DOI 10.31891/2307-5732-2025-347-1
УДК 004.05

АНДРУСЯК ІВАННА
Національний університет "Львівська Політехніка"
https://orcid.org/0000-0001-6601-4374
e-mail: ivanna.v.andrusyak@lpnu.ua
СЕВЕРИНЕНКО ДАНИЛО
Національний Університет "Львівська Політехніка"
https://orcid.org/0000-0002-1708-316X
e-mail: danylo.y.severynenko@lpnu.ua

МЕТОДИ ТА ЗАСОБИ МОНІТОРИНГУ ФУНКЦІОНУВАННЯ ВЕБ-СЕРВІСІВ

У роботі розглядаються методи та засоби моніторингу функціонування веб-сервісів. У дослідженні оцінюються існуючі рішення, такі як Datadog, New Relic, Dynatrace та Uptime Robot, з акцентом на їх функціональність, переваги та недоліки. Описано вимоги до системи та обгрунтувано вибір програмних засобів. Детально описано розробку фреймворку для моніторингу веб-сервісів, зосереджуючись на автоматизації, аналізі даних у режимі реального часу та генерації звітності.

Ключові слова: автоматизований моніторинг, веб-сервіси, надійність сервісів, управління веб-сервісами, безперервна інтеграція, інструменти моніторингу, автоматизація

IVANNA ANDRUSYAK
DANYLO SEVERYNENKO
Lviv Polytechnic National University

METHODS AND MEANS OF MONITORING THE FUNCTIONING OF WEB SERVICES

The paper explores the development of methods and tools for automated monitoring systems to ensure the reliable operation of web services. In today's digital landscape, where a large portion of business and communication depends on stable web services, ensuring their reliability and performance is critical. This research highlights the need for a robust automated monitoring system capable of real-time fault detection to address issues promptly, enhance service reliability, and improve user satisfaction.

The paper evaluates existing monitoring solutions such as Datadog, New Relic, Dynatrace, and Uptime Robot, analyzing their capabilities, strengths, and limitations. These tools are essential for monitoring IT infrastructure and applications, offering features such as real-time data analysis, automated alerts, and integration across various programming frameworks. However, challenges like high implementation costs and system complexity underscore the need for scalable, customizable solutions, especially for small and medium-sized businesses.

Additionally, the authors present a framework for web service monitoring that emphasizes automation through the use of specialized browser management tools, which convert program commands into browser actions. The framework incorporates a class-based page design to facilitate modular testing and streamline maintenance. Report generation after testing is another key component, providing valuable insights through metrics such as test duration and error frequency.

Integrating these systems into continuous integration environments is achieved through tools like Jenkins for remote script execution, Allure for structured reporting, and Docker for flexible environment deployment. This architecture ensures seamless, automated testing, efficient data analysis, and optimized resource allocation, helping address technical and operational challenges. The findings provide a practical guide for developing automated monitoring systems, contributing to improved stability and efficiency of web services in a fast-evolving digital world.

Keywords: automated monitoring, web services, service reliability, web services management, continuous integration, monitoring tools, automation

Постановка проблеми та її рішення

У сучасному світі, де значна частина нашого повсякденного життя переноситься до віртуального простору, надійне та ефективне функціонування веб-сервісів стає ключовою складовою успіху для бізнесу, комунікації та безпеки. Однак враховуючи складність роботи сервісів, необхідність вдосконалення методів моніторингу та ефективного управління функціонуванням стає очевидною. В цьому контексті виникає потреба у створенні інформаційної системи, яка забезпечить автоматизований моніторинг та швидке виявлення можливих помилок у веб-сервісі.

Актуальність роботи обумовлена низкою факторів. По-перше, інтенсивний розвиток технологій та велика конкуренція на ринку веб-сервісів призводять до зростання вимог до їхньої надійності та ефективності. Забезпечення постійного та коректного функціонування стає стратегічно важливим завданням для підтримки позитивного іміджу компаній та задоволених користувачів. По-друге, велика кількість веб-сервісів, які обслуговують чимало користувачів одночасно, ускладнює завдання моніторингу та виявлення похибок. Безумовно, традиційні методи не завжди забезпечують достатню швидкість та ефективність у реальному часі. По-третє, через зростання кількості кібератак та програмних помилок виникає серйозна потреба вдосконалити засоби захисту веб-сервісів. Саме тому, інформаційна система автоматизованого моніторингу стане ключовим елементом в системі кібербезпеки, який дозволить оперативно виявляти та вирішувати потенційні загрози. Тож, розробка інформаційної системи для автоматизованого моніторингу безперебійності роботи веб-сервісів є не лише необхідною, але й вкрай перспективним завданням, яке відповідає сучасним вимогам інформаційного суспільства.

Мета даної роботи полягає у розробці та впровадженні інформаційної системи, яка забезпечить автоматизований моніторинг та надійне виявлення потенційних несправностей у функціонуванні веб-сервісу.

Об’єкт дослідження — інформаційна система автоматизованого моніторингу коректного функціонування веб-сервісу.

Предмет дослідження — завдання та функції, пов'язані з розробкою та ефективним функціонуванням інформаційної системи, зокрема виявлення потенційних несправностей у функціонуванні веб-сервісу, оперативне реагування на можливі порушення та забезпечення його надійності та доступності.

Аналіз існуючих систем

Datadog — це хмарна служба моніторингу, яка об'єднує різні типи моніторингу в одній платформі, включаючи моніторинг серверів, контейнерів, баз даних, інструментів і сервісів, а також додатків у реальному часі. Це рішення створене для обслуговування IT-інфраструктур великого масштабу та надає розробникам, адміністраторам систем та іншим IT-професіоналам інструменти для автоматизації моніторингу та оповіщення, аналізу продуктивності, візуалізації даних та багато іншого. Datadog постійно розвивається, пропонуючи нові можливості та поліпшення, що робить його одним із найпопулярніших рішень для моніторингу сучасних хмарних інфраструктур.

New Relic — це платформа аналітики продуктивності додатків, яка дає можливість IT-спеціалістам та розробникам спостерігати за станом та ефективністю своїх програмних продуктів і інфраструктури в реальному часі. Платформа пропонує рішення для моніторингу веб-додатків, мобільних додатків, серверів та інших IT-сервісів. Аналізучи дані характеристики, до основних переваг системи New Relic слід віднести інтеграцію з широким спектром мов програмування та фреймворків, гнучку і масштабовану архітектуру, а також детальний аналіз і звітність про продуктивність. Однак, із врахуванням широкого спектру функціональності New Relic, виріс і її рівень складності - новим користувачам може знадобитися деякий час, щоб освоїтися з усіма можливостями та інструментами, які надає платформа. Крім цього, для малого та середнього бізнесу ціноутворення може бути непрозорим та відчуватись дорогим, зважаючи на обсяги їх потреб та ресурсів.

Dynatrace є однією з провідних платформ у галузі автоматизованого моніторингу веб-сервісів і додатків, яка стала визнаним лідером завдяки своїй здатності до глибокого моніторингу з повним стеком технологій в режимі реального часу. Враховуючи основні характеритики, Dynatrace – це потужна система моніторингу з AI, що підходить великим компаніям для комплексного аналізу їх IT-інфраструктури. Незважаючи на свою вартість та складність, вона пропонує глибокий інсайт і високу автоматизацію, але вимагає відповідних ресурсів та експертизи для ефективного використання.

Uptime Robot – це сервіс для моніторингу веб-сайтів, який стежить за їхнім доступом в інтернеті та оперативно сповіщає користувачів про будь-які проблеми з доступністю. Використання Uptime Robot є корисним для власників веб-сайтів та IT-фахівців, які хочуть стежити за стабільністю своїх онлайн-сервісів та оперативно реагувати на проблеми, що виникають. У той же час, простота використання робить його доступним і для недосвідчених користувачів.[1]

Автоматизація процесів та її роль у сучасному світі

Автоматизація – це процес впровадження технічних, програмних та організаційних рішень, спрямованих на виконання робіт, завдань або процедур без прямого людського втручання або із мінімальним його участю. Метою автоматизації є підвищення продуктивності, ефективності, точності, а також зниження витрат ресурсів, ризиків та людських помилок[2].

Автоматизовані системи моніторингу надають цілу низку переваг, які значно впливають на ефективність, безпеку, якість і здатність до інновацій у цих галузях.

- Підвищення продуктивності: Автоматизовані системи можуть працювати 24/7 без перерви, що важливо для процесів, де потрібне безперервне спостереження, наприклад, в енергетиці або телекомунікаціях. Це забезпечує неперервність бізнес-операцій і зменшує можливість людських помилок, які можуть виникати внаслідок втоми або неуважності.
- Точність та консистентність: Автоматизовані системи забезпечують високий рівень точності та уніформованості у вимірюваннях і зборі даних. Вони мінімізують ризики, пов'язані з людським фактором, і забезпечують точність виконання складних процедур, що критично важливо для таких галузей, як фармацевтика або аерокосмічна промисловість.
- Оптимізація процесів: Системи автоматизованого моніторингу можуть аналізувати великі об'єми даних та ідентифікувати патерни, які людина може не помітити. Це дозволяє оптимізувати процеси, виявляти та усувати неефективності, що веде до зниження витрат і підвищення загальної продуктивності.
- Раннє виявлення проблем: Через здатність постійно збирати і аналізувати дані, автоматизовані системи моніторингу можуть швидко виявити аномалії, що дозволяє своєчасно реагувати на потенційні проблеми, зменшуючи час простою і витрати на ремонт.
- Зменшення ризиків для здоров'я та безпеки: В робочих середовищах з високим рівнем ризику, таких як хімічна промисловість або гірничодобувна індустрія, автоматизовані системи можуть моніторити умови без необхідності втручання людини, знижуючи ризик травматизму та забезпечуючи безпеку персоналу.
- Підвищення якості: Автоматизація може підвищити стандарти контролю якості, забезпечуючи стабільність і однорідність продукції, що особливо важливо в харчовій промисловості та виробництві

електроніки.

- Екологічний вплив: Автоматизовані системи можуть допомогти у зниженні витрат енергії та оптимізації використання ресурсів, сприяючи сталому розвитку та зменшенню негативного впливу на навколишнє середовище.
- Комплексний аналіз: Інтеграція автоматизованих систем моніторингу з іншими бізнес-системами (наприклад, ERP або CRM) може надати глибший аналітичний огляд всіх аспектів бізнесу, дозволяючи краще розуміти процеси та приймати обгрунтовані рішення.
- Дотримання стандартів: В багатьох галузях існують строгі регуляторні вимоги щодо збору та звітності даних. Автоматизовані системи моніторингу можуть забезпечити точне та своєчасне дотримання цих стандартів, уникнувши можливих штрафів і санкцій.
- Масштабованість: Якщо бізнес розширює свою діяльність, автоматизовані системи можуть бути легко масштабовані для збільшення обсягів виробництва або моніторингу без необхідності пропорційного збільшення персоналу.

Ці переваги роблять автоматизовані системи моніторингу цінними інструментами для підвищення ефективності та конкурентоспроможності бізнесу в сучасному, швидко змінюваному світі.

Незважаючи на безліч переваг, системи автоматизованого моніторингу мають і ряд потенційних недоліків та викликів, які можуть вплинути на їхню ефективність та прийнятність.

Розглянемо деякі з цих недоліків більш детально:

- Висока вартість початкового впровадження: Встановлення автоматизованих систем моніторингу часто вимагає значних капіталовкладень для закупівлі обладнання, програмного забезпечення та інтеграції систем. Це може бути обтяжливим для малих та середніх підприємств.
- Складність інтеграції: Інтеграція нових систем моніторингу з існуючими бізнес-процесами та системами інформаційних технологій може бути складною. Це може призвести до технічних складнощів та потребує часу для правильного налаштування і тестування.
- Потреба у навчанні персоналу: Для ефективної роботи з автоматизованими системами моніторингу співробітникам може знадобитись спеціалізоване навчання, що може виявитися часозатратним і дорогим.
- Залежність від технологій: Надмірна залежність від автоматизованих систем може створити ризики у разі їх збою. Несправність однієї частини системи може призвести до зупинки всього процесу.
- Кібербезпека: З розвитком технологій збільшуються і ризики кібератак. Автоматизовані системи моніторингу, особливо ті, що підключені до інтернету, можуть бути вразливими до хакерських атак, які можуть призвести до витоку даних або шкідливих збоїв в роботі.
- Обмеження гнучкості: Певні автоматизовані системи можуть бути не настільки гнучкими, щоб швидко адаптуватися до змін у бізнес-процесах або виробничих потребах, що може обмежувати здатність компанії швидко реагувати на ринкові зміни.
- Втрата професійних навичок: Із зростанням залежності від автоматизованих систем може відбуватись втрата професійних навичок серед працівників, оскільки ручні процеси стають менш поширеними.
- Складнощі у підтримці та оновленні: Технології швидко застарівають, і системи потребують регулярного оновлення та заміни, що може бути дорогим і технічно складним процесом.
- Ризик узагальнення даних: Автоматизовані системи моніторингу можуть іноді упускати нестандартні або аномальні умови, які людина могла б виявити. Це може призвести до недооцінювання ризиків або непередбачених проблем.
- Етичні та соціальні проблеми: Застосування автоматизованих систем також може призводити до соціальних змін, таких як зменшення кількості робочих місць та етичних дилем щодо приватності та контролю над даними.

Кожна з цих проблем вимагає ретельного розгляду та планування при розробці та впровадженні систем автоматизованого моніторингу. Управління ризиками, постійне оновлення технологій, розробка заходів кібербезпеки, а також планування розвитку навичок співробітників є важливими кроками для мінімізації цих недоліків.

Моделювання об'єктів предметної області

У рамках розроблюваної системи моніторингу веб-сервісу користувачі матимуть можливість швидко оцінити загальний стан своїх онлайн-систем завдяки заздалегідь підготовленим звітам, які будуть доступні щоранку. Ці звіти, актуальні станом на початок нової доби, дають користувачам швидкий огляд основних аспектів роботи системи, дозволяючи тим самим зберігати час та ресурси, які зазвичай витрачаються на повсякденний детальний моніторинг. Такий підхід є ефективним, враховуючи малу імовірність важливих змін у стані системи за короткий період. Тим не менше, система забезпечує гнучкість: у випадках, коли є необхідність у моніторингу в реальному часі, користувачі можуть обрати для швидкої перевірки як весь спектр функціональності веб-сервісу, так і конкретні його елементи. Це стає в нагоді, коли потрібен терміновий моніторинг окремих компонент, адже щоденні всебічні перевірки зазвичай не є необхідними.

На діаграмі класів (рис. 1) представлено структуру серверної частини інформаційної системи:

- Клас broadcast_message_handler: Методи для розсилки команд з повідомленнями та обробки рішень. Взаємодіє з класами User та ChatInfo.
- Клас User: Містить атрибути для ідентифікації та статусу користувача (ID, ім'я, права доступу тощо). Реалізує методи для отримання та обробки даних користувача, а також має зв’язок з broadcast_message_handler, admin_handler, onboarding_handler.
- Клас admin_handler: Методи для адміністрування, статистики та експорту даних користувачів.
- Клас ChatInfo: Атрибути для зберігання інформації про чат (ID, тип, назва). Методи для створення та отримання даних чату. Взаємодіє з broadcast_message_handler, onboarding_handler.
- Клас onboarding_handler: Методи для сповіщень нових користувачів, реєстрації бота в бесідах та завершення сеансу. Пов'язаний з ChatInfo.
- Клас BuildInfo: Атрибути для зберігання інформації про збірки (ID, статус, URL тощо). Методи для створення та отримання інформації про збірки. Зв'язується з check_status_handler.
- Клас check_status_handler: Методи для відображення статусу та перевірки статусу збірок у реальному часі. Взаємодіє з BuildInfo.

Фреймворк для автоматизаційної перевірки веб-сервісу

Спершу побудовано фреймворк, призначений для автоматизації процесу перевірки веб-сервісів. Основним інструментом у цьому процесі є застосування спеціалізованого інструменту для управління браузером. Такий інструмент використовує протокол JSON Wire для конвертації коду програми в команди, які зрозумілі для браузерної сесії. Це дозволяє програмі, що виступає в ролі клієнта, взаємодіяти з браузером, що виконує роль сервера.

Першим кроком у реалізації фреймворку є імплементація класів сторінок за патерном Page Object. Цей підхід передбачає створення класів, що містять опис основних атрибутів веб-сторінки, логіки можливих дій користувача на цій сторінці, а також методи для отримання необхідних даних для перевірок. Такий підхід забезпечує модульність і легкість у підтримці тестових скриптів. Для розробки таких класів рекомендується використовувати об'єктно-орієнтовану мову програмування, яка підтримує принципи, аналогічні принципам реалізації класів.

Генерація звітності моніторингу після автоматизованого тестування є ключовим етапом для оцінки ефективності і виявлення проблемних аспектів системи. Основне завдання тут - обробка результатів тестів, які вже перебувають у форматі JSON, їх комбінація та сортування за типами виявлених проблем. Це сприяє більш зручному та ефективному аналізу виявлених помилок та інших проблемних моментів.

Забезпечення доступності та читабельності зібраних даних є надзвичайно важливим. Тому результуючі дані представляються у форматі веб-сторінки у браузері, що дозволяє легко структурувати інформацію і зробити її зрозумілою для ширшого кола користувачів. Такий підхід не лише спрощує доступ до результатів тестування, але й робить процес їх аналізу більш інтуїтивним і зручним.

Крім аналізу основних результатів, звіт також включає додаткові метрики, такі як тривалість виконання кожного окремого скрипта та загальний час виконання всіх тестів. Це надає цінну інформацію про продуктивність та ефективність тестів. Аналіз стабільності скриптів, з урахуванням попередніх запусків, допомагає ідентифікувати потенційні проблеми, які можуть виникати від часу до часу, а також оцінювати надійність тестового процесу в цілому.

Для реалізації цього процесу використовуються готові бібліотеки для аналізу даних та побудови звітності. Такі бібліотеки забезпечують гнучкість та легкість у налаштуванні, дозволяючи створювати інтуїтивно зрозумілі та чіткі звіти. Використання цих інструментів дозволяє командам розробників швидко і ефективно оцінювати результати тестування, що сприяє швидкому виявленню та вирішенню проблем у роботі веб-сервісів.

Розробка фреймворку та його налаштування для автоматизації моніторингу

Для подальшої роботи додано всі вищеописані бібліотеки через pom.xml, який є стандартним файлом для опису залежностей в Maven[3]. Дані інструменти будуть завантажуватись з глобального репозиторію Maven при їх відсутності у локальному сховище зберігання бібліотек (Рис.1).

Після опису знаходження необхідних елементів, було перейдено до створення класів кожної сторінки. Кожна з них наслідується від базової сторінки, яка містить у собі логіку дій з базовими інструментами, такими як меню навігації, а також сайдбар, які доступні з будь-якої сторінки веб-ресурсу. Під час ініціалізації необхідно передавати у конструктор об’єкт класу Page, логіка якого попередньо імплементована у бібліотеці Playwright[4]. За допомогою даного об’єкту виконується вся взаємодія з браузером через веб-драйвер. Було створено методи отримання потрібних елементів за їхніми ідентифікаторами, їхнього тексту, а також в окремих випадках атрибутів[5].

```xml
<properties>
    <suiteFile>healthcheck-monitoring</suiteFile>
    <playwright.version>1.39.0</playwright.version>
    <testng.version>7.7.0</testng.version>
    <lombok.version>1.18.24</lombok.version>
    <allure.version>2.22.1</allure.version>
    <assertj.version>3.23.1</assertj.version>
    <allure.testng>2.17.3</allure.testng>
    <allure.maven>2.10.0</allure.maven>
    <maven-surefire-plugin.version>2.19.1</maven-surefire-plugin.version>
    <aspectj.version>1.9.19</aspectj.version>
    <vavr.version>0.10.4</vavr.version>
</properties>

<dependencies>
    <dependency>
        <groupId>com.microsoft.playwright</groupId>
        <artifactId>playwright</artifactId>
        <version>${playwright.version}</version>
    </dependency>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <version>${lombok.version}</version>
        <scope>provided</scope>
    </dependency>
```

Рис. 1. Опис залежностей в pom.xml

```java
3 usages  Orest Torskyi
public class IsmHomePage extends BaseIsmPage {

    1 usage  Orest Torskyi
    public IsmHomePage(Page homePage) { super(homePage); }

    1 usage  Orest Torskyi
    public String getTelegramAdmissionBotLink() {
        return page.locator(XPATH_TELEGRAM_BOT_LINK.getLocator()).getAttribute(HREF.getAttribute());
    }

    1 usage  Orest Torskyi
    public String getTelegramChatLink() {
        return page.locator(XPATH_TELEGRAM_CHAT_LINK.getLocator()).getAttribute(HREF.getAttribute());
    }

    1 usage  Orest Torskyi
    public String getLinkedInLink() {
        return page.locator(XPATH_LINKEDIN_LINK.getLocator()).getAttribute(HREF.getAttribute());
    }

    1 usage  Orest Torskyi
    public String getFacebookLink() {
        return page.locator(XPATH_FACEBOOK_LINK.getLocator()).getAttribute(HREF.getAttribute());
    }

    1 usage  Orest Torskyi
    public Boolean isTelegramBotQrCodeDisplayed() {
        return page.locator(XPATH_TELEGRAM_QR_CODE.getLocator()).isVisible();
    }
```

Рис. 2. Клас головної сторінки веб-сервісу

Наступним кроком є створення скриптів – для цього використано об’єкти класів сторінок для виконання послідовних кроків навігації та отримання необхідних значень, а також інструменти TestNG для валідації значень:

Завершальним етапом у запуску валідації даних було формування с’ютів в xml-форматі – у них вказуються теги suite, де описуються параметри паралелізації, а також вкладені теги test, classes, class, які позначають саме ті тестові класи, скрипти яких повинні пройти валідацію. Для ще більш зручного розбиття було використано вкладені теги include у тегах methods для вибору окремого тестового методу з класу у тестуванні с’юта

```java
Orest Torskyi
@BeforeMethod
private void getHomePage() { homePage = new IsmHomePage(page); }

Orest Torskyi
@Test
public void verifyMainPageIsmContactInfoResources() {
    SoftAssert softAssert = new SoftAssert();

    softAssert.assertTrue(homePage.isTelegramBotQrCodeDisplayed());
    softAssert.assertTrue(homePage.isTelegramLinkDisplayed());
    softAssert.assertTrue(homePage.isLinkedInLinkDisplayed());
    softAssert.assertTrue(homePage.isFacebookLinkDisplayed());
    softAssert.assertTrue(homePage.getTelegramAdmissionBotLink().contains(TELEGRAM_ADMISSION_BOT.getSocialMediaLink()));
    softAssert.assertTrue(homePage.getTelegramChatLink().contains(TELEGRAM_ISM_CHAT.getSocialMediaLink()));
    softAssert.assertTrue(homePage.getLinkedInLink().contains(LINKEDIN.getSocialMediaLink()));
    softAssert.assertTrue(homePage.getFacebookLink().contains(FACEBOOK.getSocialMediaLink()));

    softAssert.assertAll();
}

Orest Torskyi
@Test
public void verifyBachelorProgramsList() {
    List<String> actualBachelorPrograms = homePage
            .getIsmSideBar()
            .getProgramsFor(IsmSidebar.Program.BACHELOR);

    assertTrue(actualBachelorPrograms.containsAll(BACHELOR_PROGRAMS.getActiveIsmPrograms()));
}
```

Рис. 3. Тестовий клас зі автотестами для головної сторінки

```xml
<suite name="Available Courses Tests" parallel="methods" thread-count="2">

    <test name="Verify available courses on the site">
        <classes>
            <class name="com.ism.MainPageTest">
                <methods>
                    <include name="verifyBachelorProgramsList"/>
                    <include name="verifyMagisterProgramsList"/>
                    <include name="verifyPhdProgramsList"/>
                </methods>
            </class>
        </classes>
    </test>
</suite>
```

Рис. 4. С'ют в xml-форматі для перевірки доступних курсів на головній сторінці

Далі було необхідно налаштувати генерацію звітності після завершення тестування с'юта. Для цього підключено плагін Allure у файлі опису збірки pom.xml, який буде автоматично збирати згенеровані результати скриптів з JSON-формату у структуровану сторінку для аналізу[6]. Дана звітність зберігається у пакеті allure-results.

```xml
<plugins>
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
            <source>8</source>
            <target>8</target>
        </configuration>
    </plugin>
    <plugin>
        <groupId>io.qameta.allure</groupId>
        <artifactId>allure-maven</artifactId>
        <version>2.10.0</version>
        <configuration>
            <reportVersion>2.14.0</reportVersion>
        </configuration>
    </plugin>
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-surefire-plugin</artifactId>
        <version>${maven-surefire-plugin.version}</version>
        <configuration>
            <argLine>-javaagent:"${settings.localRepository}/org/aspectj/aspectjweaver/${aspectj.version}/aspectjweaver-${asp
            <systemProperties>
                <property>
                    <name>allure.results.directory</name>
                    <value>${project.build.directory}/allure-results</value>
                </property>
```

Рис. 5. Плагін автоматичної генерації звітності Allure

Оскільки було необхідно забезпечити безперевний доступ до середовища безперервної інтеграції, використано віддалений сервер для налаштування Jenkins. Для його конфігурації виконано доступ до сервера через SSH[7]. Після успішного підключення до сервера, завантажена остання версія Jenkins, використовуючи curl-команди в терміналі.

Завантаживши файл, Jenkins запущено командою java -jar jenkins.war. Для доступу до веб-інтерфейсу Jenkins, використано IP-адресу та порт, на якому працює Jenkins-сервіс. Далі, обрано та встановлено потрібні Maven, Git, Docker та JDK плагіни, після чого налаштовано користувацький обліковий запис та права доступу [8].

Основним кроком є створення логіки вибору параметрів браузера та конкретного с'юта, створення контейнера на основі зображення з Docker Hub репозиторію, описаного у yml-файлі, стягнення фреймворку з Github-репозиторію, запуску с'юта, використовуючи Maven команду з попередньо вказаними опціями з інтерфейсу, а також прикріплення згенерованого Allure-звіту до джоби за допомогою скрипта, написаного на Groovy. Шлях до даного скрипта у проекті заданий у налаштуваннях джоби вручну[9].

```
parameters {
        string(name: 'BRANCH', defaultValue: 'master', description: 'Choose branch')
        choice(name: 'BROWSER', choices: ['chrome', 'firefox'], description: 'Choose browser for cross-browser testing
        choice(name: 'SUITE', choices: ['full health-check', 'navigation', 'localization', 'social-media', 'learning pr
}

stages {
        stage('Run suite') {
            steps {
                script {
                        // Access job parameters
                        switch(params.SUITE) {
                        case "full health-check":
                        suiteFile = "healthcheck-monitoring"
                        break
                        case "navigation":
                        suiteFile = "navigation"
                        break
                        case "localization":
                        suiteFile = "localization"
                        break
                        case "social-media":
                        suiteFile = "social-media"
                        break
                        case "learning programs":
                        suiteFile = "available-learning-programs"
                        break
                        }

                        currentBuild.displayName = "${params.BROWSER}, ${params.SUITE}"
                        // Run suite with mvn command
```

Рис. 6. Основні кроки Groovy-скрипта, використаного для конфігурації пайплайн джоби у Jenkins

Наступним кроком була реєстрація та отримання API-токена, за допомогою якого забезпечується комунікація між запитами користувача та їх опрацюванням на серверній частині бота. Для цього у Telegram-месенджері використовується інший бот BodFather, основа функція якого є генерація нових ботів та надання їм загальноприйнятих характеристик, таких як тег та назва. Після введення усіх необхідних даних був виданий необхідний API-токен.

В цей момент у середовищі неперервної інтеграції Jenkins ініціюється запуск завдання, яке включає декілька етапів: клонування репозиторію з автоматизованими скриптами, завантаження необхідних інструментів у Docker-контейнер, виконання відповідного тестового набору з урахуванням заданого параметра, а також створення звіту з результатами.

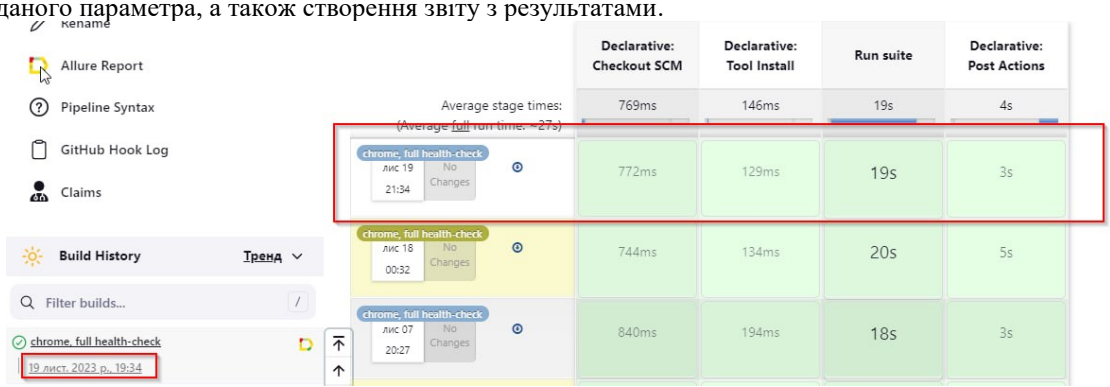

Рис. 7.Успішне проходження параметризованого запуску моніторингу з бота

Після очікування користувач повторно вибирає опцію з головного меню і переходить за посиланням. На початковій сторінці він має змогу побачити дату запуску, час виконання, ім'я користувача, який ініціював останню перевірку, а також деталі про репозиторій.

Рис. 9. Початкова сторінка звіту моніторингу

Після натискання кнопки ‘Allure Report’, користувач переходить до загального огляду звіту, який включає кількість тестів та відсоткове співвідношення між успішно виконаними та невдало проведеними тестами.

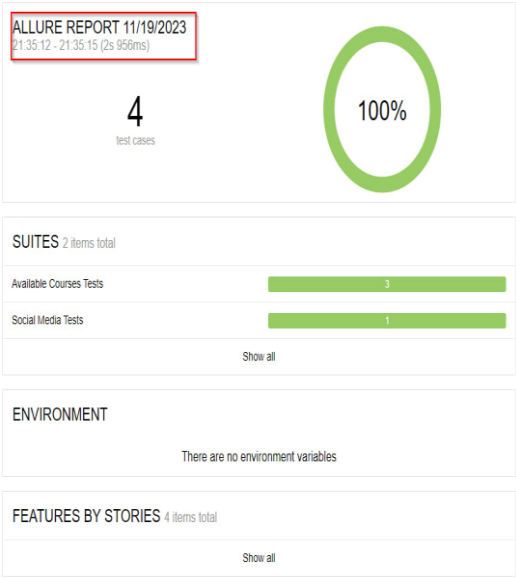

Рис. 8. Загальний опис звіту

Висновки

В результаті роботи реалізовано інформаційну систему автоматизованого моніторингу коректного функціонування веб-сервісу. Для цього було створено всі необхідні складові, а саме автоматизаційні скрипти з поділенням їх на окремі частини для можливості проведення моніторингу конкретного функціоналу, середовище безперервної інтеграції для віддаленого запуску скриптів, бота та його основної логіки функціонування на серверній частині з додатковою панеллю адміністратора, а також інтеграцію інструмента генерації звітів на основі виконаних перевіркок. Результати даної роботи можуть бути використані для подальшого розвитку та удосконалення інформаційних систем, спрямованих на підтримку стабільності та ефективності веб-сервісів у сучасному інформаційному середовищі.

Література

1. Sematext. (2023). Best server performance monitoring tools & software in 2023. *[Електронний ресурс]*. Отримано з https://sematext.com/blog/server-monitoring-tools/

2. Downey, A. B. (2015). *Think Python: How to think like a computer scientist* (2nd ed.). O’Reilly Media.

3. Apache Maven. (n.d.). Introduction to the POM. *[Електронний ресурс]*. Отримано з https://maven.apache.org/guides/introduction/introduction-to-the-pom.html

4. LambdaTest. (n.d.). What is Playwright? Playwright testing tutorial - A guide with examples. *[Електронний ресурс]*. Отримано з https://www.lambdatest.com/playwright

5. Baeldung. (n.d.). Introduction to Project Lombok. *[Електронний ресурс]*. Отримано з https://www.baeldung.com/intro-to-project-lombok

6. WebdriverIO. (n.d.). Allure reporter. *[Електронний ресурс]*. Отримано з https://webdriver.io/docs/allure-reporter/

7. DigitalOcean. (n.d.). How to use SSH to connect to a remote server. *[Електронний ресурс]*. Отримано з https://www.digitalocean.com/community/tutorials/how-to-use-ssh-to-connect-to-a-remote-server

8. Jenkins.io. (n.d.). Using a Jenkinsfile. *[Електронний ресурс]*. Отримано з https://www.jenkins.io/doc/book/pipeline/jenkinsfile/

9. freeCodeCamp. (n.d.). How to build your own Heroku with Dokku. *[Електронний ресурс]*. Отримано з https://www.freecodecamp.org/news/how-to-build-your-on-heroku-with-dokku/

References

1. Sematext. (2023). Best server performance monitoring tools & software in 2023. [Elektronnyi resurs]. Otrymano z https://sematext.com/blog/server-monitoring-tools/

2. Downey, A. B. (2015). Think Python: How to think like a computer scientist (2nd ed.). O'Reilly Media.

3. Apache Maven. (n.d.). Introduction to the POM. [Elektronnyi resurs]. Otrymano z https://maven.apache.org/guides/introduction/introduction-to-the-pom.html

4. LambdaTest. (n.d.). What is Playwright? Playwright testing tutorial - A guide with examples. [Elektronnyi resurs]. Otrymano z https://www.lambdatest.com/playwright

5. Baeldung. (n.d.). Introduction to Project Lombok. [Elektronnyi resurs]. Otrymano z https://www.baeldung.com/intro-to-project-lombok

6. WebdriverIO. (n.d.). Allure reporter. [Elektronnyi resurs]. Otrymano z https://webdriver.io/docs/allure-reporter/

7. DigitalOcean. (n.d.). How to use SSH to connect to a remote server. [Elektronnyi resurs]. Otrymano z https://www.digitalocean.com/community/tutorials/how-to-use-ssh-to-connect-to-a-remote-server

8. Jenkins.io. (n.d.). Using a Jenkinsfile. [Elektronnyi resurs]. Otrymano z https://www.jenkins.io/doc/book/pipeline/jenkinsfile/

9. freeCodeCamp. (n.d.). How to build your own Heroku with Dokku. [Elektronnyi resurs]. Otrymano z https://www.freecodecamp.org/news/how-to-build-your-on-heroku-with-dokku/