[CS304] Lab12. Automated Testing for Web Applications using Selenium

> Author: Yida Tao

Introduction

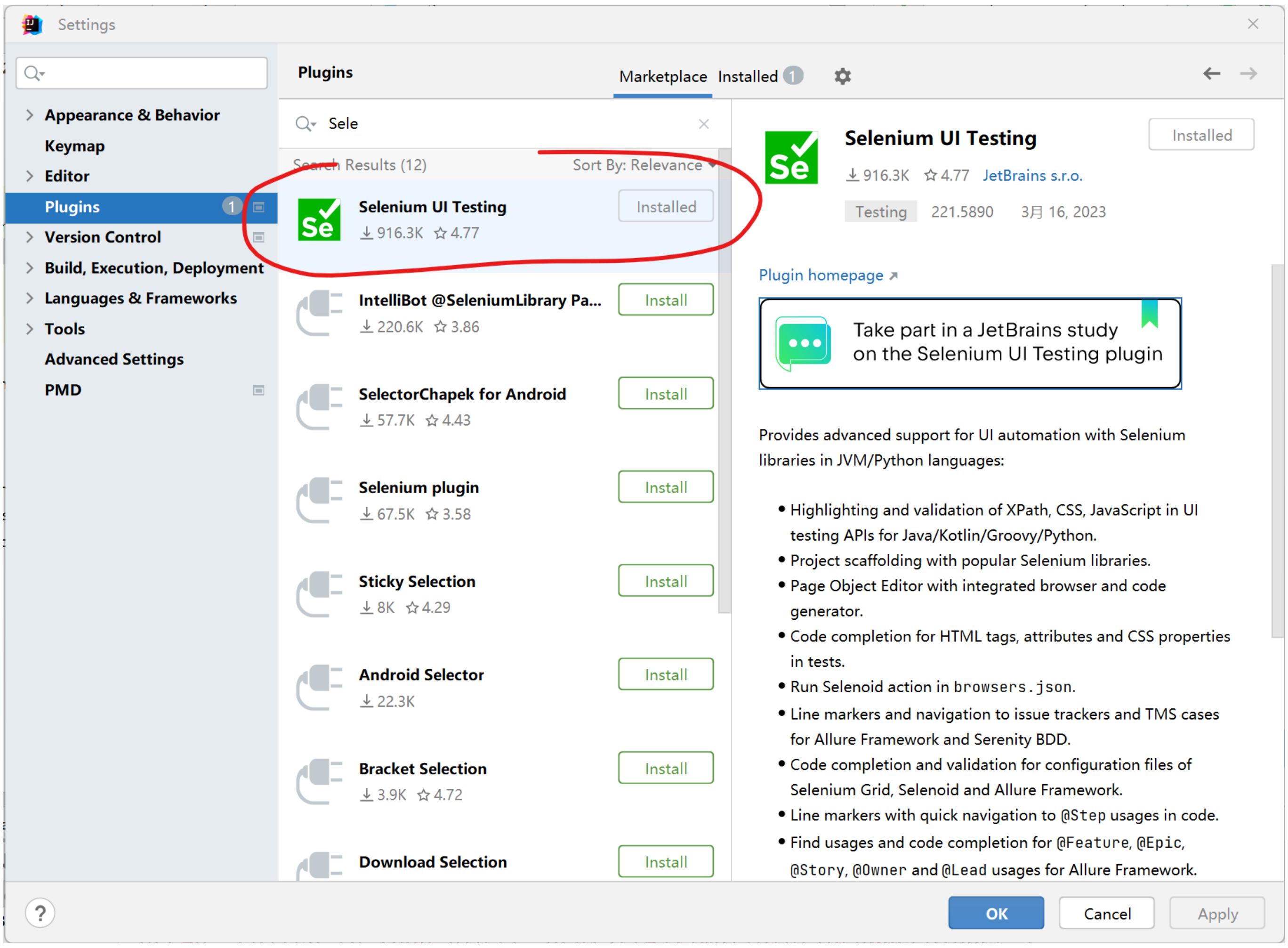
Selenium is an open source umbrella project for a range of tools and libraries aimed at supporting *browser automation*. It provides a playback tool for authoring functional tests across most modern web browsers.

At the core of Selenium is *Selenium WebDriver*, an interface to write instructions that work interchangeably across browsers. Basically, the WebDriver accepts commands, directly starts a browser instance, and send the commands to the browser.

Using the Selenium Plugin in IntelliJ IDEA

We could use Selenium in IntelliJ IDEA following the below steps.

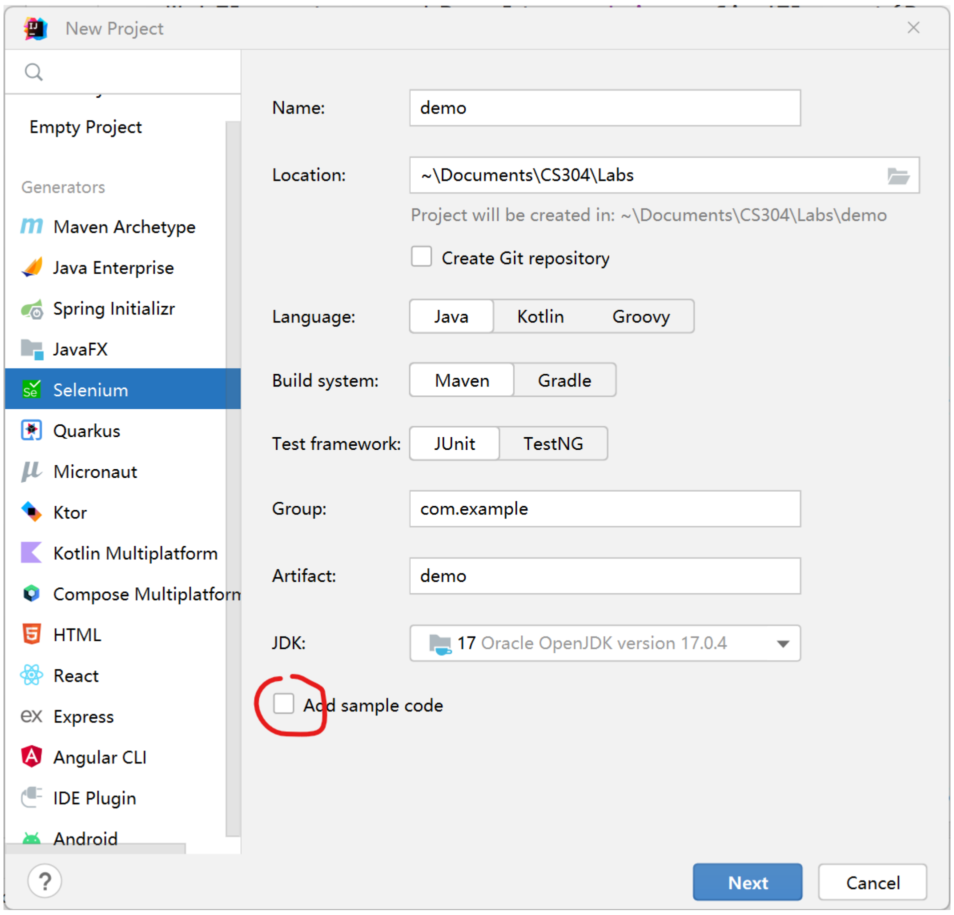
First, install the Selenium UI Testing plugin.


Settings
Appearance & Behavior
Keymap
Editor
Plugins
Version Control
Build, Execution, Deployment
Languages & Frameworks
Tools
Advanced Settings
PMD
Plugins
Marketplace
Installed
Sele
Search Results (12)
Sort By: Relevance
Selenium UI Testing ⤓916.3K ☆4.77 Installed
IntelliBot @SeleniumLibrary Pa... ⤓220.6K ☆3.86 Install
SelectorChapek for Android ⤓57.7K ☆4.43 Install
Selenium plugin ⤓67.5K ☆3.58 Install
Sticky Selection ⤓8K ☆4.29 Install
Android Selector ⤓22.3K Install
Bracket Selection ⤓3.9K ☆4.72 Install
Download Selection Install
Selenium UI Testing
Installed
⤓916.3K ☆4.77 JetBrains s.r.o.
Testing 221.5890 3月 16, 2023
Plugin homepage ↗
Take part in a JetBrains study on the Selenium UI Testing plugin
OK
Cancel
Apply

Provides advanced support for UI automation with Selenium libraries in JVM/Python languages:

- Highlighting and validation of XPath, CSS, JavaScript in UI testing APIs for Java/Kotlin/Groovy/Python.
- Project scaffolding with popular Selenium libraries.
- Page Object Editor with integrated browser and code generator.
- Code completion for HTML tags, attributes and CSS properties in tests.
- Run Selenoid action in `browsers.json`.
- Line markers and navigation to issue trackers and TMS cases for Allure Framework and Serenity BDD.
- Code completion and validation for configuration files of Selenium Grid, Selenoid and Allure Framework.
- Line markers with quick navigation to `@Step` usages in code.
- Find usages and code completion for `@Feature`, `@Epic`, `@Story`, `@Owner` and `@Lead` usages for Allure Framework.

Create a new project. After successful plugin installation, we could select "Selenium" as the project type. *Uncheck* "Add sample code" since we'll create our own sample code in this tutorial.


New Project

Empty Project

Generators
- Maven Archetype
- Java Enterprise
- Spring Initializr
- JavaFX
- Selenium
- Quarkus
- Micronaut
- Ktor
- Kotlin Multiplatform
- Compose Multiplatform
- HTML
- React
- Express
- Angular CLI
- IDE Plugin
- Android

Name: demo

Location: ~\Documents\CS304\Labs

Project will be created in: ~\Documents\CS304\Labs\demo

☐ Create Git repository

Language: Java | Kotlin | Groovy

Build system: Maven | Gradle

Test framework: JUnit | TestNG

Group: com.example

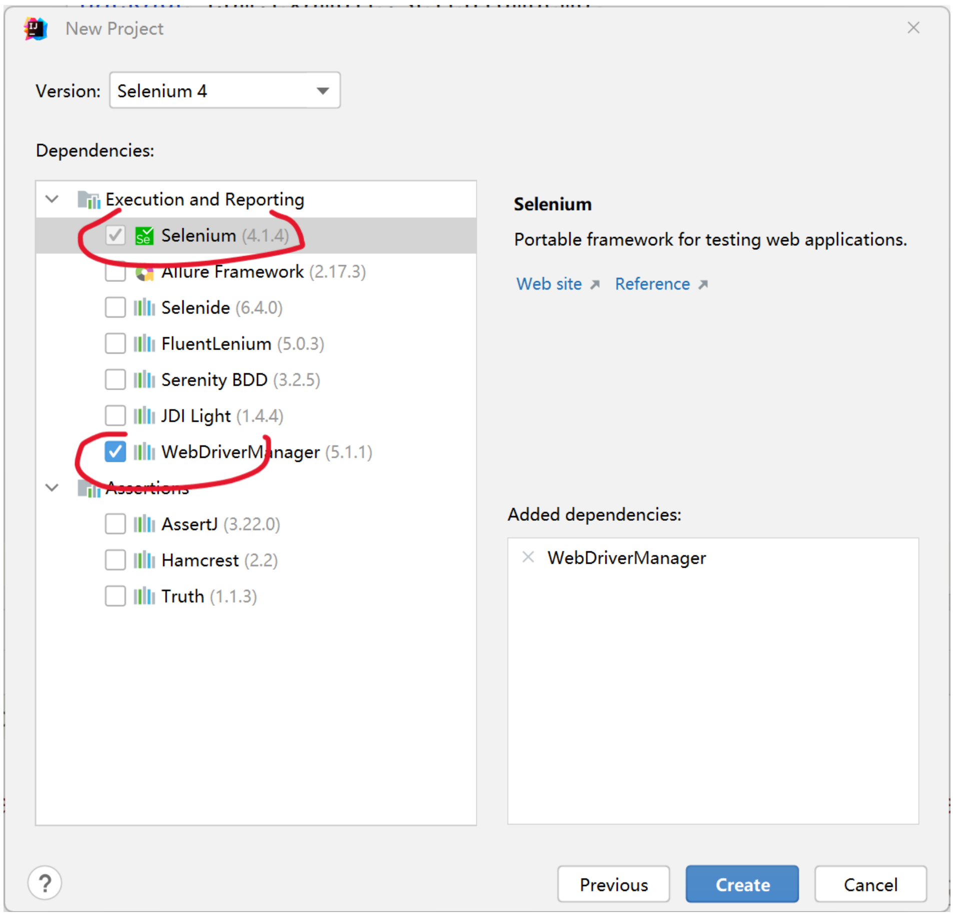
Artifact: demo

JDK: 17 Oracle OpenJDK version 17.0.4

☐ Add sample code

Next | Cancel

Click "Next". Select "Selenium" as well as "WebDriverManager", so that these two dependencies will be automatically added to our project (or you can add the dependency to `pom.xml` manually later). Note that we use `WebDriverManager` to automatically download corresponding drivers for our specified browsers.


New Project

Version: Selenium 4

Dependencies:

- Execution and Reporting
  - ✓ Selenium (4.1.4)
  - ☐ Allure Framework (2.17.3)
  - ☐ Selenide (6.4.0)
  - ☐ FluentLenium (5.0.3)
  - ☐ Serenity BDD (3.2.5)
  - ☐ JDI Light (1.4.4)
  - ✓ WebDriverManager (5.1.1)
- Assertions
  - ☐ AssertJ (3.22.0)
  - ☐ Hamcrest (2.2)
  - ☐ Truth (1.1.3)

Selenium

Portable framework for testing web applications.

Web site ↗ Reference ↗

Added dependencies:

WebDriverManager

Previous | Create | Cancel

Writing Test Code

Create `BingTest.java` as below. The code sets up the webdriver and the website we'll test (i.e., Bing). We use Chrome in this tutorial, you could set up other webdrivers based on the browsers you typically use.

```java
public class BingTest {
    private WebDriver driver;

    @BeforeAll
    public static void setupDriver() {
        WebDriverManager.chromedriver().setup();
    }

    @BeforeEach
    public void setUp() {
        ChromeOptions options = new ChromeOptions();
        options.addArguments("--remote-allow-origins=*");
        driver = new ChromeDriver(options);

        driver.get("https://www.bing.com/");
    }

    @AfterEach
    public void tearDown() {
        driver.quit();
    }
}
```

Test Case 1

Let's first create a test to test the `search bar` function. Basically, we use `driver.findElement` to find the element we want to test, by specifying the element's HTML id, name, path or other . We then use Selenium's API like `sendKeys` or `submit` to simulate user actions.

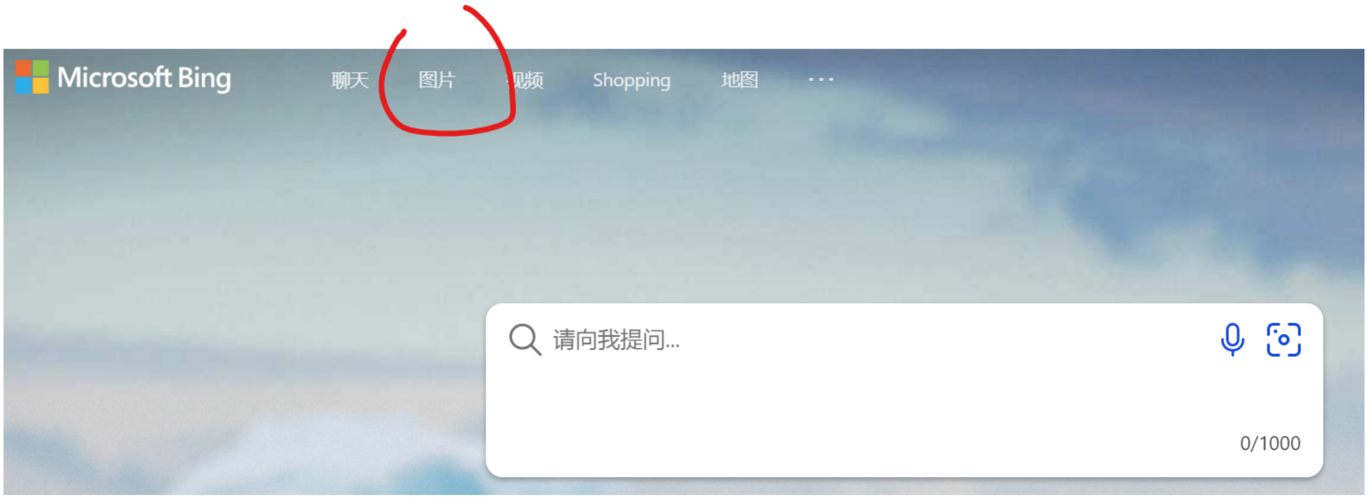
```java
@Test
void search(){
    // Find the search input box element
    WebElement searchBox = driver.findElement(By.name("q"));

    // Enter a search query
    searchBox.sendKeys("Selenium testing");
    // Submit
    searchBox.submit();

    // Verify that the search result is displayed
    WebElement searchResults = driver.findElement(By.id("b_content"));
    assertTrue(searchResults.isDisplayed());
}
```

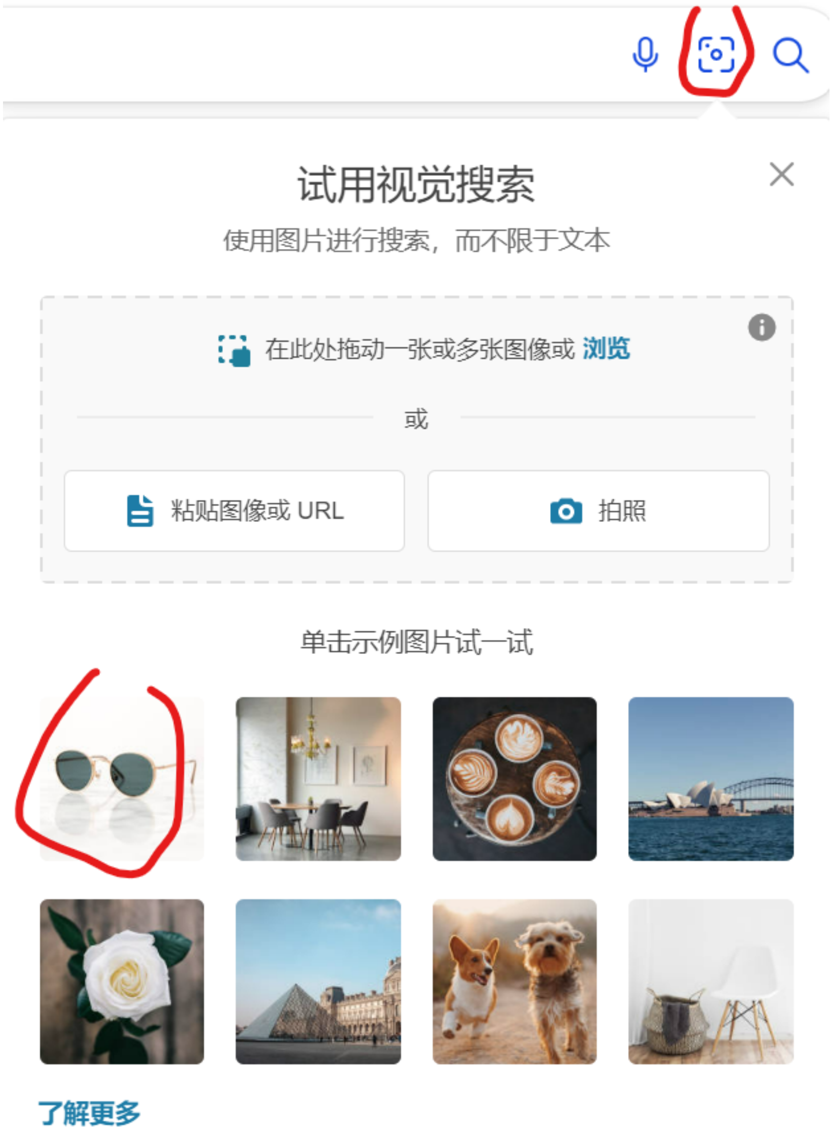
Now, execute this test to see what happens.

Test Case 2

Create the second test case as follows, which simulates a series of actions:

1. Click "图片" on the main page of Bing.
2. Click the "camera icon" (使用图像搜索).
3. Click a sample image "墨镜" and search.



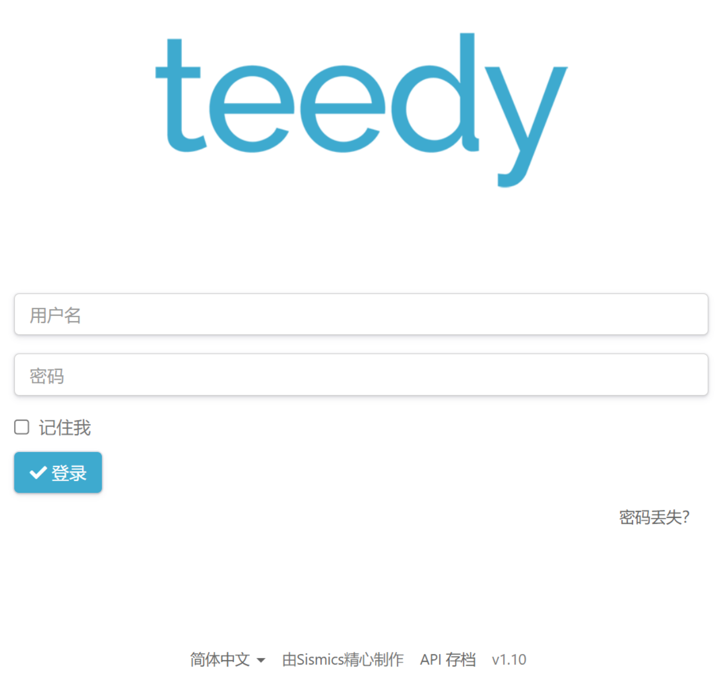

试用视觉搜索
使用图片进行搜索，而不限于文本
在此处拖动一张或多张图像或 浏览
或
粘贴图像或 URL
拍照
单击示例图片试一试
了解更多

```
@Test
void searchByImage() throws InterruptedException {

    // Find the "Images" link and click it (wait for it to be fully loaded)
    WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
    WebElement imagesLink =
wait.until(ExpectedConditions.elementToBeClickable(By.linkText("图片")));
    imagesLink.click();

    // Find the "Camera" icon and click it
    WebElement cameraIcon = driver.findElement(By.id("sb_sbip"));
    cameraIcon.click();

    // Click a sample image
    WebElement imageInput = wait.until(
            ExpectedConditions.elementToBeClickable(By.cssSelector("img[alt='墨
镜']")));
    imageInput.click();

    Thread.sleep(10000);

}
```

Execute this second test to see what happens.

Testing Teedy

You could use Selenium to automatically test Teedy by simulating client login actions.



First, set the URL correctly for the driver:

```
driver.get("http://localhost:8080/docs-web/src/");
```

The following code simulates a login test:

```
@Test
void login() throws InterruptedException {

    // Find the input box for username and password
    WebElement idInput = driver.findElement(By.id("inputUsername"));
    WebElement pwdInput = driver.findElement(By.id("inputPassword"));

    // Fill in the input box
    idInput.sendKeys("admin");
    pwdInput.sendKeys("admin");

    // Click login
    WebElement button = driver.findElement(By.xpath("//button[@ng-
click='login()']"));
    button.click();

    Thread.sleep(8000);

}
```

Now, start Teedy server as we did in the second lab. Then execute the test to simulate the login action.

You could explore more capabilities of Selenium by yourself.