Міністерство освіти і науки України

Харківський національний університет ім. В. Н. Каразіна

Факультет комп’ютерних наук

**Курсова робота**

з дисципліни «Математичні методи та технології тестування та верифікації програмного забезпечення»

Тема «Веб-орієнтоване тестування»

|  |  |
| --- | --- |
| Оцінка \_\_\_\_\_ / \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Члени комісії:  Нарєжній О. П.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Мелкозьорова О. М.\_\_\_\_\_\_\_\_\_\_\_ | Виконав:  студент 2 курсу, групи КС- 21 |
| Спеціальності:  122 «Комп’ютерні науки»  Обора Роман Сергійович |
|  |

Харків – 2020

[ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ 3](#_Toc42257155)

[ВСТУП 4](#_Toc42257156)

[1 ЗАГАЛЬНІ ВІДОМОСТІ 7](#_Toc42257157)

[1 Визначення роботи 7](#_Toc42257158)

[2 ДЕТАЛЬНА АРХІТЕКТУРА 9](#_Toc42257159)

[2.1 Компоненти веб-додатків 9](#_Toc42257160)

[1.1.1. Клієнт. 10](#_Toc42257161)

[1.1.2. Сервер. 10](#_Toc42257162)

[1.1.3. База даних. 10](#_Toc42257163)

[2.2 Особливості архітектури клієнт 11](#_Toc42257164)

[2.2.1 Кросбраузерність: різноманітність клієнтів 11](#_Toc42257165)

[2.2.2 Веб-форми на стороні клієнта 13](#_Toc42257166)

[2.3 Особливості архітектури веб-сервер 15](#_Toc42257167)

[2.3.1 Інсталяція на веб-сервер 15](#_Toc42257168)

[2.3.2 Тестування навантаження 16](#_Toc42257169)

[2.3.3 Тестування безпеки 16](#_Toc42257170)

[2.4 Чим відрізняється веб-додаток від декстопного 18](#_Toc42257171)

[2.4.1 Розрахування користувачів. 18](#_Toc42257172)

[2.4.2 Мережа 19](#_Toc42257173)

[2.4.3 Узагальнений опис 19](#_Toc42257174)

[3 ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ 23](#_Toc42257175)

[3.1 Selenium Web Driver 23](#_Toc42257176)

[3.2 Опис дій 24](#_Toc42257177)

[ВИСНОВКИ 26](#_Toc42257178)

[ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЬ 27](#_Toc42257179)

[Додаток А. 28](#_Toc42257180)

[Додаток Б. 29](#_Toc42257181)

**ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ**

Баг (англ. bug— жук) — жаргонізм, що означає помилку, ваду або дефект в програмі або системі, що викликає в ній неправильний або неочікуваний результат або неочікувану поведінку.

База даних (англ. database) – сукупність даних, організованих відповідно до концепції, яка описує характеристику цих даних і взаємозв'язки між їх елементами

Валідація - це визначення відповідності розробляється ПО очікуванням і потребам користувача, вимогам до системи.

Веб (англ. Web - павутина) - інтернет-простір.

Верифікація - це процес оцінки системи або її компонентів з метою визначення чи задовольняють результати поточного етапу розробки умов, сформульованим на початку цього етапу.

Командний рядок (консоль) - це текстовий інтерфейс, в якому команди віддаються шляхом введення текстових рядків з клавіатури.

Систе́ма управління ба́зами да́них (СУБД,  англ. Database Management System, DBMS) — набір взаємопов'язаних даних (база даних) і програм для доступу до цих даних.

Тестування програмного забезпечення - перевірка відповідності між реальним і очікуваним поведінкою програми, що здійснюється на кінцевому наборі тестів, обраному певним чином.

Фреймворк (англ. framework - остов, каркас, структура) - програмне забезпечення, що полегшує розробку і об'єднання різних компонентів великого програмного проекту.

**ВСТУП**

Актуальність даної роботи полягає в високій вагомості веб-орієнтованого тестування, адже на даний момент розробка програмного забезпечення (ПЗ) майже неможлива без даного виду тестування, до того ж завдяки цьому можна виявити такі баги, які могли б пропустити тестувальники.

Новизна роботи полягає в застосуванні засобів автоматичного тестування для виявлення несправностей в ПЗ.

У сучасному світі програмні системи оточують нас всюди. Вони вже давно увійшли в наше життя, стали звичними до користувача, інтуїтивно зрозумілими і простими для користувача. Зараз для нас їх використання - звичайна справа. Купуючи через інтернет квиток на літак, або, публікуючи фотографії в соціальної мережі, ніхто не замислюється, який великий і складний комплекс програмних і апаратних рішень стоїть за цим.

Проте, вимоги, що диктує сучасний світ до програмних систем, з кожним роком стають все складніше, і, як наслідок, самі системи стають складнішими. Зараз великі програмні рішення розробляються компаніями з сотнями і тисячами співробітників. Від діяльності програмних рішень безпосередньо залежить працездатність і прибуток великих підприємств, а в деяких випадках навіть життя людей. Всі ці фактори висувають на перший план питання надійності і відмовостійкості програмного забезпечення. Але також актуальні питання часу і передбачуваності самого процесу розробки.

З плином часу, сам процес розробки змінювався. З'являлися нові інструменти і методології, що дозволяють структурувати, прискорювати сам процес розробки, робити його більш передбачуваним щодо термінів і результату. А також дозволяти швидко підлаштовуватися під мінливі вимоги замовника прямо в процесі розробки, до того, як продукт буде випущений.

При розробці продукту, необхідно, щоб в результаті, після випуску, він працював коректно, без збоїв, а також відповідав вимогам замовника. Але також, бажано, щоб вже в процесі розробки, розробники могли мати частково функціонуючий продукт, з обмеженим, але надійно працездатним і відповідним, вимогам, функціоналом, який згодом буде розширюватися. Це завдання допомагає вирішити такі процедури, як верифікація, валідація та тестування.

Ці процедури допомагають команді об'єктивно зрозуміти, в якому стані зараз знаходиться система і, як наслідок, більш точно планувати свої подальші дії. Зараз великі програмні системи можуть розроблятися десятками людей і команд одночасно і незалежно.

Важливим етапом в житті програмного продукту є його підтримка і еволюція згодом. З'являються нові вимоги ринку, а також побажання від користувачів. Можливо, після установки програми клієнтам, в ньому виявляється якась помилка, яку не помітили при тестуванні. Внаслідок цього, розробники випускають нові версії програми з розширеним функціоналом і виправленими помилками. Але залишається питання, як доставити оновлене додаток клієнтам? А також, як переконатися, що у всіх клієнтів на абсолютно різних комп'ютерах, в різних операційних системах, в різному програмному оточенні, весь функціонал додатка працює так, як потрібно? У різний час ці питання вирішувалися по-різному, але зараз, коли інтернет став широко поширений і загальнодоступний, популярною стала модель веб-додатків. У цій моделі, сам додаток, з усім функціоналом, розташовується на серверах в інтернеті, а для користувача представляється у вигляді веб-сайту, на який він заходить за допомогою свого веб-браузера. Веб-браузер виконує тільки функцію відображення інформації, а робота з самим додатком, відбувається за допомогою запитів на сервер через інтернет. Таким чином, весь програмний код системи прихований, бо знаходиться на відстані від кінцевого користувача, а клієнтові надається тільки інтерфейс для роботи з системою. При такій моделі, постачальнику ПО, не потрібно нічого встановлювати на машини клієнтів і далі підтримувати працездатність продукту на цих машинах, вся система знаходиться в одному місці - на серверах, що робить її підтримку і оновлення простіше для розробників, а використання "прозорим" для користувача.

**1 ЗАГАЛЬНІ ВІДОМОСТІ**

**1 Визначення роботи**

Метою даної курсової роботи є вивчення методів і способів тестування веб додатків.

Об’єктом даної роботи є тестування веб додатків за допомоги засобів автоматичного тестування

Предмет цієї праці це конкретний аспект проблеми тестування додатків.

Як вже було сказано у вступі, в даний час системи, побудовані за принципом веб додатків, набувають все більшої популярності. Ця модель дуже зручна для виробників і постачальників ПО, завдяки простій підтримки системи, оновленню і контролю доступу користувачів. Система розташована в одному місці і видається, як сервіс через інтернет, а не встановлюється кожному користувачеві на комп'ютер. Розробникам потрібно підтримувати працездатність тільки однієї системи, а не всіх копій, встановлених у користувачів. Але така модель висуває також і підвищені вимоги до надійності і відмовостійкості цієї системи. Якщо раніше, при "поломці" програми на конкретному комп'ютері, вона переставала працювати тільки на ньому. То в моделі веб додатків, відмова системи на сервері, може торкнутися відразу всіх користувачів, і, як наслідок, принести великі збитки її постачальникам. Таким чином, питання тестування і перевірки якості додатків дуже актуальне в сучасній розробці.

Так як сучасні веб додатки можуть розроблятися десятками і сотнями людей незалежно, актуальним також є питання інтеграції їх незалежно розроблених модулів в одну систему, і перевірка працездатності отриманої системи. Зараз це завдання вирішується за допомогою систем безперервної інтеграції.

На даний момент найпопулярнішою платформою в світі для створення складних високонавантажених веб додатків є Java. Для створення веб додатків на цій мові існують безліч фреймворків, таких як Spring, JavaServer faces, ZK. Безліч інструментів для тестування коду, таких як JUnit, Mockito [4], Powermock, Selenium[3]. Також, для командної розробки, існують безліч різних інструментів безперервної інтеграції, таких як CruiseControl, Jenkins [6], TeamCity.

Командна розробка великих і складних веб додатків з використанням платформи Java є популярною і розвивається галуззю в сучасному світі, і, як наслідок, на ринку існують безліч продуктів, що дозволяють спростити цей процес і підвищити якість виробленого програмного продукту.

**2 ДЕТАЛЬНА АРХІТЕКТУРА**

**2.1 Компоненти веб-додатків**

Веб-додаток - це клієнт-серверний додаток, в якому клієнтом виступає браузер, а сервером - веб-сервер (в широкому сенсі). Основна частина програми, як правило, знаходиться на стороні веб-сервера, який обробляє отримані запити відповідно до бізнес-логікою продукту і формує відповідь, що відправляється користувачеві. На цьому етапі в роботу включається браузер, саме він перетворює отриману відповідь від сервера в графічний інтерфейс, зрозумілий пересічному користувачеві.

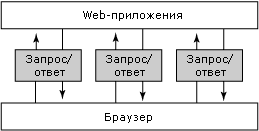


Рисунок 2.1 – схема роботи веб-додатку.

Отже, першою і однією з ключових особливостей веб-додатків є їх архітектура. Давайте більш детально розглянемо це питання, так як він представляє особливу цінність для тестування.

Веб-додаток представлено наступними складовими ( «сторонами»):

**1.1.1. Клієнт.**

Як правило, клієнт - це браузер, але трапляються й винятки (в тих випадках, коли один веб-сервер (ВС1) виконує запит до іншого (ВС2), роль клієнта грає веб-сервер ВС1). У класичній ситуації (коли роль клієнта виконує браузер) для того, щоб користувач побачив графічний інтерфейс програми у вікні браузера, останній повинен обробити отриману відповідь веб-сервера, в якому буде міститися інформація, реалізована із застосуванням HTML, CSS, JS (найбільш використовувані технології ). Саме ці технології «дають зрозуміти» браузеру, як саме необхідно «отрисовать» все, що він отримав у відповіді.

**1.1.2. Сервер.**

Веб-сервер - це сервер, що приймає HTTP-запити від клієнтів і видає їм HTTP-відповіді. Щоб уникнути можливої ​​плутанини, відзначимо, що веб-сервером називають як програмне забезпечення, яке виконує функції веб-сервера, так і безпосередньо комп'ютер, на якому це програмне забезпечення працює. Найбільш поширеними видами ПО веб-серверів є Apache, IIS і NGINX. На веб-сервері функціонує тестоване додаток, яке може бути реалізовано з застосуванням найрізноманітніших мов програмування: PHP, Python, Ruby, Java, Perl та ін.

**1.1.3. База даних.**

У класичній теорії мова йде про двох «сторін» веб-додатки, проте, якщо уважно подивитися на весь процес роботи додатків, ми можемо відзначити, що в алгоритмі роботи веб-а незримо, але досить активно бере участь ще одна «сторона» - база даних. Фактично, вона не є частиною веб-сервера, але більшість програм просто не можуть виконувати всі покладені на них функції без неї, так як саме в базі даних зберігається вся динамічна інформація додатки (облікові, призначені для користувача дані і ін).

База даних - досить широке поняття, яке використовується не тільки в сфері інформаційних технологій. В контексті моєї статті це - інформаційна модель, що дозволяє упорядковано зберігати дані про об'єкт або групі об'єктів, що володіють набором властивостей, які можна категоризувати. Бази даних функціонують під управлінням так званих систем управління базами даних (СУБД). Найпопулярнішими СУБД є MySQL, MS SQL Server, PostgreSQL, Oracle (всі - клієнт-серверні).

Також існують вбудовані і файл-серверні СУБД. Для загального розвитку зазначу лише одну популярну вбудовану СУБД - SQLite, яка використовується в деяких браузерах, Android API, Skype і інших відомих додатках. Взаємодія з перерахованими СУБД засноване на спеціальній мові структурованих запитів - SQL.

* 1. **Особливості архітектури клієнт**

Ми згадали клієнта як першу складову архітектури. У класичній ситуації клієнт представлений браузером, а тому питання тестування кросбраузерності (зважаючи на різноманіття браузерів) саме на часі. Ми також розглянемо тестування заповнюваних форм і тексту як основного джерела інформації, одержуваної через клієнта.

**2.2.1 Кросбраузерність: різноманітність клієнтів**

Що ми розуміємо під тестуванням на кросбраузерність? Це перевірка на правильність (відповідність вимогам і стандартам) відображення і функціонування веб-додатки в різних браузерах і на різних операційних системах. У сучасному світі стандартизація приймає глобальні масштаби, а тому більшість популярних браузерів однаково обробляють код. При цьому необхідність в кросбраузерному тестуванні жевріє, так як далеко не всі проблеми вирішуються стандартизацією.

Перед початком робіт я рекомендую з'ясовувати у відповідальних осіб необхідний для тестованого ПО набір браузерів. Можна самим зібрати статистику по цільовій аудиторії вашого продукту і обмежити кросбраузерне тестування набором найбільш популярних в цій аудиторії браузерів. В умовах регресійного тестування, коли обмежені часові та людські ресурси, також можна застосовувати практику розподілу браузерів: закріплюючи за кожним тестувальником певний браузер, ви зможете «покрити» великий відсоток кросбраузерних дефектів. Цей метод має і свої мінуси: він найбільш результативним у великій команді тестування, але стає практично непридатним при наявності одного тестувальника.

Що необхідно перевіряти при кросбраузерному тестуванні:

1. функціональні можливості продукту, реалізовані на стороні клієнта;
2. правильність відображення елементів графіки;
3. шрифти і розміри текстових символів;
4. доступність і функціональність різноманітних форм, включаючи їх інтерактивність.

У тестуванні потребують всі основні (серед користувальницької аудиторії) браузери, але особливу увагу необхідно приділити IE, якщо він входить в їх число. Саме в ньому дуже часто виникають проблеми, які відсутні в інших браузерах: так, в IE додатково рекомендується звертати особливу увагу на масштабованість, фокус полів і роботу JS.

Окремо рекомендую не забувати про всякого роду валідаторах верстки, наприклад https://validator.w3.org/. Навіть якщо у вас недостатньо знань, щоб оцінити відповідність верстки стандартам, можна використовувати для цього автоматичні засоби і, проаналізувавши результат, вказати розробникам на найсерйозніші «помилки». Не варто забувати, що іноді валідатори звертають увагу на самі «дріб'язкові дрібниці», які ніхто і ніколи виправляти не буде. Якщо ви і заводите баг-репорти на подібного роду зауваження, то зручніше буде зібрати їх в єдиний документ і прикріпити до репорту. До такого роду «дрібниць» можна віднести всілякі рекомендації, які не мають свого впливу на відображення і функціонування контенту.

**2.2.2 Веб-форми на стороні клієнта**

Однією з важливих складових інтернет-додатків є форми для заповнення, взаємодія з якими користувач здійснює за допомогою все того ж пильно розглянутого нами клієнта. Однак, дані форми дуже часто служать джерелом дефектів, які, влаштувавшись в «продакшені», можуть принести великі фінансові та репутаційні збитки компанії.

Як же не пропустити дефекти в формах на продакшен? Розглянемо кілька простих кроків:

1. Ретельно перевіряємо обов'язковість заповнення полів і наявність відповідного маркування у них.

2. Заповнивши і відправивши форму, переконуємося в тому, що з даними відбувається саме те, що заплановано. Якщо дані повинні бути внесені в базу даних, перевіряємо, чи коректно завершився процес (в кінці кінців, про це можна попросити розробника, якщо не вистачає своїх знань SQL або прав доступу до БД).

3. Використовуємо чит-листи для тестування форм, наприклад чит-лист реєстрації від Олексія Лупана або чит-лист по Web UI контролю від Ігоря Любина.

4. Перевіряємо, виводяться чи зрозумілі користувачеві інформаційні повідомлення про необхідність заповнення порожніх полів після спроби відправити форму.

5. Звертаємо увагу на реалізацію екранування символів в полях форм, які є потенційним джерелом вразливостей для програми та користувачів. Екранування має здійснюватися на рівні не тільки клієнта, але і сервера, відключити який в клієнті досить просто (наприклад, за допомогою спеціальних плагінів, що знімають всі можливі обмеження в кілька кліків, таких як Web Developer Toolbar - Forms).

6. Переконуємося, що після заповнення форми користувачеві приходить підтверджуючий лист із зазначенням відповідного відправника, а саме тіло листа відповідає вимогам (в тому числі і на працездатність посилань).

7. Використовуємо допоміжні спеціальні інструменти для тестування форм (наприклад, Web Developer Toolbar).

Текст, як основне джерело інформації при роботі через клієнт

З якої сторони не поглянути, але особлива цінність мережі Інтернет полягає в тому, що вона є практично безмежним джерелом інформації. Частина цієї інформації представлена ​​у вигляді текстів, з якими, знову ж таки, користувач взаємодіє за допомогою клієнта. Більшість веб-ресурсів в тому чи іншому обсязі вимагають перевірки текстів на предмет відсутності граматичних помилок і друкарських помилок.

Звичайно, значимість цього тестування не така велика в порівнянні з функціональним напрямом, але нехтувати ним не варто. На практиці мені довелося зустрітися з дуже серйозною помилкою: на продакшені одного з найбільших інтернет-гіпермаркетів Росії була допущена помилка, яка змінила горду назву величного міста Ярославль на сумне образливе слово. Назва служило частиною суб-меню, а тому було помітно всім жителям міста з півмільйонним населенням.

Так, ми не завжди маємо достатньо часу для вичитування всіх текстів, але в таких ситуаціях на допомогу приходять «SpellCheker-и» (програми для перевірки орфографії, онлайн або у вигляді плагінів для браузерів, або десктопні программи), наприклад, Яндекс.Спеллер чи Punto Switcher.

* 1. **Особливості архітектури веб-сервер**

Тестування частини веб-додатки, розміщеної на веб-сервері, можна провести і минаючи графічний (клієнтський) інтерфейс, проте це вимагає від фахівця певного рівня знань і навичок технічного характеру, а також застосування додаткових інструментів. Розглянемо веб-сервер з точки зору навантажувального і інсталяційного тестування.

**2.3.1 Інсталяція на веб-сервер**

Отже, перш ніж приступити до тестування, ми повинні встановити (інсталювати) веб-додаток на веб-сервер. Власне, в цьому є подібність з перевіркою десктоп-додатків, але існує і відмінність в нюансах, які необхідно врахувати і протестувати, особливо якщо це стосується ПО, розповсюджуваного для локальної інсталяції на веб-сервери користувачів.

У чому ж полягають ці нюанси?

1. Велика частина веб-додатків вимагає для інсталяції специфічних знань в адмініструванні ОС. Спробуйте встановити додаток на декількох веб-серверах. В оптимальному випадку це будуть найпопулярніші технології серед ваших користувачів, для встановлення переліку яких буде потрібно попереднє дослідження.

2. інсталюємо веб-додаток, звертайте увагу на те, чи дійсно додаток встановлюється в зазначену вами директорію, базу даних, використовує обраний вами префікс і дотримується інші конфігураційні моменти.

3. Переконайтеся, що додаток можна встановити як з localhost, так і віддалено.

4. Якщо процес інсталяції є інтерактивним, і кожен вибір користувача на певному етапі впливає на подальші дії, то необхідно буде пройти всі гілки, так як саме інсталяція є першим кроком користувача в роботі з вашим додатком.

5. Не забувайте про негативні тестах: перевірки в умовах недостатності ресурсів (місця на диску, оперативної пам'яті) або привілеїв, переривання процесу установки під час активної його фази (наприклад, відправляючи сервер в перезавантаження).

**2.3.2 Тестування навантаження**

Тестування навантаження імітує роботу з додатком певної кількості користувачів. Цей вид тестування здійснюється за допомогою спеціальних інструментів (наприклад, jMeter), головна мета яких - визначити профілі навантаження і штучно створити для них навантаження, що виявляє граничні можливості програми (або сервера) в умовах роботи з ним тієї чи іншої кількості користувачів.

Отримана інформація піддається ретельному аналізу з подальшим виявленням «вузьких шийок» і граничних програмних і апаратних можливостей, які в подальшому використовуються для забезпечення стабільності веб-сервера і самого додатка, що працює на ньому.

Наведемо приклад з практики функціонування великого комерційного продукту, який довгий час працював з різноманітними типами договорів. Одного разу в реліз випустили черговий особливо очікуваний тип договору, і на наступний день система повністю перестала працювати, а служба підтримки була завалена величезною кількістю звернень. Всьому виною став прорахунок в тестуванні: команда перевіряла одночасну роботу з десятками тисяч договорів, але ніхто не зміг передбачити, що на практиці мова піде про сотні тисяч, а іноді і про мільйони договорів. Тестування навантаження дозволяє виявити потенційні проблеми такого характеру ще на етапі тестування.

**2.3.3 Тестування безпеки**

Тестування безпеки - окремий напрямок тестування, яке вимагає від фахівця фундаментальних знань технічного характеру і хорошою профільної кваліфікації. Я зазначу ряд загальних моментів, які можуть допомогти будь-якому тестувальника знаходити класичні уразливості, не допускаючи їх вихід на продакшен. Питання безпеки додатків регламентуються OWASP Guide, CHECK, ISACA, NIST Guideline, OSSTMM.

Існує ряд принципів безпеки, до яких відносяться конфіденційність, цілісність і доступність:

1) Конфіденційність - обмеження доступу до тієї чи іншої інформації для певної категорії користувачів (або навпаки надання доступу лише обмеженій категорії).

2) Цілісність включає в себе:

1. можливість відновити дані в повному обсязі при їх пошкодженні;
2. доступ на зміну інформації тільки певної категорії користувачів.

3) Доступність - ієрархія рівнів доступу і чітке їх дотримання.

Перерахуємо класичні уразливості сучасних веб-додатків:

1. XSS - генерація на сторінці продукту скриптів, які становлять небезпеку для користувачів продукту;

2. XSRF - вразливість, при якій користувач переходить з довіреної сторінки на шкідливу, де крадуться представляють цінність для користувача дані;

3. Сode injection (PHP, SQL) - ін'єкція частини виконавчого коду, яка робить можливим отримати несанкціонований доступ до програмного коду або базі даних і вносити в них зміни;

4. authorization bypass - це вид уразливості, при якому можна отримати несанкціонований доступ до облікового запису або документам іншого користувача;

5. переповнення буфера - явище, якого можна досягти у шкідливих цілях, по своїй суті являє використання місця для запису даних далеко за межами виділеного буфера пам'яті.

При тестуванні рекомендую використовувати чит-листи вразливостей XSS Filter Evasion Cheat Sheet і MySQL SQL Injection Cheat Sheet.

**2.4 Чим відрізняється веб-додаток від декстопного**

Веб-додаток і повноцінний додаток звісно дуже відрізняються, і хоча їх не можна порівнювати через різні архітектурні і реалізаційні складнощі, нижче буде приведено їх основні різниці, які визначають їх.

**2.4.1 Розрахування користувачів.**

Широта аудиторії додатків накладає свій відбиток на специфіку роботи.

1. Один додаток одночасно може використовуватися величезною кількістю людей. Ми вже розглядали питання навантажувального тестування, але також слід звернути увагу на те, що в число користувачів можуть входити представники різних культур, мов і релігій. Нам необхідно пам'ятати про це, особливо якщо мова йде про тестування міжнародного додатка.

2. Кожен користувач може мати свої рівні доступу. В ідеальному варіанті тестувальник створює для себе матрицю рівнів доступу і тестує кожен доступ окремо. По кліку на картинку відкриється повна версія.

3. Користувачі з одним рівнем доступу можуть звертатися до одних і тих же сутностей, що призводить до конкурентного доступу. Тестується це досить просто. Для прикладу розглянемо систему, що має справу з договорами, які можна створювати, публікувати, редагувати, анулювати. Алгоритм роботи такий: під декількома вікнами в режимі інкогніто авторизуємся в додатку під користувачами з різними рівнями доступу; далі обрану для тесту сутність відкриваємо на редагування, а під другий обліковим записом цю ж сутність пробуємо перевести в статус «Анульовано» - на цьому етапі повинен спрацювати контроль на конкурентний доступ. Операція анулювання блокується, а користувачу видається повідомлення про те, що сутність редагується іншим користувачем (поведінку і пріоритет дій визначаються відповідно до вимог і особливостями продукту, але логіка не змінюється).

4. Широта аудиторії говорить про те, що за монітором може перебувати людина, що має злий намір щодо вашого ПО, адже з веб-ресурсу набагато легше вкрасти данні для незаконних дій.

**2.4.2 Мережа**

Веб-додатки активно використовують мережу, і це є джерелом можливих проблем. Такий, наприклад, є використання додатки в умовах низької швидкості передачі даних (в браузер Google Chrome, наприклад, вбудована функція Throttling, яка дозволяє сильно занижувати швидкість передачі даних), в умовах втрати пакетів або при відключенні мережі під час активної фази роботи програми (спосіб імітації: спочатку робимо швидкість передачі даних за допомогою Throttling мінімальної, а потім перериваємо мережеве з'єднання під час обробки запиту).

У будь-якому з описаних вище випадків програма має працювати коректно. При «падінні» запиту (time out) або іншої проблеми ми повинні, перезагрузив сторінку, знову отримати повністю працює веб-додаток без будь-якого натяку на щойно пережита «шкоду».

**2.4.3 Узагальнений опис**

1. Технологічні відмінності:

1. Класичний додаток працює з використанням однієї або сімейства споріднених технологій.
2. Web-додаток працює з використанням принципово різних технологій.

2. Структурні відмінності:

1. Класичний додаток "монолітний". Складається з одного або невеликої кількості модулів. Не використовує сервери БД, web-сервери і т.д.
2. Web-додаток - "багатокомпонентний". Складається з великої кількості модулів. Обов'язково використовує сервери БД, web-сервери, сервери додатків.

3. Відмінності режимів роботи:

1. Класичний додаток працює в режимі реального часу, тобто відомо про дію користувача відразу ж, як тільки вона виконана.
2. Web-додаток працює в режимі "запит-відповідь", тобто відомо про деяке наборі дій тільки після запиту на сервер.

4. Відмінності формування інтерфейсу:

1. Класичний додаток використовує для формування інтерфейсу користувача щодо усталених і стандартизованих технологій.
2. Web-додаток використовує для формування користувацького інтерфейсу технології, які стрімко розвиваються , безліч яких конкурує між собою.

5. Відмінності роботи з мережею:

1. Класичний додаток практично не використовує мережеві канали передачі даних.
2. Web-додаток активно використовує мережеві канали передачі даних.

6. Відмінності запуску і зупинки:

1. Класичний додаток запускається і зупиняється тільки одного разу за бажанням користувача, або в разі помилки, тобто рідко.
2. Web-додаток запускається і зупиняється за фактом надходження кожного запиту, тобто дуже часто.

7. Різниця в кількості користувачів:

1. Класичнй додаток: кількість користувачів, які одночасно використовують додаток, контрольовано, обмежено і легко прогнозовано.
2. Web-додаток: кількість користувачів, одночасно використовують додаток, складнопрогнозовано і може стрибкоподібно змінюватися в широких діапазонах.

8. Особливості збоїв і відмов:

1. Класичний додаток: вихід з ладу тих чи інших компонентів відразу стає очевидним.
2. Web-додаток: вихід з ладу деяких компонентів надає непередбачуваний вплив на працездатність додатка в цілому.

9. Відмінності в інсталяції:

1. Класичний додаток - процес інсталяції стандартизований і максимально орієнтований на широку аудиторію користувачів. Не вимагає специфічних знань. Додавання компонентів додатка виконується стандартним способом з використанням одного і того ж інсталятора.
2. Web-додаток - процес інсталяції часто недоступний кінцевому користувачеві. Інсталяція вимагає специфічних знань. Процес зміни компонентів програми не передбачається або вимагає кваліфікації користувачів. Інсталятор відсутній.

10. Відмінності в деінсталяції:

1. Класичний додаток: процес деінсталяції стандартизований і виконується автоматично або напівавтоматично.
2. Web-додаток: процес деінсталяції вимагає специфічних знань для втручання адміністратора і часто пов'язаний зі зміною коду середовища функціонування програми, БД, налаштування системної ОС.

11. Особливості середовища функціонування:

1. Класичний додаток: середовище функціонування стандартизоване і не сильно впливає на функціонування програми.
2. Web-додаток: середовище функціонування дуже різноманітне і може мати серйозний вплив на працездатність і серверної, і клієнтської частини.

**3 ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ**

* 1. **Selenium Web Driver**

Для проведення тестування було взято програмну платформу Selenium Web Driver, основні переваги:

1. Безкоштовний;
2. Має розвинене співтовариство;
3. Тести можна зберігати як html файли;
4. Можна експортувати сценарії в скрипти на різних мовах

програмування;

1. Підтримка користувальницьких розширень (використовує JS);
2. Працює з усіма A-grade браузерами;
3. Підтримка кластеризації;
4. Можливість логування в файл;

Але, у Selenium Web Driver є деякі недоліки:

1. Тестування виключно веб-додатків;
2. Вимагає JavaScript. Підтримка JS браузером є обов'язковою вимогою;
3. Selenium IDE доступний тільки для Firefox;
4. Підтримує запис тільки послідовностей дій в якості тесті;
5. Складно комбінувати складові дії, особливо після експорту в будь-який скрипт;
6. Не підтримує Flash і Java аплети;
7. Чи не дозволяє використовувати інші проксі, тому що технічно сам є проксі;
8. Обмежена підтримка Data Driven Testing;
9. Немає підтримки обробки помилок;
10. Немає влаштованої підтримки створення звітів;
11. Відсутність технічної підтримки (на відміну від комерційних альтернатив).
    1. **Опис дій**

Для початку роботи з програмним забезпеченням необхідно встановити geckodriver.exe.

Всі необхідні імпорти згенеруются під час роботи в IDE.

Після цього потрібно створити новий екземпляр драйвера Chrome, з ваказанням абсолютного шляху до geckodriver.exe. в якості другого параметру, ці дії проводятся в анотації @BeforeClass, що означає, що ці дії будуть зроблені один раз за весь клас, приклад наведено нижче:

Лістинг 1.

public static void TestBeforeMethod() throws Exception {

System.setProperty("webdriver.chrome.driver", "D:\\QA\\chromedriver.exe");

DRIVER = new ChromeDriver();

}

Після цього спробуємо дати неявне очікування - сказати WebDriver запитати DOM протягом певного часу, при спробі знайти елемент або елементи, якщо вони не одразу доступні. Установка за замовчуванням 0. Після встановлення неявного очікування встановлюється термін дії об'єкта WebDriver.:

DRIVER.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

Виконавши вищеописане далі збільшимо вікно на повний екран веб-переглядача для повноцінного розміщення елементів на екрані:

DRIVER.manage().window().maximize();

Відвідаємо сайт google.com:

DRIVER.get("http://www.google.com");

Перевіримо заголовок сторінки:

System.out.println("Page title is: " + DRIVER.getTitle());

Знайдемо елемент введення тексту за його назвою:

WebElement element = DRIVER.findElement(By.name("q"));

Введімо дещо для пошуку:

element.sendKeys("Selenium!");

Надішліть форму. WebDriver знайде форму для нас із елемента:

element.submit();

Закриваємо браузер:

public static void ActionsAfterClass () {

DRIVER.close();

}

**ВИСНОВКИ**

В процесі даної работи було освітленно питання веб-орієнтованого тестування, а також ознайомлено з його основними аспектами, а також створено програмне забезпечення що тестує сайт, отримані навички під час виконання будуть використані для подальшої роботи.

Підводячи підсумки, я хочу ще раз акцентувати увагу читача на те, що «веб» розвивався, розвивається і буде розвиватися, а кількість використовуваних технологій, як і різноманітність дефектів, - збільшуватися. Можливо, колись прийде захід ери «веб-а», але це не станеться в найближчі 5-10 років, тому знання основ і розуміння суті веб-додатків просто безцінне. Будь-який тестувальник рано чи пізно торкнеться «веб-у» своєї професійною «руйнівної» рукою, але тільки гарний фахівець отримає з цього максимально наближений до необхідного результат.

Важливо пам'ятати, що тестування ПО ставить перед кожним,хто обрав стрункі ряди сфери забезпечення якості ПО, такі завдання, які практично неможливо вирішувати однозначно і за чітким алгоритмом. Тестування - це філософія, творчість, політ думки, заснований на чітких і беззастережних технічних аспектах. Мабуть, тільки тестувальникам доводиться одночасно бути розробником, системним аналітиком, користувачем, замовником (для якого першочерговий інтерес - фінансовий чи інший успіх продукту), фахівцем в предметної області і навіть DBA.

Кожен з цих навичок QA Engineer повинен розвивати постійно, і тоді він зможе уникнути багатьох можливих проблем і залишитися затребуваним висококваліфікованим професіоналом.

**ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЬ**

1. Котляров В.П. «Основы тестирования ПО»Издательство «БИНОМ» 2006г.
2. Стаття «Continuous Integration» 2020 URL: https://martinfowler.com/articles/continuousIntegration.html (дата звернення 13.05.2020)
3. Офіційний сайт проекта Selenium URL:<http://www.seleniumhq.org/>(Дата звернення 17.05.2020)
4. Офіційний сайт проекта Mockito. URL:<http://site.mockito.org/>(дата звернення 20.05.2020)
5. Документація Java по фреймворку Mockito URL:<http://static.javadoc.io/org.mockito/mockitocore/1.10.19/org/mockito/Mo> ckito.html (дата звернення 21.05.2020)
6. Офіційний сайт проекта Jenkins. URL: https://jenkins.io/ (дата звернення 22.05.2020)
7. Стаття «Тестирование программного обеспечения: модульные тесты». URL : <http://openquality.ru/software-> testing/unit-tests.php (дата зверення 24.05.2020)
8. Стаття «Непрерывная интеграция». URL: [http://lib.custis.ru/Непрерывная\_интеграция](http://lib.custis.ru/%D0%9D%D0%B5%D0%BF%D1%80%D0%B5%D1%80%D1%8B%D0%B2%D0%BD%D0%B0%D1%8F_%D0%B8%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B0%D1%86%D0%B8%D1%8F) (дата звернення 26.05.2020)
9. Стаття «Непрерывная интеграция, Jenkins и Middleman». URL : https://mkdev.me/posts/nepreryvnaya- integratsiya-jenkins-i-middleman (дата звернення 27.05.2020)

**Додаток А.**

**Приклад застосування Selenium**

import org.junit.BeforeClass;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import java.util.List;

import java.util.concurrent.TimeUnit;

public class Test {

static WebDriver DRIVER;

@BeforeClass

public static void TestBeforeMethod() throws Exception {

System.setProperty("webdriver.chrome.driver", "D:\\QA\\chromedriver.exe");

DRIVER = new ChromeDriver();

}

@org.junit.Test

public void Text (){

DRIVER.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

DRIVER.manage().window().maximize();

DRIVER.get("http://www.google.com");

System.out.println("Page title is: " + DRIVER.getTitle());

WebElement element = DRIVER.findElement(By.name("q"));

element.sendKeys("Selenium!");

element.submit();

}

@AfterClass

public static void ActionsAfterClass () {

DRIVER.close();

}

}

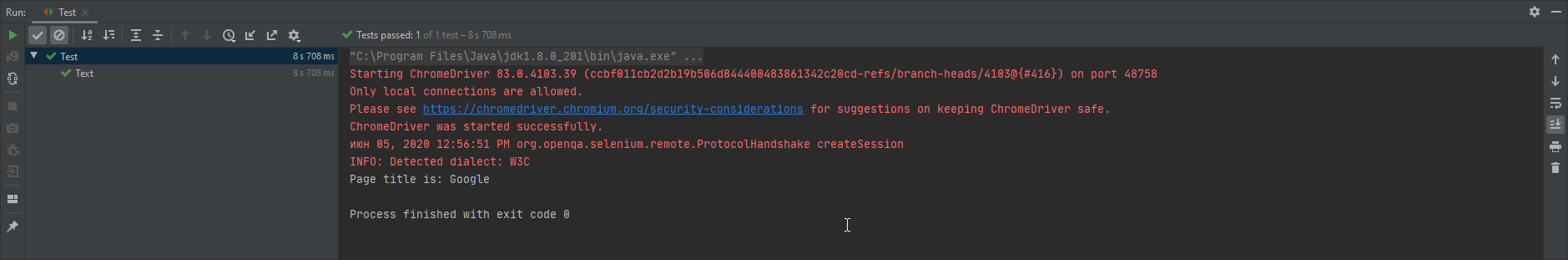
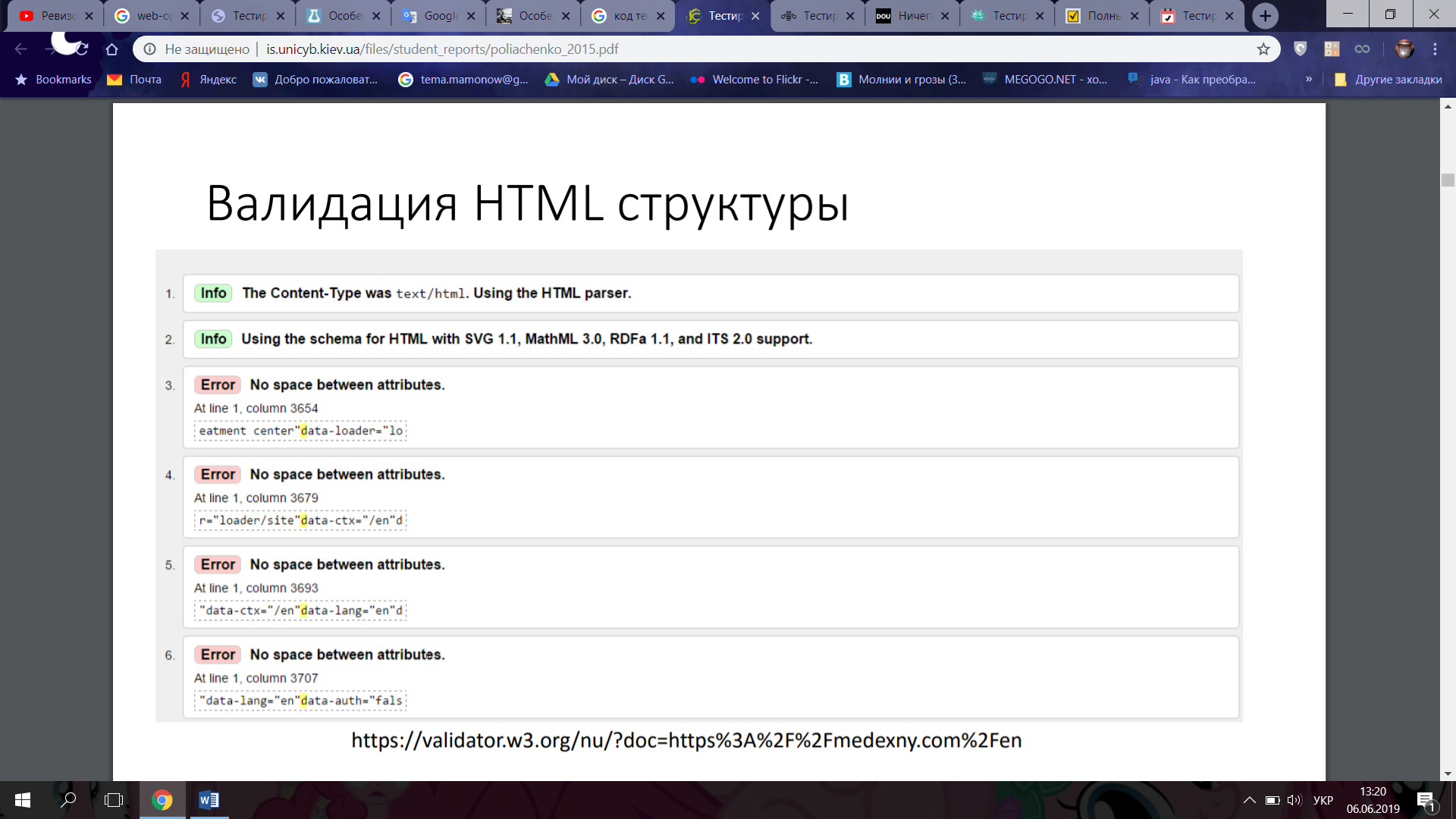


Рисунок А.1 – результат тестування.

**Додаток Б.**

**Валідація HTML структури**

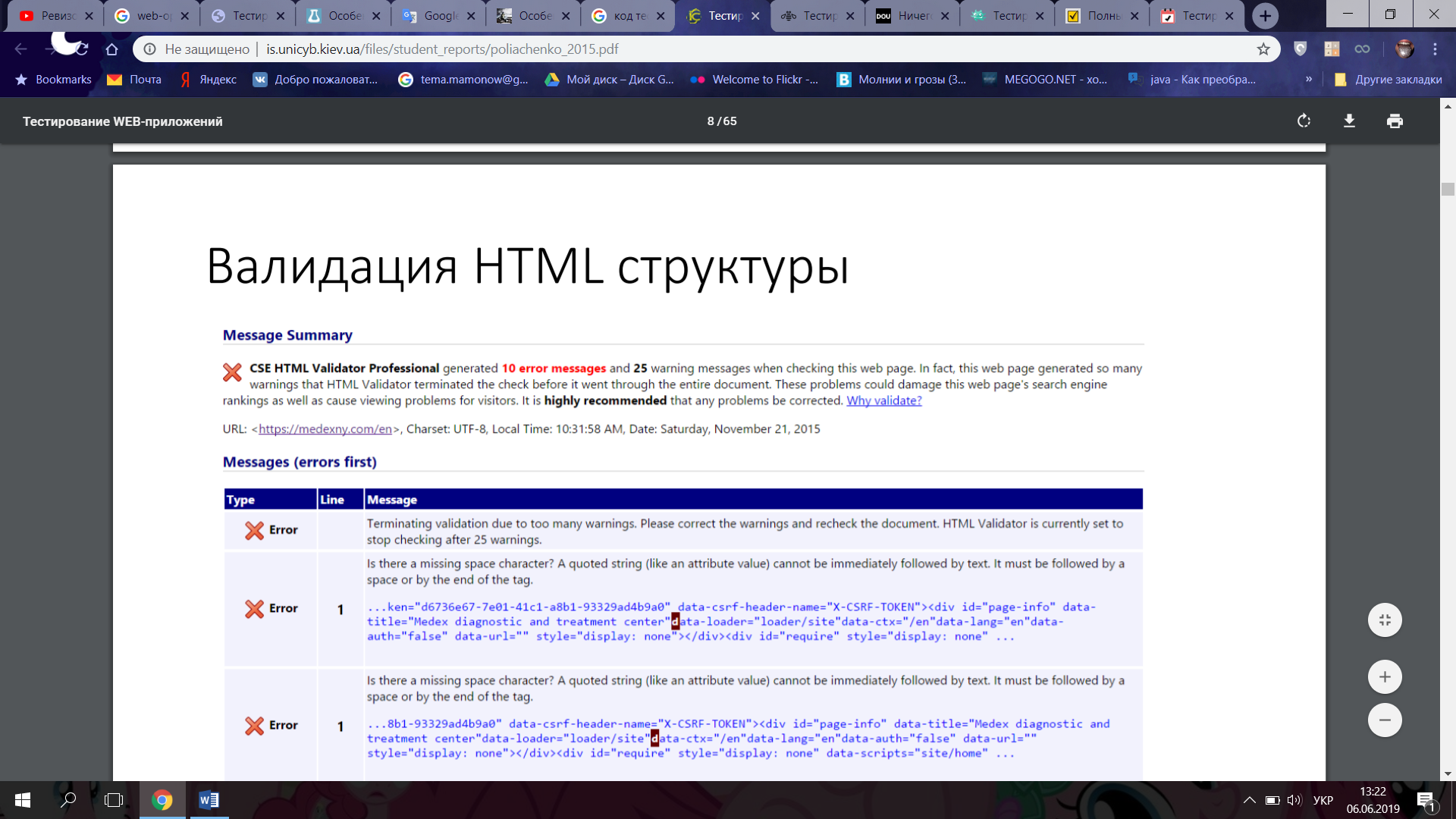
Рисунок Б.1 – Валідація HTML структури за допомоги validator.w3.org .

Рисунок Б.2 – Валідація HTML структури за допомоги onlinewebcheck.com .