Selenium Introduction

Selenium Family Overview, Selenium IDE

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://about.softuni.bg

Have a Question?

Software University

sli.do

#QA-Auto-FrontEnd

Table of Contents

What is Selenium?

Selenium Browser Automation Project

Selenium Browser Automation Project

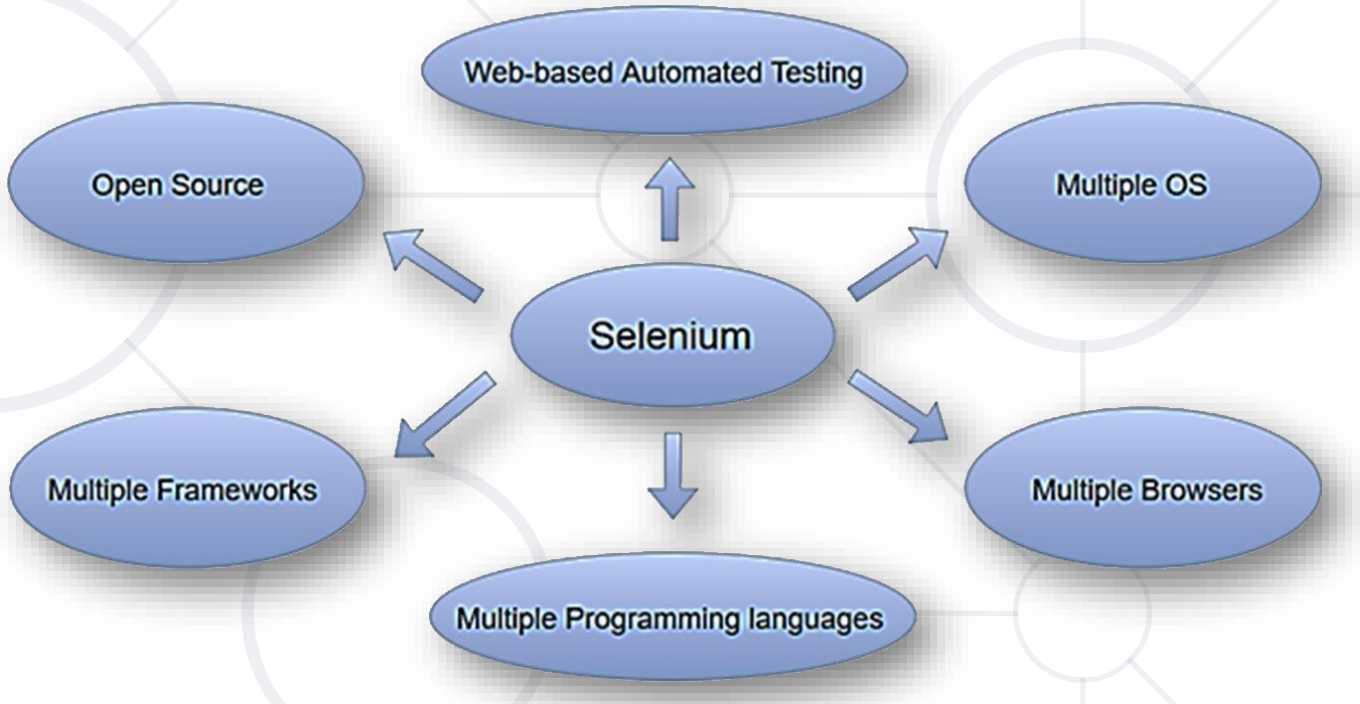
Software University

"**Selenium** is an **umbrella project** for a **range of tools** and **libraries** that **enable** and **support** the **automation of web browsers**.
It provides **extensions** to **emulate user interaction** with browsers, a **distribution server** for scaling browser allocation, and the **infrastructure for implementing the W3C WebDriver specification** that lets you write **interchangeable code for all major web browsers**."

Selenium Official Documentation

Selenium Stack

- **Supports variety of platforms**:
  - Windows, Linux, Mac OS
- **Languages supported**:
  - C#, Java, JavaScript, Perl, Python, Ruby, Kotlin
- **Browsers supported**:
  - Edge, Mozilla Firefox, Google Chrome, Opera, Safari



Selenium Tools

- **Collection** of distinct **software tools**, each tailored to fulfill specific roles in the realm of test automation
- These tools collectively contribute to a comprehensive suite for efficient testing of web applications
  - **Selenium IDE**
  - **Selenium WebDriver**
  - **Selenium-Grid**

Selenium IDE

- Short for **Integrated Development Environment**
- User-friendly **browser extension**
- Simplifies the creation and execution of automated test scripts through a **record-and-playback** mechanism
- Suitable for those with **limited programming experience**
- Accessible **entry point** into test automation
- Limited scope

Selenium WebDriver

Software University

- Most **important component** of Selenium Family
- Powerful and versatile **framework** for **automating web browsers**
- Programmatic **control** over **web elements**
- **Precise test scenarios** across different browsers
- Flexibility and support for **multiple programming languages**
- **Seamlessly integrates** into continuous integration and continuous delivery pipelines

Selenium Grid

- Run tests on different machines against different browsers in parallel
  - Run tests **simultaneously** on different machines
- Follows the **Hub-Node Architecture** to achieve parallel execution of test scripts
  - The Hub is considered as master of the network and the other are the nodes

Selenium IDE

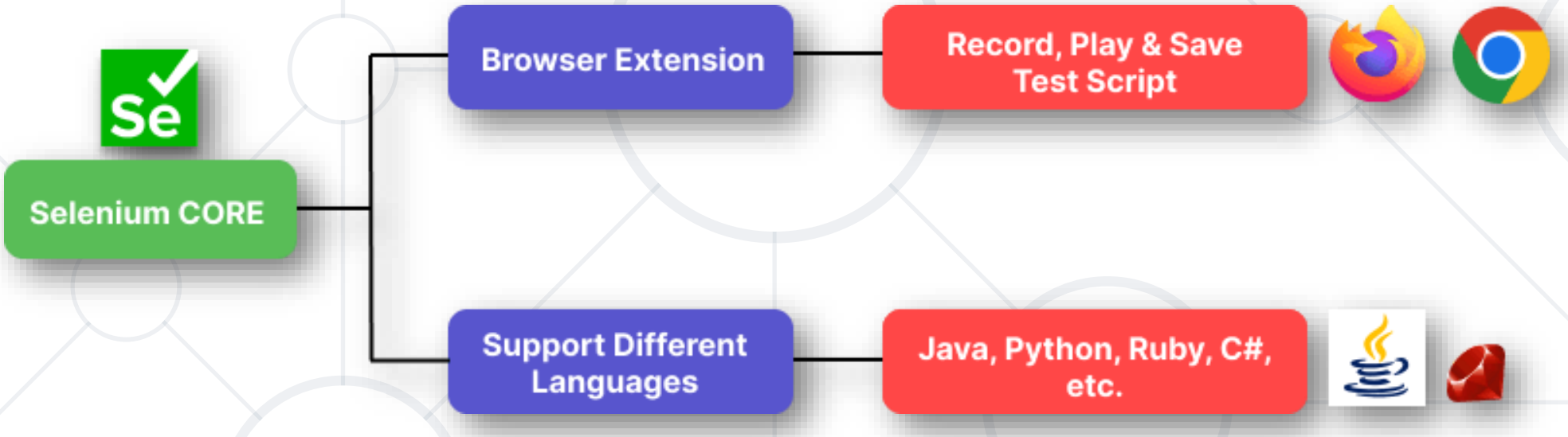
Introduction

Selenium IDE Introduction

- A tool for **building**, **editing**, and **debugging** automated test scripts
- A **browser extension**
- First **created** by **Shinya Kasatani**
- **Donated** to the Selenium project in 2006
- Actively **maintained by the Selenium project** since 2018
- Supports **multiple programming languages** and frameworks
- Provides a **simple interface** for beginners
- Features a **recording capability** that **captures user actions**
- Allows **testing** of web apps **without advanced programming skills**

Selenium IDE Architecture

- Simple and easy to understand, with **three major parts:**



- **Selenium Core**
  - The engine that powers Selenium IDE
  - Executes test scripts, interacts with the browser, provides test results

Selenium IDE Architecture

- **Browser Extension**
  - Initially designed as a Firefox extension
  - Later added support for other browsers (like Chrome)
  - Allows smooth integration with supported browsers
- **Selenium IDE User Interface (UI)**
  - Provides an intuitive interface
  - Enables users to design, modify, and run tests easily

Selenium IDE Features

Key Features, Processes

Selenium IDE Key Features

- **Record and Playback**
  - Automatically create test scripts by recording interactions
  - Perform tasks like button-clicking, form-filling, and component verification
  - Repeat actions by playing back recorded stages
- **Script Editing**
  - View and edit recorded test scripts in the code editor
  - Add extra commands, assertions, and verifications for enhanced functionality
  - Supports multiple programming languages, including JavaScript, Python, and Ruby

Selenium IDE Key Features

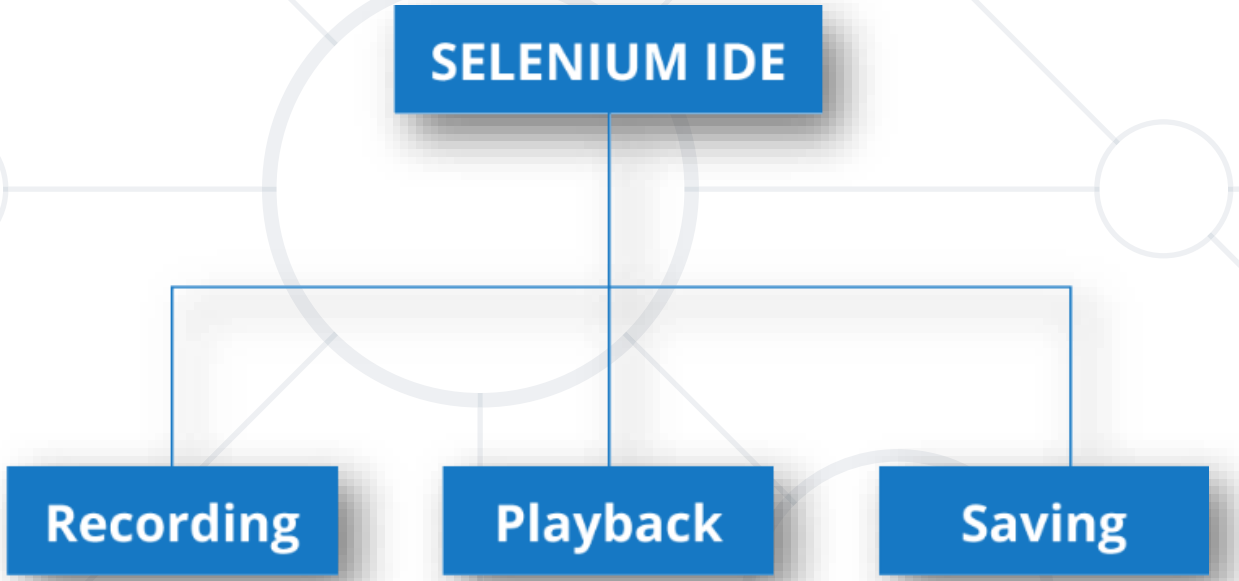
- **Element Locators**
  - Use various methods like XPath, CSS selectors, className, and linkText to locate web page components
  - Visually select components by highlighting them on the page
- **Test Debugging**
  - Debugging features to find and resolve errors in test scripts
  - Set breakpoints, step through code, and investigate variables

Selenium IDE Key Features

- **Test Suites**
  - Group test scripts into test suites for simultaneous execution
  - Configure test order, timeouts, and reporting features
- **Test Playback and Reporting**
  - Execute test scripts directly within your chosen browser
  - Use playback controls like play, pause, and stop
  - View detailed test reports with pass/fail status, execution time, and error messages
- **Export Options**
  - Export test scripts in formats like Java, C#, Python, and Ruby
  - Integrate scripts into various testing frameworks and tools

Selenium IDE Processes

- Three main processes



- Recording
  - Utilizes a browser extension to record user interactions with a web application
  - Tracks activities such as button clicks, text inputs, and page navigation

Selenium IDE Processes

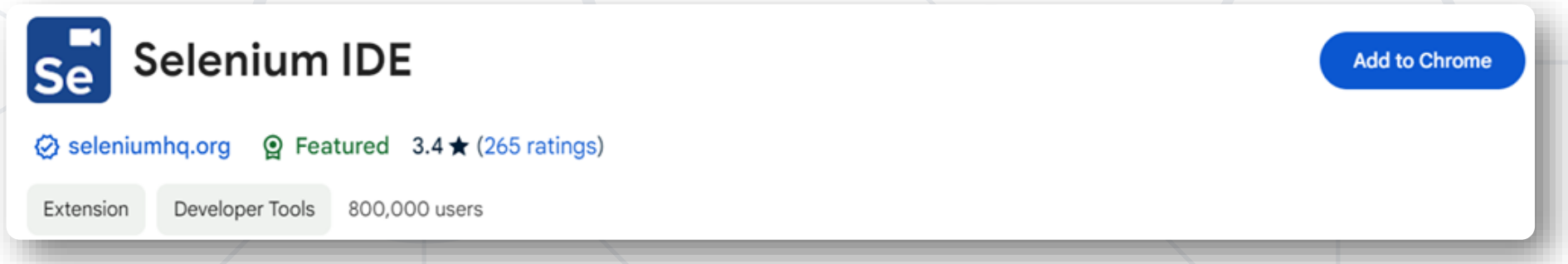
- **Playback**
  - Allows edited scripts to be played back, automating the test process
  - Selenium IDE runs the script, interacting with the browser to simulate user actions
- **Saving**
  - Enables saving the recorded test case script for future use
  - Saves the file with the extension ".side"

Getting Started with Selenium IDE

Installation, UI, Managing Tests and Suites

Installation

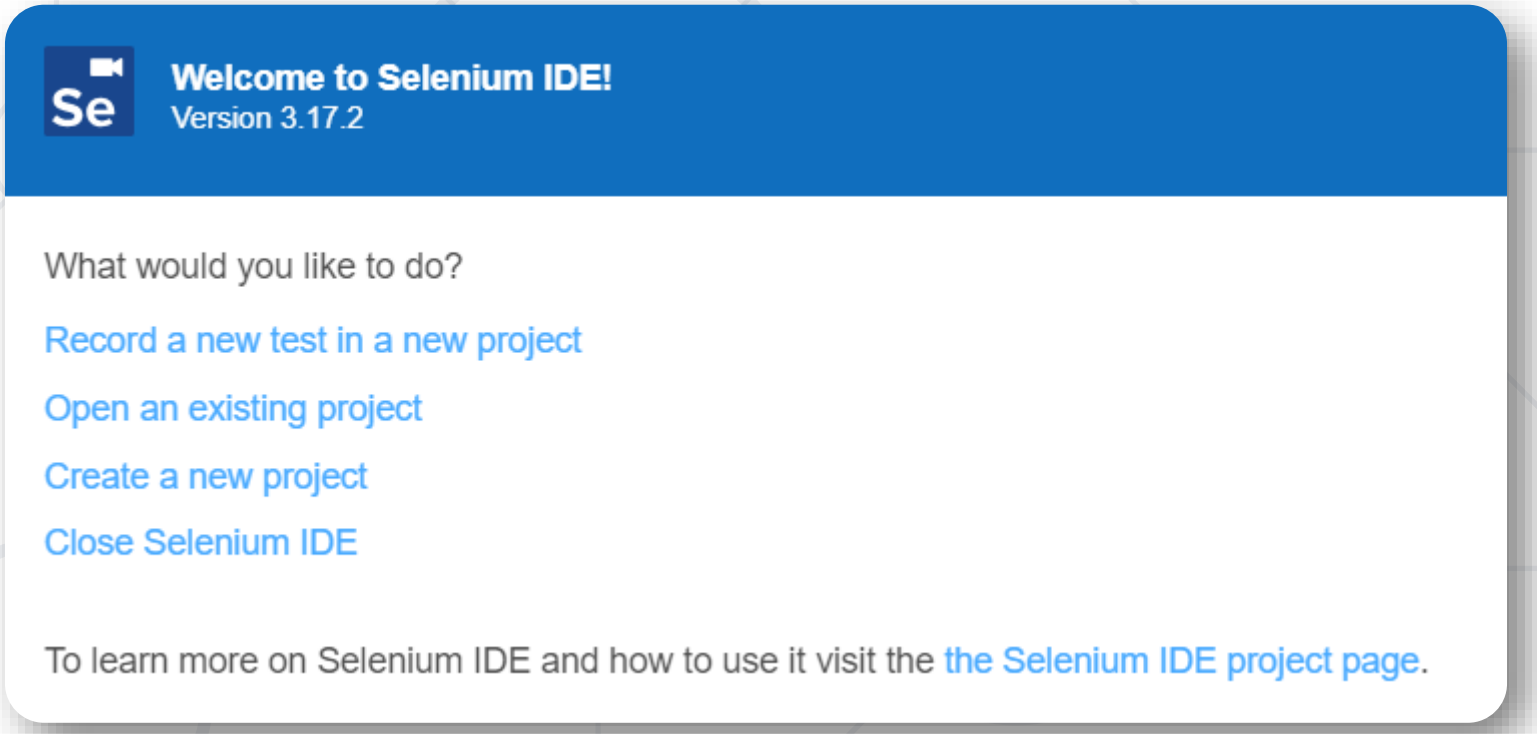
- **Supported browsers:**
  - **Chrome**
  - **Firefox**
  - **Edge**
  - Both **Brave** and **Opera** support the chrome web store natively, so Selenium IDE works with those, too



- Sadly, Selenium **IDE** has **no official release for Safari**

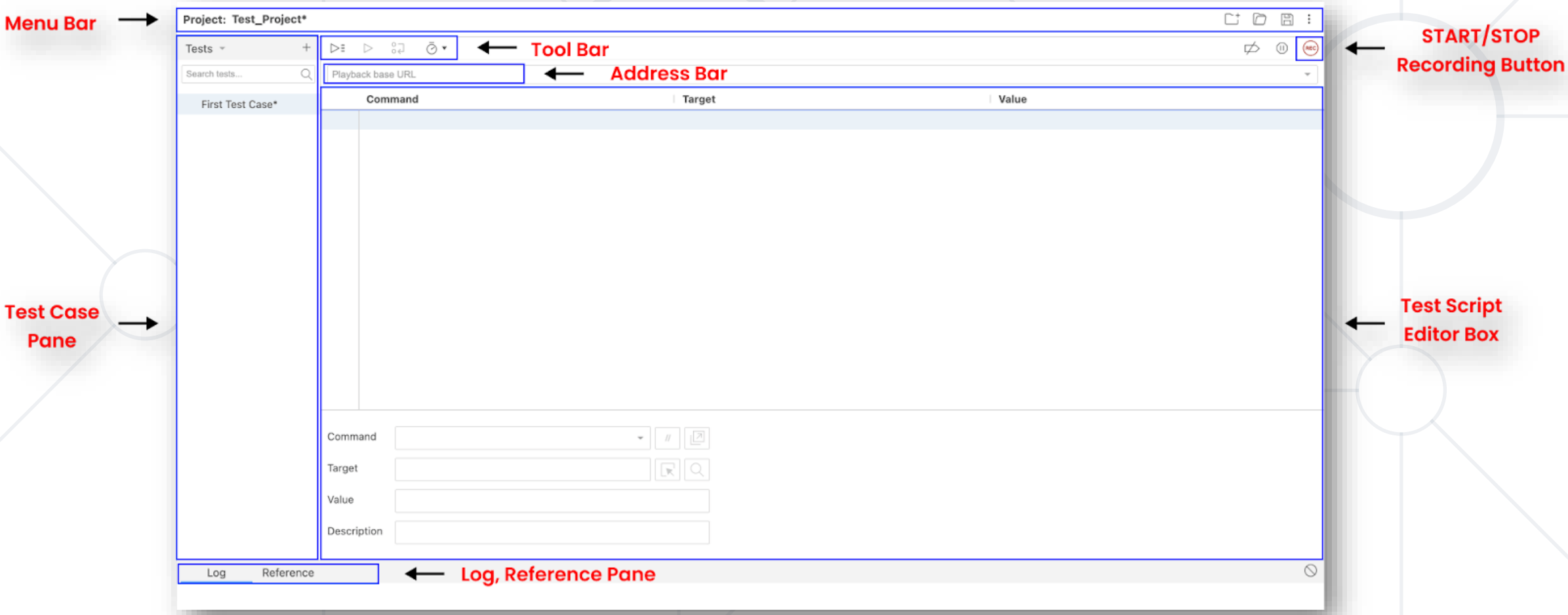
User Interface – Welcome Screen

- Launching the IDE will present you with a welcome screen



- Select, "Create a new project"
- Give a desired name to the project

Project Screen



Project Screen

- **Menu Bar:** Controls for opening, saving, and exporting test cases; Access to preferences and test suite management
- **Tool Bar:** Setup options such as playback speed, element locators, and other
- **Address Bar:** Field to enter the URL of the site where test cases will be run
- **START/STOP Recording Button:** Start or stop recording of test cases
- **Test Case Pane:** Create and manage individual test cases; Add, delete, and modify test steps using various commands and assertions
- **Test Script Editor Box:** Manage test suites, or groups of test cases; Add, modify, and delete test suites and their test cases
- **Log, Reference Pane:** Displays the outcomes of test runs, including errors or failures; Provides detailed information about test procedures and assertions

First Script Live Demo

Recording, Playback, Debugging, Refactoring, Saving

First Script

Software University

- Record and play a script (test) for an incorrect login attempt on a website
- Base URL: https://katalon-demo-cura.herokuapp.com

	Command	Target	Value
1	*open*	/	
2	*set window size*	1552x832	
3	*click*	css=.fa-bars	
4	*click*	linkText=Login	
5	*click*	id=txt-username	
6	*type*	id=txt-username	pepina
7	*click*	id=txt-password	
8	*type*	id=txt-password	1234
9	*click*	id=btn-login	
10	*close*		

Log | Reference

Running 'IncorrectLogin'

1. open on / OK
2. setWindowSize on 1552x832 OK
3. click on css=.fa-bars OK
4. click on linkText=Login OK
5. click on id=txt-username OK
6. type on id=txt-username with value pepina OK
7. click on id=txt-password OK
8. type on id=txt-password with value 1234 OK
9. click on id=btn-login OK
10. close OK

'IncorrectLogin' completed successfully

First Test

- **Add assertions** to check:
  - The home page main text
  - The text on the login page
  - The error message for incorrect login

		Command	Target	Value
1		*open*	/	
2		*set window size*	1552x832	
3		*assert text*	css=h1	CURA Healthcare Service
4		*click*	id=menu-toggle	
5		*click*	linkText=Login	
6		*assert text*	css=h2	Login
7		*type*	id=txt-username	pepina
8		*type*	id=txt-password	1234
9		*click*	id=btn-login	
10		*assert text*	css=.text-danger	Login failed! Please ensure the username and password are valid.
11	//	*close*		

Selenium IDE Commands

Selenese

Selenium Commands (Selenese)

- **Selenese** is the **set of commands** that Selenium IDE uses to interact with your web applications during testing
- These commands **serve as building blocks for creating test scripts**, essentially forming a specialized language for web testing
- A **sequence of these commands constitutes a test script**, dictating actions and checks to be performed

Selenium Commands (Selenese)

- Selenese commands can have a maximum of two parameters:
  - **Target**
  - **Value**
- Parameters are not always required. It depends on the chosen command
- Broadly classified into three categories:
  - **Actions**
  - **Accessors**
  - **Assertions**

Selenium Parameters

Software University

- **Parameters** are typically
  - A locator for identifying a **UI element** within a page
  - A **text pattern** for **verifying** or **asserting** expected page **content**
  - A text **pattern** or a selenium **variable** for **entering text** in an **input field** or for **selecting an option** from an option list

Command types - Actions

- **Directly interact with the browser**
- For example:
  - "**Click**" command - It is the Actions command because it directly interacts with the element on the page by clicking on it
  - "**Type**" command - It is a two-way interaction. Enters text/values into the field, and the field displays them to us
  - "**Close**" command - Mimics the user's action of clicking the "close" button of a window

Accessors

- **Store values in a variable**
- For example:
  - "**storeTitle**" command - Reads the page title and stores it in a variable
  - "**storeText**" command - Stores the text of a specified element into a variable

Assertions

- **Check whether a certain condition has been met**
- **Modes**:
  - "**Assert**" command - The test or the entire test suite will be aborted immediately if the "assert" command fails
  - "**Verify**" command - When the "verify" command fails, the Selenium IDE will log an error in the logs. However, the test or set of tests will continue

Assert vs. Verify

- Choosing between "**assert**" and "**verify**" comes down to convenience and management of failures
  - There is no point checking a paragraph if you are not on the correct page
  - On the other hand, you may want to check many attributes of a page without aborting the test case
  - Usually each command group is started with an "**assert**" followed by one or more "**verify**" test commands

Popular Commands

- **open**: Opens the page using a URL
- **click**: Clicks a specific item, optionally waits for a new page to load
- **type**: Enters text/values into the indicated fields
- **verify/assert title**: Compares the specified page title with the actual one
- **verify/assert text**: Checks whether the specified text exists on the page
- **verify/assert not text**: Checks that the specified text does not exist on the page

Popular Commands

- **verify/assert element present**: Checks whether the specified element exists on the page
- **verify/assert element not present**: Checks that the specified element is not present on the page
- **pause**: Stops script execution for the specified time. For example, pause 5000 will cause the script to stop for 5 seconds
- **close**: Closes the browser
- **set window size**: Resizes the browser window to emulate various screen resolutions
- **verifyEditable**: Verifies the expected element is editable

Popular Commands

- **verifyElementPresent**: Verifies an expected UI element, as defined by its HTML tag, is present on the page
- **verifyText**: Verifies expected text and its corresponding HTML tag are present on the page
- **verifyTable**: Verifies a table's expected contents
- **waitForPageToLoad**: Pauses execution until an expected new page loads. Called automatically when clickAndWait is used
- **waitForElementPresent**: Pauses execution until an expected UI element, as defined by its HTML tag, is present on the page

Control Flow

Control Flow

- Selenium IDE supports commands for adding **conditional logic** and **loops**
- **Execute commands** (or sets of commands) based on **specific conditions** in an application
- Use loops to execute command(s) repeatedly based on **pre-defined criteria**
- Check conditions using **JavaScript expressions**
- **Run** JavaScript **snippets at any point** using execute script or execute async script commands
- **Store results** in **variables** for use in control flow commands
- JS **expressions** can be **used directly in control flow** commands

Available Commands

- Control Flow commands work by specifying **opening** and **closing commands** to denote a set (or block) of commands
- Here are each of the **available control flow commands** accompanied by their companion and/or closing commands
  - **if, else if, else, end**
  - **times, end**
  - **do, repeat if**
  - **while, end**

Conditional Branching

- **Enables you to change the behavior in your test**
  - **if**: Starts a conditional block; Evaluates a JavaScript expression in the target field; Executes subsequent commands if true; Skips to the next conditional command if false
  - **else if**: Used within an if block; Evaluates a JavaScript expression in the target field; Executes the following commands if true; Skips to the next conditional command if false
  - **else**: Final condition in an if block; Executes if none of the previous conditions are met; Proceeds to the end command after execution
  - **end**: Terminates the conditional command block; Required to complete the block and avoid errors

Conditional Branching Example

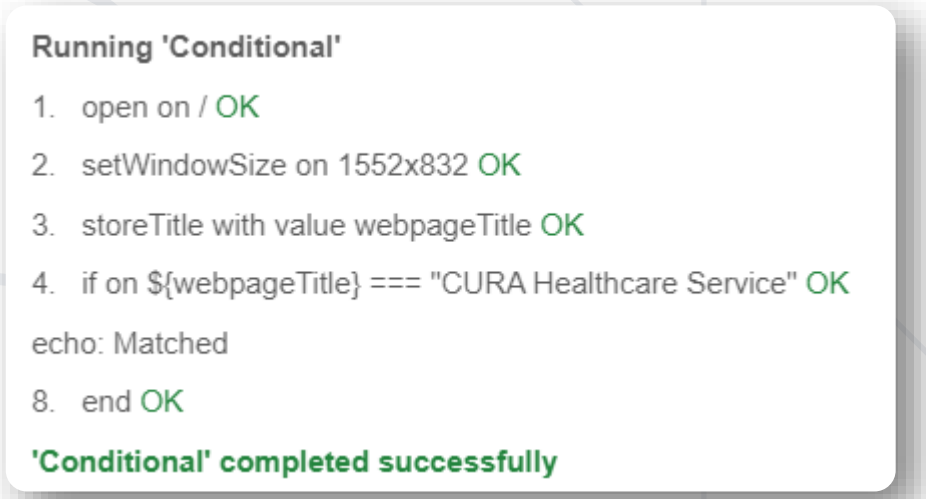
- Fetches the title of the webpage and prints "Matched" if it is "CURA Healthcare Service", else prints "Unmatched"

https://katalon-demo-cura.herokuapp.com

	Command	Target	Value
1	*open*	/	
2	*set window size*	1552x832	
3	*store title*		webpageTitle
4	*if*	${webpageTitle} === "CURA Healthcare Service"	
5	*echo*	Matched	
6	*else*		
7	*echo*	Unmatched	
8	*end*		

Conditional Branching Example Explained

- **store title** command: It will store the title of the webpage in the variable you provide in the value input field. Here the variable name is webpageTitle
- Creating **if** block with the condition as $webpageTitle==="CURA Healthcare Service"
- **echo** is a print statement to print statement in logs
- Once the script is executed it will give the output in the log pane



Looping

- **Looping enables repeating a set of commands multiple times or until a certain condition is met**
  - **times**: Repeats commands a specific number of times
  - **do and repeat if**: Executes commands at least once and repeats based on a condition
  - **while**: Repeats commands as long as a condition is true
  - **forEach**: Iterates over a collection, executing commands for each item

Looping Example with "times"

- **times**: specifies the number of iterations for a set of commands. The number of iterations is provided in the target input field
- **end**: This command closes the times command block. It indicates the end of the loop

5	✓ *times*	3	
6	✓ *type*	id=txt-username	pepina
7	✓ *type*	id=txt-password	1234
8	✓ *click*	id=btn-login	
9	✓ *end*		

Looping Example with "do"

- **do and repeat if**: Executes the enclosed commands at least once
- **Repeats based** on the **condition** specified **in repeat if**
- The **condition** is **evaluated after** the commands are executed

	Command	Target	Value
1	✓ *execute script*	return 0	check
2	✓ *do*		
3	✓ *execute script*	return ${check} + 1	check
4	✓ *repeat if*	${check} < 3	3
5	✓ *close*		

- **Note**: The loop continues until the condition is false or 1000 attempts. Override with a number in the repeat if value field

Looping Example with "while"

- **while**: Executes the enclosed commands while a condition is true
- **Repeats** based on the **condition specified** in the **while** command
- The **condition** is **evaluated before** the commands are executed

	Command	Target	Value
1	✓ *execute script*	return 0	x
2	✓ *while*	${x}<3	
3	✓ *execute script*	return ${x} + 1	x
4	✓ *end*		

Looping Example with "forEach"

- **forEach**: Iterates over each item of a collection (e.g., a JS array)
- **Specify** the **variable name** of the array in the target field
- **Specify** the **iterator variable** name in the value field
- **Commands** inside the loop **execute** for **each array entry**
- The **current entry** is **accessible** through the **iterator** variable

1	✓ *open*		
2	✓ *execute script*	return ["Audi","Volvo","BMW","Opel","Ford"]	x
3	✓ *for each*	x	itr
4	✓ *echo*	${itr}	
5	✓ *end*		

- **Note**: The loop continues until all items in the array are processed

Command-line Runner

Command Line Runner

- Possibility to run your tests from the command line
- The following dependencies are needed for the command line runner to work:
  - **node** (the Node.js programming language) version 8 or 10
  - **npm** (the NodeJS package manager) which typically gets installed with node
  - **selenium-side-runner** (the Selenium IDE command line runner)

```
> npm install -g selenium-side-runner
```

Browser Driver

- And then the **browser driver** you want
  - For **Chrome**

```
> npm install -g chromedriver
```

  - For **Firefox**

```
> npm install -g geckodriver
```

  - For **Edge**

```
> npm install -g edgedriver
```

Launching Runner

- Once installed simply **run the tests** using:

```
> selenium-side-runner /path/to/your-project.side
```

- **Filter tests**:

```
> selenium-side-runner --filter smoke
```

- **Output** test results:

```
> selenium-side-runner --output-directory=results --output-
format=jest
```

Code Export

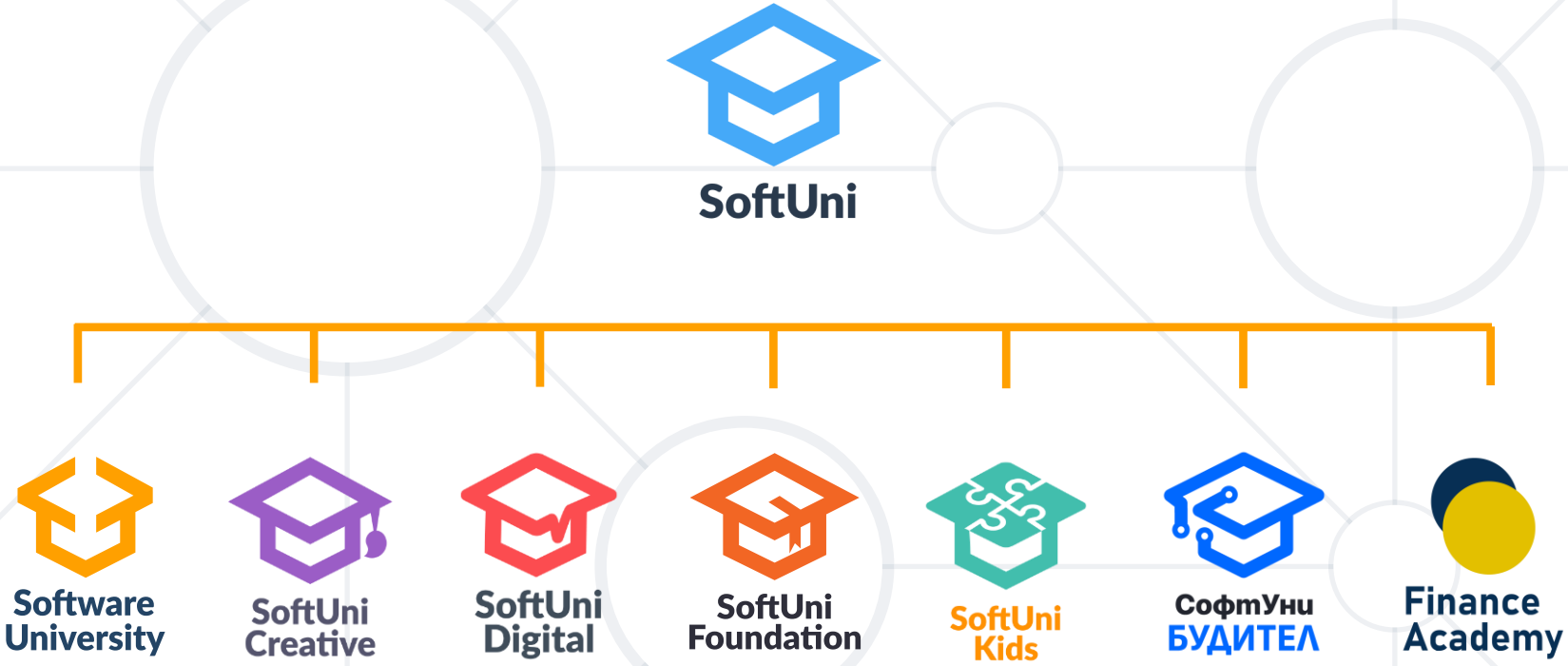
Code Export

- Possibility to **export suite** or tests to **WebDriver code**
- You can export either a test or suite
- Export to the following languages and test frameworks
  - **C# NUnit**
  - **Java JUnit**
  - **JavaScript Mocha**
  - **Python pytest**

Summary

- Selenium Browser Automation Project
- Selenium IDE Introduction
- Key Features, Processes: Recording, Playback, Saving
- Installation, UI, Managing Tests and Suites
- Writing Scripts: Creating First Test
- Selenese: Selenium IDE Commands
- Selenium IDE Advanced: Control Flow, Command-line Runner, Code Export

Questions?



SoftUni Diamond Partners

Software University

SmartIT

Coca-Cola HBC Bulgaria

SUPER HOSTING .BG

createX

Postbank

Решения за твоето утре

DRAFT KINGS

VIVACOM

AMBITIONED

INDEAVR

Serving the high achievers

SOFTWARE GROUP

PHAR VISION

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg/
- © Software University – https://softuni.bg