Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

(повна назва факультету)

(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

(назва освітнього ступеня)

на тему: _____

Виконав(ла): студент(ка) _____ курсу, групи _____

спеціальності _____

(шифр і назва спеціальності)

(підпис) (прізвище та ініціали)

Керівник _____ _____
(підпис) (прізвище та ініціали)

Нормоконтроль _____ _____
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ _____
(підпис) (прізвище та ініціали)

Рецензент _____ _____
(підпис) (прізвище та ініціали)

Тернопіль
20___

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет _____
(повна назва факультету)

Кафедра _____
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ _____
(підпис) (прізвище та ініціали)
« » 20___ р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня _____
(назва освітнього ступеня)

за спеціальністю _____
(шифр і назва спеціальності)

студенту _____
(прізвище, ім'я, по батькові)

1. Тема роботи _____

Керівник роботи _____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «___» _____ 20___ року № _____

2. Термін подання студентом завершеної роботи _____

3. Вихідні дані до роботи _____

4. Зміст роботи (перелік питань, які потрібно розробити)

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка

Студент _____ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота на тему «Розробка додатку автоматизації та тестування веб-додатків з використанням технологій JavaScript» на здобуття освітньо-кваліфікаційного рівня «Бакалавр» зі спеціальності «Інженерія програмного забезпечення» написана на 54 сторінках і містить 15 ілюстрацій, 3 таблиці і 12 списків використаних джерел та літератури.

Метою цієї дипломної роботи є створення системи автоматизованого тестування програмного забезпечення веб-додатків шляхом поєднання різних методів тестування та налагодження безперервної інтеграції проекту.

Методологія дослідження. Взято до уваги існуючі технології. Порівняно та проаналізовано технології, визначені в мові Javascript. Система була реалізована з використанням безперервної інтеграції та управління цією конфігурацією.

Результати дослідження містять порівняльну характеристику та опис методів тестування на всіх рівнях тестування веб-додатків, а також програмну реалізацію для автоматизованого тестування інтерфейсів прикладного програмного забезпечення, що, як очікується, підвищить якість програмного забезпечення та полегшить його розробку. Результати дослідження можуть бути використані для побудови унікального фреймворку автоматизованого тестування на JavaScript та створення системи звітності за проектом.

Ключові слова: веб-додатки, безперервна інтеграція, JavaScript, автоматизоване тестування, програмне забезпечення, піраміда тестування.

SUMMARY

The thesis on the topic "Development of additional automation and testing of web applications using JavaScript technologies" for obtaining the educational and qualification level "Bachelor" in the specialty "Software Engineering" is written on 54 pages and contains 15 illustrations, 3 tables and 12 lists of used sources and literature.

The method of this thesis is to create a system of automated software testing of web applications by combining different testing methods and overlaying continuous integration of the project.

Research methodology. Existing technologies are taken into account. The technologies defined in the Javascript language are compared and analyzed. The system was implemented using continuous integration and management of this configuration.

The results of the study compare the benchmarking and description of testing methods at all levels of web application testing, as well as the software implementation for automated testing of application software interfaces, which is expected to improve software quality and facilitate its development. The research results can be used to build a unique JavaScript automated testing framework and create a project reporting system.

Key words: web applications, continuous integration, JavaScript, automated testing, software, testing pyramid.

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

АТ – автоматизоване тестування

ЖЦАТ – життєвий цикл автоматизації тестування

ПЗ – програмне забезпечення

VM – віртуальна машина

БД – база даних ОС – операційна система

CI – сервер інтеграції

API – прикладний програмний інтерфейс

GUI – графічний інтерфейс користувача

ВСТУП

У повсякденному житті програмне забезпечення має великий вплив практично на усі сфери нашого життя. Розробка і підтримка певного продукту забезпечення набуває чи малих обертів кожного дня, а задача з конструюванням і зменшенням витрати часу на виготовлення того чи іншого продукту зазнає не малих змін адже все це є дуже актуальним в теперішній час. Практично усі сфери діяльності починаючи від комфортного для нас обслуговування в магазині до високотехнологічної медицини використовують ті самі програмні забезпечення. Постійно іде впровадження нових технологічних систем для покращення роботи в майбутньому і зменшення витрати часу на обробку того чи іншого продукту. Але не всі замислюються, наскільки важкий цей процес налагодження процесу розробки і його тестування, наскільки складна ця робота у користувача, який це все перевіряє, налаштовує.

Однак вимоги до команди розробників з кожним роком стрімко зростають. На сьогодні програмне забезпечення рішення для сотень співробітників, які працюють одночасно, вартість пошуку помилок під час розробки не дуже висока адже необхідно з’ясувати в програмному коді, що саме викликає це, виправити це та перевірити знову. Але якщо кінцеві користувачі знайдуть деякі програмні помилки – вартість виправлення багаторазово збільшується і іноді це може навіть забрати життя, якщо ми говоримо про медицину або військову сферу. Тому більшість сучасних IT-компаній витрачають значний бюджет для тестування програмного забезпечення де це можливо. Існує звісно безліч методів, напрямків і випробувань.

Існує два основних види тестування – ручне і автоматизоване. Все більше компаній прагнуть автоматизувати процес тестування сам по собі, але повністю позбутися ручного тестування все одно неможливо. Іноді потрібно один раз

перевірити і втратити час, тому що автоматизувати цей процес просто немає сенсу.

Ручне тестування: Це традиційний підхід, де тестування виконується вручну тестувальниками. Вони вручну перевіряють різні аспекти додатку, такі як функціональність, взаємодія з користувачем та сумісність з різними пристроями та браузерами. Ручне тестування може бути ефективним для виявлення непередбачених проблем та візуального аналізу, але воно може віднімати досить багато часу і не ефективним для виконання повторюваних та об’ємних тестів.

Автоматизоване тестування: Цей підхід використовує автоматично написані тести та інструменти для їх виконання. Використання JavaScript-фреймворків, таких як Selenium або Cypress, дозволяє автоматизувати взаємодію з веб-додатком та перевірку його функціональності. Автоматизоване тестування забезпечує швидке виконання тестів, регулярне виконання регресійних тестів та зниження ризику помилок, пов’язаних з ручним тестуванням.

Метою цього проекту є підвищення якості програмного забезпечення шляхом його створення. Автоматизовані тестові моделі повинні забезпечувати надійність роботи. Простота веб-додатку та його подальший розвиток і підтримка.

Науково-практична тема, яка вирішується в даній дипломній роботі, включаючи такі завдання:

1) Аналіз та порівняння існуючих технологій автоматизації тестування веб-додатків
2) Побудова автоматизованих моделей для всіх рівнів тестування
3) Написати автоматизовані тести користувача інтерфейсу
4) Налаштувати безперервний процес інтеграції даних модель

1. АНАЛІЗ ВИДІВ ВЕБ-ДОДАТКІВ ТА СПОСОБІВ ЇХ ПОБУДОВИ

1.1. Механізми створення та види веб-додатків

Веб-додаток — це попередньо створене програмне забезпечення, яке на запити користувачів надає попередньо створений гіпертекстову інформацію. Можна вважати сайт і веб-додаток одним і тим же поняттям. З точки зору архітектури, кожен веб-додаток повинен бути створений на основі клієнт-серверу. Користувач надсилає запит на веб-додаток, веб-додаток його обробляє і надсилає через мережу на веб-сервер. У відповідь сервер створює веб-додаток та надсилає її назад через мережу. Веб-додаток у свою чергу інформацію, отриману з сервера, відображає користувачеві. Як правило, веб-додатки використовують браузер, як інтерфейс користувача.

Сучасні веб-додатки – це власні розробки, орієнтовані на вирішення певного кола завдань. Вони є власними розробками, орієнтованими на вирішення певного кола завдань. Попит на них зростає через те, що це дає змогу ефективніше здійснювати бізнес-процеси, збільшити кількість сервісів. Для того, щоб збільшити кількість послуг і поліпшити якість і зручність їх подачі, компанії замовляють розробку веб-додатків, компанії замовляють розробку веб-додатків. Гроші вкладені в його розробку швидко виправдовують себе окупаючись.

Виділяють основні види веб-додатків:

- форуми
- форми відправки повідомлень
- веб-пошта
- системи голосування
- рахівники
- гостьові книги
- форми для завантаження та скачування

• движки сайту

• веб-системи, що надають послуги для своїх користувачів [1].

Більшість систем у сучасному суспільстві не є досконалими, і про їхні вразливі місця потрібно говорити. Про вразливість також необхідно згадати:

• При роботі з файлами, коли вони передаються ззовні в додаток. (GET або POST);

• Некоректна обробка вхідних даних;

• Некоректне зберігання, передача або обробка паролів;

• Неврахування особливостей роботи програми для завантаження файлу на сервер;

• Неврахування особливостей роботи програми завантаження файлів на сервер;

• Неврахування особливостей роботи програми

• Неправильна логіка під час роботи веб-додатку, що доводить до непередбачуваних результатів, якщо вхідні дані знаходяться в допустимих межах

• Відображення додаткової службової інформації, яка може бути використана зловмисником.

• Недостатня безпека бази даних (наприклад, введення паролів або виведення службової інформації, величезна кількість запитів при обробці).

• Уразливості в обробці вхідних даних при роботі з базами даних (різні види SQL-ін’єкцій).

• Неоптимізований програмний код може створювати велике навантаження на веб-сервер.

• Вразливість веб-додатків і систем до DDoS і DoS-атак.

• Відсутність валідації даних під час введення користувачем.

Класифікуємо різні типи веб-додатків. Ця класифікація базується на тому, як веб-додаток відображає отриманий зміст. На основі цієї класифікації маємо до 6 різних типів веб-додатків.

Статичні веб-програми:

Якщо ви створюєте статичний веб-додаток, по-перше, що потрібно усвідомлювати це те, що цей тип веб-додатків показує дуже мало контенту і він не є особливо гнучким.

Статичні веб-програми зазвичай розробляються за допомогою HTML і CSS. Це не єдина платформа для розробки статичних додатків. Також можна використовувати jQuery та Ajax. Крім того також дозволяють легко відображати анімовані об'єкти, такі як банери, GIF-файли та відео та інші.

На жаль, змінити вміст статичного веб-додатку не так просто. Для цього потрібно спочатку завантажити HTML-код, потім змінити його і відправити назад на сервер. Тільки веб-майстер або команда розробників, яка планувала і розробляла веб-сайт, може вносити ці зміни.

Приклади розроблених статичних веб-додатків включають професійні портфоліо та цифрові резюме. Вони дуже прості в розробці і їх можна спроектувати та розробити за кілька хвилин. Аналогічно, сторінка профілю компанії. Цей тип веб-додатків можна використовувати для відображення інформації про компанію.

Динамічні веб-додатки:

Динамічні веб-додатки є більш складними на технічному рівні. Вони використовують базу даних для завантаження даних, вміст якої оновлюється щоразу, коли користувач звертається до неї або коли відбувається якась подія в місці. Список можливих подій в DOM дуже довгий: drag (перетягування), click (натиск мишею), touch (торкання), load (завантаження), input (введення), change (зміна), error (помилка), resize (зміна розміру) і так далі. Події можуть бути ініційовані для будь-якої частини документа в результаті взаємодії користувача або браузера. Події можуть запускатися в будь-якій частині документа в результаті взаємодії користувача або браузера. Події не просто починаються і закінчуються в одному місці. Вони циклічно проходять через весь документ, проходячи свій власний життєвий цикл. Ось чому DOM-події є надзвичайно гнучкими та корисними. Розробники та тестувальники повинні розуміти, як працюють DOM-події, щоб мати можливість використовувати їхній потенціал та створювати інтерактивні інтерфейси [2]. Зазвичай динамічні веб-додатки мають панель

адміністрування (так звану CMS), яка дозволяє адміністратору редагувати або змінювати вміст, наприклад, текст і зображення.

Для розробки динамічних веб-додатків використовуються різні мови програмування. Найчастіше вони використовуються для опису внутрішньої частини:

• Сценарні: PHP, Ruby, Node.js, TypeScript, Python

• Функціональні мови: Scala, Erlang, Elixir, Clojure, Haskell

• Мультипарадигмні мови: Rust, Goland

• Корпоративні рішення: Java, .NET [3].

Front-end – це розробка користувацького інтерфейсу та функціональності, яка працює на клієнтській стороні веб-додатку. Це те, що бачить користувач, коли відкриває веб-сторінку, а також те, з чим він взаємодіє.

До основних компонентів frontend-розробки відносяться:

HTML (HyperText Markup Language) – мова для розмітки документів, створення структур сторінок, таких як заголовки, абзаци, списки.

CSS (Cascading Style Sheets) – це мова опису та стилізації зовнішнього вигляду документа. Завдяки коду CSS браузер розуміє, як відображати елементи. CSS визначає налаштування кольорів і шрифтів, а також те, як розташовуються різні блоки сайту. Це дозволяє відобразити один документ у різних стилях, показувати передачі на екрані або читати його в голос.

JavaScript – це мова, створена для того, щоб оживляти веб-сторінки. Її завдання – реагувати на дії користувача, обробляючи кліки миші, рухи курсору та натискання клавіш. Вона може надсилати запити до сервера, завантажувати дані без перезавантаження сторінки, друкувати повідомлення та багато іншого [4].

У цьому типі додатків оновлення вмісту настільки просте, що серверу навіть не потрібен доступ до зміненого вмісту. Навіть не потрібні права доступу для внесення змін. Також можна реалізувати багато функцій, таких як форуми, бази даних. Веб-додаток можна змінювати відповідно до налаштувань системного адміністратора.

Якщо веб-додаток являє собою інтернет-магазин, його розробка

найімовірніше, буде схожа на розробку веб-сайту m-commerce чи e-commerce. Розробка додатків такого типу складніша, оскільки необхідно забезпечити можливість електронних платежів за допомогою кредитних карт або платіжних систем, таких як PayPal. Також необхідно створити адміністративну панель управління, яку можна використовувати для перерахування та оновлення нових продуктів, видалення записів і управління додатками та платежами.

Веб-додатки повинні підходити для мобільних пристроїв так само, як і мобільні додатки, і мати можливість працювати так само, як нативні додатки

Портальний веб-додаток

Належить до типу застосунку, який дає змогу отримати доступ до різних розділів і категорій з головної сторінки. Це належить до типу додатків, які дають змогу отримати доступ до різних розділів і категорій з головної сторінки. Ці додатки можуть включати форуми, електронну пошту, чати, браузери, області для реєстрації, останні матеріали та багато іншого.

Анімовані веб-програми:

Коли мова заходить про анімацію, неминуче виникає образ технологій Flash. За допомогою цього підходу програмування можна відображати контент з анімованими ефектами. Цей тип додатків має більш сучасний вигляд і є однією з ключових технологій, які використовують дизайнери та креативні директори. Однак цей тип технології не підходить для цілей веб-розміщення або SEO-оптимізації, оскільки пошукові системи не можуть правильно прочитати інформацію, що включає в себе.

Веб-додатки із системами управління контентом:

Вміст таких веб-додатків необхідно постійно оновлювати. Під час розроблення такого типу веб-застосунків слід впровадити систему керування контентом (CMS). Щоб адміністратори могли самі змінювати й оновлювати контент за допомогою CMS.

Ці системи управління контентом інтуїтивно зрозумілі й дуже прості у використанні. Деякі приклади систем управління контентом включають:

WordPress: найпоширеніша система управління контентом. Тримається високо серед лідерів ринку.

Joomla: ця CMS поступається тільки WordPress. У неї не так багато користувачів, але є сильна спільнота, а також вона дуже інтуїтивно зрозуміла.

Drupal: це безкоштовне програмне забезпечення CSM дуже чуйне, вона особливо рекомендується для створення спільнот [5].

1.2 Структура веб-додатку

У більшості випадків необхідна наявність щонайменше 3 основних компонентів:

1. Клієнтська частина – це система, яку ми використовуємо на наших комп'ютерах. Цю роль виконує браузер. Сам браузер може тільки правильно відобразити інформацію, отриману з будь-якого ресурсу.

2. Серверна частина – це програма, яка знаходиться на сервері та обробляє запити користувачів (браузерів). Взаємодіє з клієнтами для надання інформації. Вона відповідає за зберігання та обробку даних, підтримання користувацького веб-інтерфейсу і взаємодію з обліковою системою. Наприклад, на схемі серверна частина реалізована на PHP, але зараз існує безліч мов програмування для створення Back-End. Залежно від дії користувача, наприклад, натискання на посилання або відповідну кнопку, запит надсилається на сервер, який обробляє цей запит і виконує певні завдання відповідно до того, що описав програміст. Нова сторінка може бути надіслана користувачеві у вигляді HTML-файлу, який браузер уже має обробити та відобразити для користувача.

3. База даних – це програмне забезпечення на сервері, завдання якого – зберігати дані та швидко витягувати їх за запитом користувача. Сама база даних розташовується на сервері. База Даних несе відповідальність за зв'язок із сервером.

Серверна частина веб-додатка звертається до бази даних і запитує дані, необхідні для генерації сторінок, запитуваних користувачем. Інтегровані (вбудовані) СУБД використовуються як компоненти інших програмних продуктів, таких як словники, електронні енциклопедії та інтернет-магазини. Ці системи не потребують окремого встановлення і можуть мати обмежену функціональність для управління базою даних, наприклад, зміна даних не повинна бути можливою. Доступ до даних має здійснюватися через прикладне програмне забезпечення у яке інтегровано СУБД.

На рисунку 1.1 представлено класифікацію основних СУБД.

Рисунок 1.1 – Схема основної класифікації систем, що керують базами даних.

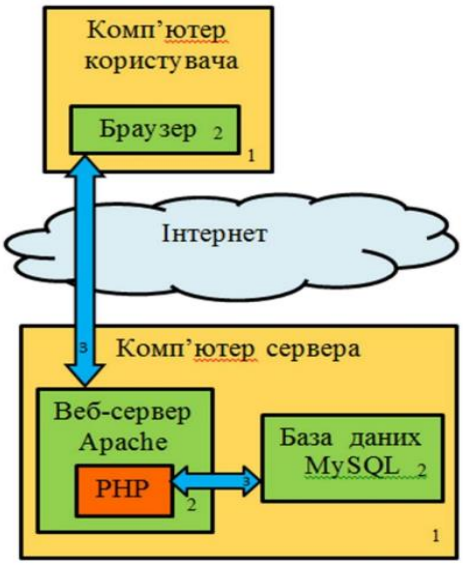

Рисунок 1.2 – Графічна діаграма, що показує взаємодію між базою даних, клієнтом і сервером: 1 – зв'язок між комп'ютерами, 2 – програми, 3 – зв'язок між програми.

На рисунку 1.2 показано взаємодію між компонентами веб-додатка. Браузер надсилає HTTP-запит на веб-сервер через інтернет. У відповідь веб-сервер викликає сценарій, написаний розробником веб-додатка. За необхідності цей скрипт звертається до бази даних. У результаті клієнту провертається веб-сторінка, яку має відобразити браузер.

Висновки до розділу 1:

Насамкінець було розглянуто методи і типи створення веб-додатків. Надійність веб-додатків – це потреба, про яку не можна забувати і якій необхідно приділяти належну увагу. Для задоволення цієї потреби було розглянуто основні методи тестування ПЗ. Під час аналізу було сформовану загальну картину структури побудови автоматизованого тестування системи та можливих програмних реалізацій.

2. АНАЛІЗ ПІДХОДІВ ДО АВТОМАТИЗАЦІЇ ТЕСТУВІАННЯ

2.1 Життєвий цикл автоматизації тестування

Рисунок 2.1 – Схема життєвого циклу автоматизації тестування

Керівники проектів і розробники часто стикаються з проблемою створення додатків з недостатніми ресурсами і постійно з скорочуванням термінів. Незалежно від того наскільки досвідченими є програмісти при створені ПЗ, вони повинні перевіряти чи дійсно воно відповідає вимогам. Тому організації звертаються до автоматизації тестування для ефективного досягнення цієї мети.

Багато хто вважає автоматизацію тестування лише однією з частин SDLC (життєвого циклу розроблення програмного забезпечення), але для досягнення найкращих результатів необхідно слідувати всьому циклу, який називається життєвим циклом автоматизації тестування.

Реалізація життєвого циклу автоматизації тестування проходить паралельно з життєвим циклом розробки програмного забезпечення.

Сам життєвий цикл є багатоетапним процесом, що підтримує діяльність, необхідну для використання та впровадження інструментів автоматизованого тестування, розроблення та виконання тестових випадків, розроблення тестових проектів, створення та обробки тестових даних і створення середовища.

У методології дизайн тесту будують так, щоб відобразити роботу з тестування і переконатися, що команди проекту і тестування розуміють обсяг роботи з тестування.

У методології автоматизованого тестування життєвого циклу існує 6 основних етапів:

- Визначення обсягу робіт для автоматизації тестування
- Вибір вірного способу технологій для автоматизації
- Тест План, Тест Дизайн, Тест Стратегія
- Створення тестового середовища
- Розробка коду для виконання автоматизованих тестів та їх запуск тестів
- Аналіз і створення звітів про тестування [6].

Визначення обсягу робіт з тестування автоматизації:

Це етап самий перший життєвого циклу АТ, мета якого – визначити доцільність автоматизації. Тут необхідно розглянути, скільки часу є в наявності для побудови процесу і наскільки складно буде створити модель для системи автоматизації. Крім того, важливо провести аналіз здійсненності ручних пакетів тестових ситуацій для інженерів з автоматизації для розробки програмних сценаріїв.

У цьому етапі слід ставитися обережно до наступних речей:

1. Які модулі застосунку можуть бути автоматизовані, а які ні?
2. Які тестові випадки можуть бути автоматизовані і як?
3. Необхідно також врахувати такі фактори, як вартість, розмір команди і досвід також має бути взято до уваги.

Перш ніж приступити до автоматизації тестування, необхідно перевірити техніко-економічне обгрунтування:

- Можливість автоматизувати тестові випадки
- Автоматизація автоматичної автоматизованості

Давайте згадаємо всі компоненти програмного інтерфейсу і виділимо ті, які найважливіші для користувача. Насамперед слід звернути увагу на ті, що може призвести до збою у роботі які можуть понести завдаток у великій кількості грошей. Також необхідно визначити відсоток компонентів користувацького інтерфейсу, які мають бути автоматизовані засобами автоматизованого тестування. Постарайтеся знайти інструмент автоматизованого тестування, який може автоматизувати компоненти користувацького інтерфейсу з невеликими змінами. Це полегшить виконання наступних кроків.

Вибір правильної технології для автоматизації:

АТ сильно залежить від інструменту. Тому пошук відповідного інструменту для автоматизації тестування є важливим аспектом для ЖЦАТ. Під час пошуку інструменту автоматизації необхідно враховувати бюджет, технологію, використовувану в проекті, знання розробників і тестувальників у команді та гнучкість. Також слід обирати інструмент, який підтримують розробники і який добре документований.

Наприклад, якщо ви шукаєте автоматизований інструмент для перевірки сумісності браузерів, то необхідно чітке розуміння різних підтримуваних браузерів. Відеозвіти, можливість захоплення метаданих для автоматизованих сценаріїв у різних браузерах і пристроях. Механізми виявлення та відстеження помилок, а також способи запису програмного коду.

Тест план + Тест дизайн + Тест стратегія:

Це найважливіший етап методу ЖЦАТ, що визначає, яким чином має бути досягнута основна мета АТ. Автоматизоване планування тестування – перше, на що варто звернути увагу на цьому етапі.

Вибір інструменту залежить від технології, що використовується в застосунку. Перш ніж приступити до автоматизації, необхідно повністю зрозуміти продукт.

Наприклад, якщо це настільний додаток, дізнайтеся, якою мовою його створено. Якщо це веб-додаток, дізнайтеся функціональність фреймворка для тестування, використовувані компоненти та функції, з якими вони працюють. Знайте використовувані компоненти та функції, які з ними працюють.

На етапі планування тестування, група з тестування повинна визначити стандарти та рекомендації щодо створення процедур тестування, апаратне та програмне забезпечення і мережу, що підтримує тестове середовище, попередні графіки тестування і вимоги до тестових даних. Мають бути розроблені чіткі процедури відстеження дефектів і відповідні інструменти відстеження для моніторингу тестової конфігурації та середовища постановки.

Після розроблення моделі стратегії випробувань команда інженерів-випробувачів розробляє архітектуру випробувань, яка описує структуру програми випробувань і спосіб управління процедурами випробувань.

Після проектування створюється архітектура випробувань. Вона описує загальну структуру програми випробувань і способи управління нею.

Під час розробки плану тестування не забудьте врахувати наступне:

Зберіть усі ручні тестові випадки з інструменту управління тестуванням і визначте, які тестові випадки ви хочете автоматизувати.

Зрозуміти плюси та мінуси інструментів тестування, перш ніж вирішити, який фреймворк використовувати. Створіть набір автоматизованих тестових прикладів в інструменті управління тестуванням.

Переконайтеся, що план тестування охоплює припущення, ризики та залежності між інструментом і додатком. Отримайте схвалення замовника та зацікавлених сторін на стратегію тестування.

Створення тестового середовища:

Як випливає з найменування, цей життєвий цикл автоматизації тестування завбачити налаштування або віддаленої машини для виконання тестових

прикладів. Навіщо нам потрібні віддаленні машини? Тому що, якщо ви не живете в ідеальному світі, користувачі використовують різні машини для доступу до веб-сайтів і веб-додатків в Інтернеті.

Крім контролю коректності роботи веб-додатків на різних пристроях, нам також необхідно знати про різні браузери та їхні версії. Це пов'язано з тим, що наші веб-сайти можуть по-різному відображатися в різних браузерах. Тестування сумісності браузерів також відоме як крос-браузерне тестування. Також відоме як крос-браузерне тестування, це процедура під час якої веб-сайт або веб-додаток тестується в декількох версіях браузерів, щоб переконатися, що він забезпечує безперебійну роботу для всіх користувачів.

Етап створення середовища вимагає ретельного планування. Необхідно забезпечити максимальне покриття тестами якомога більшої кількості сценаріїв. Необхідно правильно налаштувати середовище, встановити програмне забезпечення, мережеві ресурси та обладнання, удосконалити базу даних тестування і розробити сценарії тестового покриття, які постійно інтегруються в процес розробки.

Під час проведення крос-браузерного тестування браузерів налаштування сотень браузерів і версій браузерів на великій кількості пристроїв може бути дуже складним завданням, а купівля лабораторії пристроїв не всім по кишені. Саме тут на допомогу приходять хмарні інструменти, такі як Azure Devops, Amazon Webing Services і Google Web Services, які пропонують понад 2000 браузерів і версій браузерів і розміщують віртуальні машини для великої кількості настільних і мобільних пристроїв.

Основні сфери для налаштування тестового середовища:

1. Тестові дані – немає необхідності постійно створювати великі обсяги тестових даних для функціональної верифікації. Не завжди необхідно створювати великі обсяги тестових даних для функціональної перевірки. Якщо є можливість – це можливо, все необхідне для виконання тестових прикладів має бути створено заздалегідь. Це може значно скоротити час виконання тесту і запобігти помилковим результатам.

2. Середовище для навантажувального тестування має важливе значення, необхідно підготувати окреме середовище для навантажувального тестування і проаналізувати здатність обробляти веб-трафік, підтримувати велику кількість користувачів і обробляти достатню кількість запитів одночасно.

3. Контрольний список усіх систем, модулів і додатків, що підлягають тестуванню.

4. Ізольовані сервери баз даних.

5. Тестування на різних клієнтських операційних системах.

6. Максимальна кількість тестів із різними версіями браузерів.

Переконайтеся, що ви протестували свій сайт у мережах із низькою та високою швидкістю передавання даних, щоб зрозуміти різницю в часі доставки та загальному вигляді вашого сайту або веб-додатка.

Документація має вирішальне значення. Важливо забезпечити, щоб усі посібники з конфігурації, посібники з установлення та посібники користувача мають управлятися централізовано.

Налаштування тестового середовища включає в себе такі завдання:

1. Ліцензії та інструменти

2. Засоби налаштування, такі як сучасні текстові редактори та інструменти для порівняння.

3. Впровадження фреймворку автоматизації

4. Доступ до AUT і дійсні облікові дані

5. Ліцензії на додаткові модулі

Більшість організацій мають окреме тестове середовище для тестування програмного забезпечення. Найкращою практикою є копіювання даних із користувацького середовища та використання їх для тестування. Таким чином, інженери з тестування можуть знайти проблеми без необхідності возитися з виробничими даними.

Найкращі практики з налаштування управління тестовим середовищем включають:

• Розуміння середовища тестування та навчання членів команди.

• Необхідне програмне забезпечення, ліцензії та необхідне обладнання.

• Кросбраузерне тестування, тестування в декількох браузерах і версіях з урахуванням пріоритетів і частки ринку.

• Використання трафіку в реальному часі. Забезпечити стійкість змін.

• Плануйте використання середовища та тестування.

Розробіть код для запуску автоматизованих тестів і запустіть їх:

Після налаштування тестового середовища настав час запустити тестовий сценарій. Запуск тестових сценаріїв. На цьому етапі життєвого циклу автоматизації всі тестові сценарії будуть присвячені виконанню всіх тестових сценаріїв.

Передача підписаних і перевірених тестових прикладів команді автоматизованого тестування для виконання сценарію.

Важливо переконатися, що всі сценарії тестування працюють правильно. Тому перед розробкою сценаріїв тестування необхідно врахувати таке:

• Створення сценаріїв тестування має грунтуватися на реальних вимогах.

• Розробіть загальні функціональні методи, які можна використовувати протягом усього процесу тестування.

• Створюйте багаторазово використовувані, структуровані та прості сценарії, які можуть бути зрозумілі кожному.

• Доручайте перевірку коду іншому члену команди, щоб поліпшити якість продукту і підвищити розуміння загальної структури членами команди.

• Використовуйте звіти для чіткішого відображення результатів АТ і загального процесу.

Після того як ви успішно створили сценарій тестування, вам слід пам’ятати таке:

1. Тестові сценарії повинні включати всі функціональні аспекти відповідно до тестових випадків.

2. Тестові сценарії мають виконуватися в різних середовищах і на різних платформах.

3. Там, де це можливо, можливе паралельне виконання для економії часу та зусиль.

4. Якщо збій пов’язаний з конкретною функцією, необхідно написати звіт про помилку.

Для виконання сценарію тестування група тестування повинна слідувати графіку, затвердженому в процедурі. На цьому етапі проводиться оцінювання результатів тестування і складається документ про результати тестування.

Уся система тестується шляхом виконання планів, складених для модульного тестування, системного тестування, тестування прийняття користувачем та інтеграційного тестування. Під час модульного тестування проводиться профілювання коду. Профілювання дає змогу знайти випадки, коли масштабування алгоритмів, використання ресурсів, використання екземплярів тощо є необгрунтованим.

Аналіз і генерація тестових звітів:

Після завершення всіх видів тестування група тестування аналізує та визначає компоненти, у яких було виявлено специфічні функції або конкретні проблеми.

Результати, отримані під час аналізу, можуть бути використані для перевірки того, чи можуть виконані сценарії/процедури тестування виявити помилки.

Це остання фаза життєвого циклу автоматизованого тестування на якій складаються звіти про тестування. Звіти про тестування важливі для аналізу того, наскільки добре веб-додаток справляється з труднощами. Хоча можна використовувати старі робочі таблиці Excel, сучасні генератори тестів надають звіти про застосунок і всі тестові випадки, які були запущені за допомогою сценаріїв автоматизації на хмарній платформі Selenium Grid.

2.2 Піраміда тестування у автоматизації

Рисунок 2.2 – Піраміда ідеального тестування

На рисунку 2.2 показано так звану ідеальну піраміду автоматизації тестування.

Почнемо з того, що зазвичай робить тест автоматизованим. Які тести можна назвати автоматизованими? Насамперед, це ті, які не потребують присутності людини. Існує безліч способів запуску та налагодження тестів. Тести можуть бути налаштовані на запуск після певних дій програміста, таких як новий коміт або просто виконуватися за певним розкладом.

На нижчому рівні існує автоматизоване модульне тестування, метою якого є перевірка правильності роботи окремих частин коду (часто функцій) окремо одна від одної. Основна ідея полягає в тому, щоб гарантувати, що подальші зміни наявного коду не призведуть до зламу вже протестованих частин додатка. Як правило, модульні тести пишуть розробники коду або члени команди, які

витрачають на один модульний тест у кілька разів менше часу, ніж на будь-яких з рівнів вище. Таким чином, модульне тестування – це перший рівень боротьби з помилками. Юніт-тестування відіграє важливу роль при розробці великих додатків. Якщо над продуктом починають працювати кілька людей, стає дуже складно вносити зміни в наявний код, не зламавши жодного компонента системи. Саме тут на допомогу приходять автоматизовані модульні тести, які можна запускати в системі безперервної інтеграції після внесення змін до коду.

Однак, якщо ви самостійно розробляєте невеликий додаток, вам потрібно мати на увазі, що написання модульних тестів тільки сповільнить і ускладнить процес розробки.

Основна відмінність між компонентним і модульним тестуванням полягає в тому, що компонентне тестування використовує об’єкти або драйвери як параметри функції, тоді як модульне тестування використовує конкретні значення [7].

Під час інтеграційного тестування модулі ПЗ інтегруються локально один з одним і тестуються як група. Типове ПЗ складається з окремих модулів, створених різними програмістами. Основним завданням цього рівня є пошук несправностей у взаємодії цих частин. Навіть якщо кожен модуль окремо працює ідеально, необхідно переконатися, що під час їхньої взаємодії нічого не пошкоджено. Можна тестувати як взаємодію компонентів, так і різні системи. Щоб ізолювати окремі частини системи, можна використовувати так звані заглушки, які можуть емулювати поведінку певних частин веб-додатка або просто зупинити його.

Існує 3 основні підходи до інтеграційного тестування:

Знизу вгору (Bottom Up Integration)

Всі низькорівневі процедури, модулі та функції збираються, а потім тестуються. Наступний рівень модулів потім включається в інтеграційний тест. Цей підхід корисний, коли більшість модулів, які потрібно створити, вже підготовлені. На основі результатів тестування цей підхід допомагає поступового виявляти вразливості на кожному рівні програми.

Зверху вниз (Top Down Integration)

Спочатку тестуються всі високорівневі модулі, потім по одному додаються низькорівневі. Всі низькорівневі модулі імітуються заглушками з однаковою функціональністю і коли вони готові їх замінюють реальними активними компонентами. Так працює дане тестування.

Великий вибух («Big Bang» Integration)

Зберіть усі модулі як цілісну систему, а потім проведіть інтеграційні тести. Цей метод є дуже хорошим способом заощадити час. Однак, якщо тестові кейси та їх результати записані неправильно, сам процес інтеграції може стати дуже складним і може стати перешкодою для досягнення основної мети інтеграційного тестування командою тестувальників [8].

Прикладний програмний інтерфейс (API – англ. Application Programming Interface) забезпечують обмін даними та комунікацію між двома програмними системами. Програмна система, що реалізує API, повинна містити функціональність, яку може виконувати інша програмна система. Для того, щоб програми могли спілкуватися між собою. Їхні API повинні бути побудовані на єдиному наборі принципів. Наразі найпопулярнішими є REST – стандарт архітектури взаємодії додатків на основі HTTP. Тестування API базується на бізнес-логіці програмного продукту. Для тестування використовуються спеціальні інструменти, які дозволяють створювати запити і перевіряти точність вихідних даних.

Чим вище по піраміді тестування ви піднімаєтеся, тим більше навичок і знань потрібно для проведення тестування. Для тестування API потрібно розуміти систему і те, як влаштовані запити в ній. Помилки HTTP-сервера можна розділити на ряд:

- 100-199 Інформація. Інформує про те, що запит прийнято і він обробляється.
- 200-299 Запит успішно оброблено і сервер надіслав клієнту запитуваний документ.
- 300-399 Запит змінено, агент повинен ініціювати дію для виконання зміненого запиту.

- 400-499 Виникла проблема на стороні клієнта.
- 500-599 Помилка сервера.

Нижче наведено відмінності між модульним тестуванням та тестуванням API.

Таблиця 2.1 – Порівняльна таблиця API і модульного тестування

Модульне тестування	Тестування API
Зазвичай тести пишуть розробники	Тести пишуть тестувальники, менше розробники
Тестують відокремлені методи, функції	Тестується основна система функціональності
Розробник має право вплинути на програмний код	Тестувальник зазвичай не має можливості вплинути на програмний код
Є можливість тестувати з урахуванням графічного інтерфейсу	Лише тестуються функції з API
Підходить лише для базових функцій	Підходить для усіх функціональних частин
Є певні обмеження в застосуванні	Відсутність в сфері застосування
В основному проходить під час написання самого програмного коду	В основному проходить вже після самого створення функціоналу основного

Тривалий час SOAP був переважним протоколом для обміну даними між веб-сервісами, але останнім часом гнучкіша архітектура REST стрімко набирає

популярності, оскільки від розробників вимагається створювати компактні та швидкі додатки. Давайте розглянемо основні відмінності між двома архітектурами, представленими в таблиці 2.2, і визначимо основні переваги на недоліки [8].

Таблиця 2.2 – Порівняння таблиць REST і SOAP архітектур

	SOAP	REST
Значення	Simple Object Access Protocol	Representational State Transfer
Дизайн	Стандартизований протокол з чіткими заданими правилами оформлення	Архітектурний стиль, який просто має свої рекомендації до оформлення
Підхід	Function-driven	Data-driven
Безпечність	WS-Security з підтримкою SSL. Вбудована ACID compliance	Підтримує HTTPS і SSL
Формат повідомлень	Тільки XML	Plain text, XML, HTML, YAML, JSON та інші
Трансферні протоколи	HTTP, SMTP, UDP and others	HTTP
Використання ресурсів	Велика	Менша

Обидві архітектури дають змогу користувачам створювати власні API; SOAP, імовірно, буде, як і раніше, використовуватися підприємствами, яким потрібен високий ступінь безпеки і які проводять багато складних транзакцій. Найважливішими з них є банківські установи, системи управління ідентифікацією та платіжні шлюзи та телекомунікаційні компанії. Навіть PayPal, найбільша платіжна система, повністю побудована на SOAP API. Також важливо пам'ятати про підтримку наявних систем. Відомі веб-додатки можуть уже мати велику кількість користувачів і, як і раніше, підключатися до сервісів через SOAP API. Ось деякі веб-додатки можуть мати певну частину архітектури веб-сервісів на SOAP, а іншу – на REST.

Зі свого боку у 2018 році REST став найпоширенішим вибором розробників для створення публічних API. Існує безліч соціальних мереж з відкритими інтерфейсами REST API, що дає змогу розробникам легко втілити свої веб-додатки з будь-якою платформою. Ці публічні API постачаються з добре документованою документацією, де ви можете отримати всю необхідну інформацію.

Загалом, основними перевагами SOAP є високий рівень безпеки, стандартизація та масштабованість. З іншого боку до недоліків належать тривалий час виконання, складність і негнучкість.

З іншого боку, REST є більш масштабованим, швидким, гнучким і сумісним із сучасними браузерами. До недоліків належать найнижчий рівень безпеки та нижча продуктивність у розподілених середовищах.

На вершині піраміди знаходиться UI End-to-End тестування. За допомогою цього тестування можна повністю протестувати саму систему або продукт і переконатися, що весь призначений для користувача інтерфейс працює правильно. Тестування призначеного для користувача інтерфейсу займає більшу частину часу і дає змогу протестувати безліч компонентів одночасно. Є можливість переконатися, що кожен елемент на сторінці відображається відповідно до вимог, що переходи між сторінками відбуваються і що розмітка не плаває. Такі тести можуть повністю емулювати поведінку типового користувача і перевіряти поведінку кількох сервісів одночасно. Існує низка інструментів, написаних на

різних мовах програмуваннях, які дають змогу налагодити процес автоматизованого тестування веб-додатків. Ці інструменти можна задіяти на заключному етапі, щоб забезпечити максимальне охоплення і найвищу якість кінцевого продукту.

Тестування E2E вимагає складної інженерної роботи, сполучення датчиків, виконання всіх необхідних тестів і сценаріїв або хоча б опису цих сценаріїв.

Висновки до розділу 2:

Інакше кажучи, просуваючись угору пірамідою тестування, важливо пам’ятати, що з кожним новим рівнем такі параметри, як складність, важкість і ціна, збільшуються, а швидкість виконання зменшується. Тому в ідеалі кількість написаних тестів має збільшуватися зверху вниз. Аналогічно у тестуванні, звісно, E2E тестування потрібне лише тоді, коли вбудовано і юніт-тести, і тести вищого рівня; розумно побудувати всю систему тестування лише завдяки End-to-End тестуванню, але це скоріше питання розподілу бюджету та стратегії, що використовується під час розроблення ПЗ. Саме кінцеве тестування може гарантувати найбільшу надійність, оскільки воно імітує поведінку реальних користувачів [9].

3. ПОРІВНЯЛЬНА ХАРАКТЕРИСТИКА ІНСТРУМЕНТІВ ДЛЯ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ВЕБ-ДОДАТКУ

3.1 Вибір мови програмування для автоматизації

Розпочинаючи автоматизацію, команди приймають рішення, опираючись на потреби замовника та власних навичках і знаннях. Останнім часом з’являється дедалі більше можливостей для вибору мови автоматизації на основі технологічного стека конкретного проекту. Як правило, намагаються обирати мови програмування, які використовуються в back-end і front-end. Це дає змогу залучити до процесу автоматизації більше членів команди, що підвищує якість створення коду і дає змогу всім брати участь у процесі створення та рецензування коду.

Відмінності між мовами стають дедалі меншими та менш помітними. Тому сучасні тенденції в автоматизації показують, що дедалі більше проектів використовують JavaScript для різних рівнів і типів тестування. Приклад можна побачити у статистиці Github за популярністю мов за 2022 рік, рисунок 3.1 [10]. Front-End завжди написаний на JavaScript / TypeScript, а проекти, що обирать Node.js як Back-End мову, є такими. Ми знаємо, що кількість проектів зростає, тому в нас є можливість будувати цілі проекти, використовуючи лише JavaScript. Тому для нас вибір JavaScript має бути найбільш оптимальним.

Рисунок 3.1 – Популярність мов програмування у 2022 році.

JavaScript виявляється більш ніж удвічі швидшим, ніж Java і Python. Однак слід також звернути увагу на наявні рішення на ринку автоматизації. Це також відіграє важливу роль у виборі правильної мови.

3.2 Порівняння JS-фреймворків для Unit-тестування

Рисунок 3.2 – Найпопулярніші фреймворки для Unit-тестування, написаний на JavaScript, TypeScript

Jest, ймовірно, один із найпопулярніших фреймворків для тестування. Це фреймворк для тестування JavaScript з більш ніж 22 000 зірок на github, створений і постійно підтримуваний командою Facebook. Він не вимагає конфігурації, рекомендований React і є найпростішим у використанні. Jest має дуже вражаючий показник прийняття в 2022 році спільнотою JavaScript.

Він працює швидко і має зручний інтерфейс. Він також пропонує миттєве тестування і поставляється з вбудованим інструментом покриття коду. Він напрочуд швидкий і є одним із найкращих варіантів для новачків, які хочуть протестувати код JavaScript. Крім того, в Інтернеті можна знайти безліч ресурсів і документації [11].

Основні плюси JEST:

1. Сумісність із NodeJS, React, VueJS, Angular та іншими проектами на основі Babel.

2. Стандартний синтаксис із підтримкою документації

3. Дуже швидкий і високоефективний

4. Можливе тестове керування великими об’єктами з допомогою Live Snapshots

MochaJS – найпопулярніший фреймворк для тестування, що підтримує Back-End та Front-End тестування. Mocha JS – гнучний фреймворк для розроблення тестів на вимогу. Запуск тестів асинхронно на рушії Chrome v8 та інших браузерів.

Розробникам надається тільки базова основа для тестування. Твердження, шпигунство, мокінг та інші функції додаються за допомогою інших бібліотек і плагінів.

Додаткові параметри та конфігурації, необхідні для Mocha, - це, безумовно, те, що потрібно перевірити, особливо якщо вам потрібна гнучка конфігурація, що включає необхідні бібліотеки.

На жаль, вищеописаний варіант має і недолік, який полягає в тому, що для перевірки необхідно включити додаткові бібліотеки. Це означає, що інсталяція трохи складніша, якщо не довша, ніж в інших випадках.

Mocha включає структуру тесту як глобальну, що економить час.

Гнучкість тверджень і заглушок дуже корисна. Загалом, Mocha охоплює основи і дозволяє розробникам розширювати його.

До основних переваг Mocha належить:

1. Працює як на Front-End, так і на Back-End

2. Підтримує відладчик NodeJS

3. Надає чистий фреймворк для розробки тестів у зручний для розробника час

4. Підтримує всі браузери, включно з бібліотеками для Google Chrome

5. Підтримує об’єктні заглушки для виконання гнучких тестів

Jasmine - все необхідне надається з коробки. Це симулятор поведінки користувача, який дає змогу запускати тестові випадки, що нагадують поведінку користувачів на вашому сайті Jasmine корисний для тестування видимості сайту, кількості кліків, чуйності користувацького інтерфейсу в різних режимах роздільної

здатності тощо. Jasmine можна використовувати для автоматизації поведінки користувачів з мінімальною затримкою і затримкою для імітації реальної поведінки користувачів.

Основні переваги використання Jasmine:

1. Низькі накладні витрати, оскільки є мало зовнішніх залежностей
2. Майже всі необхідні інструменти включені в комплект
3. Підтримується як фронтенд-тестування, так і бекенд-тестування
4. Написання коду дуже схоже на написання природною мовою
5. Велика документація про те, як використовувати його з іншими фреймворками
6. Великі вступні онлайн-курси

AVA – мінімалістична легка структура тестування, що використовує асинхронну природу JavaScript. В основному орієнтований на виконання тестів проти коду на базі NodeJS; AVA може виконувати тести паралельно.

Це дає практично повний контроль. Основна увага приділяється виконанню тестів для коду на базі NodeJS. Переваги включають:

1. Тести виконуються асинхронно і паралельно
2. Швидше, ніж багато інших фреймворків для тестування
3. Простий синтаксис для тестування на JavaScript
4. Очищувач стека відстежує потенційні виявлені помилки

Karma – це високопродуктивне середовище тестування, що підтримує всі поширені фреймворки для тестування. Забезпечує підтримку додатків для запуску тестів у різних середовищах. Має широку підтримку для запуску тестів на різних пристроях і додатках.

Основними факторами, що зумовили вибір Karma, стали підтримка інтеграції з рушіями CI / CD та наступними можливостями:

1. Можна використовувати для запуску тестів у браузері, безголових середовищах, таких як PhantomJS, і на пристроях
2. Підтримує тести, написані на поширених фреймворках

3. Тести можна запускати віддалено на інших пристроях, просто надіславши файл

4. Можливе налагодження тестових прикладів за допомогою Chrome, а також WebStorm

3.3 Порівняння JS-фреймворків для End-To-End-тестування

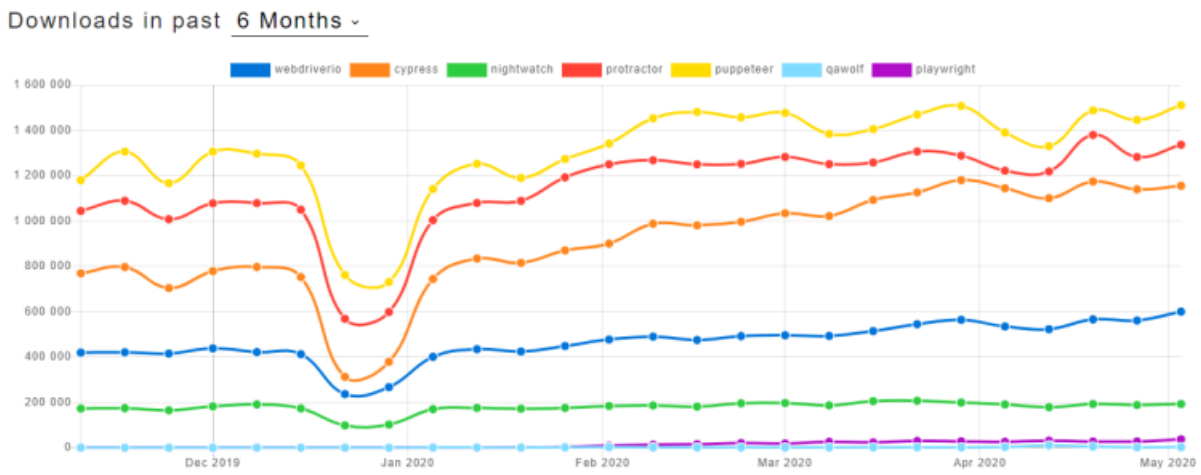

Рисунок 3.3 - Статистика завантажень для фреймворків End-To-End тестування, написаних на JavaScript and TypeScript, з грудня 2019 по травень 2020 років.

Під час вибору фреймворка для тестування важливо звернути увагу на його популярність на GitHub, а також підтримку спільноти, кількість відкритих проблем, сумісність із технологією, яка використовується в системі, та характеристики кожного додатка, що використовується для автоматизації. Слід звернути увагу на характеристики кожного додатка, що використовується для автоматизації.

	stars	forks	issues	updated	created	size
webdriverio	5 675	1 663	119	May 15, 2020	Aug 30, 2011	minzipped size 238.4 KB
cypress	20 103	1 205	1 249	May 15, 2020	Mar 4, 2015	minzipped size 365.2 KB
nightwatch	10 194	999	120	May 11, 2020	Mar 17, 2012	
protractor	8 542	2 370	610	May 14, 2020	Jan 16, 2013	bundlephobia timeout
puppeteer	61 323	6 323	1 032	May 14, 2020	May 10, 2017	minzipped size 50.8 KB
qawolf	1 324	35	15	May 14, 2020	Sep 17, 2019	bundlephobia timeout
playwright	12 631	365	119	May 15, 2020	Nov 15, 2019	minzipped size 117.3 KB

Рисунок 3.4 – Інформація про актуальність та підтримку фреймворків End-To-End тестування, написаних на JavaScript / TypeScript

Фреймворк	Приорітет	webdriverio	cypress	testcafe	puppeteer	protractor
Screenshots	Високий	y	y	y	y	y
Parallel Testing	Високий	y	y	y	yn	y
Easy debug	Високий	yn	yn	yn	yn	yn
Sending native keyboard/mouse events	Високий	y	n	y	y	y
Fast tests execution	Високий	yn	y		y	y
CI: good compatibility	Високий	y	y	y	y	y
Synchronous code execution	Середній	y	yn	yn	yn	yn
Cross-browser testing	Середній	y	y	y	yn	y
Iframes support	Середній	y	yn	y	y	y

Продовження таблиці 3.1						
Mobile Testing	Середній	y	yn	y	yn	y
Easy base-install for test execution	Середній	yn	y	y	y	y
Detailed documentation	Середній	y	y	y		y
Multiple domains	Середній	y	n	y	y	y
Files: uploading/downloading	Середній	y	yn	y	y	y
Low entry threshold	Середній	y	y	y		
Access to browser console	Низький	y	y	y	y	y
Shadow DOM	Низький	y	yn	y	y	y
Time Travel	Низький	n	y		n	n
Automatic Waiting	Низький	yn	y	y	n	yn
Network Traffic Control	Низький	yn	y		y	n
Videos	Низький	y	y	y	n	yn
Visual Testing	Низький	y	y	yn	yn	yn
Custom asserts	Низький	y	y	yn	n	
Retries: scenario or step	Низький	y	yn		n	yn

Продовження таблиці 3.1						
Multiple tabs	Низький	y	n	n	y	y
Multiple windows	Низький	y	n	yn	n	y
User-friendly command syntax	Низький	y	y	yn		y
TypeScript	Низький	y	y	y		y
Possibillity to intercept request		y	y	y	y	n
Possibillity to mock responses		y	y	y	y	n
Bundled with Chromium		n	y		y	
Reporters: good compatibility		y	y	y		yn

Таблиця 3.1 – Порівняння фреймворків для End-To-End тестування, написаних на JavaScript/TypeScript

У таблиці такі статуси: y(yes) - це означає підтримується, yn(yes/no) - частична підтримка або якась специфіка, n(no) - взагалі не підтримується.

Protractor. Ідеально підходить для проектів, написаних на Angular, з вбудованим функціоналом для специфічних завантажувачів елементів Angular.

Плюси:

• Protractor – єдиний фреймворк, який підтримує кастомний AngularJS. Якщо ви використовуєте, використовуйте Protractor.

• Зручна підтримка TypeScript та різних фреймворків також підтримується модульне тестування (Jasmine, Mocha, Cucumber тощо).

• Широкі допоміжні функції.

Недоліки:

• Немає підтримки мобільного тестування.

• Написаний як обгортка для WebDriverJS. Якщо ви використовуєте WebDriverJS, якщо у вас виникнуть проблеми, то автоматично буде використано ProtractorJS.

• Підтримка від розробників більше не доступна.

Приклад базовий на ProtractorJS:

```
describe(‘Protract One testing’, function()  {
it(‘should entering Two numbers’, function()  {
browser.get(‘https://www.madewithangular.com’);
element(by.models(‘press one’)).sendKeying(1);
element(by.models('press two’)).sendKeying(2);
element(by.id('press')).click();
expect(element(by.binding(‘last’)).getText()).
toEqual('10');
});
});
```

Nightwatch.js

Переваги:

• Подібно до WebdriverIO, є кастомною реалізацією API W3C WebDriver.

• Легко додавати нову функціональність.

• Не потрібно вибирати між Jasmine і Mocha.

Недоліки:

• Не підтримує Mocha та мобільне тестування.

• Менша підтримка, ніж у WebdriverIO та Cypress.

Приклад базовий на Nightwatch.js:

```
module.exporting = {
        after.function(website) {
                console.log(Website.loaded. ')
                },

'Nighting watch simple testing': function (website) {
        browser.url('https://nightingwatch.net.com/')
        .waitForElementVisible('[data-nw=name-input] ')
        .setValue('[data-nw=name-input] ', 'You want to text')
```

```
.pause(800)
.assert.containsText('[data-nw        =welcome-message] ', Welcome my friend')
.end()
}
```

Cypress

Плюси:

• На відміну від багатьох E2E фреймворків, тут не використовується Selenium. Архітектура браузера повністю визначена, і Cypress працює безпосередньо з додатком без віддалених команд по мережі.

• Швидке і просте встановлення. Для встановлення всіх пакетів потрібна лише одна команда, тому немає необхідності встановлювати все окремо.

• Вся документація знаходиться в одному місці, тому немає необхідності шукати окремо, як керувати або використовувати Chai.

• Використовується необхідний інтерфейс з описом того, які команди виконуються.

Мінуси:

• Той факт, що Cypress не використовує Selenium для наскрізного тестування, є позитивним, але водночас і негативним. Це викликає ряд проблем при роботі з ним. Робота в браузері не завжди стабільна, наразі на GitHub викладено понад 900 проблем.

• Підтримки мобільного тестування немає, і судячи з коментарів творців про локальну підтримку, ніколи не буде.

• Підтримується обмежена кількість браузерів: Canary, Chromium, Chrome та Electron.

• Платне розпаралелювання тестів. Якщо ви хочете запустити в кілька потоків, вам доведеться заплатити.

```
describe(‘Cypress simple Testing', () => {
```

```
it('clicking «type» Press to a new url', () => {
        cy.visit('https://cyexample.press.io')

        cy.contains('button').click()
        cy.url().should('include', '/commands/actions')
        })
})
```

WebdriverIO

Переваги:

• WebdriverIO – це кастомна реалізація W3C WebDriver API; вона не повинна залежати від реалізації WebDriverJS.

• Підтримує синхронний код. Можна забути про асинхронний JavaScript

• Легке базове налаштування через вбудований інтерфейс у командному рядку wdio

• Підтримує майже всі тестові фреймворки BDD і TDD.

• Підтримує бібліотеку webdrivercss, корисну для порівняння стилів CSS елементів на сторінці.

Недоліки:

• Існують значні відмінності між останніми версіями (WebdriverIO 4 і 5).

• Гірше, ніж Protractor для автоматизації AngularJS.

```
describe('Simple WebdriverIO Test', () => {
    beforeEach(async(() => {
        TestBed.configureTestingModule({
            declarations: [
             AppComp
            ],
        }).compileComponents();
    }));
it('add the app', () => {
    const Fixture = TestBed.createComponent(AppComp);
    const app = Fixture.componentInstance;
    expect(app).toBeTruthy();
```

```
});
```

Наостанок перерахуємо переваги та недоліки використання JavaScript-фреймворку для автоматизації тестування, про які йшлося вище.

Переваги:

- Швидкість написання тестів набагато вища, ніж на Java або C #.
- Низький поріг для запуску проекту.
- Краща взаємодія в команді, якщо її члени розмовляють однією мовою програмування.
- Багато готових рішень для різних завдань.

Недоліки:

- Низька стабільність рішень. Іноді виявляється, що проблема на стороні
- Написання стабільних тестів вимагає розуміння того, як працює JavaScript.
- Кожне рішення/мова/технологія має свої переваги та недоліки. Вибір стеку технологій залежить від вас.
- Цей вибір залежить від структури вашої команди, знань і досвіду її членів та проблеми, яку ви хочете вирішити. Якщо дозволяють обставини, намагайтеся активно вивчати нові технології.

Висновки до розділу 3:

Отже, під час аналізу було вивчено найпопулярніші інструменти, що використовуються сьогодні для автоматизації тестування веб-додатків. Було досліджено їх продуктивність, виміряно швидкість та можливості, а також побудовано графік залежностей характеристик. На основі аналізу графіків були зроблені висновки про сильні та слабкі сторони методів та зроблені висновки про їх ефективність. Проаналізовано перспективи та можливості використання методів неперервної інтеграції.

4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Використання ергономічних досліджень в системі безпеки життєдіяльності людини.

Основа ергономіки – це комплексний системний підхід з дослідження того чи іншого об’єкта, тому є методи ергономічних досліджень мають комплексний системний характер. Специфічним предметом ергономічних досліджень є не відокремлена техніка, не тільки людина як суб’єкт виробництва, не окреме середовище, а система «Людина – машина – середовище», всі елементи якої розглядаються в єдності й взаємодії з кінцевою метою узгодженості фізичних і психічних можливостей людини, її естетичних смаків та інших якісних показників с параметрами сучасних технічних засобів і виробничого середовища. Ергономіст вивчає проблеми оптимального розподілу та узгодженості функцій між людиною, машиною й виробничим середовищем, проектує процес діяльності людини, обґрунтовує оптимальні вимоги до засобів та умов праці, розробляє методи їх урахування при створенні та експлуатації техніки, яка обслуговується людиною. Раціональне поєднання можливостей людини, характеристик машин і відповідний розподіл функцій всередині системи суттєво підвищує її ефективність, обумовлює оптимальне використання людиною технічних засобів за їх призначенням. Основними завданнями ергономіки є: — вивчення умов праці; — виявлення конструктивних недоліків виробничого обладнання; — оцінка організації робочих місць з точки зору забезпечення нормальної робочої зони, допустимих швидкостей, траєкторій, кількості рухів і зусиль, необхідних для обслуговування виробничого обладнання; — вивчення інформаційної взаємодії оператора й машини. Для розв'язання різних ергономічних завдань застосовують методи дослідження характеру та організації праці, методи спостереження та опитування, операційно-структурного опису трудової діяльності, хронометражні, антропометричні, біомеханічні, фізичні, фізіологічні, психологічні, гігієнічні, економічні методи та ін. Залежно від особливостей досліджуваної системи "людина — машина —

середовище" добирають комплекс методів, який може в одних випадках бути націлений на розкриття конструктивних недоліків виробничого обладнання, що призводять до погіршення умов праці, а в інших — на оцінку конструктивних особливостей органів керування, організації робочого місця і т. ін. Спеціальний комплекс методів повинен застосовуватись при вивченні інформаційної взаємодії оператора й машини. Важливе значення має застосування адекватних методів досліджень, за допомогою яких можна виявити працездатність найбільш завантажених систем організму людини. Під час проведення ергономічних досліджень необхідно отримати перш за все достовірні дані, тобто такі, які б відрізнялися стійкістю, відтворенням і разом з цим відповідали завданням досліджень. При порівнянні результатів двох або більше дослідів оцінюються відмінності між ними й середнє значення з урахуванням величини дисперсії. Однак у всіх випадках велике значення мають дані про число спостережень або досліджуваних об'єктів. Перш за все повинні досліджуватися форми і способи праці, характер трудових операцій та вимог до них у межах конкретного технологічного процесу для того, щоб виявити операції, які доцільно покладати на людину в системі "людина — машина". Дослідження при цьому треба здійснювати, виходячи із завдань всебічного підвищення ефективності функціонування системи, підвищення продуктивності, надійності й зручності роботи з повним усуненням небезпеки для здоров'я працюючих. Для цього доцільно порівнювати можливості людини й машин, на основі яких можна було б відібрати операції, які слід покладати на людину (табл. 1). Людині доручають операції найбільш варіативні за способом виконання, які не завжди забезпечені достатньо чіткою інформацією й не потребують великої швидкості рухів. Операції, які могли б виконуватися людиною, але вимагають значної напруги, може виконувати машина. Необхідно зазначити, що наведені в табл. 1 характеристики та можливі варіанти розподілу функцій між людиною й машиною безперервно змінюються з розвитком техніки, вдосконаленням технічних засобів, як відносно безпосереднього виконання тих чи інших операцій, так і відносно керування ними.

4.2 Вплив комп’ютера на здоров’я користувача

Всеохоплююча інформатизація суспільства як глобальний, загальносвітовий процес та багатопланові зміни в організації праці спонукає багатьох людей використовувати різноманітні засоби електронно-обчислювальної (комп'ютерної) техніки та опановувати відповідні інформатичні технології.

При цьому, упровадження комп’ютерних засобів (КЗ) в усі сфери життєдіяльності людини виявило не тільки позитивні, а й негативні наслідки їх використання, про що свідчать скарги користувачів комп’ютера (КК). Зокрема, щодо погіршення власного здоров’я та зниження функціональності організму.

Загальновизнано, що організм людини в цілому не індиферентний до роботи з персональними комп’ютерами (ПК). Найбільш вразливими виявляються зір, центральна нервова і кістково-м’язова системи організму КК, про що стверджують експерти ВООЗ. Зокрема, у висновках експертів ВООЗ, розроблених на основі проведених у різних державах світу досліджень, чітко визначено, що:

- найбільше навантаження під час роботи за ПК припадає на зоровий аналізатор;
- робота із засобами обчислювальної техніки є стресовим фактором для користувача;
- людина, яка працює з комп’ютерними засобами, зазнає впливу фізичних факторів різної природи й малої інтенсивності, а про деякі з них поки що немає достатніх наукових даних, щоб визначити рівень їх впливу на здоров’я людини. Велика ймовірність наявності ефекту комбінованої дії, коли вплив кожного з окремих факторів сам по собі незначний, а їхня сукупність викликає помітну шкідливу дію на організм людини.

Інтенсивне застосування комп’ютерних засобів у навчально-виховному процесі вимагає уважного й відповідального розгляду питань забезпечення безпеки

учнівської й студентської молоді та вчителів і викладачів, а також розробки відповідних рекомендацій, виконання яких дозволить захистити фізичне й психічне здоров’я людей від негативного впливу технічних і програмних засобів інформатичних технологій.

Загалом, на функціональний стан молодої людини та на її здоров’я під час навчання в комп’ютерному класі впливає комплекс об’єктивних і суб’єктивних чинників, зокрема, зміст і обсяг навчальної інформації, інтенсивність і тривалість роботи за ПК, складність навчального предмета, якість і досконалість використовуваних програмних продуктів, їхні ергономічні, педагогічні, психогігієнічні властивості та рівень "дружності" інтерфейсу. Окрім того, об’єктивними, гігієнічно значимими також вважають чинники внутрішнього середовища навчального приміщення, які виникають під час роботи комп’ютерних комплексів – показники мікроклімату, освітленість, яскравість, контрастність і колір зображення на екрані дисплея, іонізуюче та неіонізуюче опромінення, шум тощо.

Загалом, прийнято виділяти чотири групи основних об’єктивних факторів, які можуть негативно вплинути на здоров’я будь-якого користувача персонального комп’ютера:

- візуальні параметри дисплеїв ПК у сполученні зі світловим кліматом у робочому приміщенні (комп’ютерному класі);
- електростатичне і електромагнітне поля комп’ютера, дисплея та інших периферійних пристроїв (емісійні параметри);
- ергономічні параметри робочого місця та приміщення;
- режим праці й відпочинку, види й напруженість роботи за комп’ютером.

Перша група факторів. Користувачі комп’ютерів, які щоденно працюють за ПК, ризикують захворіти на "комп’ютерний зоровий синдром" (Computer Vision Syndrome – CVS). Ознаками прояву CVS є один з таких симптомів: головні болі, напруженість очей, двоїння зображення, стомлені, червоні або сухі очі, тимчасова

короткозорість, випадкове “змазування” зображень на екрані, зростаюче подразнення очей, зміни колірного сприйняття.

Друга група факторів. Комп’ютери, дисплеї та інші периферійні пристрої генерують електромагнітні поля у широкому діапазоні частот. Вплив цих полів на здоров’я людини досі залишається повною мірою невивченим, а результати досліджень є досить суперечливими. Проте, не заперечується потенційна небезпека для здоров’я, яку спричиняє довготривале перебування у зоні неіонізованих електромагнітних полів вкрай низьких частот (5¸2000 Гц) та дуже низьких частот (2¸400 кГц). При цьому, неіонізоване електромагнітне поле, створюване дисплеєм, подібно звичайним телевізійним пристроям, складається з електричного й магнітного полів.

Серед усіх пристроїв, що входять у стандартну комплектацію персонального комп'ютера, найбільш "шкідливим" є монітор. Монітор – це джерело різного виду випромінювань, а саме м'якого рентгенівського, оптичного ультрафіолетового, інфрачервоного, радіочастотного та низькочастотного діапазонів електромагнітних і електростатичних полів.

Третя група факторів. Робоче приміщення (кабінет), обладнаний комп'ютерною технікою, зокрема, в навчальних закладах, має розміщуватись в окремій кімнаті із природним освітленням та організованим обміном повітря (наприклад, за допомогою кондиціонерів), бути досить просторим, ясним, тихим, зі сприятливими умовами мікроклімату в усі пори року.

Фахівці з ергономіки, експерти ВООЗ вказують, що неувага до робочого крісла або економія на ньому призводять до деформації хребта КК та викликають негативну дію на нервові шляхи, викликають больові відчуття в поперековій ділянці, загальний дискомфорт і нерідко знижують працездатність. При цьому перевагу слід віддавати кріслам, які обертаються, пересуваються і які можуть змінювати свою висоту і кут нахилу спинки. Правильне сидіння полегшує працю м'язів. Тому найкращими є крісла, що дозволяють індивідуально підігнати всі параметри і цим забезпечити оптимальну робочу позу.

Правильне встановлення дисплея, пюпітра, клавіатури та загальне облаштування робочого місця зменшує можливість появи і розвитку хвороб у КК.

Загалом, правильне використовування комп'ютера вимагає особливої уваги стосовно обладнання робочого місця. Виконання всіх правил гігієни в цьому відношенні значно зменшить кількість порушень функціонального стану організму і збільшить працездатність КК.

Четверта група факторів. Характер, тривалість та інтенсивність роботи за комп’ютером, режим праці і відпочинку є визначальними факторами впливу на здоров’я користувача комп’ютера. Регламентація видів і режим роботи за комп’ютером важливо для всіх КК, проте, найбільш важливо для дітей, школярів і студентської молоді. Діючі нормативи дозволяють частково врахувати вікові особливості організму оператора, проте не дають відповідь на питання, зумовлені необхідністю врахування індивідуальних особливостей зорового аналізатора конкретної людини, особливо, дитини. А тим більше – на питання, пов’язані з порушеннями зорових функцій.

Для студентів припустимий час роботи за комп'ютером має складати на 1–2 курсі дві години на день, на старших – три, за умови, що робота за відеотерміналом становить не більше 50% від усього часу роботи з використанням комп'ютера. Через кожні 20–25 хвилин занять слід робити паузи для виконання вправ для очей, а через 40–50 хвилин роботи варто влаштовувати 10–15-хвилинну перерву з фізкультурними вправами. Під час перерви доцільно виконати спеціальний комплекс фізичних вправ чи просто походити, наприклад, у коридорі чи іншому приміщенні. Забороняється витрачати перерву на комп'ютерні ігри.

Отже, комп'ютерна техніка може негативно впливати на здоров'я і фізичний стан людини. Тому, при облаштуванні й обладнанні комп’ютерних кабінетів, нормуванні тривалості роботи, зокрема, учнів (студентів) і вчителів (викладачів), необхідно неухильно дотримуватися санітарних, ергономічних, гігієнічних норм та проводити певні фізкультурно-оздоровчі заходи. Це дозволить усім працюючим за комп'ютерами значно зменшити їх вплив на здоров’я, фізичний стан та психіку людини.

ВИСНОВКИ

У рамках своєї бакалаврської роботи я досліджував наявні методи автоматизації тестування веб-додатків на JavaScript і на основі отриманих даних розробив програму тестування.

Аналізовано різні підходи до автоматизованого тестування. користувацький інтерфейс, функціональне тестування, модульне тестування. В умовах стрімкого розвитку інтернет-технологій існує необхідність приділяти достатньо часу тестуванню надійності. Ми розглянули це. Основні методи тестування програмного забезпечення для задоволення цієї потреби. Розглянули основні типи веб-додатків, їхнє призначення та класифікацію; виявили вразливості веб-додатків у зв'язку з класифікацією їхньої побудови. А також інструменти та методи їх створення. Було розглянуто найпоширеніші інструменти тестування веб-додатків для кожного методу, досліджено їхні характеристики та продуктивність. Аналізуючи зібрані дані, було виявлено слабкі та сильні сторони методів і розраховано їхню ефективність. Було створено систему безперервної інтеграції, яку можна використовувати для створення цілого проекту тестування. Було обрано конкретні бібліотеки та сервіси, необхідні для автоматизованого тестування інтерфейсів програмного забезпечення веб-додатка. Цю систему було реалізовано на прикладі сайту.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Веб-додаток і його характеристики [Електронний ресурс] https://www.centum-d.com/uk/veb-dodatok-yogo-harakteristiki
2. Introduction to DOM elements [Електронний ресурс] https://frontender.info/an-introduction-to-dom-events/
3. Сучасний веброзробник [Електронний ресурс] https://medium.com/nuances-ofprogramming/%D1%81%D0%BE%D0%B2%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D1%8B%D0%B9backend%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D1%87%D0%B8%D0 %BA-2018-a43d51a7bcd1
4. Front-end і Back-end розробка [Електронний ресурс] https://skillbox.ua/media/code/frontend_i_backend_razrabotka/
5. 6 Different Types of Web Application Development [Електронний ресурс] https://www.clustox.com/6-different-types-of-web-application-development
6. Automation Testing Life Cycle [Електронний ресурс] https://www.lambdatest.com/blog/all-you-have-to-know-about-automation-testing-life-cycle
7. End-to-End тестування [Електронний ресурс] https://habr.com/ru/post/417395
8. UI автоматизація, чому слід подивитись в бік JavaScript [Електронний ресурс] https://dou.ua/lenta/articles/automation-js-frameworks

9. JavaScript unit testing frameworks in 2020: A comparison [Електронний ресурс] https://raygun.com/blog/javascript-unit-testing-frameworks/

10. 11 Best JavaScript Unit Testing Framework and Tools [Електронний ресурс] https://geekflare.com/javascript-unit-testing/

11. QA Wolf [Електронний ресурс] https://docs.qawolf.com/docs/

12. Slack API [Електронний ресурс] https://api.slack.com/web

ДОДАТКИ

ДОДАТОК А

Програмна реалізація автоматизованого тестування прикладного програмного інтерфейсу

Впровадження системи тестування користувацького інтерфейсу для сторінки https://tntu.edu.ua/

Для цього вам потрібно встановити Node.js – середовище в якому можна запускати JavaScript-код, QA Wolf – це бібліотека Node.js, і для її запуску потрібен Node.js. npm постачається в комплекті з Node.js і є менеджером пакетів Node Package Manager. Він може керувати пакетами, які потрібно запустити.

Щоб перевірити чи встановлені Node.js і npm, виконайте наступну команду в CLI:

```
вузол -v
npm -v
```

Тепер інсталюємо Node.js та добавимо QA Wolf до проекту [12].

Спочатку виберіть проект, до якого буде додано QA Wolf. Ви можете або створити новий проект або вибрати наявний проект. Щоб створити новий проект, виконайте в командному рядку такі дії (ім’я проекту змінювати не обов’язково):

```
mkdir my-easy-project
cd my-easy-project
npm init -z
```

Щоб справно налаштувати QA Wolf, необхідно створити файл qawolf.config.js у корені проекту та встановити необхідні залежності.

```
npm init qawolf
```

Вам буде запропоновано вказати:

1. Каталог у якому буде створено тест (rootDir). За замовчуванням: .qawolf

2. Постачальник послуг CI. За необхідності ви можете створити файл робочого процесу для запуску тестів за допомогою CI. За замовчуванням: GitHub Actions.

QA Wolf також визначає, чи використовується TypeScript, і, якщо застосовується, оновлює конфігурацію для створення тестів на TypeScript замість JavaScript.

Після встановлення залежностей виконайте такі дії, щоб переконатися, що QA Wolf був успішно встановлений

```
npx qawolf
```

Далі, давайте створимо простий тестовий скрипт, щоб переконатися, що на сторінці для бакалаврів є «ПОСИЛАННЯ на завершення бакалаврського проекту».

Наш програмний код буде мав такий вигляд:

```
const qawolf = require(“qawolf”);
```

```
let website, text;

beforeAll(async => {
 website = await qawolf.launch();
 const myContext = await browser.newContext();
 text = await myContext.newPage();
});

afterAll(async () => {
 await qawolf.stopVideos();
});

test("Info", async () => {
 await page.goto(https://tntu.edu.ua/);
 await    page.click('[href=https://dl.tntu.edu.ua/login.php/]');
 await    expect(page).toHaveText('Посібник з виконання робіт/проектів');
 await qawolf.create();
});
```

Imports

Для тестування вам спочатку знадобиться бібліотека qawolf, яка побудована на базі бібліотеки Playwright від Microsoft:

```
const qawolf = require('qawolf');
```

beforeAll

Щоб запустити тест, у блоці beforeAll у Jest відбувається кілька дій. Тест запускає браузер і створює новий контекст, секретний сеанс браузера. Цей контекст передається в метод qawolf.register, щоб QA Wolf міг отримати до нього доступ. Нарешті, створюється нова сторінка:

```
let website, text;

beforeAll(async () => {
 website = await qawolf.launch();
 const myContext = await website.newContext();
 await qawolf.register(my.Context);
 text = await myContext.newPage();
});
```

afterAll

Після закінчення тесту браузер закривається за допомогою AfterAll у Jest. Після закінчення тесту запис зупиняється, і відео зберігається.

```
afterAll(async () => {
 await qawolf._stopVideos();
 await browser._close();
});
```

Сам тест міститься в тестовому блоці Jest із заданим ім’ям. (у цьому прикладі “bakalavrInfo”). Спочатку тест переходить на вказаний URL. Потім він перевіряє, чи містить він текстову сторінку https://dl.tntu.edu.ua/login.php/. В кінці

перевіряється, що на сторінці присутній текст «Посібник з виконання робіт/проектів»

Для цього необхідно створити файл робочого процесу; щоб створити файл робочого процесу для вашого постачальника CI, виконайте таку команду в каталозі проекту:

```
npm init qawolf
```

Буде запропоновано вам вибрати свого постачальника IC із списку, вибираємо GitHub Actions.

Файл робочого процесу буде створено в наступному місці ./.github/workflows/qawolf.yml.

Цей тест працює завжди, незалежно від того, яку гілку проекту хтось коммітить. Він спрацьовує кожного разу, коли ви фіксуєте. Файл /qawolf.yml містить:

```
name: myTesting
on:
        push:
         # test every branch
         branches: “*”
        # schedule:
        # - cron: “() * * * *” # every hour
jobs:
 test:
        runs-on: ubuntu-18.04

        steps:
         - uses: actions/checkout@v2
         - uses: actions/setup-node@v1
```

```
- uses: Microsoft/playwright-github-action@v2
- uses: actions/cache@v2

with:
        path: ~/.nmp
        key: ${{ runner.os }}-node-$ {{ hashFiles('root/package-lock.json')
        restore-keys:
          ${{ runner.os }} -node-

-run: nmp install

#name to start our local server
#  run: nmp run start & npx wait-on http://localhost:3000

-run: nxp qawolf test (--headless)
-env:
  FFMPEG_PATH: /user/bin/ffmpeg # to record our video
  QAW_ARTIFACT_PATH: $ {{ github.workspace }}/artifacts
  #LOGIN_PASSWORD: $ {{ secrets.PASSWORD }}

-name: Upload Artifacts
 if: always()
 uses: _actions/uploading-artifacts@pro
 with:
   name: qawolf
   path: $ {{ github.workspace }}/artifact
```

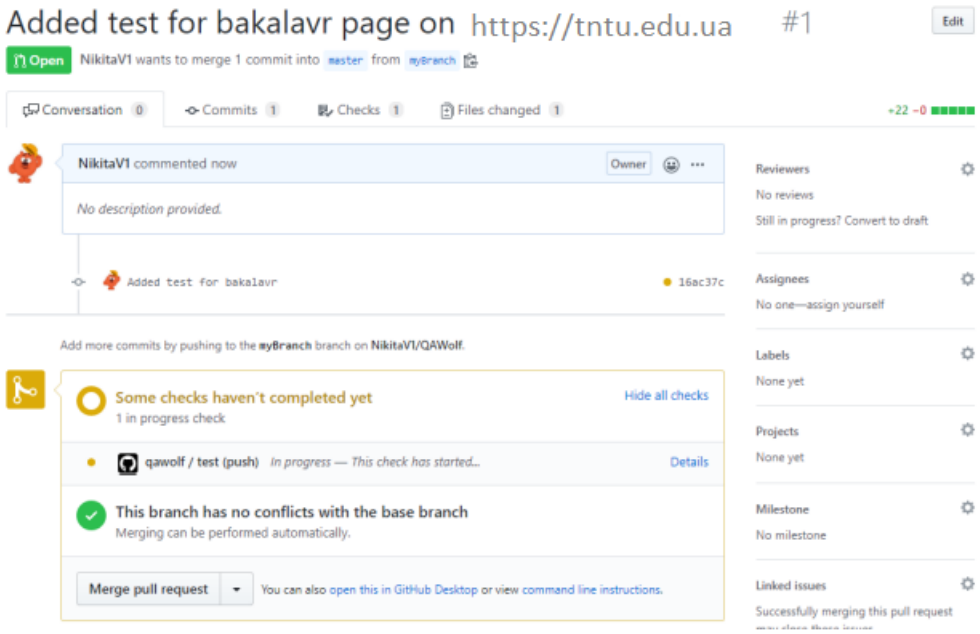

Рисунок 4.3 – Приклад запуску автоматичних тестів після коміту до репозиторію

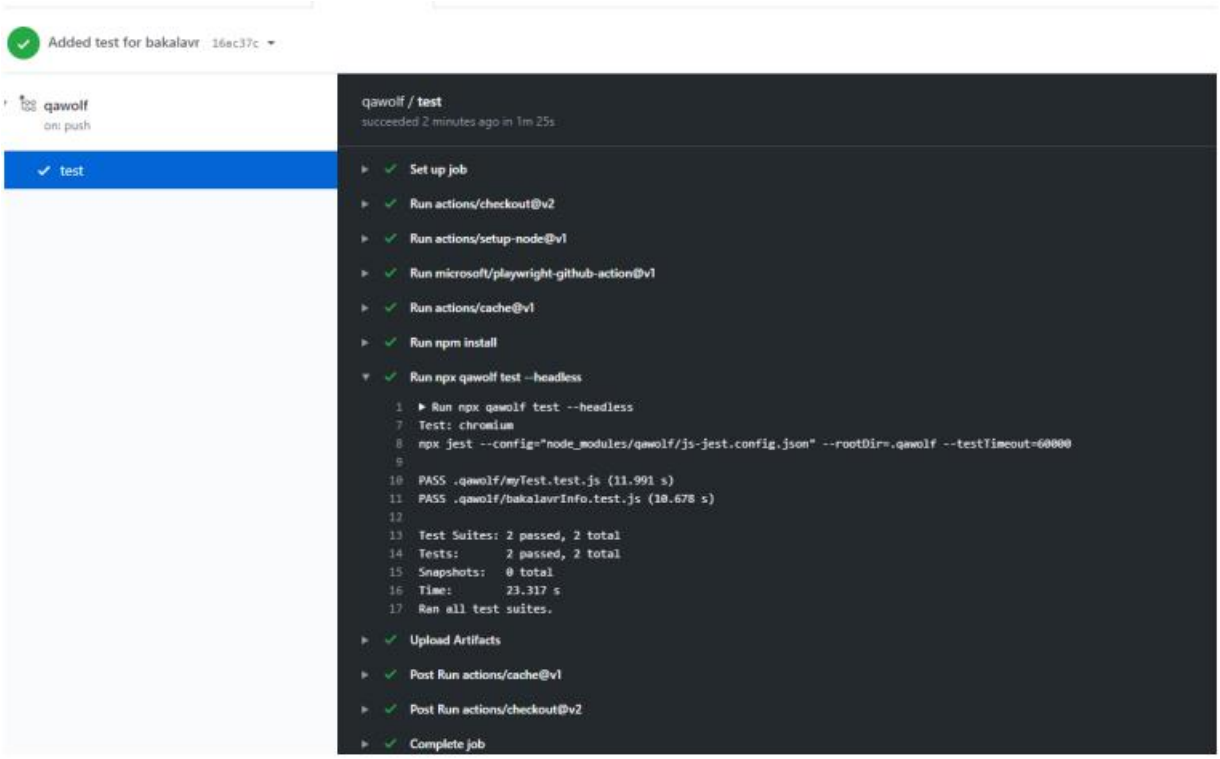

Рисунок 4.4 – Результат прогону тестів на віддаленому сервері

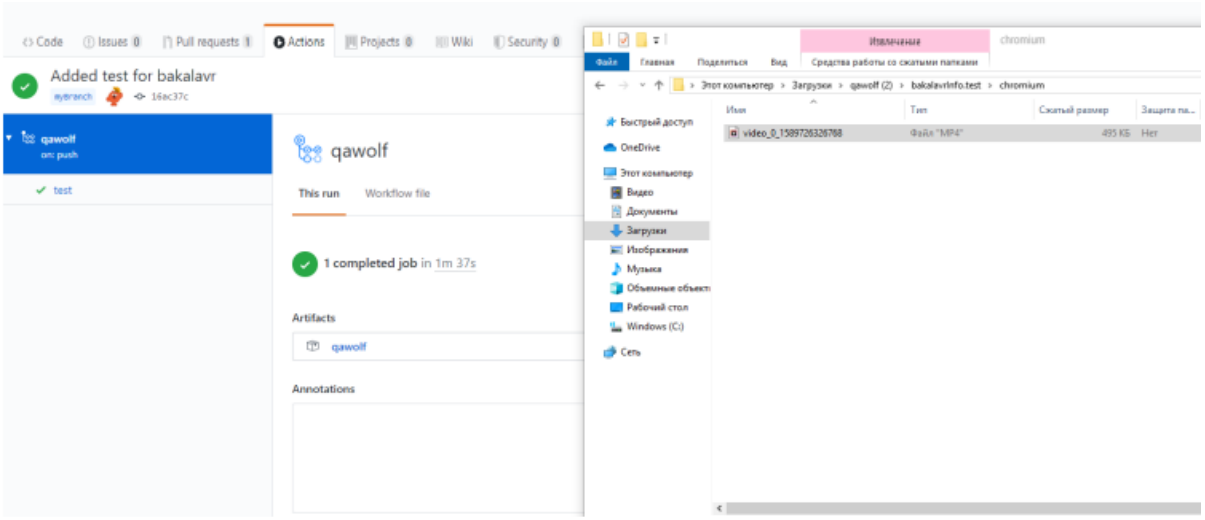

Рисунок 4.5 – Відеозапис результату і час виконання тестів

4.3 Програмна реалізація автоматизованої звітності до проекту

Для звітування використовується месенджер Slack. Він безкоштовний і може бути інтегрований з низкою зовнішніх серверів. Його API можна використовувати для кастомізації додатків для Android та IOS, а також десктопних версій системи звітності.

Створіть веб-хук Slack. Це URL-адреса, яка дозволяє програмно надсилати повідомлення Slack. Коли тест не пройде успішно, ви зробите POST-запит на цю URL-адресу.

Спочатку вам потрібно створити додаток Slack, який відповідатиме за надсилання повідомлень про помилки. Спочатку перейдіть на сайт Slack API. У правому верхньому куті ви побачите зелену кнопку з надписом «Створити новий додаток».

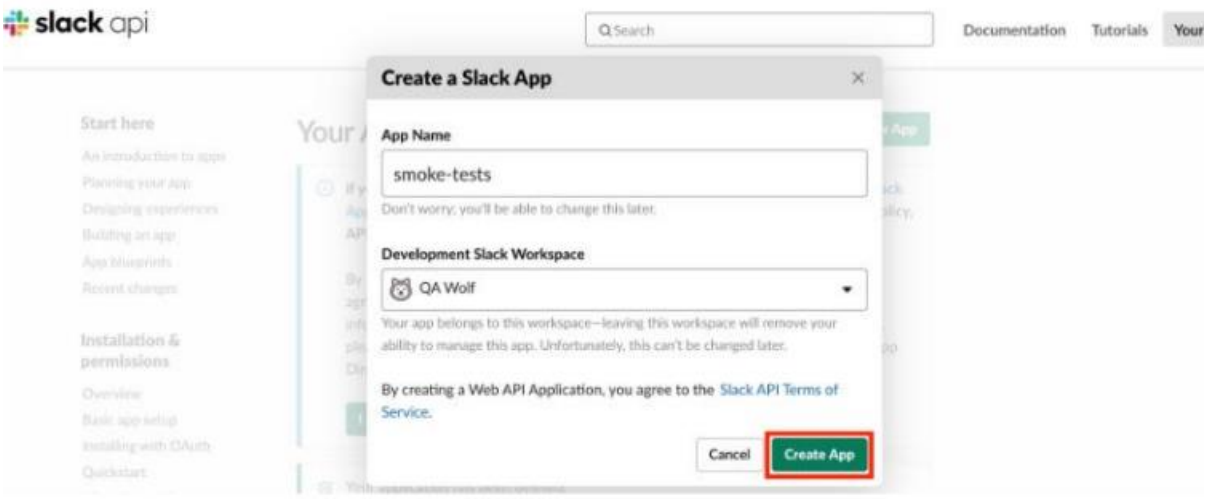

Рисунок 4.6 – Створення програми у Slack API

Далі потрібно зробити запит:

curl -Z POST -C та ‘Content-types: app/json’ -data ‘{“press texting”:”Your test failed!”}
‘https://hooks.slack.com/services/SECRET

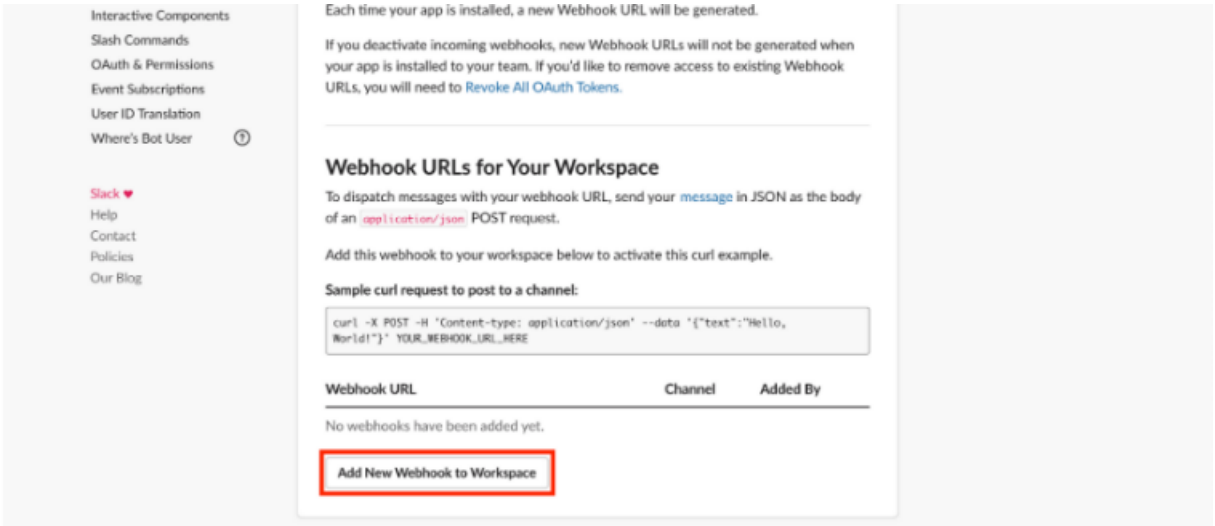

Рисунок 4.7 - Налаштування веб-кроку у Slack

Тепер коли ви створили свій веб-крок у Slack, наступним кроком буде оновлення робочого процесу. Файл робочого процесу GitHub Actions потрібно оновити. Додайте крок для відправки POST-запиту після завершення тесту.

Замість того, щоб вставляти URL-адресу безпосередньо в файл сценарію, ми добавимо її в секрет в репозиторії. Секрет - це зашифрована змінна середовища, яка зберігає конфіденційну інформацію. Зберігаючи URL-адресу веб-камери в таємниці, ви запобігаєте можливості її перегляду та використання іншими особами. Додайте новий секрет до конфігурації сховища: встановіть SLACK_WEBHOOK_URL як URL-адресу для веб-хука Slack.

Тепер оновіть файл робочого процесу. Додайте наступні рядки в кінець файлу «.github / workflows / qawolf.yml». Ці рядки вказують GitHub робити POST-запит до веб-хука Slack, коли тест зупиняється з помилкою.

```
name: Process of posting Slack Message to our account

    if: failure()

    run:

     curl -Z POST -C та ‘Contenting-types: app/json’ -data ‘{“press text”:”TNTU main pages test fail”}’${{ secrets.SLACK_WEBHOOK_URL }}
```