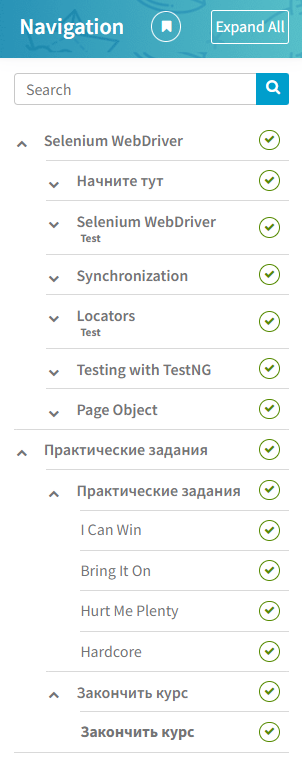
**#Lab9 “Webdriver”**

1. Проходите курс “WebDriver” - <https://learn.epam.com/detailsPage?id=93e65671-c99b-42d7-ae15-f77a60cdfa53>

 2. В помощь вам идёт книга - https://comaqa.gitbook.io/selenium-webdriver-lectures/

3. Ваша задача здесь - научиться работать с браузером, локаторами и писать правильные ассерты. Остальной код должен быть минимально адекватным, соблюдающим хотя бы базовый code convention в плане нейминга.

То есть, результатом вашей работы по #Lab9, станет код в репозитории + ваши знания из курса. Книга - ваше подспорье в правильных ответах.

**--Why Selenium--**

UI может быть единственной доступной частью приложения, которую мы можем протестировать и автоматизированной частью

UI автоматизация может быть тем этапом, когда задействуется user - like interaction

UI автоматизация дает возможность кросс-браузерного тестирования

**Why?**

1. Бесплатный

2. Пионер в области open source

3. Библиотека автоматизации, потому на базе seleniuma можно разработать свой фреймворк/инструментарий

4. Selenium — это ядро, и оно входит в состав фреймоворков

5. Является стандартом W3C

**--What is Selenium--**

Selenium — это набор компонентов, каждый из которых вполне самодостаточный, но работает с четкой привязкой:

1. Selenium WebDriver - библиотека программирования

2. Selenium Server (решают инфраструктуру и удаленные тест раны)

3. Selenium Grid (решают инфраструктуру и удаленные тест раны)

4. Selenium IDE (предлагает легкий варианты работы для разработки в селениум)

5. Selenium RC-Deprecated

Selenium IDE— это расширение для браузера, которой легко загрузить для сайта, позволяет записывать тесты и воспроизводить их.

Записывать в режиме запоминаний действий, которые делает пользователь,

потом тесты можно сохранить в тестовый набор, в окошке IDE увидеть переменные, команды и атрибуты авто-теста

Selenium RC (Remote Controll)-вначале был как попытка реализовать прослойку между ЯП, давая возможность писать тесты и позволить взаимодействия с браузером.

Selenium RC выполнял 2 задачи:

1. Управление браузером (проксирование запросов),

2. Конвертация в js команд, приходящие по http протокола

**Недостатки:**

1. JS можно получить доступ к параметрам, даже к тем, к которым пользователь не может иметь доступа

2. Результаты авто тестов не достоверны

3. Конверсия из языка программирования в js имела свои издержки

4. Большая задержка выхода на (Новые версии селениума не успевали за появлениями новых версий браузеров)

потому Selenium RC в утиль

На смену Selenium RC пришел Selenium WB

Selenium WB - появились новые узлы: Selenium Server и Browser Drivers

Selenium Server - удобен, когда нам нужно построить унифицированную среду для запуска тестов. (RemoteWebDriver)

Сам код разворачивается и пишется в Test Script (локально)

Selenium Grid-когда нужна сложная инфраструктура для тестов, когда должны быть множественные конфигурации и версии браузеров.

(запускается с параметром либо с hub, либо с node)

Selenium - является комплексным проектом, который представляет из себя набор компонентов, используемые для UI автоматизации

**--Synchronization issues--**

Синхронизация-согласованность выполнение кода с событиями на вэб приложении

Факторы, приводящие к проблемам:

1. Уверенность разработчиков в том, что любое действие будет занимать некое константное время

2. Клиент-серверная спецификация

3. Разница перфомансов js

4. Сетевая инфраструктура

5. Неверное указание критериев определения статуса страницы

**--WebDriver - Synchronization solutions--**

Инструменты WebDriver API на уровне кода:

1. Настройка общих временных параметров.

2. Ожидания(waits), направленные на определения состояния переменных

Не стоит использовать: Thread.Sleep

Почему он не является методом решения вопроса синхронизации:

1. Используется константное время (1 ошибка)

2. Не является частью драйвера WebDriver API

3. У него есть слишком много обёрток в WebDriverWait|ExpectedConditions

Когда стоит использовать Thread.Sleep:

1. Ищем проблемы синхронизации

2. Заимплиментить какой-то тест(?) (типа чтобы отправлять пакеты по таймеру)

3. Когда удобно затормозить весь джава процесс/код во время отладки.

Есть строго допустимые ситуации, когда стоит использовать Thread.Sleep и они не зависят от внешних условий (debug, demo, counters, only)

**---Sync issues solutions - general timeouts (Настройка общих временных параметров.)---**

Что это за настройки (implicit waits, но лучше называть general timeouts):

1. Период загрузки страницы (pageLoadStrategy())

2. Время ограничения работы асинхронного скрипта(executeAsyncScript())

3. Тайм-аут ожидания

**--Sync issues solutions - Waits--**

1. Explicit/fluent wait

2. Implicit wait (исторически не входил в API, вернулся лишь недавно, лучше не использовать)

Используется для кратковременного решения проблемы синхронности. Применяется в крайнем случае.

Считаем точкой риска. Не стоит устанавливать таймаут не нулевым. Стоит реализовать все варианты.

**--Sync issues solutions - implicitlyWait vs explicit waits--**

1. implicitlyWait может замедлять весь тестовый запуск, используем, когда хотим посмотреть исчезновение объекта, может ломать explicit waits.

2. implicitlyWait вложен в функцию findelement(). Используя implicitlyWait и explicit wait вместе implicitlyWait, может никогда не сработать

**--Webdriver locators and selectors--**

Можно выбрать элемент через:

1. id

2. name

3. class name

4. tag name

5. link text

6. partial link text

7. CSS selector

8. XPath

**--DOM(Document Object Model)--**

Обходим для описания структуры различных документов в унифицированном виде

**--Reliable locators--**

Правила:

1. Убрать большой процент обслуживания локаторов (делаем стабильные локаторы)

2. Обращаем внимание на ID

3. Находим якорь с наименьшим риском изменением, с легкоузнаваемыми атрибутами

4. Избегаем цифровых индексов (первый/последний элемент)

5. Используем статические локаторы, которые привязаны к функциональным локаторам

6. Используем "золотое правило"-локаторы устойчивы к изменениям, но со строгими критериями

7. Менее детально описываем ступеньки перехода (важна конечная точка)

8. Обращаем внимание на потенциально рисковые места и отслеживаем значения атрибутов

9. Локаторы нужно составлять с учетом цели

10. Фреймы используются

11. Не доверяем плагинам

Рекомендации:

1. Сохраняем холоднокровие

2. Быть рассудительным

3. Иметь здоровый скептицизм

**---Intro---**

WebDriver не является тестом без юнит фреймворка-набор джава методов и аннотаций для тестирования(!!)

С фреймоворком WebDriver уже может протестировать

**--Page Object--**

1. Нужно разделять тестовые задания и тестовый контент

2. Все взаимодействия со страницей должны быть в одном месте

**---Ideas vs Development---**

1. DSL - Domain Specific Language (именование страниц и методов должны говорить сами за себя)

2. dry - не повторяйся

3. KISS - делая все просто

4. yagni - не нужно сходу все реализовывать