ЛАБ-3-Защита

ВАЖНО!
Написано @kkettch по лекциям Клименкова и иногда с помощью ChatGPT, использовалась для защиты ЛР. Информация может оказаться старой или недостоверной.

1. Функциональное тестирование. Основные понятия, способы организации и решаемые задачи.

- Строится на готовой системе, в рамках модульного и интеграционного тестирования.
- В качестве исходников для тестов функционального тестирования берутся **Use-Case'ы** либо сценарии использования системы. Обычно указывают внутри Use-Case'а конкретные данные, которые мы хотим подставить в работающий функционал и посмотреть как это работает.

Применяют и ручное и автоматизированное тестирование

- *Ручное*: сделали новый функционал, протестировали руками
- *Автоматизированное*: писать функциональное тестирование без проверки вручную чаще всего бессмысленно
  - Открытое: Selenium, Sahi, Watir
  - Коммерческое: от HP, Rational (IBM)
- Особенности: в качестве входа - интерфейс приложения для пользователя верхнего уровня. Необходимо каким-то образом воздействовать на этот интерфейс.

2. Система Selenium. Архитектура, принципы написания сценариев, способы доступа к элементам страницы.

Selenium - состоит из нескольких компонентов

1. IDE - позволяет записывать действия, запоминая их, которые позже можно повторить. Однако сценарии получаются не очень качественными из-за чего использовать его для реального тестирования не очень хорошо.
2. SeleiumServer / WebDriver - пишется самостоятельно или получается из кода сформированного IDE. Удобство в том, что его можно отредактировать и изменить логику отработки тестовых сценариев необходимым образом.
3. Grid

- Разработка возможна, например, на следующих языках:
  - Java

- PHP
- Python
- C#

Selenium является кросс-бразуерным. Есть драйверы для Firefox, Chrome, Exlporer и др. Однажды написав программу, можно выполнить код на многих браузерах сразу. Возможность протестировать одну программу в многих браузерах является плюсом.

- Имеет встроенные конструкции assert для проверки отработки сценариев.
- Встроенный механизм логирования ошибок

Selenium IDE - изначально разработан как плагин для Firefox. Интегрированная среда исполнения и разработки тестов. Поддерживает запись тестов в виде Java, Ruby, HTML

Пример команд для Selenium IDE:

1. open – открыть URL
2. click – клик по элементу
3. type – ввести текст в поле
4. select – выбрать элемент из выпадающего списка
5. check – установить флажок
6. uncheck – снять флажок
7. assertText – проверить текст элемента
8. verifyElementPresent – проверить наличие элемента
9. waitForElementVisible – ждать, пока элемент станет видимым
10. storeText – сохранить текст элемента в переменную
11. verify - для проверки условий, при этом тест продолжается даже при неудаче (в отличие от `assert`, который останавливает тест)

Assertion & Verification

- Предназначены для проверки содержимого элемента UI
  - Элемент присутствует?
  - Есть ли необходимый текст на странице?
  Если Verification неуспешна - тест продолжается
  Если Assertion неуспешна - тест останавливается

Ограничения **SeleniumIDE**:

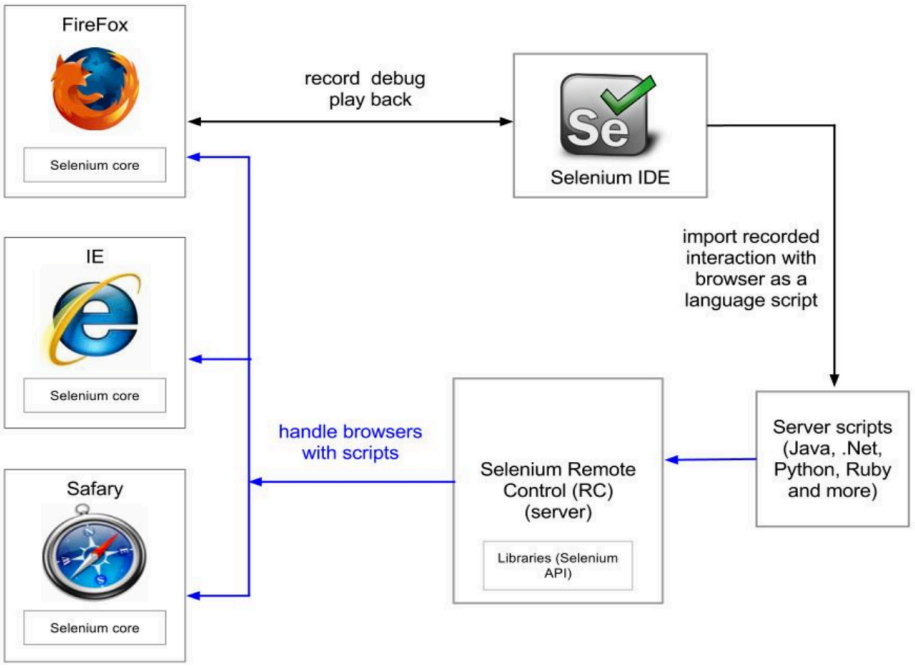
- Слабо развитое управление логикой теста
- Запускает только свои сценарии
- Сложно использовать с динамическим содержимым - современные сервера приложений генерируют html разный при каждом запросе (тк динамически

подставляют идентификаторы елементов и тд). Поэтому на что полагаться, когда пришла страница не всегда понятно

SeleniumServer - компонент для кросс-бразуерного тестирования

- Сервер для тестирования разработан на Java
- Работает как прокси для web-запросов совместно с WebDriver для каждого браузера.
- Используется совместно с многими языками программирования
- Разработка - плагины к NetBeans и Eclipse
- Запуск тестов - maven и ant

Принцип работы:



Если IDE работает с Firefox, то можно записать скрипты на любом ЯП, SeleniumServer воспринимает запросы и запускает несколько браузеров, чтобы в каждом из них протестировать скрипт.

Процесс разработки:

- Записать сценарий в виде java кода
- Создать проект в IDE и добавить необходимые jar файлы
- Скопировать сценарий в IDE
- Запустить

Selenium Grid - позволяет запустить параллельно на нескольких браузерах большое количество тестов

- Ускоряет выполнение тестов (за счёт параллельности).

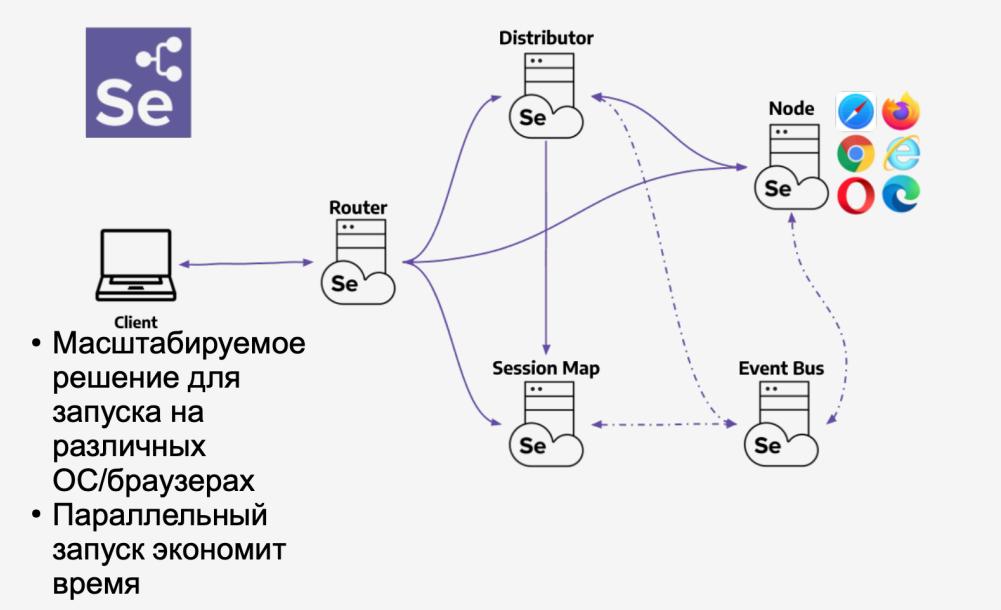
- Позволяет тестировать в разных браузерах и ОС одновременно.
- Удобен для распределенного тестирования на нескольких устройствах.

Состоит из двух основных компонентов:

Компонент	Описание
Hub	Главный узел. Принимает команды от клиента и распределяет их на узлы.
Node	Исполнитель. Устанавливается на удаленной машине. Запускает браузер и выполняет тест.

Как работает

1. Hub получает тест от клиента (например, Selenium WebDriver).
2. Определяет подходящий node (по типу браузера, ОС).
3. Направляет туда тест.
4. Node выполняет команды и возвращает результат.



3. Язык XPath. Основные конструкции, системные функции, работа с множествами элементов.

XPath локатор - метод поиска узлов в DOM модели, язык запросов к элементам DOM модели. В Selenium XPath помогает находить элементы в DOM, особенно когда:

- У элемента нет ID
- Или ID динамический
- Или требуется точное позиционирование в сложной структуре
- Не всегда можно использовать идентификаторы для поиска элементов. В современных серверах приложений, порталах нельзя привязываться к

компонентам по ID — он динамический

Основные типы XPath

1. Абсолютный XPath
   Путь от корня документа (`html`) до элемента:
   `/html/body/div[2]/table/tbody/tr[1]/td[1]`
   Минусы:
   - Очень чувствителен к изменениям в структуре
   - Используется редко
2. Относительный XPath
   Поиск от любого уровня, часто через `//`:
   `//input[@type='email']`
   `//div[@class='product']//span[contains(text(),'Цена')]`
   - Гибкий, удобен для сложных структур
   - Основной тип XPath в тестировании

Навигация по DOM: Оси XPath (Axes)

XPath поддерживает оси — способы перемещения от текущего узла.

Ось	Описание	Пример
`self::`	Сам узел	`//p/self::p`
`parent::`	Родитель	`//input/parent::form`
`child::`	Прямые дети	`//ul/child::li`
`descendant::`	Все потомки	`//div/descendant::span`
`ancestor::`	Все предки	`//button/ancestor::form`
`following::`	Все элементы **после**	`//h2/following::p`
`preceding::`	Все элементы **до**	`//p[@id='para2']/preceding::h2`
`following-sibling::`	Соседние элементы справа	`//label/following-sibling::input`
`preceding-sibling::`	Соседние элементы слева	`//input/preceding-sibling::label`
`attribute::`	Доступ к атрибутам	`//a/attribute::href`

Предикаты: Фильтрация элементов

Предикаты — условия в `[]`, позволяющие уточнять выбор

1. По индексу:
   `//ul/li[1]` - первый элемент списка
   `//table/tr[last()]` — последняя строка таблицы`

2. По атрибуту:
   `//button[@type='submit']`
   `//input[@checked]`
3. По тексту:
   `//a[text()='Вход']`
   `//span[contains(text(),'Цена')]`

Часто используемые XPath-функции

Функция	Описание	Пример
`text()`	Текст внутри тега	`//h1[text()='Главная']`
`contains()`	Частичное совпадение текста или атрибута	`//a[contains(@href, 'login')]`
`starts-with()`	Начало строки	`//div[starts-with(@id, 'user_')]`
`not()`	Отрицание	`//input[not(@disabled)]`
`last()`	Последний элемент в множестве	`//li[last()]`
`position()`	Позиция элемента	`//li[position() < 4]`
`normalize-space()`	Удаляет лишние пробелы	`//p[normalize-space(text())='Подтвердить']`