



# Web UI Testing Using Selenium

# Selenium Overview

Selenium is not just one tool or API but it composes many tools:

- **WebDriver** - If you are beginning with desktop website or mobile website test automation, then you are going to be using WebDriver APIs. WebDriver uses browser automation APIs provided by browser vendors to control browser and run tests. This is as if a real user is operating the browser. Since WebDriver does not require its API to be compiled with application code, it is not intrusive.
- **IDE - IDE (Integrated Development Environment)** is the tool you use to develop your Selenium test cases. It's an easy-to-use Chrome and Firefox extension and is an efficient way to develop test cases. It records the users' actions in the browser for you, using existing Selenium commands, with parameters defined by the context of that element. This is not only a time-saver but also an excellent way of learning Selenium script syntax.

# Selenium Grid

- **Grid** - **Selenium Grid** allows you to run test cases in **different machines** across **different platforms**. The control of triggering the test cases is on the local end, and when the test cases are triggered, they are automatically executed by the remote end. Generally speaking, there's two reasons why you might want to use Grid:
  - To run your tests against multiple browsers, multiple versions of browser, and browsers running on different operating systems.
  - To reduce the time it takes for the test suite to complete a test pass.
- After the development of the **WebDriver** tests, you may face the need of running your tests on **multiple browser and operating system combinations**. This is where **Grid** comes into the picture.

# Selenium Grid - II

- **Grid** is used to speed up the execution of a test pass by using multiple machines to **run tests in parallel**. For example, if you have a **suite of 100 tests**, but you set up **Grid** to support **4 different machines** (VMs or separate physical machines) to run those tests, your test suite will complete in (roughly) one-fourth the time as it would if you ran your tests sequentially on a single machine. For large test suites, and long-running test suite such as those performing large amounts of data-validation, this can be time-saver.
- **Grid** is also used to support running tests against multiple runtime environments, specifically, against different browsers at the same time – e.g. **machine 1** has Internet **Explorer 8**, **machine 2**, Internet Explorer 9, **machine 3** the latest **Chrome**, and **machine 4** the latest **Firefox**. When the test suite is run, Selenium-Grid receives each test-browser combination and assigns each test to run against its required browser.

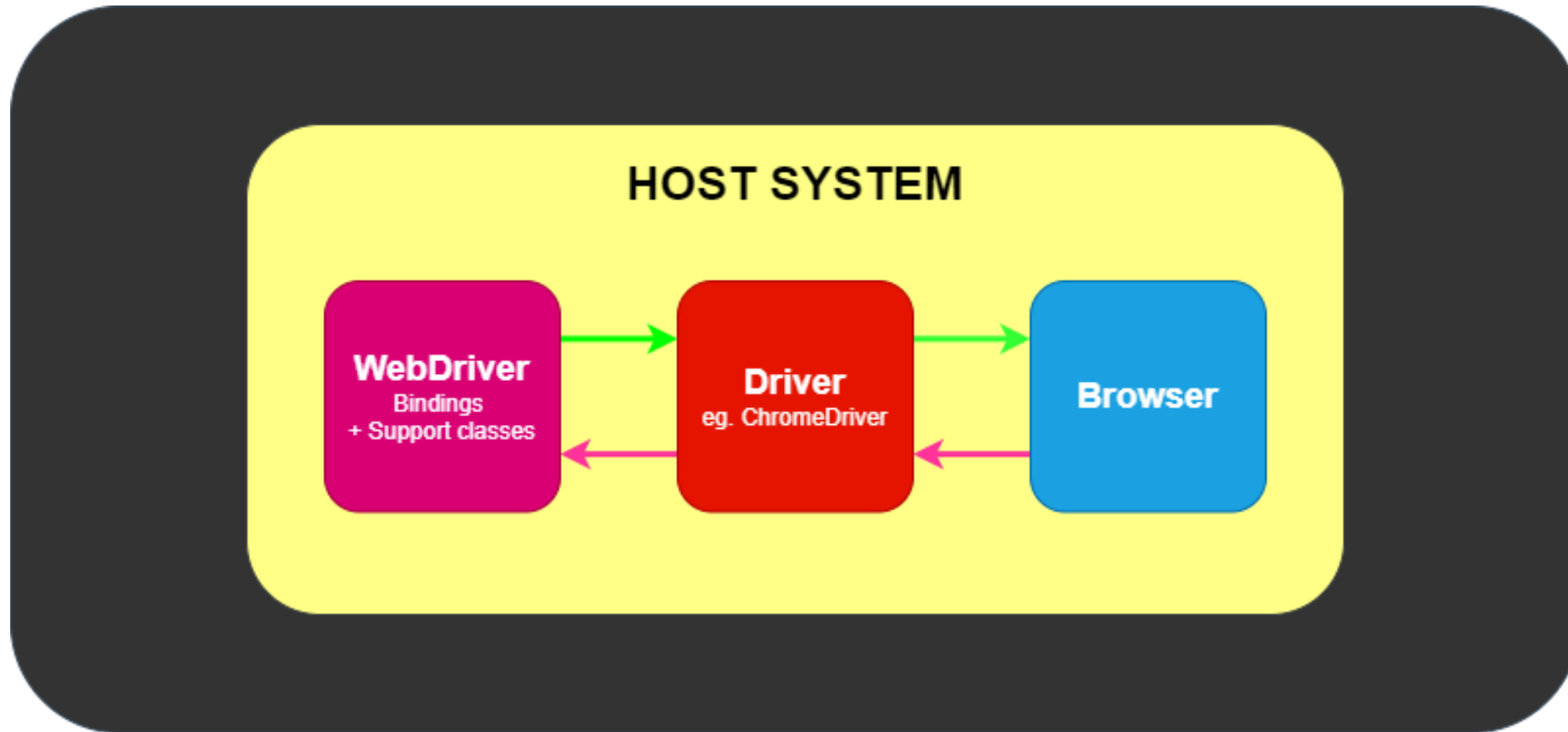
# Selenium Grid Advantages

- Central entry point for all tests
- Management and control of the nodes / environment where the browsers run
- Scaling
- Running tests in parallel
- Cross-platform testing
- Load balancing

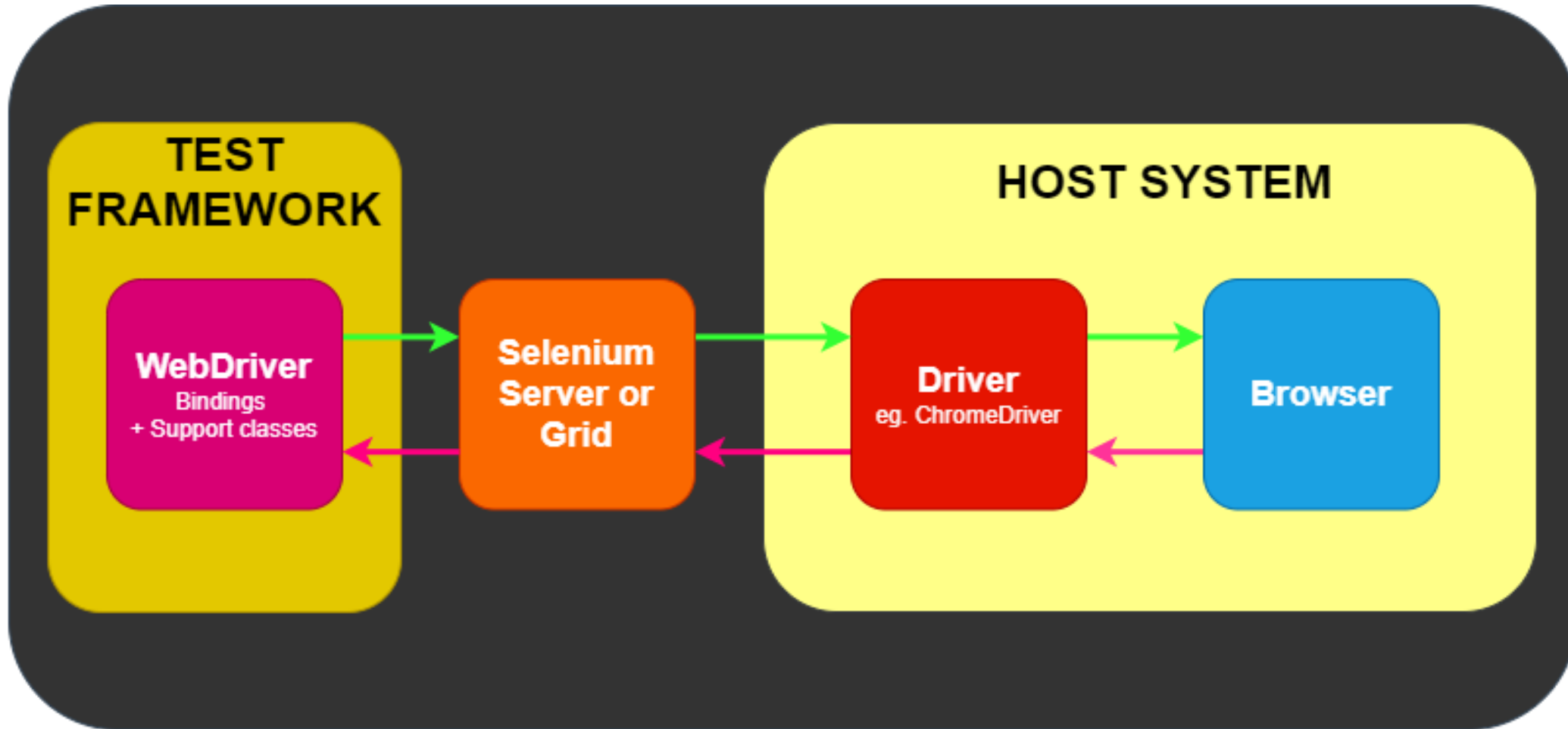
# Terminology

- API – a set of “commands” you use to manipulate WebDriver.
- Library - code module which contains the APIs and the code necessary to implement them. Libraries are specific to each language binding, eg .jar files for Java, .dll files for .NET, etc.
- Driver - responsible for controlling the actual browser. Most drivers are created by the browser vendors themselves. Drivers are generally executable modules that run on the system with the browser itself, not on the system executing the test suite. (Although those may be the same system.) NOTE: Some people refer to the drivers as proxies.
- Framework: An additional library used as a support for WebDriver suites. These frameworks may be test frameworks such as JUnit or TestNG.

# Selenium Components



# Selenium Components - II





# WebDriver

- WebDriver drives a browser natively, as a user would, either locally or on a remote machine using the Selenium server, marks a leap forward in terms of browser automation.
- Selenium WebDriver refers to both the language bindings and the implementations of the individual browser controlling code. This is commonly referred to as just WebDriver.
- Selenium WebDriver is a W3C Recommendation
  - WebDriver is designed as a simple and more concise programming interface
  - WebDriver is a compact object-oriented API
  - It drives the browser effectively

# Install a Selenium library

```
<dependency>  
  <groupId>org.seleniumhq.selenium</groupId>  
  <artifactId>selenium-java</artifactId>  
  <version>4.1.4</version>  
</dependency>  
<dependency>  
  <groupId>io.github.bonigarcia</groupId>  
  <artifactId>webdrivermanager</artifactId>  
  <version>5.1.1</version>  
  <scope>test</scope>  
</dependency>
```

# Install Browser Drivers

Browser	Supported OS	Maintained by	Download	Issue Tracker
Chromium/Chrome	Windows/macOS/Linux	Google	<a href="#">Downloads</a>	<a href="#">Issues</a>
Firefox	Windows/macOS/Linux	Mozilla	<a href="#">Downloads</a>	<a href="#">Issues</a>
Edge	Windows/macOS	Microsoft	<a href="#">Downloads</a>	<a href="#">Issues</a>
Internet Explorer	Windows	Selenium Project	<a href="#">Downloads</a>	<a href="#">Issues</a>
Safari	macOS High Sierra and newer	Apple	Built in	<a href="#">Issues</a>

# Example

```
@Test public void chromeSession() {  
    System.setProperty("webdriver.chrome.driver",  
"D:\\CourseJavaQA\\chromedriver_win32\\chromedriver.exe");  
    ChromeOptions options = new ChromeOptions();  
    WebDriver driver = new ChromeDriver(options);  
  
    // Exercise  
    driver.get("https://bonigarcia.dev/selenium-webdriver-java/");  
    String title = driver.getTitle();  
  
    // Verify  
    assertThat(title).contains("Selenium WebDriver");  
    driver.quit();  
}
```

# Example with WebDriverManager - I

```
public class WebDriverManagerTest {  
    WebDriver driver;  
    @BeforeAll  
    static void setupClass() {  
        WebDriverManager.chromedriver().setup();  
    }  
    @BeforeEach  
    void setupTest() {  
        driver = new ChromeDriver();  
    }  
    @AfterEach  
    void teardown() {  
        driver.quit();  
    }  
}
```

# Example with WebDriverManager - II

```
public class WebDriverManagerTest {  
  
    @Test  
    void test() {  
        // Exercise  
        driver.get("https://bonigarcia.dev/selenium-webdriver-java/");  
        String title = driver.getTitle();  
  
        // Verify  
        assertThat(title).contains("Selenium WebDriver");  
    }  
}
```

# Selenium Commands

- **Actions** – allow to manipulate or change the state of applications (e.g. click on some link or select an option from a page).
- **Accessors** - enable verification and storage of the application state
- **Assertions** - allow compare expected and actual results. They act like checkpoints and if both the values are equal, only then the test case passes or else it fails. Thus, Assertions help verify whether the state of the application after executing the test case conforms to the desired state. Assertions have three modes:
  - Assert
  - Verify
  - WaitFor

# Web Elements

- **Locator strategies** - Ways to identify one or more specific elements in the DOM.
- **Finding web elements** - Locating the elements based on the provided locator values.
- **Interacting with web elements** - A high-level instruction set for manipulating form controls.
- **Information about web elements** - What you can learn about an element.
- **Working with select list elements** - Select lists have special behaviors compared to other elements.



# Web Elements

Locator	Description
class name	Locates elements whose class name contains the search value (compound class names are not permitted)
css selector	Locates elements matching a CSS selector
id	Locates elements whose ID attribute matches the search value
name	Locates elements whose NAME attribute matches the search value
link text	Locates anchor elements whose visible text matches the search value
partial link text	Locates anchor elements whose visible text contains the search value. If multiple elements are matching, only the first one will be selected.
tag name	Locates elements whose tag name matches the search value
xpath	Locates elements matching an XPath expression

# Page Object Model

- Within your web app's UI there are **areas that your tests interact with**. A **Page Object** simply models these as objects within the test code. This reduces the amount of duplicated code and means that if the UI changes, the fix need only be applied in one place.
- **Page Object** is a Design Pattern which has become popular in test automation for enhancing test maintenance and reducing code duplication. A page object is an **object-oriented class** that serves as an interface to a page of your AUT. The tests then use **the methods of this page object class** whenever they **need to interact with the UI of that page**. There is a clean separation between test code and page specific code such as locators (or their use if you're using a UI Map) and layout. Advantages:
  - Clean separation between **test code** and **page specific code** such as **locators**.
  - Single **repository for the services or operations** offered by the page.

# Page Object Model

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.CacheLookup;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.How;
import org.openqa.selenium.support.PageFactory;
```

```
public class ArticleFormPageObject {
    public ArticleFormPageObject(WebDriver driver) {
        PageFactory.initElements(driver, this);
    }
}
```

```
@FindBy(how = How.NAME, using = "title")
public WebElement title;
```

```
@FindBy(how = How.NAME, using = "content")
@CacheLookup // for better test performance
public WebElement content;
}
```

# Using Page Object Model in Selenium Test Scripts

- @Test

```
public void TestTitleAndContentFields() throws InterruptedException {
    System.setProperty("webdriver.chrome.logfile", "TestLog.txt");
    WebDriver driver = WebDriverManager.getInstance(ChromeDriver.class).capabilities(handlingSSL).create();
    driver.get("https://localhost:8081/articles/article-form");

    // Initialize the Page object
    ArticleFormPageObject pageObject = new ArticleFormPageObject(driver);

    // Write some values to title and content
    pageObject.title.sendKeys("Using Page Objects"); // A FindBy call is triggered to fetch title
    pageObject.content.sendKeys("Page objects make selenium testing more robust ..."); // A FindBy fetching content

    // Read values from the Text box.
    log.info("TITLE: {}", pageObject.title.getAttribute("value")); // A FindBy call is triggered to fetch title
    log.info("CONTENT: {}", pageObject.content.getAttribute("value")); // A FindBy call is triggered to fetch content

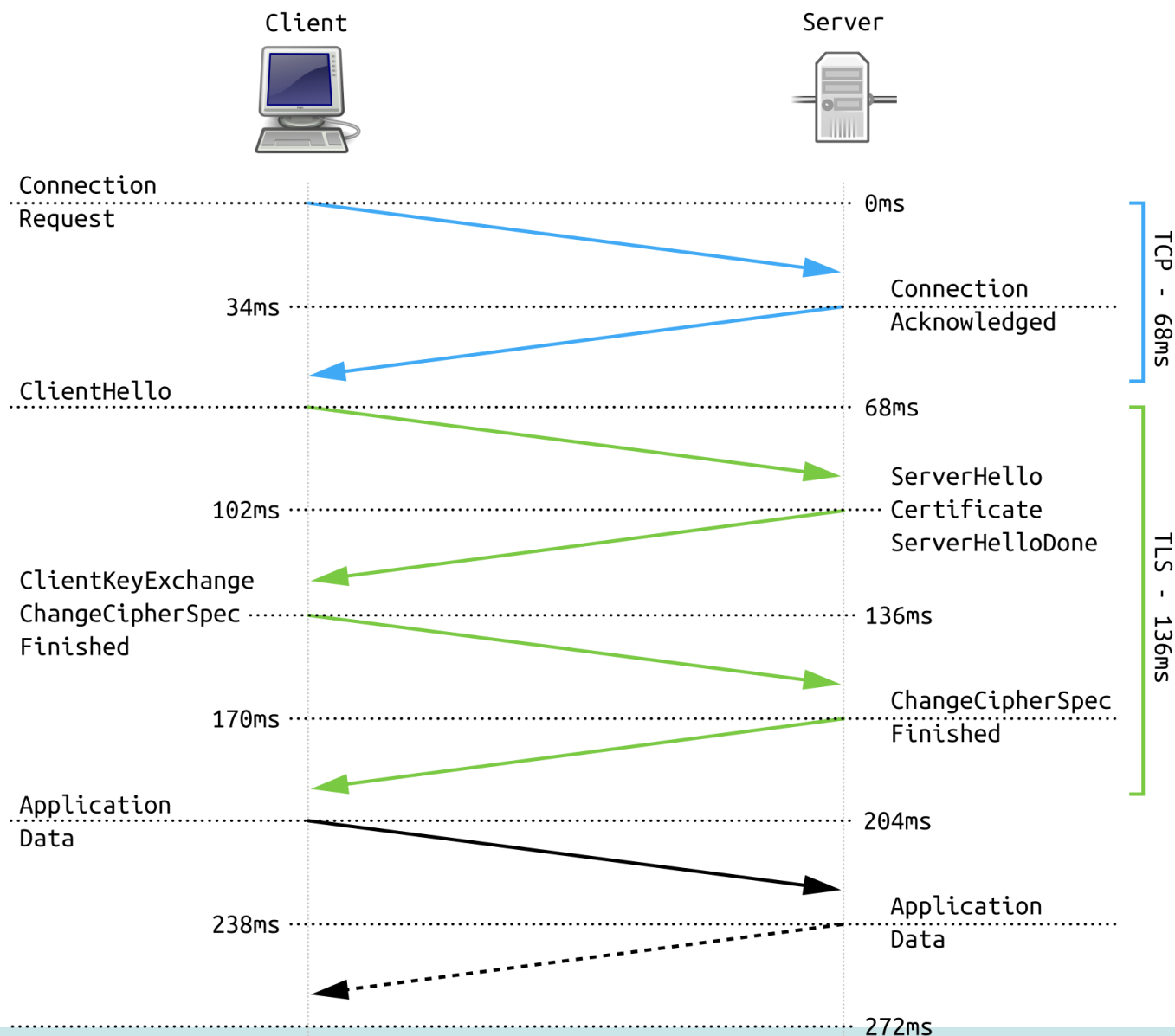
    assertThat(pageObject.title.getAttribute("value")).isEqualTo("Using Page Objects");
    assertThat(pageObject.content.getAttribute("value")).isEqualTo("Page objects make selenium testing more robust ...");

    driver.close();
    driver.quit();
}
```

# Transport Layer Security (TLS)

- **Transport Layer Security (TLS)** is a cryptographic protocol designed to provide communications security over a computer network. The protocol is widely used in applications such as email, instant messaging, and voice over IP, but its use in securing HTTPS remains the most publicly visible.
- The **TLS protocol** aims primarily to provide **cryptography**, including **privacy (confidentiality)**, **integrity**, and **authenticity** through the use of **certificates**, between two or more communicating computer applications. It runs in the **application layer** and is itself composed of two layers: the **TLS record** and the **TLS handshake protocols**.
- **TLS** is a proposed **Internet Engineering Task Force (IETF) standard**, first defined in 1999, and the current version is **TLS 1.3**, defined in August 2018.
- **TLS** is the successor of the now-deprecated **Secure Sockets Layer (SSL)**.

# Simplified TLS 1.2 Handshake



# Using Self-Signed Certificates

- `keytool -genkeypair -alias springboot -keyalg RSA -keysize 4096 -storetype JKS -keystore springboot.jks -validity 3650 -storepass changeit`
- `keytool -genkeypair -alias springboot -keyalg RSA -keysize 4096 -storetype PKCS12 -keystore springboot.p12 -validity 3650 -storepass changeit`
- `keytool -list -v -keystore springboot.jks`
- `keytool -list -v -keystore springboot.p12`
- `keytool -importkeystore -srckeystore springboot.jks -destkeystore springboot.p12 -deststoretype pkcs12`
- `keytool -import -alias springboot -file myCertificate.crt -keystore springboot.p12 -storepass password`
- `keytool -export -keystore springboot.p12 -alias springboot -file myCertificate.crt`

# Spring Boot SSL Configuration

server.ssl.key-store: classpath:springboot.p12

server.ssl.key-store-password: changeit

server.ssl.key-store-type: pkcs12

server.ssl.key-alias: springboot


server.ssl.key-password: changeit

server.port=8443



# Selenium IDE

- Provides you the capability of automatically recording your test cases based upon the interactions with the browser
- Gives developers greater flexibility in executing the test cases - developer can run the entire test suite or execute a single test case
- Operates on the basis of the rich set of Selenese commands, which helps the IDE understand what needs to be done
- Allows the test developers to set breakpoints for the purpose of debugging  
Test cases can be re-used using the run command. (e.g. allowing you to re-use the logic of login or reload on multiple places in the entire suite)
- Use of multiple-locators for each element in the IDE ensures successful execution



http://localhost:8081

Command		Target	Value
1	open	/articles	
2	set window size	1722x936	
3	click	linkText=Add Article	
4	click	id=title	
5	type	id=title	Selenium IDE
6	click	id=content	
7	type	id=content	Selenium IDE is easy to use
8	click	id=file	
9	type	id=file	C:\Users\office27\Pictures\java2days.png
10	click	id=submit	
11	click	xpath=//tr[last()]/td[6]/a/img	
12	mouse over	css=tr:last-child .img-thumbnail	
13	mouse out	css=tr:last-child .img-thumbnail	

Command	<input type="text" value="open"/>	<input type="button" value="//"/>	<input type="button" value="↗"/>
Target	<input type="text" value="/articles"/>	<input type="button" value="↖"/>	<input type="button" value="🔍"/>
Value	<input type="text"/>		
Description	<input type="text"/>		



# Best Practices Structuring Selenium Projects

- 16 Selenium Best Practices For Efficient Test Automation - <https://www.lambdatest.com/blog/selenium-best-practices-for-web-testing/>
- Selenium Java Testing: Page Object Model - <https://www.lambdatest.com/blog/selenium-java-testing-page-object-model/>
- @CacheLookup in PageObjectModel - <https://toolsqa.com/selenium-webdriver/cachelookup-in-pageobjectmodel/>

# Thank's for Your Attention!



Trayan Iliev

IPT – Intellectual Products & Technologies

<http://iproduct.org/>

<http://robolearn.org/>

<https://github.com/iproduct>

<https://twitter.com/trayaniliev>

<https://www.facebook.com/IPT.EACAD>