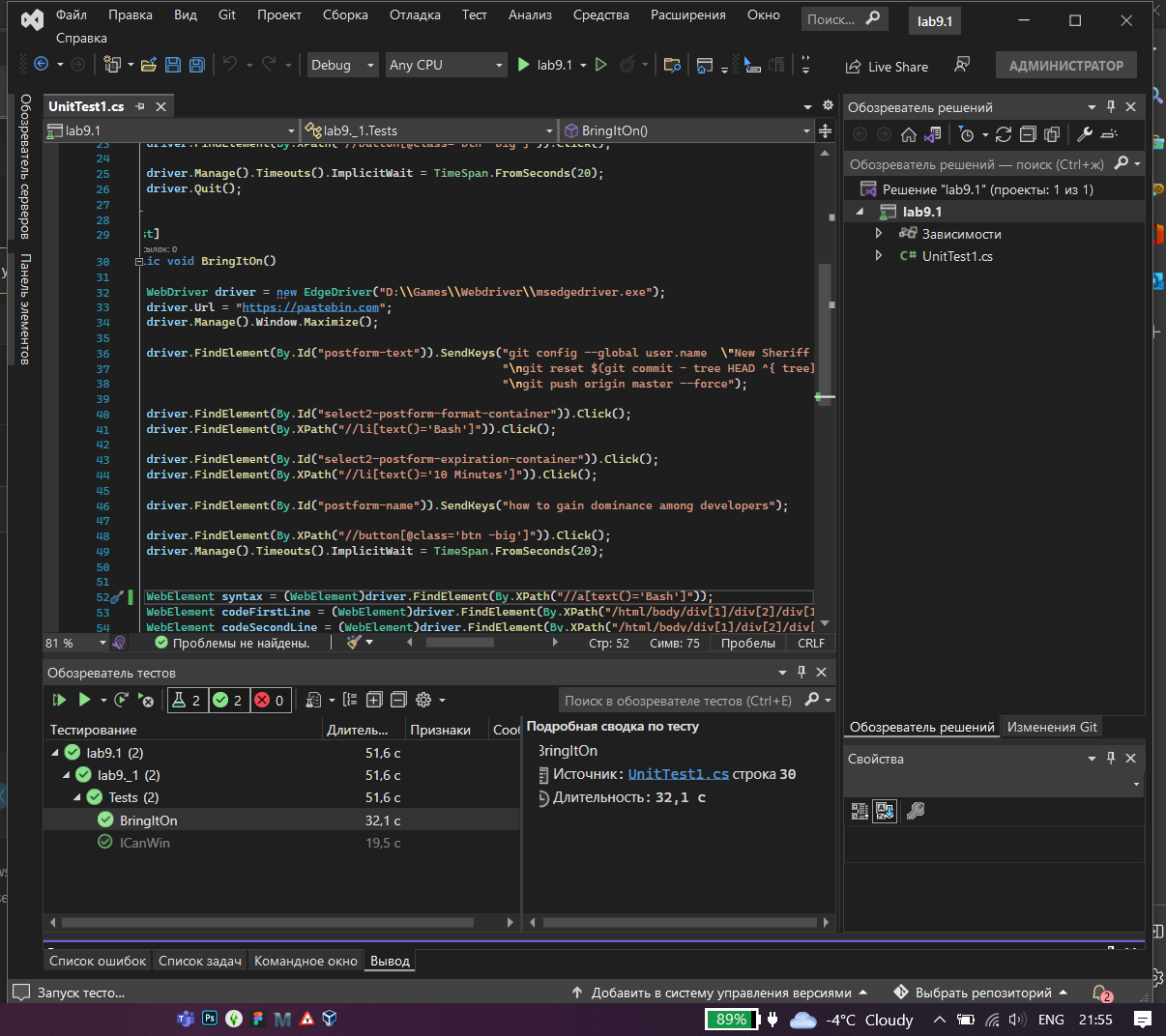
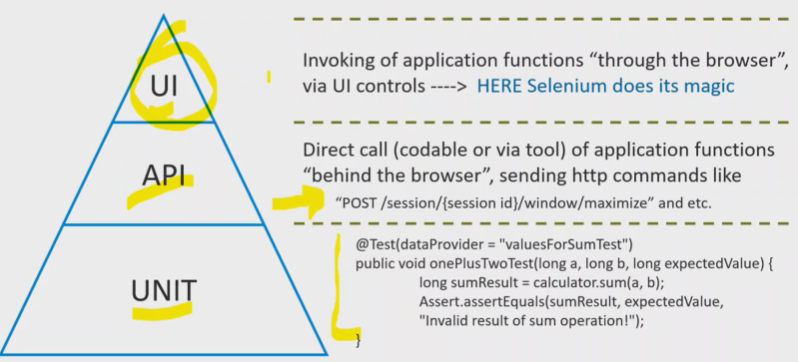
**Lab9.2 – 4 лаба 3ий тест**

****

**Why Selenium – инструмент автоматизации**



* UI может быть единственной частью приложения, доступной для автоматизации и тестирования
* UI автоматизация может быть тем этапом, когда задействуется в полной мере эзер экспиреанс. Тестовые сценарии запускаются именно так, как делал бы их человек. Это страхует разработчика и автоматизатора от разных проблем.
* UI автоматизация делает возможным именно в кросс-браузерном режиме приложения.

**Почему Selenium получил популярность, как инструмент UI автоматизации?**

* Он open source – бесплатный и исходный код селениума доступен любому желающему.
* Не надо покупать какие-то дорогие лицензии.
* Библиотека автомтизации, потому на базе селениума можно разработать свой фреймворк/инструментарий.
* Именно он был родоначальником в области опен-сорсных методов автоматизации. Переболел всякими болячками. Крч очень крутой
* Селениум – ядро, то, что работает под капотом. Входит в состав более сложных фреймворков.
* Является стандартом W3C для разработчиков браузеров.

**Итоги:**

* Автоматизация – это не только применение селениума.
* UI автоматизация – сфера, в которой селениум доминирует.

**What is Selenium**

**Selenium** – это набор компонентов, используемых для ui автоматизации. Позволяет гибко реагировать на запросы времени, выполнять или отвечать нужных автоматизаторам при разработке тестов.

представляет собой **драйвер** браузера, то есть программную библиотеку, которая позволяет разрабатывать программы, управляющие поведением браузера

1. **Selenium Webdriver** – в классическом виде – библиотека программирования, набор классов и методов.
2. **Server и Grid** – решают инфаструктурные задачи. Грамотно задать режим выполнения тестов на удал узлах сети и т.д.
3. **IDE** – предлагает легкий вариант для разработчика. 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.
4. **RC** – родоначальник, но устаревший. Выполнял 2 задачи: управление браузером( проксирование запросов) и конвертацию в js команд, приходящие по http протокола.

**Setup Steps**

**Настройка среды разработки и написание первых строчек кода.**

1. Скачать и установить jdk – набор библиотек разработчиков.
2. Скачать драйвер для браузера.
3. Настройка переменных среды.
4. Скачать и установить ide для джавы.
5. Создать maven для сборки проекта.
6. Добавить.

**Synchronization issues**

Проблемы **синхронизации** – согласованность выполнения кода, использующего веб-драйвера и событиями, происходящими на веб-странице приложения.

**Факторы, приводящие к появлению этих проблем:**

1. Уверенность разработчика в том, что время выполнения одни и тех же сценариев на одних и тех же страницах будет занимать определенное время, которое можно зафиксировать, используя специализированные инструменты.
2. Клиент – серверная спецификация. Клиент – рабочая станция, на котором запускается код автоматизированного теста. Сервер – то, где хостинг и происходит жизненный цикл самого приложения.
3. Разница перфоменсов производительности и быстродействия js.
4. Сетевая инфаструктура. Она может быть очень распределенной.
5. Нечеткое указание критериев определения статуса страницы и тд. Важно понимать, какие шаги будут предприняты после загрузки страницы.

**WebDriver - Synchronization solutions**

**Решение проблем на уровне кода:**

1. **General timeouts**. Настройка каких-либо общих таймаутов, временных параметров поведения методов веб-драйвера, которые имеют отношение к поиску элементов на странице и определения их глобальных состояний.
2. **Waits**. Ожидание. Направлены на состояние и параметров конкретно выбранного элемента на странице.

**Webdriver** — популярный инструмент для управления реальным браузером, который можно использовать как для автоматизации тестирования **веб** приложений, так и для выполнения других рутинных задач, связанных с работой в **вебе**.

**Не стоит использовать** Thread.sleep(). **Почему?**

1. Он не является частью веб-драйвера API.
2. У него есть достаточно объектов, которые реализованы в WebDriverWait|ExpectedConditions. Лучше пользоваться ими.

**Когда использовать?**

1. Когда ищем проблемы синхронизации, а не решаем их.
2. Заимплиментить какой-то тест (чтобы отправлять пакеты по таймеру)
3. Когда удобно затормозить весь джава процесс/код в режиме отладки.

**Использовать:**

1. Debug
2. Demo
3. Counters
4. Only

**Sync issues solutions - general timeouts**

**Настройка общих временных параметров**

1. Период загрузки страницы (pageLoadStrategy())
2. Время ограничения работы ассинхронного скрипта(executeAsyncScript())
3. Тайм-аут ожидания появления элементов. Глобальный, влияет на все элементы.

**Sync issues solutions - waits**

Методы вебдрайвера, позволяющие наблюдать за изменением статуса конкретных элементов, без оказания влияния на сторонние элементы или глобальные настройки.

1. Explicit/fluent wait
2. Implicitly wail (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 – Explicit waits**

То, с чего начинать правильно решение проблем на уровне кода.

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

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

**Webdriver locators and selectors**

**Локаторы (селекторы)** - выражения, с помощью которых осуществляется поиск элементов на странице.

Селекторы как-то в css-технологии, локаторы – поиск элементов.

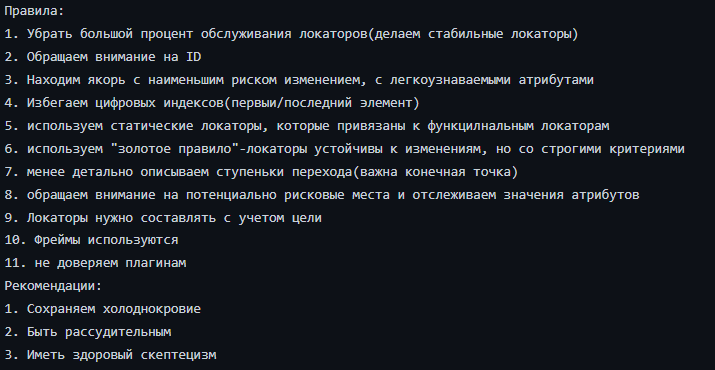
**Можно выбрать элемент через:**

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.

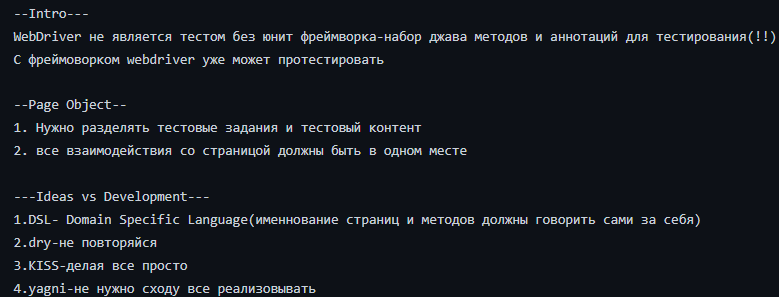
**XPath**



Css

Можно читать справа-налево

**Page Object – паттерн, позволяет абстрагироваться от отдельных html**



Усталь…………=(