Selenium to simplify the management of web driver binaries, such as ChromeDriver, GeckoDriver, and others. It offers several advantages and conveniences for handling web driver setup:

**Automatic Driver Downloads:** WebDriverManager can automatically download the appropriate web driver binary for the web browser you intend to use. This eliminates the need for manually downloading and managing driver binaries, which can be error-prone and time-consuming.

**Driver Version Compatibility:** It ensures that the downloaded driver version is compatible with the browser version you have installed. This helps avoid compatibility issues between the browser and driver.

**Simplified Configuration:** WebDriverManager simplifies the configuration of web drivers in your Selenium project. You don't need to specify the exact path to the driver executable; WebDriverManager handles it for you.

**Cross-Platform Compatibility:** It works across different platforms (Windows, macOS, Linux) and supports various web browsers (Chrome, Firefox, Edge, etc.).

- **close() vs quite()**

The **close()** method is used to close the currently focused browser window or tab. The **quit()** method is used to gracefully terminate the WebDriver session and close all open browser windows or tabs associated with that session.

- **Invoke browser via testng.xml**
- **Selenium Options**

In Selenium, the "Options" class refers to a set of classes and interfaces that provide various configuration and customization options for different aspects of web browsers, such as browser settings, capabilities, and behavior. These options are typically used when initializing a WebDriver instance to configure how the browser should behave during test automation.

```
ChromeOptions options = new ChromeOptions();  
options.addArguments("--start-maximized"); // Maximize the browser window
```

```
options.addArguments("--disable-extensions"); // Disable extensions
WebDriver driver = new ChromeDriver(options);
```

```
FirefoxOptions options = new FirefoxOptions();
options.setHeadless(true); // Run Firefox in headless mode
options.addPreference("network.proxy.type", 1); // Set proxy settings
WebDriver driver = new FirefoxDriver(options);
```

```
EdgeOptions options = new EdgeOptions();
options.setPageLoadStrategy("normal"); // Set page load strategy
options.setCapability("ms:edgeChromium", true); // Use Chromium-based Edge
WebDriver driver = new EdgeDriver(options);
```

## - Waits in selenium

**Thread.sleep()** is a static method in Java used for pausing the execution of a program for a specified period, often used in Selenium testing to introduce delays. **implicitlyWait** is a setting in Selenium that tells the WebDriver to wait for a certain amount of time when trying to locate elements before throwing an exception. **WebDriverWait is an explicit wait** in Selenium, which allows you to wait for specific conditions (e.g., element visibility) to be met before proceeding with test execution. **Wait with FluentWait** is another explicit wait in Selenium that offers more flexible polling and exception handling options. Lastly, **"PageLoad timeout"** is a setting to define the maximum time a WebDriver should wait for a page to load completely, useful for ensuring that web pages are fully loaded before interacting with them during testing, and it can be set via the WebDriver instance.

- `pageLoadTimeOut()`
- `implicitlyWait()`
- Explicit Wait - `WebDriverWait`
- FluentWait - `Wait`

## - Page object model in Selenium

The Page Object Model (POM) is a design pattern used in Selenium automation testing to enhance the maintainability and readability of test code. It involves organizing your test automation code into separate classes or components, each representing a different web page or part of the application being tested.

### What is the Page Object Model (POM)?

**Page Objects:** In POM, each web page or component of the application is represented by a separate class known as a "Page Object." These classes encapsulate the interaction with elements on the page, such as locating and interacting with web elements like buttons, text fields, and links.

**Separation of Concerns:** POM promotes a clear separation of concerns between test code and page interaction code. Test scripts are written independently of the specific details of the web page, making them more maintainable and easier to read.

**Reusability:** Page Objects can be reused across multiple test cases, reducing code duplication and making it easier to update the application's locators or interaction methods in a single place.

### **Why Use the Page Object Model in Selenium?**

**Enhanced Maintainability:** POM reduces code duplication and promotes a modular approach, making it easier to update the test code when there are changes in the application's UI or functionality. Changes are typically confined to the Page Object classes, minimizing the impact on test scripts.

**Improved Readability:** POM improves the readability of test scripts by abstracting the low-level interactions with web elements. Test cases become more self-explanatory and easier to understand, even for non-technical team members.

**Reusability:** Page Objects can be reused across multiple test scripts or projects, promoting code reusability and reducing the effort required to write new tests.

**Maintenance Efficiency:** When the structure or locators of web elements change, you only need to update the relevant Page Object class rather than modifying every test case. This streamlines maintenance efforts and reduces the risk of introducing errors during updates.

**Collaboration:** POM facilitates collaboration between automation engineers and manual testers or other team members since the code is more organized and less complex.

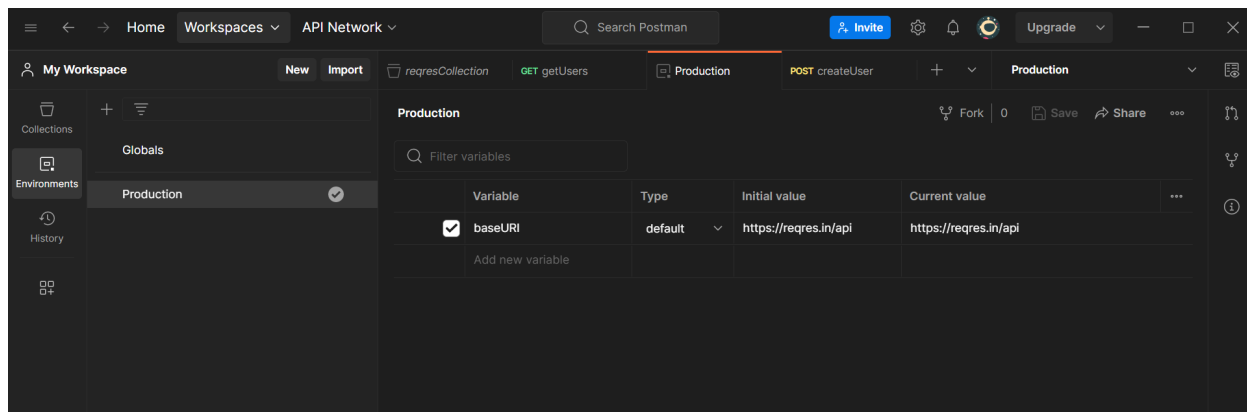
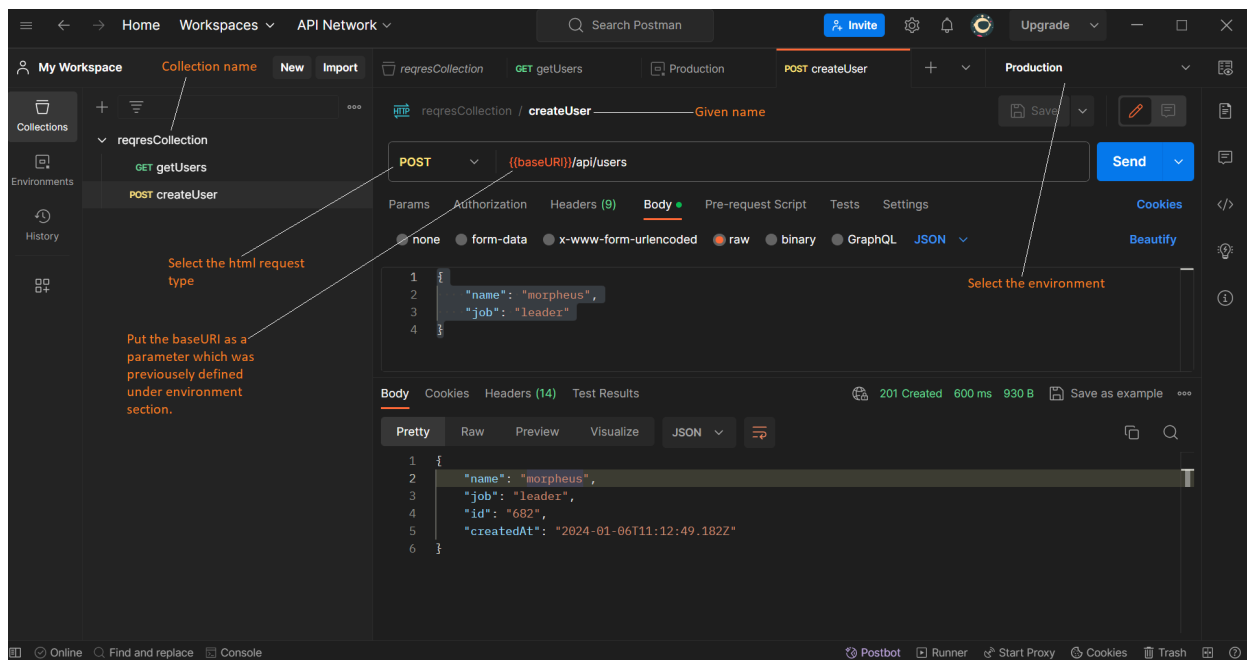
**Scalability:** As your test suite grows, POM helps maintain a structured and scalable framework that can handle a large number of test cases and web pages efficiently.

- What and Why POM
- Basic program structure of POM
- Use of PageFactory with different return types
- Use aggregation in PO
- Data driven(optional)

## **API testing**

### **Handling API via postman**

- Download postman from <https://www.postman.com/>.
- You can use <https://reqres.in/> as example api provider



## How to use Rest assured

Install rest-assured maven library as dependency. It's better to use TestNG along with.

MVN REPOSITORY


[Categories](#) | [Popular](#)








---

Repository

**Found 11068 results**

---

Sort: [relevance](#) | [popular](#) | [newest](#)

            	<div style="margin-bottom: 10px;"><b>Central</b></div> <div style="margin-bottom: 10px;"><b>Sonatype</b></div> <div style="margin-bottom: 10px;"><b>Spring Plugins</b></div> <div style="margin-bottom: 10px;"><b>Spring Lib M</b></div> <div style="margin-bottom: 10px;"><b>JCenter</b></div> <div style="margin-bottom: 10px;"><b>Boundless</b></div> <div style="margin-bottom: 10px;"><b>JBoss Releases</b></div>	<div>8.0k</div> <div>1.9k</div> <div>1.1k</div> <div>1.0k</div> <div>407</div> <div>406</div> <div>306</div>
---	--	--

See the class example given for your references.