Міністерство освіти і науки України
Державний університет « Житомирська політехніка »
Факультет інформаційно- комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки

ПОЯСНЮВАЛЬНА ЗАПИСКА

до випускної кваліфікаційної роботи магістра

на тему : «Методи та технології кібербезпеки мануального тестування»

Виконав студент II- го курсу, групи КБм -22-1
спеціальності 125 « Кібербезпека »
_____ Башинський В.С.

Керівник
професор кафедри комп'ютерної інженерії та кібербезпеки, доктор технічних наук, доцент
_____ Воротніков В.В.

Рецензент
Доцент кафедри комп'ютерної інженерії та кібербезпеки, кандидат технічних наук, доцент
_____ Россінський Ю.М

ДЕРЖАВНИЙ УНІВЕРСИТЕТ « ЖИТОМИРСЬКА ПОЛІТЕХНІКА »
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО- КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА КІБЕРБЕЗПЕКИ
СПЕЦІАЛЬНІСТЬ 125 « КІБЕРБЕЗПЕКА »

ЗАТВЕРДЖУЮ
Завідувач кафедри
комп'ютерної інженерії
та кібербезпеки
Єфіменко А. А.
13 жовтня 2023р.

ЗАВДАННЯ
на випускну кваліфікаційну роботу

Студента Башинського Вадима Станіславовича
Тема роботи: Методи та технології кібербезпеки мануального тестування.

Затверджена Наказом університету від « 13 » жовтня 2023 р. №572/с
Термін здачі студентом закінченої роботи « 11 » грудня 2023 р.
Вихідні дані роботи (зазначається предмет і об'єкт дослідження):

Консультанти з випускної кваліфікаційної роботи із зазначенням розділів, що їх стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Єфіменко А.А	13.10.23	13.10.23
2	Єфіменко А.А	27.10.23	27.10.23
3	Єфіменко А.А	15.11.23	15.11.23

Керівник _____

Календарний план

№ з/п	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка задачі. Пошук, огляд та аналіз аналогічних розробок. Формулювання технічного завдання. Опрацювання літературних джерел.	13.10.23-18.10.23	Виконано
2	Аналіз проблем кібербезпеки в тестуванні програмного забезпечення	19.10.23 - 23.10.23	Виконано
3	Пошук видів автоматичного та мануального тестування	23.10.23 - 26.10.23	Виконано
4	Пошук інструментів тестування кібербезпеки	27.10.23 - 30.10.23	Виконано
5	Розробка методики тестування	31.10.23 - 5.11.23	Виконано
6	Налаштування програмного середовища	6.11.23 - 14.11.23	Виконано
7	Реалізація практичних кейсів з тестування кібербезпеки	15.11.23 - 25.11.23	Виконано
8	Аналіз результатів	25.11.23 - 7.11.23	Виконано
9	Захист	25.12.23	

Студент _____
(підпис)

Керівник _____
(підпис)

РЕФЕРАТ

Випускна кваліфікаційна робота магістра складається з тестування можливих кіберзагроз в програмному забезпеченні використовуючи програми якими користуються при мануальному та автоматизованому тестуванні. Пояснювальна записка до випускної роботи містить 77 сторінок, 43 ілюстрацій.

Метою роботи є встановлення взаємозв'язку між кібербезпекою та мануальним тестуванням а також показати як тестувальник може покращити тестування свого продукту на кіберзагрози завдяки програмам. В рамках цього дослідження, ми розглянемо загрози, які виникають під час мануального тестування та які можуть призвести до ризику для інформаційної безпеки. Ми також розглянемо методи виявлення цих загроз та інструменти, які можуть допомогти зробити процес мануального тестування більш безпечним і надійним.

КЛЮЧОВІ СЛОВА : МАНУАЛЬНЕ ТЕСТУВАННЯ , ВИДИ ТЕСТУВАНЬ, K6, POSTMAN, CHARLES

ABSTRACT

The master's thesis focuses on testing potential cyber threats in software using tools employed in both manual and automated testing. The accompanying explanatory note for the thesis comprises 78 pages with 43 illustrations.

The objective of the study is to establish the correlation between cybersecurity and manual testing, demonstrating how a tester can enhance the testing of their product for cyber threats using specific programs. Within this research, we will explore threats arising during manual testing that may pose risks to information security. Additionally, we will delve into methods for detecting these threats and tools that can contribute to making the manual testing process more secure and reliable.

KEYWORDS: MANUAL TESTING, TESTING TYPES, K6, POSTMAN, CHARLES

ЗМІСТ

ВСТУП…….. 7

1. АНАЛІЗ ПРОБЛЕМ КІБЕРБЕЗПЕКИ В ТЕСТУВАННІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ…… 9

1.1 Основні поняття мануального тестування…… 9

1.2 Види автоматичного та мануального тестування програмного забезпечення…….. 14

1.3 Проблеми кібербезпеки QA Engineer Програмного забезпечення 19

2. АНАЛІЗ МЕТОДІВ ТА ІНСТРУМЕНТІВ ТЕСТУВАННЯ КІБЕРБЕЗПЕКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ……. 26

2.1Види кіберзагроз та інструменти тестування кібербезпеки…….. 26

2.2 Обґрунтування вибору інструментарію для проведення досліджень……. 50

2.3 Розробка методики тестування…….. 52

3. ПРАКТИЧНА РЕАЛІЗАЦІЯ МЕТОДИКИ ТЕСТУВАННЯ…….. 53

3.1 Налаштування програм та середовища тестування……. 53

3.2 Практичні кейси з реалізації тестування кібербезпеки……... 59

3.3 Аналіз результатів тестування та розробка рекомендацій з застосування…….. 73

ВИСНОВОК……. 76

СПИСОК ЛІТЕРАТУРИ……. 77

ВСТУП

Актуальність теми. У постійно змінному цифровому світі, де технологічні інновації швидко впроваджуються в усі галузі діяльності, безпека інформації стає вкрай важливою та актуальною проблемою. Кіберзлочинці та зловмисники намагаються використовувати кожну можливу можливість для вчинення атак на цифрові системи та мережі, що призводить до загрози для конфіденційності, цілісності та доступності даних.

У цьому контексті, мануальне тестування програмного забезпечення визначається як критична складова процесу розробки. Цей метод використовується для виявлення помилок, вразливостей та можливих проблем, які можуть стати точкою входу для атак ззовні. Завдяки мануальному тестуванню розробники можуть бути впевнені в безпеці свого продукту перед випуском його на ринок.

Але, незважаючи на важливу роль, яку мануальне тестування відіграє в процесі розробки програмного забезпечення, саме через цю важливість воно може стати об'єктом кібератак та атак на інформаційну безпеку. Процес тестування може виявити вразливості, які, якщо залишити без відповідної уваги, стануть точкою доступу для зловмисників, порушуючи конфіденційність і цілісність даних та наносячи значний збиток організаціям та користувачам.

Метою кваліфікаційної роботи є глибокий розгляд важливої проблеми в галузі кібербезпеки: взаємозв'язку між кібербезпекою та мануальним тестуванням. В рамках цього дослідження, ми розглянемо загрози, які виникають під час мануального тестування та які можуть призвести до ризику для інформаційної безпеки. Ми також розглянемо методи виявлення цих загроз та інструменти, які можуть допомогти зробити процес мануального тестування більш безпечним і надійним.

Об’єкт дослідження є процес підходу в створенні практичних кейсів з реалізації тестування кібербезпеки.

Предметом дослідження є методи і засоби забезпечення захисту системи під час мануального тестування.

По закінченню цієї дипломної роботи, ми сподіваємося внести важливий внесок в галузі мануального тестування та кібербезпеки. Ми віримо, що ця робота надасть інструменти та рекомендації для покращення практик тестування та забезпечення надійності програмного забезпечення в умовах зростаючих загроз для цифрової безпеки.

1 АНАЛІЗ ПРОБЛЕМ КІБЕРБЕЗПЕКИ В ТЕСТУВАННІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Основні поняття мануального тестування.

Мануальне тестування - це процес, який передбачає ручну перевірку програмного забезпечення з метою виявлення дефектів та валідації його функціональності та якості. Тестувальники, зазвичай з глибоким розумінням бізнес-вимог та користувацьких очікувань, виконують різноманітні тестові сценарії, взаємодіючи з програмою, спостерігаючи за реакцією та документуючи виявлені проблеми.

Роль мануального тестування у розробці програмного забезпечення

Мануальне тестування відіграє стратегічну та невід'ємну роль у процесі розробки програмного забезпечення. Воно забезпечує високу якість та надійність програмного продукту та допомагає виявляти та виправляти проблеми на ранніх етапах розробки. Розглянемо важливі аспекти ролі мануального тестування:

- Виявлення дефектів та вразливостей: Однією з ключових мет мануального тестування є виявлення дефектів та вразливостей в програмному забезпеченні. Тестувальники встановлюють відповідні тестові сценарії та виконують їх з метою виявлення помилок та недоліків. Це допомагає усувати проблеми на ранніх стадіях розробки, що значно зменшує витрати на подальше виправлення.
- Валідація відповідності вимогам: Мануальне тестування допомагає переконатися, що програма відповідає вимогам та специфікаціям, визначеним на початку проекту. Тестувальники перевіряють, чи виконуються заплановані функції, чи відповідає робота програми очікуванням користувачів.

- Валідація якості програмного забезпечення: Мануальне тестування дозволяє оцінити якість програмного продукту в цілому. Тестувальники перевіряють коректність відображення інтерфейсу, швидкість реакції, стабільність роботи та інші аспекти, які впливають на задоволення користувачів.
- Тестування в реальних умовах: Мануальне тестування дає можливість відтворити реальні умови використання програмного продукту. Тестувальники можуть виконувати тести в різних умовах та на різних пристроях, що допомагає виявити проблеми, які можуть виникнути в реальному середовищі користувача.
- Підготовка користувачів до використання продукту: Мануальне тестування може бути корисним для підготовки користувачів або співробітників до використання програмного продукту. Тестувальники можуть надавати рекомендації та підказки щодо ефективного використання системи, що сприяє легкості впровадження продукту в реальному середовищі.

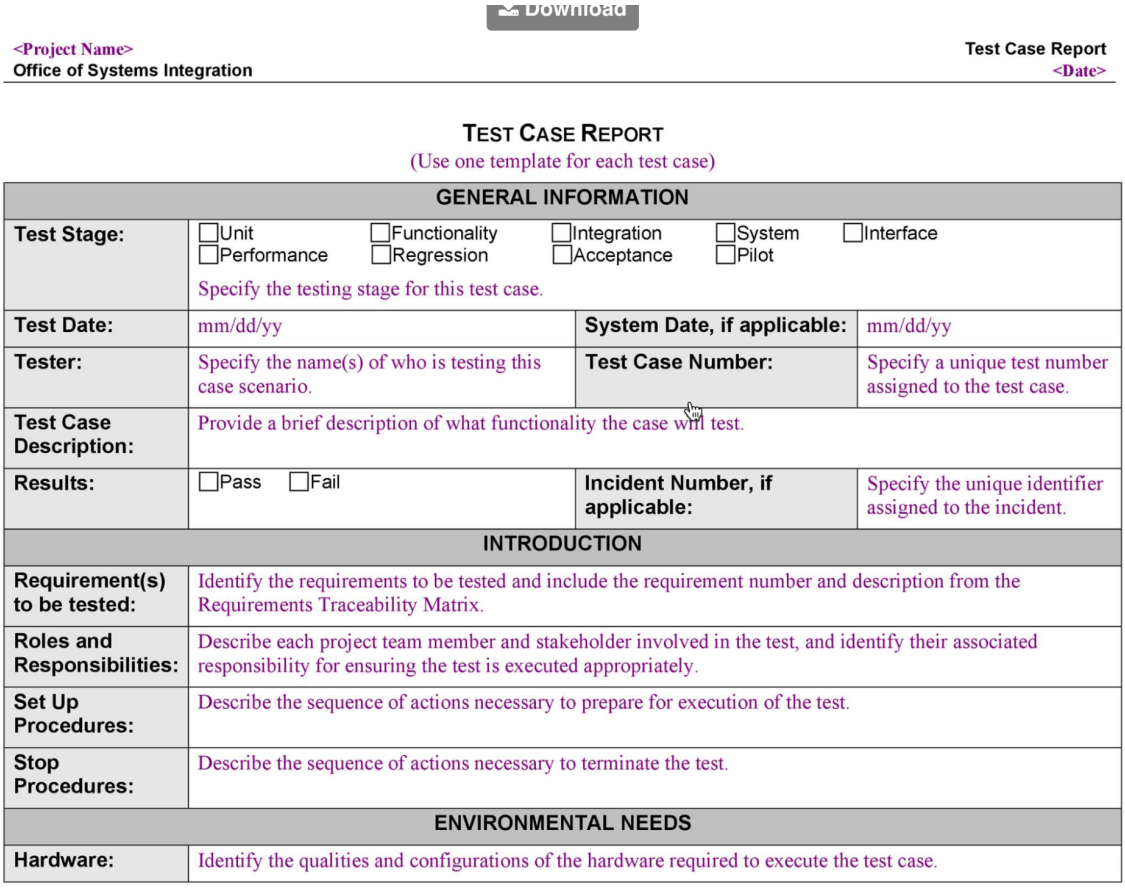
Мануальне тестування є ключовим елементом у забезпеченні якості та безпеки програмного забезпечення. Воно дозволяє ефективно виявляти та виправляти проблеми, підтверджувати відповідність вимогам та забезпечувати задоволення користувачів продуктом.

Обов'язки QA-інженера включають в себе наступні аспекти:

- Аналіз вимог[1]: QA-інженер повинен детально аналізувати вимоги до програмного продукту, які надаються замовником або бізнес-аналітиками. Він повинен розуміти, яким чином продукт має працювати і які функціональні вимоги в ньому мають бути задоволені.
- Планування тестування[1]: QA-інженер розробляє план тестування, в якому визначається стратегія, методи, терміни та ресурси, необхідні для

виконання тестів. План тестування допомагає структуровано підходити до процесу тестування.

- Написання тест-кейсів/чеклістів: QA-інженер створює тест-кейси або сценарії тестування, приклад на рисунку 1.1, які описують послідовність дій для перевірки конкретних аспектів програмного продукту. Це включає в себе визначення вхідних даних, очікуваних результатів та кроків для їх досягнення.
- Виконання тестів: QA-інженер проводить тестування, виконуючи розроблені тест-кейси. Він перевіряє, як програмний продукт реагує на вхідні дані та чи відповідає він вимогам.



Download

<Project Name> Test Case Report
Office of Systems Integration <Date>

TEST CASE REPORT
(Use one template for each test case)

GENERAL INFORMATION			
Test Stage:	☐Unit ☐Functionality ☐Integration ☐System ☐Interface ☐Performance ☐Regression ☐Acceptance ☐Pilot Specify the testing stage for this test case.		
Test Date:	mm/dd/yy	**System Date, if applicable:**	mm/dd/yy
Tester:	Specify the name(s) of who is testing this case scenario.	**Test Case Number:**	Specify a unique test number assigned to the test case.
Test Case Description:	Provide a brief description of what functionality the case will test.		
Results:	☐Pass ☐Fail	**Incident Number, if applicable:**	Specify the unique identifier assigned to the incident.
INTRODUCTION			
Requirement(s) to be tested:	Identify the requirements to be tested and include the requirement number and description from the Requirements Traceability Matrix.		
Roles and Responsibilities:	Describe each project team member and stakeholder involved in the test, and identify their associated responsibility for ensuring the test is executed appropriately.		
Set Up Procedures:	Describe the sequence of actions necessary to prepare for execution of the test.		
Stop Procedures:	Describe the sequence of actions necessary to terminate the test.		
ENVIRONMENTAL NEEDS			
Hardware:	Identify the qualities and configurations of the hardware required to execute the test case.		

Рисунок 1.1 - Приклад оформлення Тест кейсу

- Виявлення та документування дефектів: Якщо QA-інженер виявляє помилки або недоліки у програмному продукті, він документує їх докладно, включаючи опис помилки, кроки для її відтворення та іншу важливу інформацію.
- Спілкування з розробниками: QA-інженер взаємодіє з розробниками для обговорення виявлених дефектів, їх виправлення та перевірки виправлень.
- Відстеження життєвого циклу помилок[2]: QA-інженер слідкує за станом документованих помилок та відстежує їх життєвий цикл, включаючи виправлення та перевірку. Детально цикл можна побачити на рисунку 1.2.

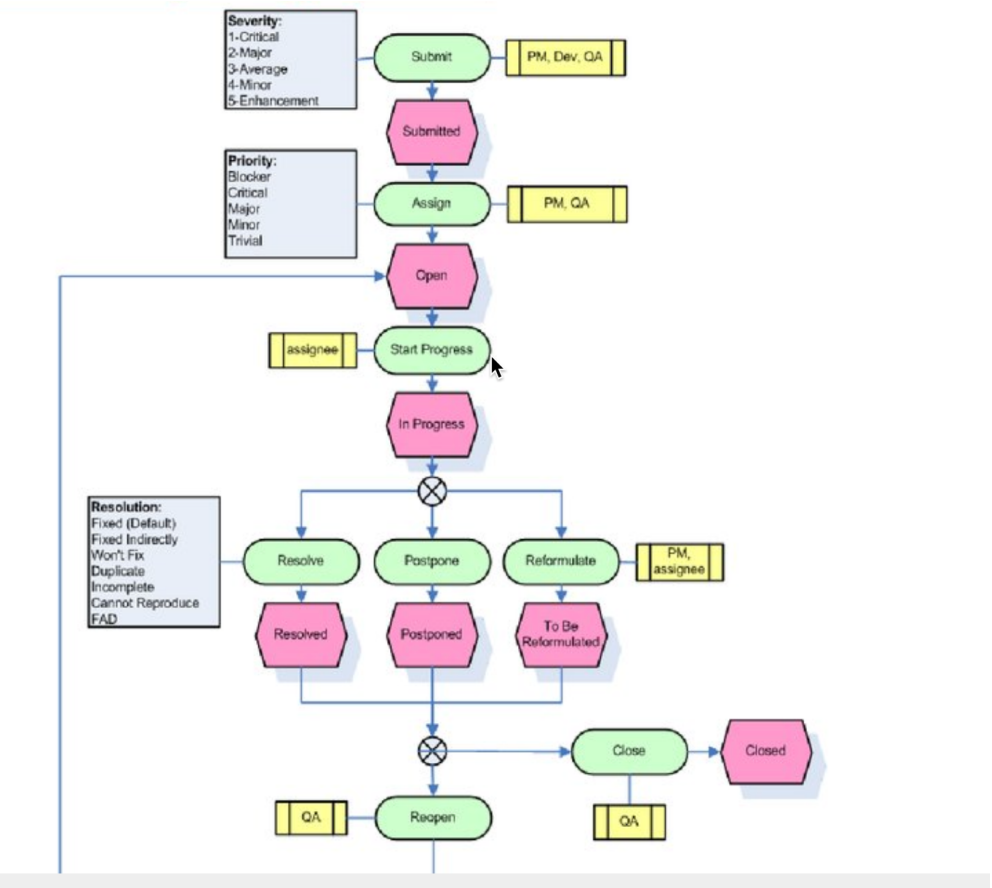


Рисунок 1.2 - Життєвий цикл помилки

- Ретестування виправлених дефектів: Після виправлення дефектів QA-інженер повторно виконує тестування, щоб переконатися, що виправлення були успішними.
- Аналіз тестування: QA-інженер аналізує результати тестування та надає звіти про якість продукту, виділяючи помилки та рекомендації для покращення.
- Оптимізація процесу тестування: QA-інженер постійно покращує та оптимізує процес тестування, шукаючи ефективніші методи та інструменти.
- Аналіз робочих процесів в команді: QA-інженер вивчає процеси роботи в команді та шукає можливості для їх поліпшення.
- Покращення процесів: На основі аналізу, QA-інженер розробляє та реалізує ініціативи для покращення якості та продуктивності роботи команди.
- Ведення тестової документації: QA-інженер дбає про актуальність та належне оновлення тестової документації, включаючи тест-кейси, звіти та інші документи, пов'язані з тестуванням.

Специфічні навички (Hard skills) для тестувальника включають[3]**:**

1. Орієнтування в операційних системах та робота з командним рядком;
2. Написання тест-кейсів та автоматизоване тестування;
3. Робота з базами даних;
4. Розуміння мови запитів SQL;
5. Взаємодія з командними програмами JIRA/Confluence;
6. Робота з системами контролю версій, такими як Git;
7. Читання професійної літератури англійською мовою.

Слід зауважити, що рівень володіння конкретними навичками може відрізнятися на різних проектах і в компаніях, а також залежати від конкретних завдань. Проте на основі аналізу понад 2 000 вакансій для

тестувальників можна сказати, що важливе знання теорії та вміння застосовувати її на практиці.

Особисті якості (Soft skills), корисні для тестувальника, включають[3-4]**:**

1. Уважність і готовність до монотонної роботи через рутину, яка часто супроводжує цю спеціальність. Для людей, які не люблять стабільність, може бути важко зберігати увагу.
2. Розвинуті аналітичні здібності та критичне мислення. Потрібно не лише знайти помилку, але й зрозуміти, що призвело до її виникнення та як її уникнути у майбутньому.
3. Впертість. Не повинно упустити жодної дрібниці від уважного тестувальника.
4. Самостійність та педантичність. Не можна допустити, щоб користувач виявив помилку.
5. Вміння чітко формулювати думки та вести діалог з колегами. Розвинені комунікативні навички спростять спілкування в команді.

1.2 Види автоматичного та мануального тестування програмного забезпечення

Існує дуже багато видів тестування, ми розглянемо деякі базові з них[5].

Функціональне тестування: Цей підхід дозволяє перевірити, чи відповідає програмний продукт специфікаціям та вимогам. Висновок: Функціональне тестування необхідне для забезпечення, що продукт відповідає функціональним вимогам і очікуванням користувачів.

Тестування відповідно до вимог безпеки: Цей підхід допомагає забезпечити безпеку програмного продукту, ідентифікуючи потенційні загрози та вразливості. Висновок: Тестування безпеки є важливою складовою для захисту даних та користувачів.

Інтерфейсне тестування: Воно допомагає забезпечити, що інтерфейс користувача є інтуїтивно зрозумілим та зручним для використання. Висновок: Інтерфейсне тестування важливе для поліпшення користувацького досвіду.

Тестування сумісності та переносимості(Cross Platform Testing): Цей підхід допомагає забезпечити, що продукт працює на різних платформах і пристроях. Висновок: Тестування сумісності важливо для досягнення максимальної аудиторії користувачів.

Тестування відновлення та стійкості(stress testing, load testing): Воно гарантує, що програмний продукт може відновити роботу після аварій або помилок, рисунок 1.3. Висновок: Тестування відновлення та стійкості сприяє покращенню надійності продукту.

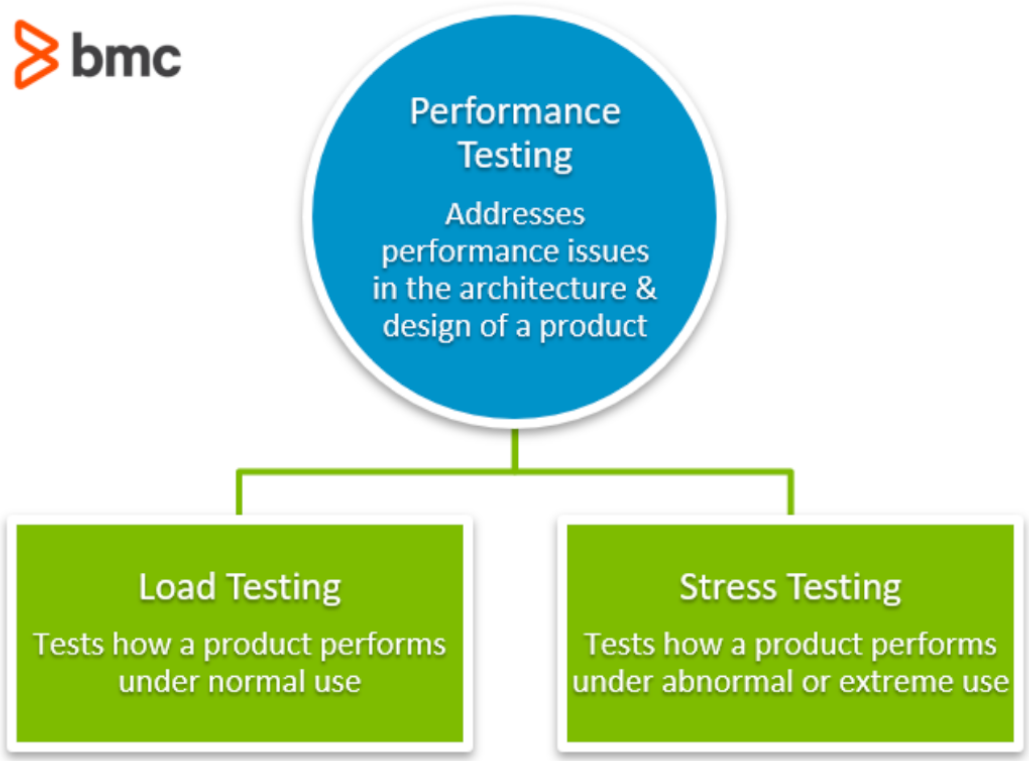


Рисунок 1.3 - Performance Testing

Тестування документації: Важливо переконатися, що документація відповідає фактичному функціоналу продукту. Висновок: Тестування документації сприяє збереженню актуальності та інформативності документів.

Тестування регресії: Воно важливо для забезпечення того, що нові зміни не руйнують існуючий функціонал. Висновок: Тестування регресії допомагає зберегти стабільність продукту. Рисунок 1.4.

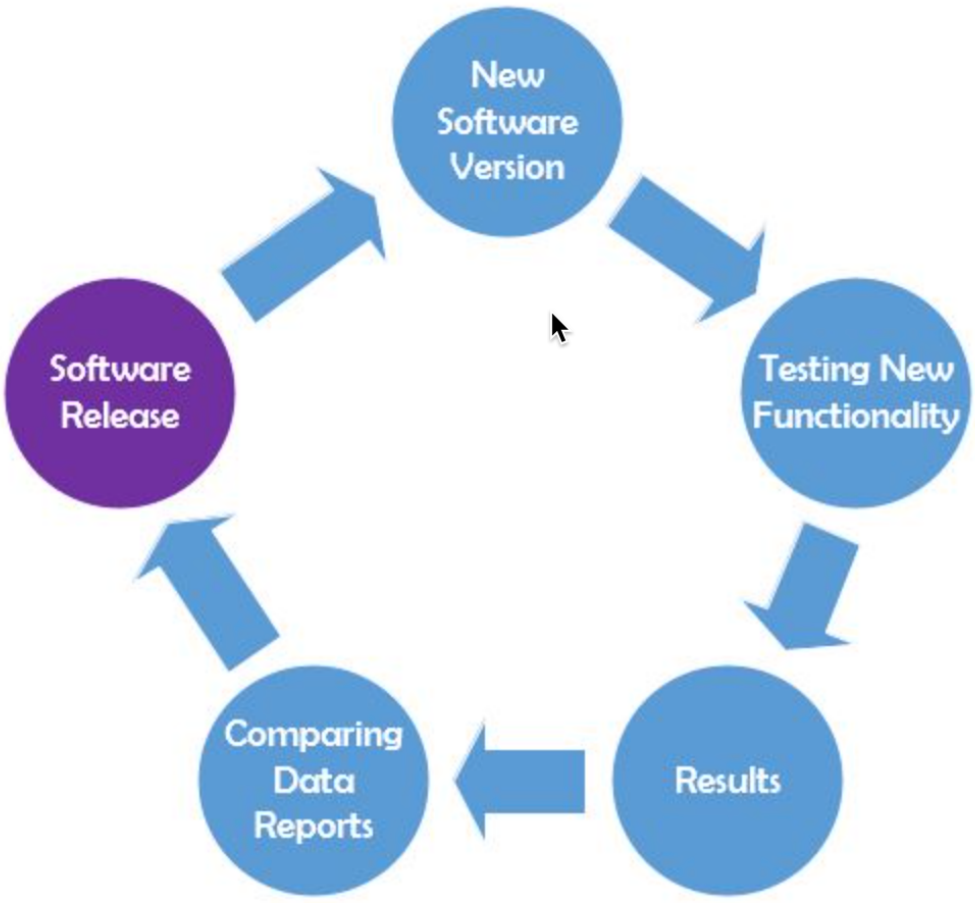


Рисунок 1.4 - Цикл тестування під час релізу продукту

Тестування в реальних умовах або тестування на продакшені: Цей підхід важливий для перевірки, як продукт веде себе в реальних ситуаціях

користування. Висновок: Тестування в реальних умовах сприяє виявленню можливих проблем в реальному використанні.

Чорний ящик(Black box testing) - це коли в тестувальника не має доступу до коду, бази даних та інших сервісів до яких має доступ розробник. Приклад: це тестування веб сайту[6]. Рисунок 1.5

Білий Ящик(White box testing) - це навпаки доступ до всіх ресурсів

Сірий ящик (Gray box testing) - це коли в нас є частковий доступ наприклад до бази даних але немає доступу до коду

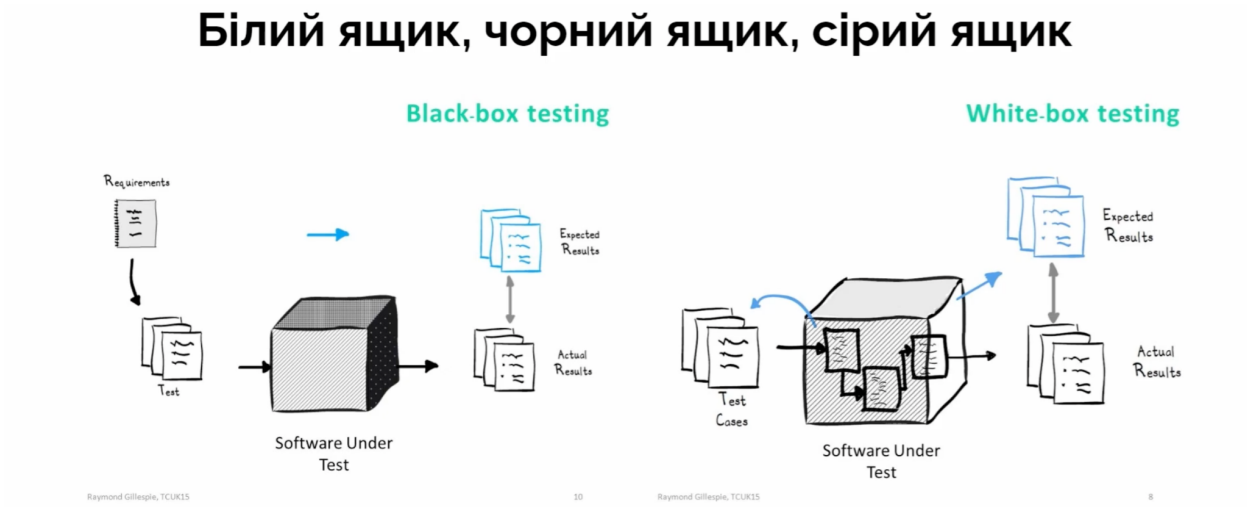


Рисунок 1.5 - Схема білого та чорного ящика

Також повну картину видів та підходів тестування можна побачити на мапі - Рисунок 1.6, або можна перейти за посиланням - https://coggle.it/diagram/We-qF3iayQABgwGL/t/testing. На цій мапі зазначені майже всі види тестування.

Рисунок 1.6 - Види тестування[7]

Висновок

Мануальне тестування включає в себе різні підходи, які допомагають забезпечити якість та надійність програмного продукту. Вибір конкретного підходу залежить від характеру програми, її вимог та цілей тестування. Ретельне мануальне тестування може сприяти своєчасному виявленню та виправленню помилок, покращенню функціональності та забезпеченню задоволення від користування продуктом.

Мануальне тестування є важливою складовою процесу розробки програмного забезпечення, яке полягає в ручному виконанні тестів з метою перевірки функціональності та якості продукту. В даному документі було надано визначення мануального тестування та розглянуто його роль у розробці програмного забезпечення.

Мануальне тестування дозволяє тестувальникам виконувати тести, перевіряти коректність роботи програми, ідентифікувати помилки та відстежувати їх життєвий цикл. Цей вид тестування вимагає людського втручання для створення тест-кейсів, виконання тестів та аналізу результатів.

Роль мануального тестування у розробці програмного забезпечення полягає в забезпеченні якості та надійності продукту. Мануальні тести дозволяють виявити помилки, які можуть залишитися непоміченими при автоматизованому тестуванні. Вони також важливі для перевірки коректності функціональності, інтерфейсу користувача та відповідності вимогам замовника.

Мануальне тестування вимагає професіоналізму, уваги до деталей та систематичного підходу до виконання тестів. В процесі розробки програмного забезпечення важливо поєднувати мануальне та автоматизоване тестування для досягнення найкращих результатів у забезпеченні якості продукту.

1.3 Проблеми кібербезпеки QA Engineer Програмного забезпечення

Зв'язок між кібербезпекою та мануальним тестуванням

Зв'язок між кібербезпекою та мануальним тестуванням визначається необхідністю перевірки безпеки програмного забезпечення вручну з метою ідентифікації потенційних загроз та вразливостей. Ось декілька напрямів за якими мануальний тестувальних може допомогти забезпечити безпеку продукту:

- Виявлення вразливостей і проблем безпеки: Мануальний тестувальник може активно шукати вразливості в програмному забезпеченні. Це включає в себе аналіз коду та тестування на проникнення, під час якого тестувальники

намагаються використати потенційні слабкі місця для отримання несанкціонованого доступу чи впливу на систему. Мануальне тестування дозволяє ідентифікувати вразливості, які можуть залишитися непоміченими при автоматизованому тестуванні.

- Аналіз безпеки даних[8]: QA-інженери, під час мануального тестування, також може перевіряти як програмне забезпечення обробляє конфіденційні дані. Вони перевіряють, чи правильно зберігаються та передаються дані, чи враховуються вимоги щодо шифрування та безпеки даних. Це важливо для захисту конфіденційності користувачів та запобігання витоку інформації.
- Тестування аутентифікації та авторизації: QA-інженери перевіряють системи аутентифікації та авторизації, переконуючись, що користувачі мають лише той рівень доступу, який відповідає їхнім правам і ролям. Вони перевіряють, чи правильно відбувається ідентифікація користувачів та чи забезпечено захист від несанкціонованого доступу.

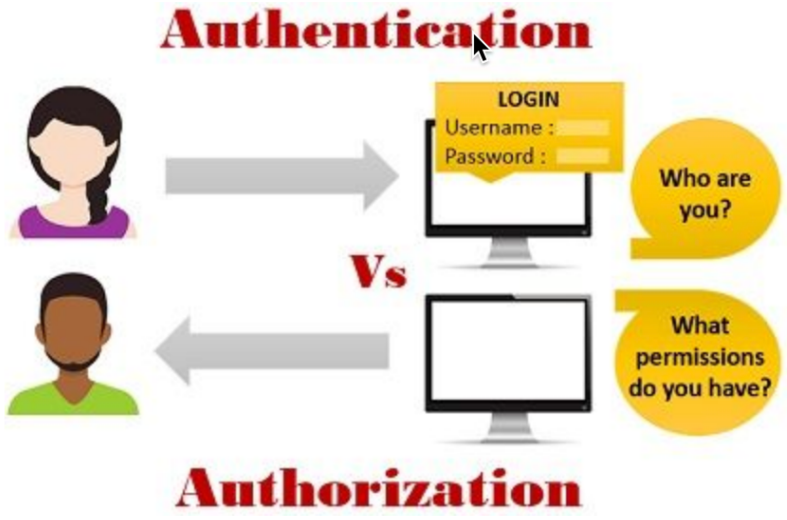


Рисунок 1.7 - Аутентифікації та авторизації

- Тестування на проникнення(Pentest): Цей аспект тестування включає в себе спроби використовувати потенційні вразливості для виявлення слабких місць у системі. Тестувальники намагаються використати можливості для атак та вимагають удосконалення заходів безпеки для запобігання проникненню.

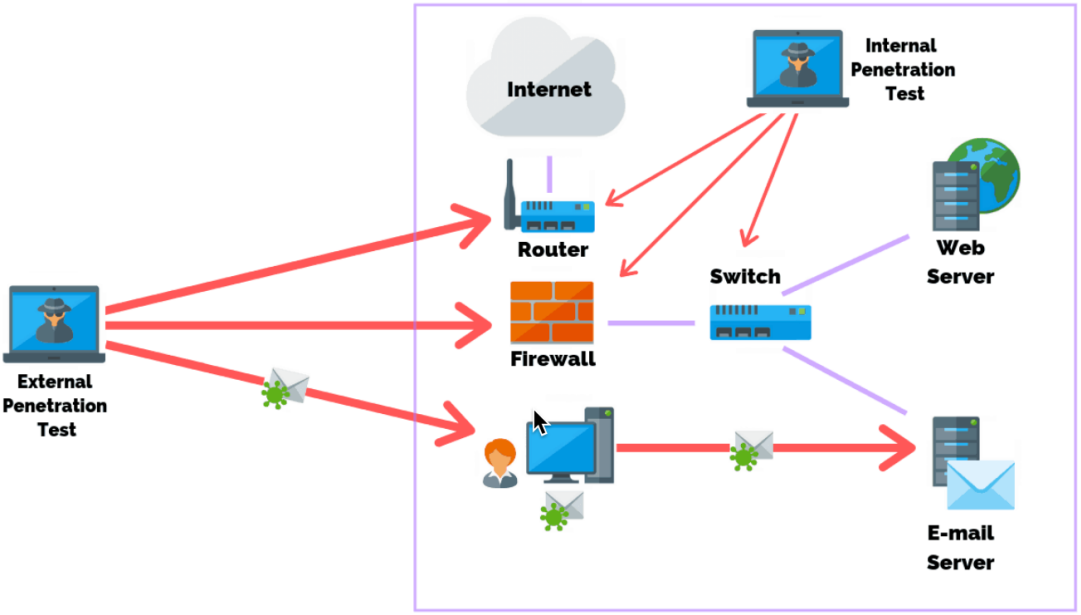


Рисунок 1.8 - Можливості атаки в середині та зовні

- Аналіз контролю доступу: QA-інженери перевіряють правильність реалізації контролю доступу до функціональності та даних. Вони переконуються, що правильно налаштовані права доступу та що користувачі не мають доступу до функцій, до яких вони не мають права.

Зв'язок між кібербезпекою та мануальним тестуванням полягає в тому, що мануальне тестування може допомогти ідентифікувати потенційні загрози та вразливості, забезпечуючи більшу безпеку програмного забезпечення.

Важливо включати тестування безпеки в процес розробки та переконатися, що програмне забезпечення відповідає вимогам щодо кібербезпеки перед його впровадженням.

Якщо говорити про проблеми QA в кібербезпеці включають в себе різні аспекти, які стикаються з тестуванням та забезпеченням безпеки програмного забезпечення. Я би виділив основні моменти чому для мануального тестувальника (QA-інженера) є проблемою перевірка системи на безпеку. Ось декілька прикладів:

Спеціалізація: QA-інженери зазвичай спеціалізуються на тестуванні функціональності та якості продукту, а не на безпеці. Їхні головні завдання полягають у виявленні багів, тестуванні вимог та забезпеченні якості виробу. Кібербезпека може вимагати спеціальної експертизи і знань, які не входять в основний спектр обов'язків QA-інженера.

Відсутність необхідних знань: QA-інженерам може бракувати необхідних знань і навичок у галузі кібербезпеки. Ця галузь швидко змінюється, і для розуміння всіх аспектів безпеки необхідно постійно навчатися та оновлювати знання.

Фокус на інших аспектах тестування: QA-інженери можуть бути спрямовані на інші аспекти тестування, такі як функціональність, сумісність, відповідність вимогам тощо. Вони можуть не мати часу або ресурсів для вивчення ідентифікації загроз та вразливостей, пов'язаних з кібербезпекою.

Обмежений обсяг завдань: В деяких проектах тестування кібербезпеки може бути покладено на інших спеціалістів, таких як експерти з кібербезпеки або інженери з безпеки. QA-інженерам може бути призначено інші завдання, які відповідають їх основній спеціалізації.

Специфіка проекту: В деяких проектах рівень кібербезпеки може не бути високим пріоритетом, особливо якщо програмне забезпечення не

містить важливих конфіденційних даних або не стоїть перед великими загрозами з боку зловмисників.

Недостатні ресурси: Не всі організації мають достатньо ресурсів для підтримки спеціаліста з кібербезпеки в команді тестування. У таких випадках QA-інженери можуть бути обмежені у можливості зосередитися на кібербезпеці.

З урахуванням цих факторів, важливо пам'ятати, що кібербезпека - це спеціалізована галузь, і вона може вимагати спеціалізованих спеціалістів для забезпечення безпеки програмного забезпечення. Однак QA-інженери можуть покращити свої знання в цій галузі, якщо це важливо для їхньої команди та проекту. Необхідність для мануального тестувальника поглиблюватись в тестування кібербезпеки стає все більш актуальною у світі, де безпека програмного забезпечення стає ключовим питанням. Сучасні загрози, пов'язані з кібербезпекою, вимагають від професіоналів з тестування більшої обізнаності та спроможності виявляти вразливості та недоліки в програмному забезпеченні. Ось декілька важливих аргументів, які підкреслюють важливість цієї необхідності:

Зростання кількості кіберзагроз: Сучасний кіберпростір насичений різноманітними загрозами, такими як хакерські атаки, витоки даних, фішинг і багато інших. Тестувальникам необхідно розуміти ці загрози та бути здатними ідентифікувати вразливості, які можуть бути використані зловмисниками для атак.

Захист конфіденційності користувачів: Безпека даних користувачів - це один із найважливіших аспектів розробки програмного забезпечення. Тестувальники повинні бути здатними перевіряти, як програмне забезпечення обробляє та зберігає конфіденційні дані, та виявляти можливі вразливості, які можуть призвести до витоку цих даних.

Забезпечення надійності продукту: Вразливості в програмному забезпеченні можуть призвести до його нестабільної роботи, аварій та інших проблем. Тестувальники повинні бути здатними ідентифікувати такі вразливості та допомагати розробникам виправити їх, забезпечуючи надійну роботу продукту.

Захист від репутаційних ризиків: Порушення безпеки може пошкодити репутацію компанії та призвести до втрати довіри користувачів. Тестувальники мають важливу роль у запобіганні таким ситуаціям та забезпеченні високої якості продукту.

Вимоги регуляторів та стандартів безпеки: Багато галузей регулюються стандартами безпеки, а також законами та нормативами, які вимагають від програмного забезпечення високого рівня безпеки. Тестувальники повинні бути ознайомлені з цими вимогами та здатні визначити відхилення від них.

Висновок

Усі ці аргументи свідчать про те, що мануальні тестувальники повинні активно розвивати свої знання та навички в галузі кібербезпеки. Це вимагає постійного навчання, оновлення та вдосконалення своєї експертизи. Такий підхід допомагає забезпечити високий рівень безпеки та надійності програмного забезпечення та відповідає вимогам сучасного кіберпростору. На завершення, важливо відзначити, що в сучасному світі програмне забезпечення може бути безпечним та функціональним лише завдяки спільним зусиллям та співпраці між різними групами фахівців, включаючи мануальних тестувальників, експертів з кібербезпеки та розробників.

2 АНАЛІЗ МЕТОДІВ ТА ІНСТРУМЕНТІВ ТЕСТУВАННЯ КІБЕРБЕЗПЕКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Види кіберзагроз та інструменти тестування кібербезпеки

Кіберзагрози — це потенційні ризики, що виникають в результаті дії зловмисників в інформаційних системах та комп'ютерних мережах. Ось деякі види кіберзагроз:

- Віруси: Програми, які можуть внедритися в комп'ютер чи систему та виконувати шкідливі дії, такі як видалення чи пошкодження даних.
- Черв'яки (Worms): Це саморозповсюджуючі програми, які можуть розповсюджувати свої копії через мережу, не потребуючи активної дії користувача.
- Троянські коні (Trojans): Програми, які приховуються під корисними, але небезпечними додатками, щоб здійснювати несанкціонований доступ чи збирати інформацію.
- Фішинг (Phishing): Атаки, які спрямовані на отримання конфіденційної інформації, такої як паролі чи кредитні картки, шляхом використання видимо надійних джерел.
- ДДоС-атаки (Distributed Denial of Service): Спроби перевантажити систему або мережу, заборонити доступ користувачам до ресурсів.
- Рейнджомери (Ransomware): Програми, які шифрують файли чи весь комп'ютер та вимагають викуп для їх розблокування.
- Атаки на слабкі місця в захисті (Exploits): Використання вразливостей в програмах чи операційних системах для незаконного врізання в систему.
- Соціальний інжиніринг (Social Engineering): Використання маніпуляції та обману людей для отримання конфіденційної інформації.

- Зловживання правил доступу (Privilege Escalation): Спроби здобуття несанкціонованого доступу до привілейованих облікових записів або ресурсів.
- Шпигунство (Spyware): Програми, які надсилають інформацію про користувача без його відома і згоди.

Це лише кілька прикладів, і кіберзагрози можуть походити з різних джерел та мати різні форми.

Зараз існує велика кількість програм та інструментів для мануального тестування кібербезпеки, і вибір правильного інструменту може бути завданням не менш важливим, ніж саме тестування. Отже, у цьому розділі ми поглибимося в світ інструментів для мануального тестування кібербезпеки та розглянемо популярні програми, які знаходяться на ринку. Наша мета - вибрати ті інструменти, які найкраще підходять для дослідження та демонстрації їхнього потенціалу.

У світі кібербезпеки засоби для виявлення вразливостей інформаційних систем постійно розвиваються та змінюються, тому важливо тримати руку на пульсі новацій і розуміти, які з них найкраще підходять для конкретних завдань. Відмінність між успішним та неуспішним тестуванням може полягати в правильному виборі інструментів.

В даному розділі ми розглянемо важливі аспекти вибору і застосування інструментів для мануального тестування кібербезпеки. Вивчивши їх характеристики та можливості, ми зможемо визначити, які з них допоможуть нам досягти наших цілей та завдань у цій області.

Для веб-застосунків[9]:

- Burp Suite: Використовується для тестування безпеки веб-застосунків.

- Selenium: Допомагає автоматизувати та виконувати тестування веб-застосунків в різних браузерах.
- Postman: Використовується для тестування та налагодження API.
- Chrome DevTools: Інструмент для відлагодження та аналізу веб-сторінок в браузері Google Chrome.
- Charles: Використовується як сніфер трафіку
- Fiddler: Використовується як сніфер трафіку
- K6 library: Використовується для тестування нагрузки
- Wireshark: Використовується Для аналізу мережевого трафіку.
- Apache JMeter - Використовується для тестування нагрузки та стрес тестування

Для мобільних додатків:

- Appium: Допомагає тестувати мобільні додатки на платформах iOS та Android.
- Calabash: Використовується для тестування мобільних додатків на різних пристроях.
- Xcode: Для тестування мобільних додатків на платформі iOS.

Для десктопних програм:

- AutoIt: Допомагає автоматизувати тестування десктопних програм на платформі Windows.
- SikuliX: Використовується для тестування програм, коли потрібно виконувати дії на основі зображень на екрані.

Для ігор:

- Unity Test Runner: Використовується для тестування ігор, створених у Unity.
- VirtualBox або VMware: Для створення віртуальних середовищ для тестування операційних систем.

Це лише деякі з прикладів програм, які можуть використовуватися мануальними тестувальниками в залежності від сфери і типу програмного забезпечення. Тестувальники можуть також розробляти власні тестові сценарії та використовувати інші інструменти для тестування, які найбільше підходять для їхніх потреб.

Burp Suite

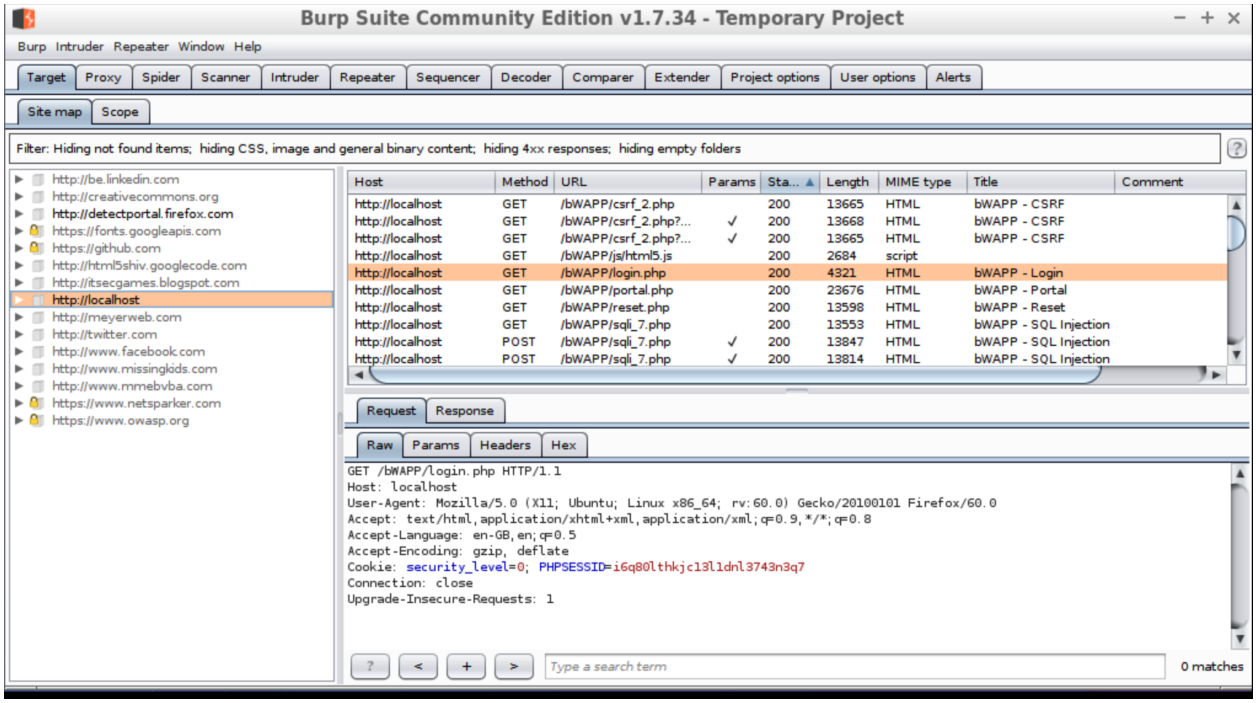


Рисунок 2.1 - Програма Burp Suite

Burp Suite - це один з найпопулярніших та потужних інструментів для тестування вразливостей веб-застосунків та виявлення багів у їх безпеці. Розроблений компанією PortSwigger, цей інструмент став важливою частиною набору програм для тестування кібербезпеки і безпеки веб-додатків. Розглянемо основні характеристики та можливості Burp Suite докладніше:

Proxy: Burp Suite дозволяє перенаправлять весь трафик між вашим браузером та веб-застосунком через свій власний проксі-сервер. Це дозволяє

тестувальникам аналізувати та змінювати HTTP-запити та відповіді перед тим, як вони досягають сервера.

Scanner: Burp Suite має вбудований сканер вразливостей, який автоматично перевіряє веб-застосунок на популярні баги та вразливості, такі як кросс-сайт скрипти, SQL-ін'єкції, переповнення буфера, ін'єкції команд та інші.

Repeater: Інструмент Repeater дає можливість повторити HTTP-запити та перевірити, як веб-застосунок реагує на різні варіації запитів. Це корисно для тестування та виявлення помилок та незахищених ресурсів.

Intruder: Intruder - це інструмент для атаки з перебором, який дозволяє тестувальникам автоматично відправляти багато варіацій запитів з різними параметрами для пошуку вразливостей.

Sequencer: Sequencer використовується для аналізу випадкових чисел, які генеруються веб-застосунком. Це може бути корисно для виявлення слабких алгоритмів генерації випадкових чисел.

Spider: Spider відстежує всі посилання та сторінки на веб-сайті та дозволяє створювати карту сайту для подальшого аналізу.

Comparer: Інструмент для порівняння двох HTTP-запитів або відповідей, щоб виявити різницю між ними.

Extender: Ви можете розширити функціональність Burp Suite, додавши власні розширення, які дозволяють виконувати специфічні завдання або аналізувати нові види вразливостей.

Target: Target допомагає ідентифікувати та відстежувати цільові веб-застосунки для тестування.

Session handling: Burp Suite дозволяє ефективно керувати сесіями, включаючи керування куками та авторизацією.

Burp Suite має безкоштовну версію (Community Edition) та платну версію (Professional Edition) з додатковими функціями та підтримкою. Цей інструмент користується популярністю серед тестувальників безпеки та пенетраційних тестерів завдяки своїм зручним можливостям та розширеній функціональності для виявлення вразливостей в веб-застосунках.

Selenium

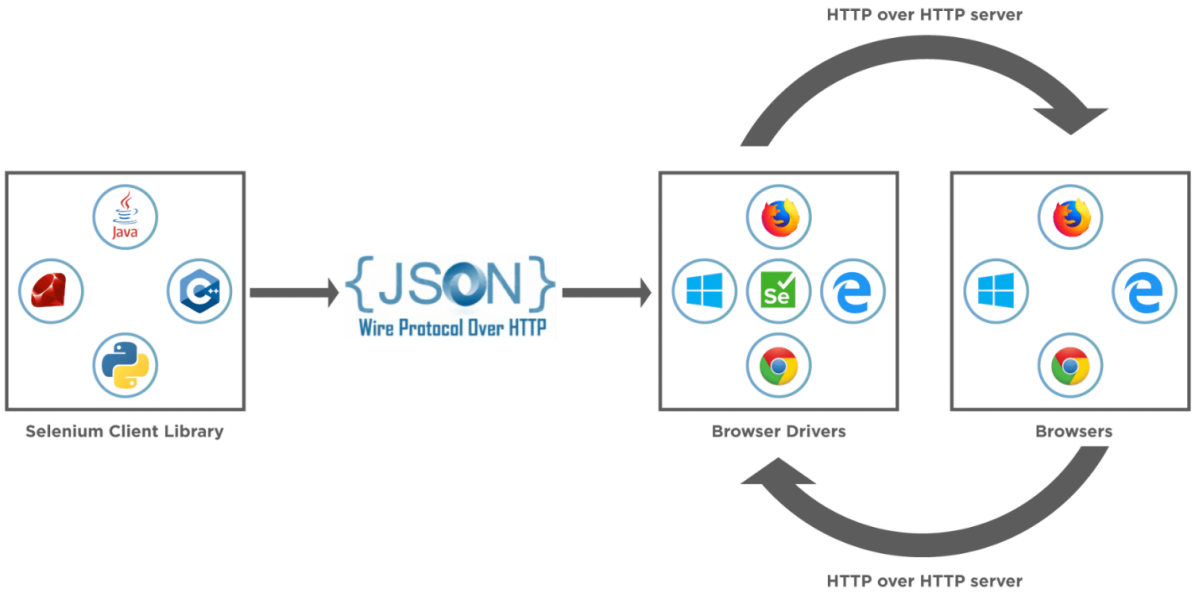


Рисунок 2.2 - Схема взаємодії бібліотеки Selenium

Selenium - це набір інструментів для автоматизованого тестування веб-додатків, який надає засоби для взаємодії з веб-сторінками та виконання різних тестових сценаріїв на веб-застосунках. Selenium дозволяє автоматизувати виконання дій, які користувачі виконують в браузері, такі як

натискання на кнопки, введення даних, навігація між сторінками та перевірка результатів. Основні характеристики та компоненти Selenium включають:

- Selenium WebDriver: Це основна частина Selenium, яка дозволяє взаємодіяти з браузерами. Selenium WebDriver підтримує багато відомих браузерів, такі як Chrome, Firefox, Edge, Safari та інші. Ви можете написати тестові скрипти на різних мовах програмування, таких як Java, Python, C#, Ruby, і багатьох інших, і взаємодіяти з веб-сторінками через API.
- Selenium IDE: Це розширення для браузерів, таких як Chrome та Firefox, яке дозволяє записувати та відтворювати тестові дії на веб-сторінці без потреби в програмуванні. Selenium IDE корисний для швидкого створення простих тестових сценаріїв.
- Selenium Grid: Цей компонент дозволяє запускати тести одночасно на різних браузерах та платформах, що робить його ідеальним для великих проектів, де необхідно тестувати на різних конфігураціях.
- Selenide: Це розширення Selenium для Java, яке спрощує та полегшує написання тестових сценаріїв та взаємодію з веб-сторінками. Воно надає вищий рівень абстракції та більш зрозуміле API для автоматизації веб-додатків.

Інші інструменти і розширення: Спільнота Selenium постійно розробляє нові інструменти та розширення для полегшення автоматизованого тестування, такі як Selenoid, Selendroid, Appium (для мобільних додатків) та інші.

Selenium користується популярністю серед тестувальників та розробників завдяки своїй універсальності та здатності працювати з різними браузерами та мовами програмування. Він широко використовується для автоматизованого тестування веб-додатків та дозволяє прискорити процес

тестування, зменшити помилки та покращити якість програмного забезпечення.

Postman

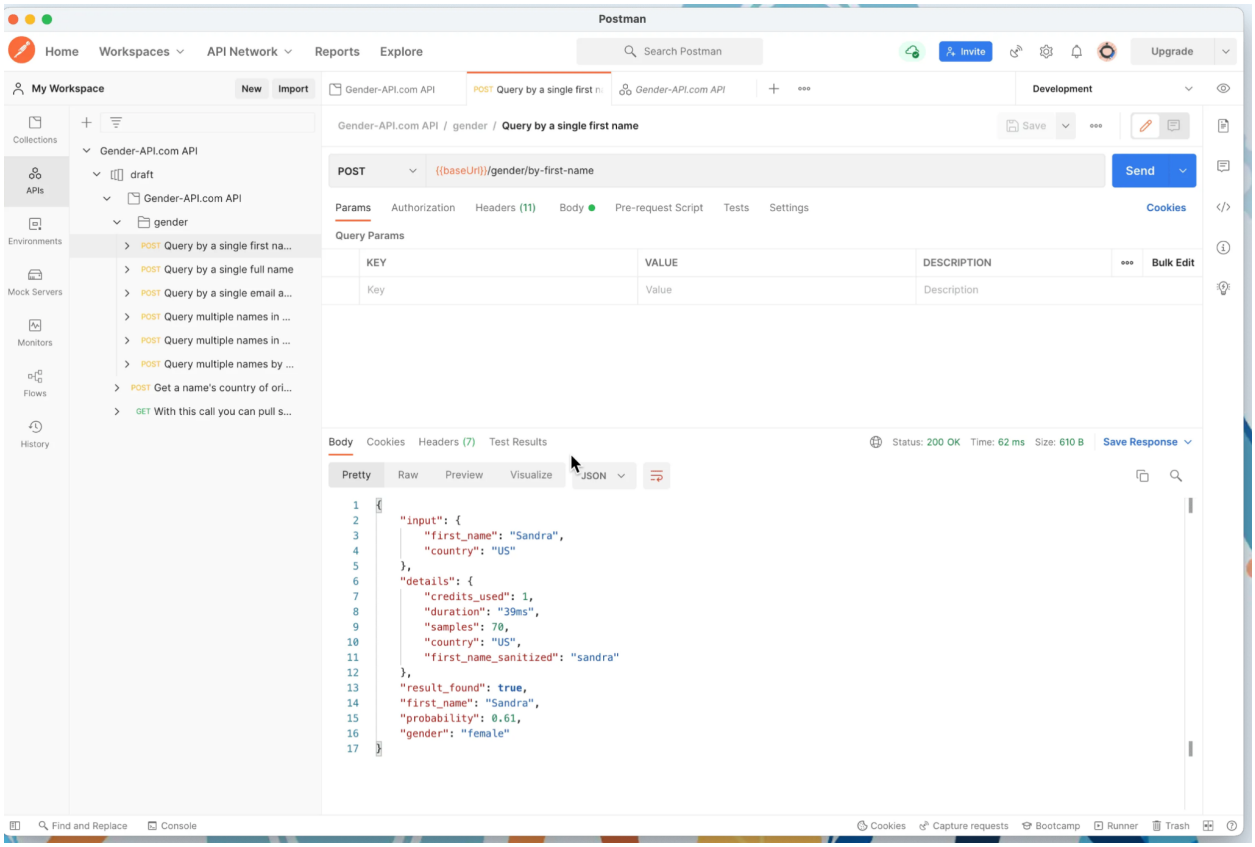


Рисунок 2.3 - Програма Postman

Postman - це популярний інструмент для тестування та налагодження API (інтерфейсів програмування застосунків). Він надає зручний інтерфейс для взаємодії з веб-сервісами та виконання HTTP-запитів для тестування функціональності API. Postman створений для тестувальників, розробників та всіх, хто працює з API. Основні характеристики та можливості Postman включають:

- Надсилання HTTP-запитів: Postman дозволяє відправляти HTTP-запити (GET, POST, PUT, DELETE і багато інших) до веб-сервісів та перевіряти їхню відповідь.
- Збереження та організація запитів: Ви можете зберігати та організовувати ваші запити в колекціях, щоб легко повторювати їх та спрощувати керування тестовими сценаріями.
- Змінні середовища: Postman дозволяє використовувати змінні для динамічної зміни параметрів запитів, що дуже корисно для тестування різних конфігурацій.
- Автоматизація тестових сценаріїв: Postman підтримує автоматизацію тестів, що дозволяє створювати тестові сценарії, використовуючи колекції запитів та сценарії зі змінними.
- Спільна робота та обмін колекціями: Postman дозволяє команді обмінюватися колекціями та спільно працювати над тестовими сценаріями.
- Зручний інтерфейс для редагування JSON і XML: Postman надає засоби для редагування та перегляду структури JSON і XML даних в запитах та відповідях.
- Налаштування аутентифікації та авторизації: Postman підтримує налаштування автентифікації для взаємодії з захищеними API.
- Тестові сценарії та засоби звітування: Ви можете додавати тестові сценарії для перевірки відповідей ваших запитів та створювати звіти про результати тестів.
- Середовища: Postman дозволяє налаштовувати середовища для різних стадій розробки (наприклад, локальна, тестова, продуктивна) та змінювати середовища зі змінними.
- Моніторинг роботи API: Postman підтримує моніторинг веб-сервісів для виявлення проблем та помилок в роботі API.

Postman допомагає автоматизувати процеси тестування та налагодження API, збільшити продуктивність тестувальників а також запобігти деяким кібер загрозам

Chrome: DevTools

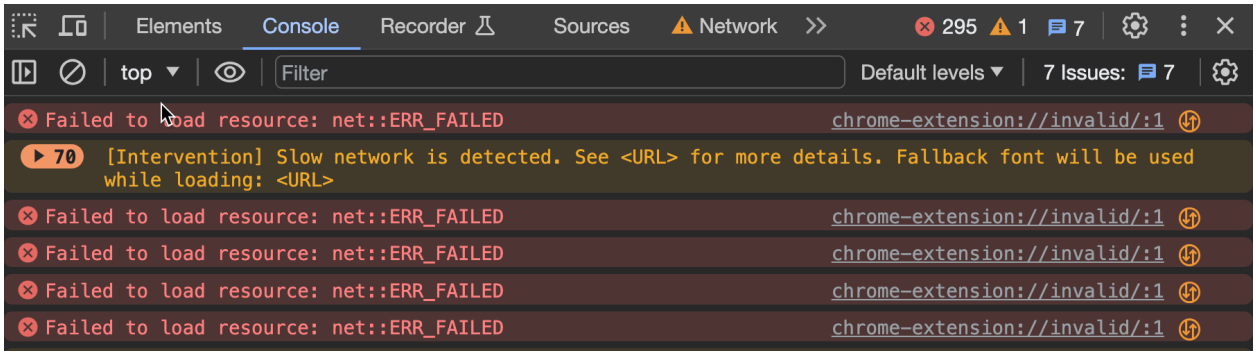


Рисунок 2.4 - Програма Chrome: DevTools

Chrome DevTools - це вбудований інструмент для розробників у браузері Google Chrome. Він надає широкий спектр можливостей для аналізу та налагодження веб-сайтів та веб-додатків. Chrome DevTools дозволяє розробникам взаємодіяти з HTML, CSS, JavaScript та іншими ресурсами в реальному часі, дебажити проблеми та оптимізувати продуктивність веб-застосунків. Основні можливості та компоненти Chrome DevTools включають:

- Elements (Елементи): Цей панель дозволяє переглядати та редагувати структуру HTML-сторінки. Ви можете інтерактивно вибирати та редагувати елементи, перевіряти CSS-стилі, змінювати атрибути та властивості, а також перевіряти бокс-модель елементів.
- Console (Консоль): У цій панелі ви можете виконувати JavaScript-код та переглядати видачу консолі для відладки. Це корисно для виявлення помилок в коді та вивчення повідомлень, які генерує ваш веб-сайт.

- Sources (Джерела): Панель Sources надає доступ до вихідного коду вашого веб-сайту, включаючи HTML, CSS та JavaScript. Ви можете дебажити код, встановлювати точки зупинки, відстежувати виконання коду та багато іншого.
- Network (Мережа): Ця панель служить для аналізу мережевого трафіку вашого веб-сайту. Ви можете переглядати запити та відповіді, вимірювати час завантаження ресурсів та виявляти проблеми зі швидкістю завантаження.
- Performance (Продуктивність): Панель Performance дозволяє вам записувати та аналізувати профілювання веб-сайту для виявлення шляхів оптимізації продуктивності.
- Application (Додаток): У цій панелі ви можете налаштовувати ресурси, які зберігаються на веб-сайті, такі як локальне сховище, покладка кукісів, маніфести та інше.
- Security (Безпека): Панель Security надає інформацію про сертифікати SSL, забезпечує аналіз безпеки вашого веб-сайту та виявлення можливих загроз.
- Audits (Аудити): Ця панель допомагає виявляти проблеми з продуктивністю, доступністю та іншими аспектами вашого веб-сайту, надаючи рекомендації щодо виправлення проблем.
- Lighthouse: Інструмент Lighthouse дозволяє вам оцінювати якість та продуктивність вашого веб-сайту за різними метриками, такими як швидкість завантаження, доступність та інші.

Chrome DevTools надає широкий спектр засобів для розробки та відладки веб-застосунків, що допомагає покращити якість та продуктивність вашого веб-сайту. Цей інструмент використовується розробниками та тестувальниками для аналізу та відладки веб-додатків та веб-сайтів.

Charles

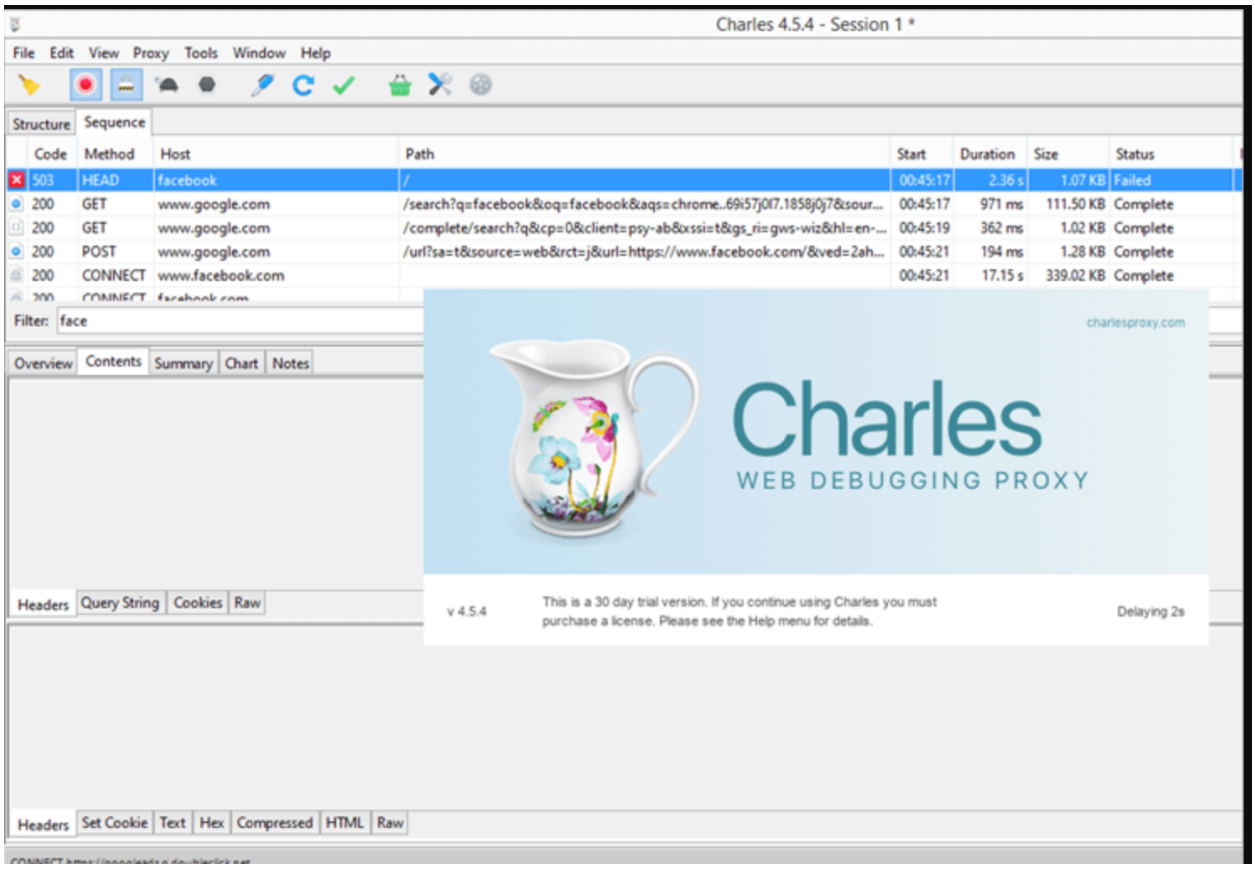


Рисунок 2.5 - Програма Charles

Charles - це інструмент для перехоплення та аналізу мережевого трафіку, який допомагає розробникам та тестувальникам налагоджувати та аналізувати взаємодію між веб-застосунками та серверами. Цей інструмент особливо корисний для роботи з мережевим трафіком мобільних додатків та веб-сайтів. Основні характеристики та можливості Charles включають:

- Перехоплення HTTP-запитів та відповідей: Charles дозволяє перехоплювати та аналізувати HTTP-запити та відповіді між клієнтами (наприклад, браузерами або мобільними додатками) та серверами. Ви можете переглядати вміст запитів та відповідей, включаючи тексти, зображення, стилі CSS, дані JSON та інші ресурси.

- SSL-декодування: Charles підтримує SSL-декодування, що дозволяє аналізувати зашифрований трафік між клієнтами та серверами. Це корисно для відлагодження проблем з SSL-сертифікатами та безпекою.
- Середовища: Ви можете створювати різні середовища для тестування, що дозволяє вам швидко переключатися між різними конфігураціями та параметрами.
- Завантаження та збереження сесій: Charles дозволяє вам зберігати та завантажувати сесії для подальшого аналізу та відлагодження.
- Маніпуляція запитами та відповідями: Ви можете змінювати запити та відповіді, додавати або видаляти параметри, модифікувати заголовки та тіло запитів, щоб вивчати різні сценарії.
- Визначення точок зупинки (breakpoints): Charles дозволяє встановлювати точки зупинки для HTTP-запитів та відповідей, що дозволяє вам налагоджувати та аналізувати трафік на конкретних етапах.
- Спостереження за швидкістю завантаження сторінок: Charles надає інформацію про час завантаження різних ресурсів на сторінці, дозволяючи вам виявляти проблеми з продуктивністю.

Charles є потужним інструментом для відлагодження та аналізу мережевого трафіку, який широко використовується розробниками, тестувальниками та адміністраторами для забезпечення якості та безпеки веб-додатків та веб-сайтів.

Fiddler

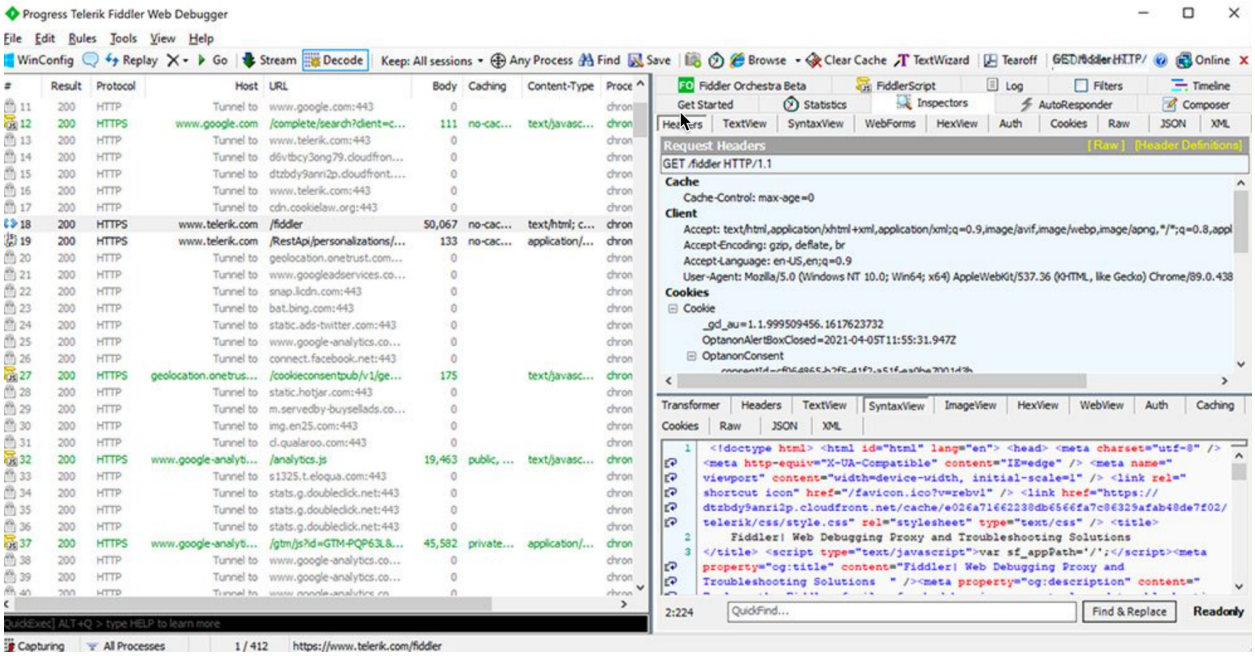


Рисунок 2.6 - Програма Fiddler

Fiddler - це інструмент для перехоплення, аналізу та налагодження HTTP-трафіку між клієнтами та серверами. Він використовується для розробки, відлагодження та тестування веб-додатків та API. Fiddler надає багато можливостей для моніторингу та маніпулювання HTTP-запитами та відповідями. Основні характеристики та можливості Fiddler включають:

- Перехоплення HTTP-трафіку: Fiddler дозволяє перехоплювати та аналізувати HTTP-запити та відповіді між клієнтами (наприклад, браузерами або мобільними додатками) та серверами. Ви можете вивчати заголовки, тіло запитів та відповідей, параметри та інші дані.
- SSL-декодування: Fiddler підтримує декодування SSL-з'єднань, що дозволяє аналізувати зашифрований трафік між клієнтами та серверами.
- Автоматизація та сценарії: Ви можете створювати скрипти для автоматизації тестів та маніпулювати запитами та відповідями за допомогою FiddlerScript, який базується на мові програмування C#.

- Завантаження та збереження сесій: Fiddler дозволяє зберігати сесії для подальшого аналізу та відлагодження.
- Моніторинг швидкості завантаження сторінок: Fiddler надає інформацію про час завантаження різних ресурсів на сторінці, дозволяючи вам виявляти проблеми з продуктивністю.
- Маніпуляція запитами та відповідями: Ви можете змінювати запити та відповіді, додавати або видаляти параметри, модифікувати заголовки та тіло запитів.
- Тестові сценарії: Fiddler підтримує створення тестових сценаріїв для відлагодження та тестування веб-додатків.
- Експорт даних: Ви можете експортувати дані про HTTP-трафік у різні формати, такі як HAR (HTTP Archive), що дозволяє поділитися даними та аналізувати їх за допомогою інших інструментів.

Fiddler як і Charles є потужним інструментом для відлагодження та аналізу HTTP-трафіку, який широко використовується розробниками та тестувальниками для забезпечення якості та безпеки веб-додатків та веб-сайтів. Він допомагає виявляти та вирішувати проблеми зі швидкістю завантаження, безпекою та функціональністю веб-додатків.

K6 library

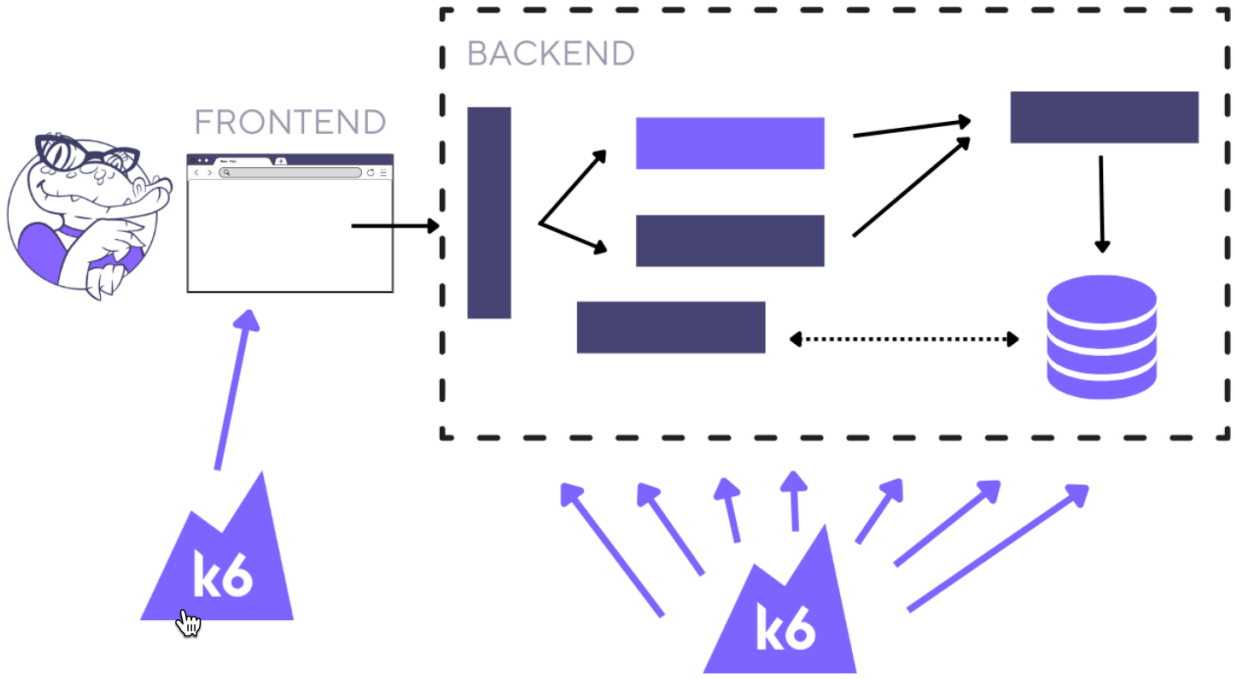


Рисунок 2.7 - Вплив бібліотеки K6

K6 - це інструмент для навантажувального тестування (тестування з навантаженням) та вимірювання продуктивності веб-застосунків. Він дозволяє розробникам та тестувальникам створювати сценарії для імітації одночасних запитів до сервера, щоб перевіряти якість та продуктивність веб-додатків під навантаженням. Основні характеристики та можливості K6 включають:

- JavaScript-сценарії: K6 використовує мову програмування JavaScript для створення тестових сценаріїв. Ви можете програмувати сценарії для відправки HTTP-запитів, обробки відповідей та вимірювання продуктивності.
- Гнучка конфігурація навантаження: Ви можете налаштовувати кількість користувачів, частоту запитів, тривалість навантаження та інші параметри для симуляції різних сценаріїв навантаження.

- Завдання та сценарії: K6 дозволяє розділяти ваші сценарії на окремі завдання, що дозволяє вам легко організовувати тестові сценарії та виконувати їх одночасно.
- Моніторинг продуктивності: K6 надає інформацію про швидкість відповідей сервера, час завантаження сторінок, кількість помилок та інші метрики для аналізу продуктивності.
- Графіки та звіти: K6 генерує графіки та звіти, які допомагають вам аналізувати результати тестів та відстежувати зміни в продуктивності з часом.
- Інтеграція з CI/CD: K6 можна інтегрувати з системами Continuous Integration/Continuous Deployment (CI/CD) для автоматизації тестування та вимірювання продуктивності під час розробки.
- Розширення та плагіни: K6 підтримує розширення та плагіни для розширення можливостей інструменту.
- Спільнота та підтримка: K6 має активну спільноту користувачів та отримує регулярні оновлення та підтримку.

K6 є потужним інструментом для тестування продуктивності та вимірювання навантаження на веб-додатки. Він дозволяє вам виявляти проблеми з продуктивністю та ефективністю веб-застосунків, забезпечуючи їхню стабільність під великим навантаженням. K6 широко використовується для тестування масштабних веб-систем та мікро сервісів.

Wireshark

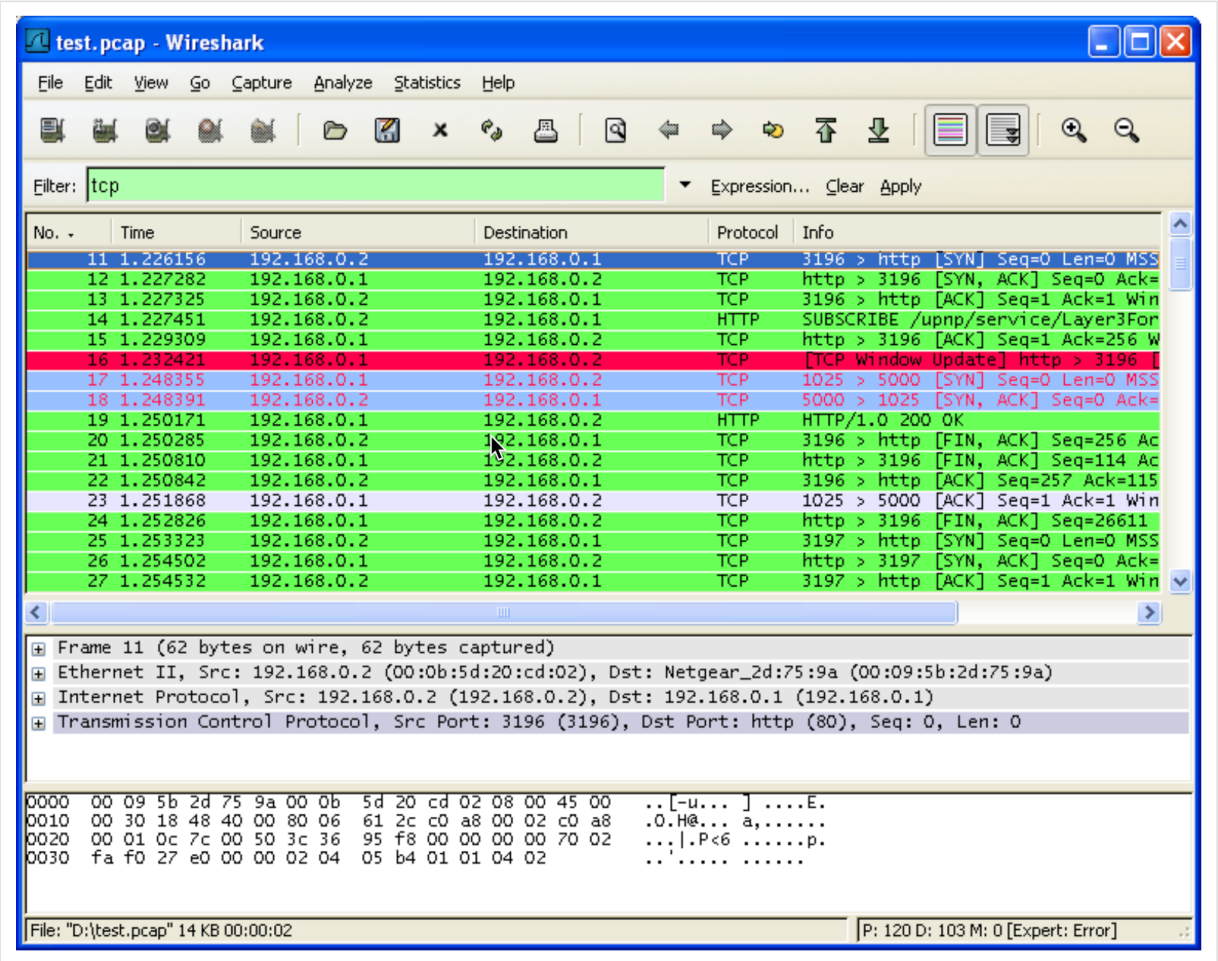


Рисунок 2.8 - Програма Wireshark

Wireshark - це безкоштовний та відкритий інструмент для аналізу мережевого трафіку. Він дозволяє захоплювати, налагоджувати та аналізувати пакети даних, які пересилаються в мережі, що допомагає розробникам, системним адміністраторам та тестувальникам виявляти проблеми та вирішувати їх. Основні характеристики та можливості Wireshark включають:

- Захоплення пакетів даних: Wireshark дозволяє захоплювати пакети даних, які пересилаються по мережі. Ви можете вибирати мережевий інтерфейс та фільтри для обмеження обсягу захопленого трафіку.

- Аналіз пакетів: Wireshark надає докладну інформацію про кожний захоплений пакет, включаючи заголовки, дані, адреси відправника та отримувача, порти, протоколи та іншу інформацію.
- Фільтрація та пошук: Ви можете застосовувати фільтри для пошуку пакетів за певними умовами, що допомагає виявляти конкретні події або проблеми.
- Візуалізація даних: Wireshark надає графічні засоби для візуалізації мережевого трафіку, включаючи діаграми, графіки та графи.
- Підтримка багатьох протоколів: Wireshark підтримує велику кількість мережевих протоколів, включаючи Ethernet, TCP, IP, HTTP, DNS, SSL, SSH та інші.
- Завантаження та збереження сесій: Ви можете зберігати сесії для подальшого аналізу та навчання.
- Розширення через плагіни: Wireshark підтримує розширення через плагіни, що дозволяє додавати додаткову функціональність та протоколи.
- Підтримка різних операційних систем: Wireshark доступний для Windows, macOS та багатьох дистрибутивів Linux.

Wireshark є незамінним інструментом для аналізу мережевого трафіку та відлагодження мережевих проблем. Він допомагає виявляти помилки, оптимізовувати мережеву інфраструктуру та забезпечувати безпеку мережі.

Appium

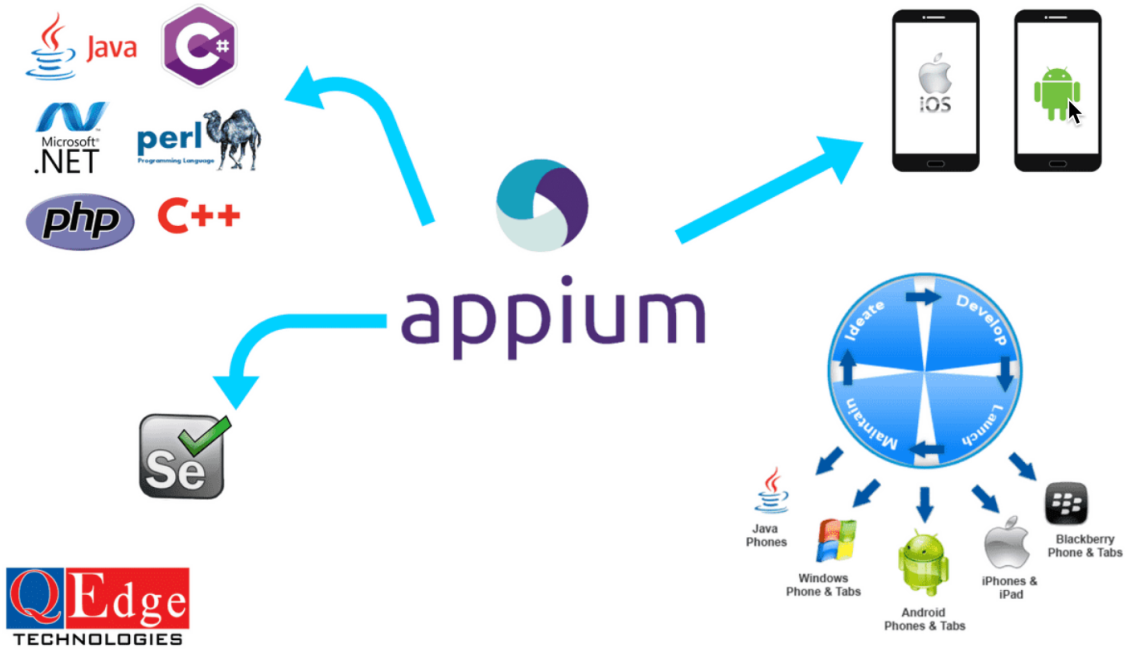


Рисунок 2.9 - Взаємодія з програмою Appium

Appium - це відкритий інструмент для автоматизації тестування мобільних додатків (iOS та Android) на різних пристроях. Він дозволяє розробникам та тестувальникам автоматизувати та виконувати функціональне, UI та навантажувальне тестування мобільних додатків. Основні характеристики та можливості Appium включають:

- Підтримка крос-платформності: Appium підтримує автоматизацію як для iOS, так і для Android, що дозволяє використовувати один набір тестових сценаріїв для обох платформ.
- Мови програмування: Ви можете використовувати популярні мови програмування, такі як Java, JavaScript, Python, Ruby, C#, для написання тестових сценаріїв.

- Інтеграція з різними тестовими фреймворками: Appium інтегрується з різними тестовими фреймворками, такими як JUnit, TestNG, XCTest, інші, що дозволяє легко виконувати тестування та аналізувати результати.
- Підтримка різних апаратних платформ: Appium дозволяє використовувати як реальні пристрої, так і емулятори або симулятори для тестування.
- Підтримка гібридних та нативних додатків: Appium підтримує як нативні, так і гібридні мобільні додатки, що робить його універсальним інструментом для різних типів додатків.
- Взаємодія з реальними пристроями: Appium дозволяє взаємодіяти з реальними мобільними пристроями, що допомагає відтворювати реальні умови тестування.
- Автоматизація жестикуляцій і вводу на сенсорних екранах: Appium підтримує автоматизацію різних жестів, таких як тапи, свайпи, мультитач та інші, що важливо для тестування сенсорних екранів.
- Інтеграція зі зборкою та CI/CD: Appium легко інтегрується з системами Continuous Integration/Continuous Deployment (CI/CD) для автоматизації тестування під час розробки.
- Спільнота та підтримка: Appium має активну спільноту користувачів та отримує регулярні оновлення та підтримку.

Appium є потужним інструментом для автоматизації тестування мобільних додатків, який допомагає виявляти та вирішувати проблеми, підвищувати якість та продуктивність мобільних додатків та забезпечувати їхню стабільність під різними умовами.

Apache JMeter

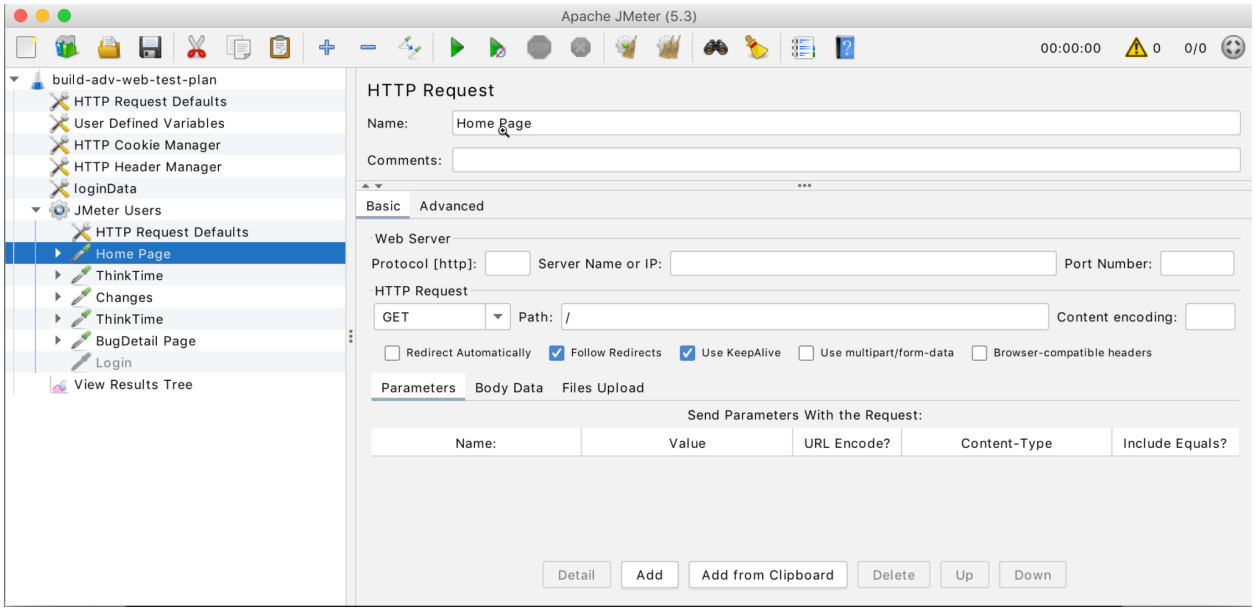


Рисунок 2.10 - Програма Apache JMeter

Apache JMeter - це безкоштовний інструмент для тестування продуктивності та навантаження веб-додатків та послуг. JMeter розроблений Apache Software Foundation і надає можливість створювати тестові плани для вимірювання продуктивності серверів, мереж та додатків під навантаженням. Основні характеристики та можливості JMeter включають:

- Захоплення та відтворення тестових сценаріїв: JMeter дозволяє захоплювати HTTP-запити та відповіді, створювати тестові сценарії, а потім відтворювати їх під навантаженням для вимірювання продуктивності.
- Підтримка різних протоколів: JMeter підтримує багато мережевих протоколів, таких як HTTP, HTTPS, FTP, SOAP, REST, JDBC, SMTP, POP3, IMAP, та інші.
- Гнучкість та розширюваність: JMeter дозволяє створювати складні тестові плани за допомогою компонентів, які можна розширювати за допомогою плагінів.

- Симуляція багатокористувацького доступу: Ви можете налаштовувати кількість користувачів, швидкість навантаження та інші параметри для симуляції багатокористувацького доступу до додатку.
- Збір та аналіз результатів: JMeter забезпечує можливість збирати та аналізувати результати тестування у форматі графіків, таблиць та звітів.
- Інтеграція з CI/CD: JMeter можна інтегрувати з системами Continuous Integration/Continuous Deployment (CI/CD) для автоматизації тестування під час розробки.
- Підтримка реальних та віртуальних пристроїв: Ви можете тестувати як реальні сервери, так і віртуальні пристрої.
- Розширена функціональність: JMeter підтримує роботу з базами даних, авторизацію, обробку файлів, генерацію корпоративного трафіку та багато інших функцій.

JMeter є дуже популярним інструментом для тестування продуктивності та навантаження веб-додатків і дозволяє виявляти проблеми з продуктивністю та швидкістю доступу до додатків під різними умовами навантаження.

Xcode

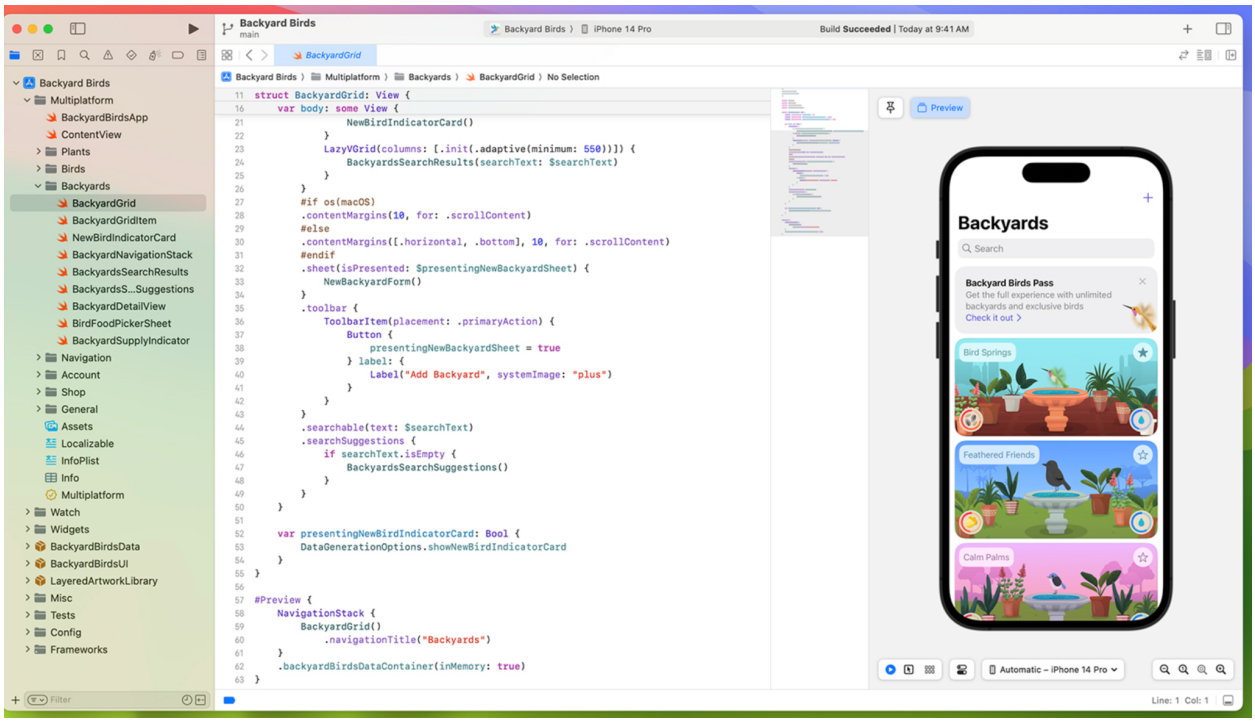


Рисунок 2.11 - Програма Xcode

Xcode - це інтегроване середовище розробки (IDE) від Apple для розробки програмного забезпечення для платформи iOS, macOS, watchOS та tvOS. Xcode є основним інструментом для розробки мобільних додатків та програмного забезпечення для пристроїв Apple, і включає в себе багато корисних інструментів та можливостей. Основні характеристики та можливості Xcode включають:

- Редактор коду: Xcode надає потужний редактор коду з підтримкою автодоповнення, перевірки синтаксису та інструментами для роботи зі сніпетами.
- Інтерфейс розробника і розмітка: Xcode має інтерфейс розробника для створення і редагування інтерфейсу користувача та можливість використовувати візуальний редактор для розмітки інтерфейсу.

- Відлагодження коду: Xcode підтримує відлагодження коду з можливістю додавання точок зупинки, перегляду значень змінних та інших інструментів для відлагодження.
- Симулятори та пристрої: Xcode має вбудовані симулятори для різних пристроїв Apple, що дозволяють тестувати додатки без реальних пристроїв. Ви також можете підключити реальні пристрої для тестування.
- Інструменти для аналізу продуктивності: Xcode включає в себе інструменти для аналізу продуктивності додатків, включаючи Profiler для виявлення проблем з продуктивністю.
- Інструменти для роботи з базами даних та API: Xcode підтримує інтеграцію з базами даних та можливість роботи з API для спрощення розробки додатків.
- Інструменти для роботи з мультимедіа: Xcode підтримує обробку мультимедійних вмісту, включаючи зображення, аудіо та відео.
- Інтеграція з Git та іншими системами контролю версій: Xcode має вбудовану підтримку Git і дозволяє легко керувати версіями вашого проекту.
- Можливості для автоматизованого тестування: Xcode підтримує автоматизоване тестування додатків з використанням XCTest.
- Розгортання додатків: Xcode надає можливість розгортати додатки на App Store або власних серверах.
- Підтримка Swift та Objective-C: Xcode підтримує мови програмування Swift і Objective-C для розробки додатків.

Xcode є незамінним інструментом для розробників, які створюють програмне забезпечення для платформ Apple, і надає широкі можливості для розробки та тестування додатків під iOS, macOS, watchOS та tvOS.

2.2 Обґрунтування вибору інструментарію для проведення досліджень

Для подальшого дослідження я вибрав ряд характеристик які мені потрібні в ході виконання практичної роботи в наступному розділі. Отже характеристики які потрібні для подальшого використання в тестуванні:

- Перехоплення HTTP-трафіку
- Додавання API запитів через curl
- Написання тестів в програмі
- Тестування навантаження
- Інтеграція CI/CD
- Виведення результатів нагрузочного тестування
- Простота налаштування
- Легкість в навчанні
- Використання в мобільному тестуванні

Як видно з рисунку 2.12, створена порівняльна таблиця за моїми характеристиками щоб обрати програму для подальшого використання

Характеристики	Burp Suite	Selenium	Postman	Chrome DevTools	Charles	Fiddler	K6 library	Wireshark	Apache JMeter	Appium	Xcode
Перехоплення HTTP-трафіку	☑	☐	☐	☑	☑	☑	☐	☑	☐	☐	☐
Додавання API реквестів через curl	☑	☐	☑	☐	☐	☐	☐	☐	☐	☐	☐
Написання тестів в програмі	☑	☑	☑	☐	☐	☐	☑	☐	☐	☑	☑
Нагрузочне тестування	☐	☐	☐	☐	☐	☐	☑	☐	☑	☐	☐
Інтеграція CI/CD	☐	☑	☑	☐	☐	☐	☑	☐	☐	☑	☑
Виведення результатів нагрузочного тестування	☐	☐	☐	☐	☐	☐	☑	☐	☐	☐	☐
Простота налаштування	☐	☐	☑	☐	☑	☐	☑	☐	☐	☐	☐
Легкість в навчанні	☐	☐	☑	☑	☑	☐	☐	☐	☐	☐	☐
Використання в мобільному тестуванні	☑	☑	☐	☑	☑	☑	☐	☐	☐	☑	☑

Рисунок 2.12 - Порівняльна таблиця

Отже для подальшого практичного застосування та можливості виявлення загроз я зупинився ось на цих інструментах:

- Charles: Charles є потужним інструментом для аналізу мережевого трафіку, і він був обраний для того, щоб показати які проблеми можуть бути

під час тестування перехвату трафіку, під час мануального тестування. Він надає можливість перевірити, як додаток взаємодіє з сервером, і виявити можливі проблеми з мережевим трафіком, такі як помилки, затримки чи проблеми з безпекою.

- Postman: Postman є ідеальним інструментом для мануального тестування API. Його було обрано, оскільки він допомагає легко створювати, відправляти та аналізувати HTTP-запити до API. Це дозволяє перевірити, чи працює API належним чином та відповідає специфікації, та забезпечує зручний інтерфейс для мануального тестування.А також знайти загрози безпеки під час маніпуляцій з API.
- K6: K6 був обраний для тестування продуктивності та навантаження, оскільки цей інструмент дозволяє вимірювати, як додаток веде себе під навантаженням. Він надає можливість створювати навантаження на сервер і робити DOS атаки для визначення продуктивності та стійкості додатку, що також будет плюсом для мануального тестування.

Обрані інструменти доповнюють один одного та надають повний набір засобів для виконання мануального тестування на різних рівнях. Вони допомагають виявляти та вирішувати проблеми як мережевого трафіку, так і API, а також вимірювати продуктивність додатків під навантаженням. Такий комбінований підхід дозволяє забезпечити якість та стабільність продукту під час мануального тестування.

2.3 Розробка методики тестування

Отже в ході нашого практичного тестування ми будемо використовувати такі методики:

- Витік конфіденційної інформації: Будемо використовувати програму Charles, яка може допомогти виявити витік конфіденційної інформації, якщо

ви бачите, що конфіденційні дані передаються без шифрування або зберігаються в не зашифрованому вигляді.

- Перехоплення інформації для аналізу: також будемо використовувати програму Charles, яка дозволяє змінювати реквест/респонс і таким чином впливати на систему.
- Тестування аутентифікації: Будемо використовувати Postman для перевірки аутентифікації JWT (або JSON Web Token). Це допоможе переконатися, що ваша система обробляє ці процеси належним чином.
- Тестування обробки помилок: Ви можете надсилати запити з неправильними даними та спостерігати, як програма або сервіс реагує на ці помилки.Використовуючи Charles або Postman.
- Навантаження тестування найбільш важливого вузла системи - використовуючи бібліотеку K6 будемо писати скрипт і пробувати навантажувати нашу систему кількістю запитів.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ МЕТОДИКИ ТЕСТУВАННЯ

3.1 Налаштування програм та середовища тестування

Налаштування програми Charles

Для операційної системи Windows і пов'язаних з нею браузерів Charles може автоматично налаштувати параметри проксі, щоб програми ОС використовували тільки Charles[10].

Для початку роботи слід завантажити Charles https://www.charlesproxy.com/ та встановити сертифікат для роботи на ПК Help > SSL Proxying > Install Charles Root Certificate. Сертифікат необхідно імпортувати в сховище «Надійні кореневі центри сертифікації» як показано на рисунках 3.1-3.2. На цьому встановлення та налаштування завершені.

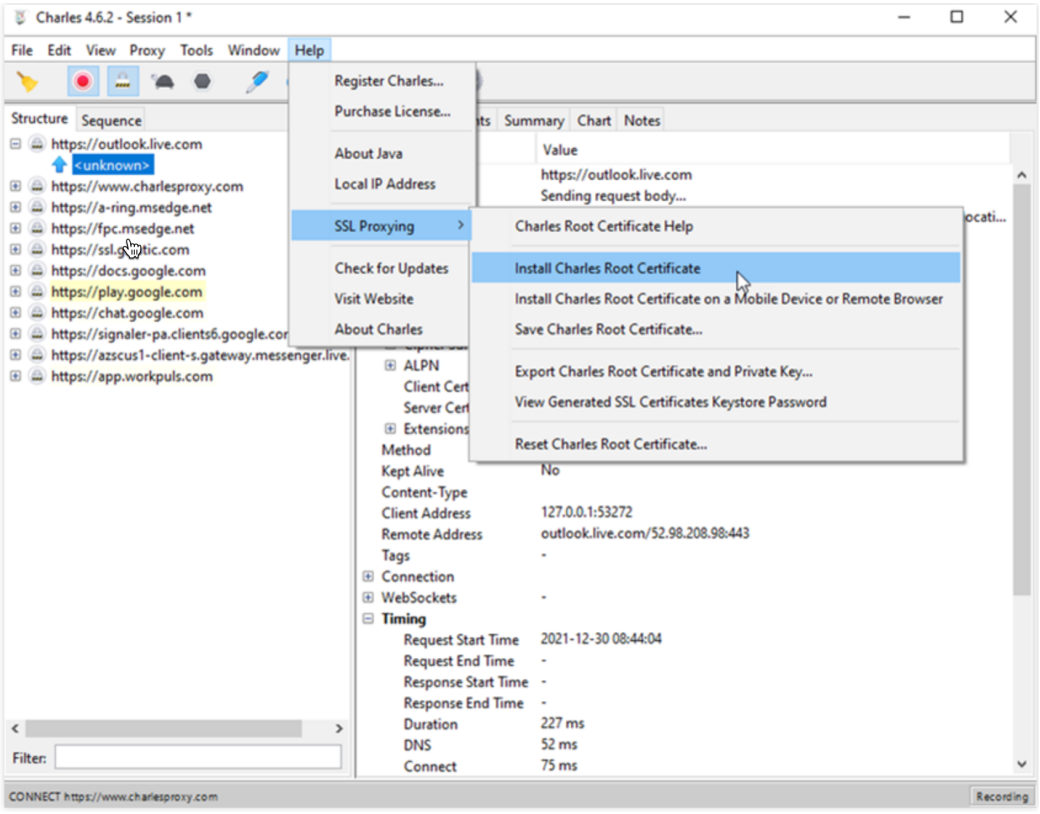


Рисунок 3.1 - Налаштування Charles на ПК

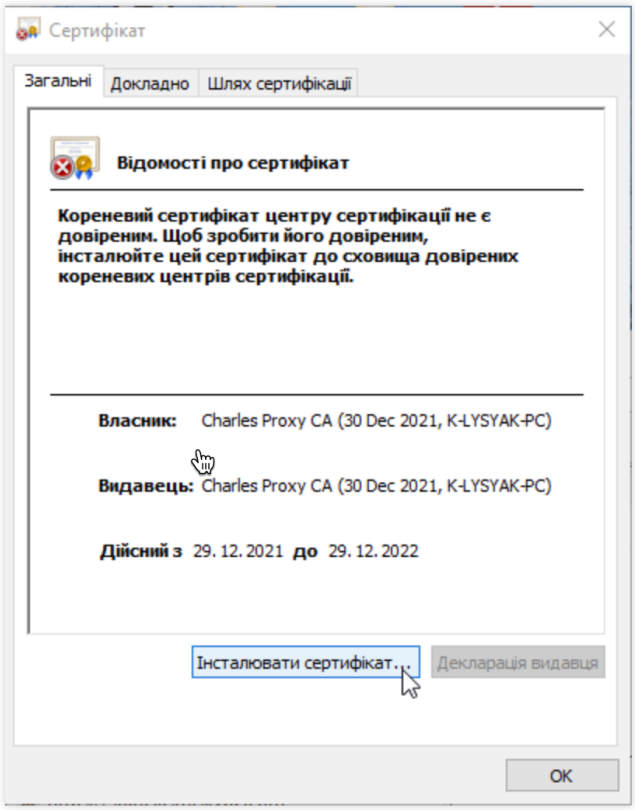


Рисунок 3.2 - Встановлення сертифікату

Для операційної системи Mac OS X та пов’язаних з нею браузерів (наприклад, Safari) під час першого встановлення Charles буде запропоновано надати дозвіл. Після цього інструмент автоматично налаштує та переналаштує параметри проксі.

Для браузера Mozilla Firefox користувачеві потрібно буде завантажити доповнення Firefox, після чого Charles зможе автоматично налаштувати параметри проксі.

Для всіх браузерів налаштування Charles можна налаштувати в меню проксі.

Установка та налаштування на девайсі

Для початку роботи необхідно налаштувати проксі на девайсі. Комп'ютер з якого працює Charles і девайс повинні бути в одній локальній мережі. IP

адресу мережі до якої підключений комп’ютер можна переглянути в Charles > Help > Local IP Address

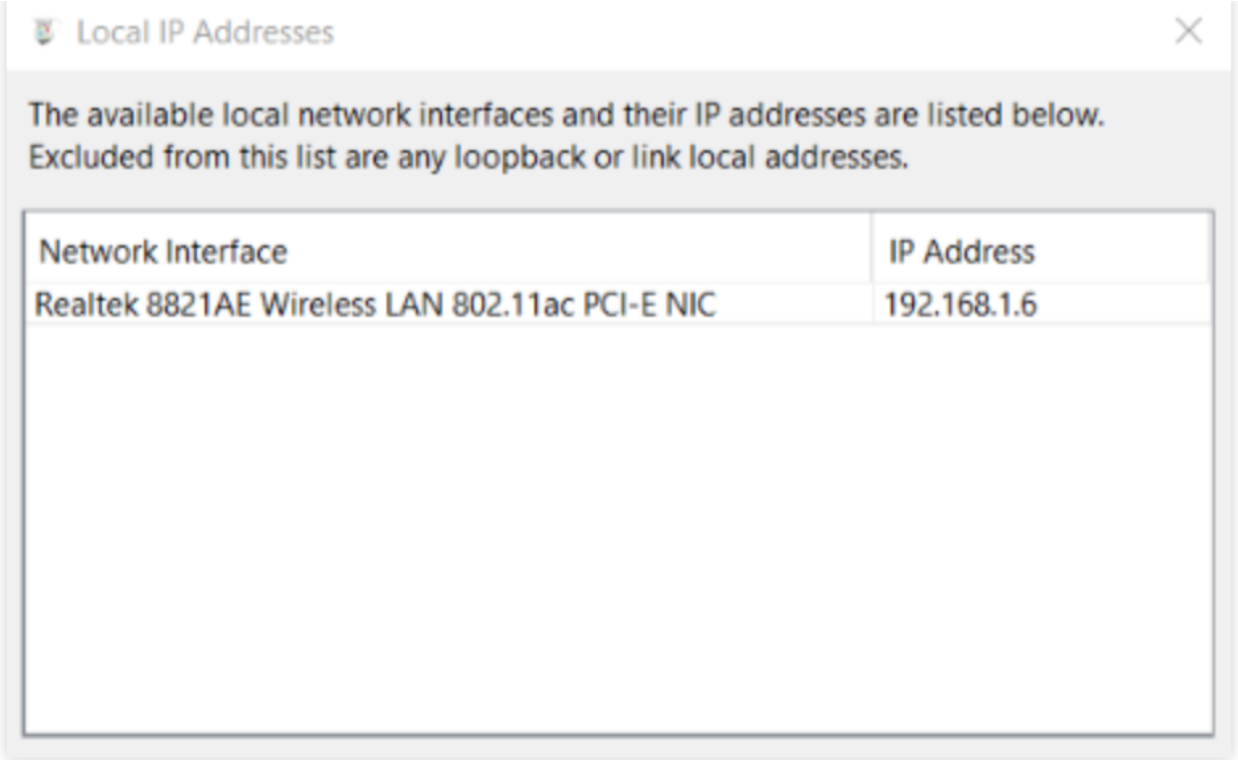


Рисунок 3.3 - Перегляд IP Addresses

В налаштуваннях девайсу необхідно обрати Wi-Fi.

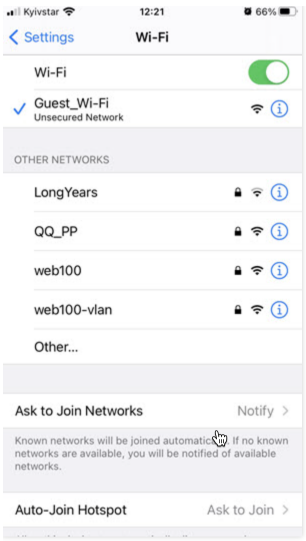


Рисунок 3.4 - Перегляд доступної мережі Wi-Fi

Обрати налаштування мережі, для того щоб ПК та девайс були в одній локальній мережі.

Натиснути на кнопку «Configure Proxy» та обрати налаштування «Manual». В поле «Server» ввести IP мережі до якої підключений ПК. Адресу порта ввести «8888» за замовчуванням як показано на рисунку 3.5.

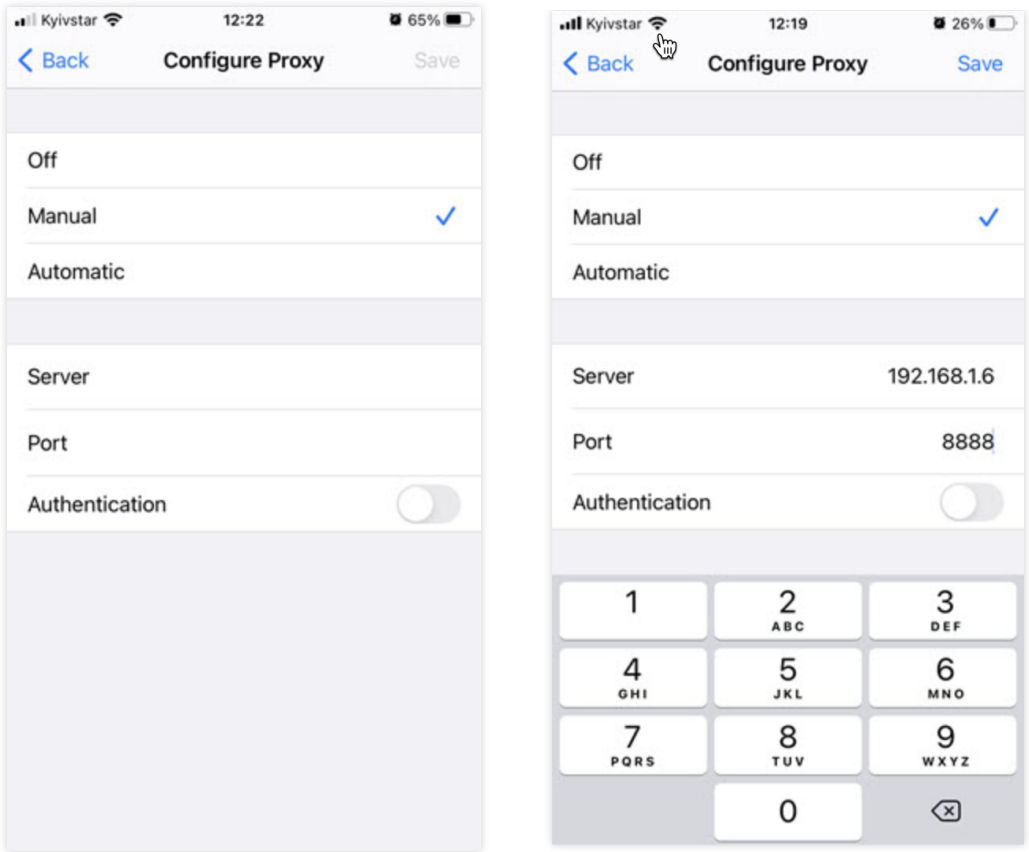


Рисунок 3.5 - Налаштування мережі та проксі

Після цього всього налаштування, якщо девайс підключений через Charles на ПК відобразиться подібний запит як показано на рисунку 3.6.

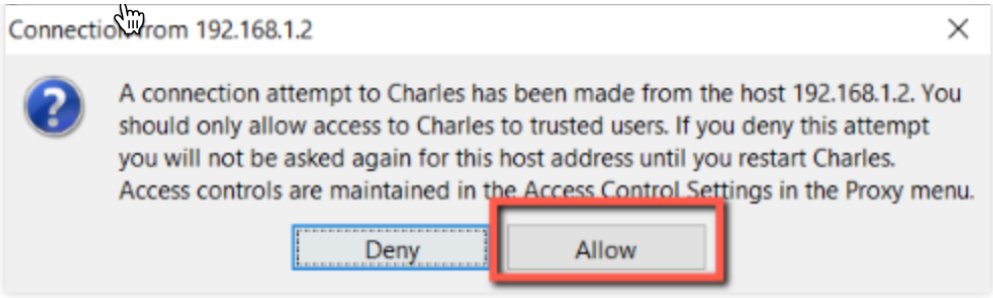


Рисунок 3.6 - Запит на підключення до ПК

Після натискання на кнопку «Allow» весь HTTP трафік буде йти через Charles.

Налаштування програми Postman

Для початку роботи з постменом його достатньо встановити з офіційного сайту на свій компьютер - https://www.postman.com/downloads/ для ПК.

Для MacOS - можна використати Homebrew та виконати запит в терміналі, рисунок 3.7.

```
brew install --cask postman
```

Рисунок 3.7 - Команда інсталяції через Homebrew

Для цілей цього проекту це більше ніж достатньо від налаштувань.

Налаштування бібліотеки K6[11]

Встановіть Node.js: K6 працює під керуванням Node.js, тому перш за все переконайтеся, що ви маєте встановлену Node.js на вашому комп'ютері. Ви можете завантажити Node.js з офіційного веб-сайту (https://nodejs.org/) та встановити його.

Створіть папку проекту: Створіть нову папку для вашого проекту, якщо її ще немає, і перейдіть до цієї папки за допомогою командного рядка.

Ініціалізуйте проект Node.js: В командному рядку введіть наступну команду, щоб ініціалізувати ваш проект Node.js:

```
npm init -y
```

Рисунок 3.8 - Команда ініціалізації проекта Node.js

Ця команда створить файл package.json, де будуть збережені дані про ваш проект.

Встановіть K6: Для встановлення K6 виконайте наступну команду:

```
brew install k6
```

Рисунок 3.9 - Команда інсталяції бібліотеки через Homebrew

Ця команда установить K6 як залежність вашого проекту та додасть її до package.json.

Далі вже можна створювати файл де буде лежати наш скрипт та сценарії. Як бачити сама установка досить проста для такої потужної бібліотеки

3.2 Практичні сценарії з реалізації тестування кібербезпеки

Сценарії 1: Аналіз Реквестів API та пошук загроз завдяки підміні респонсу

Отже уявімо собі що в нас є програмне забезпечення в якому є функціонал покупки доменних імен і можливість використовувати скидочні купони. Перш за все нам потрібно увімкнути нашу програму Charles та пройти по флоу покупки роздивляючись всі запити на пошук атаки. Отже на рисунку 3.10 ми бачимо що ми можемо купити домен ‘iwanabuydomain.tv але без використання купону.

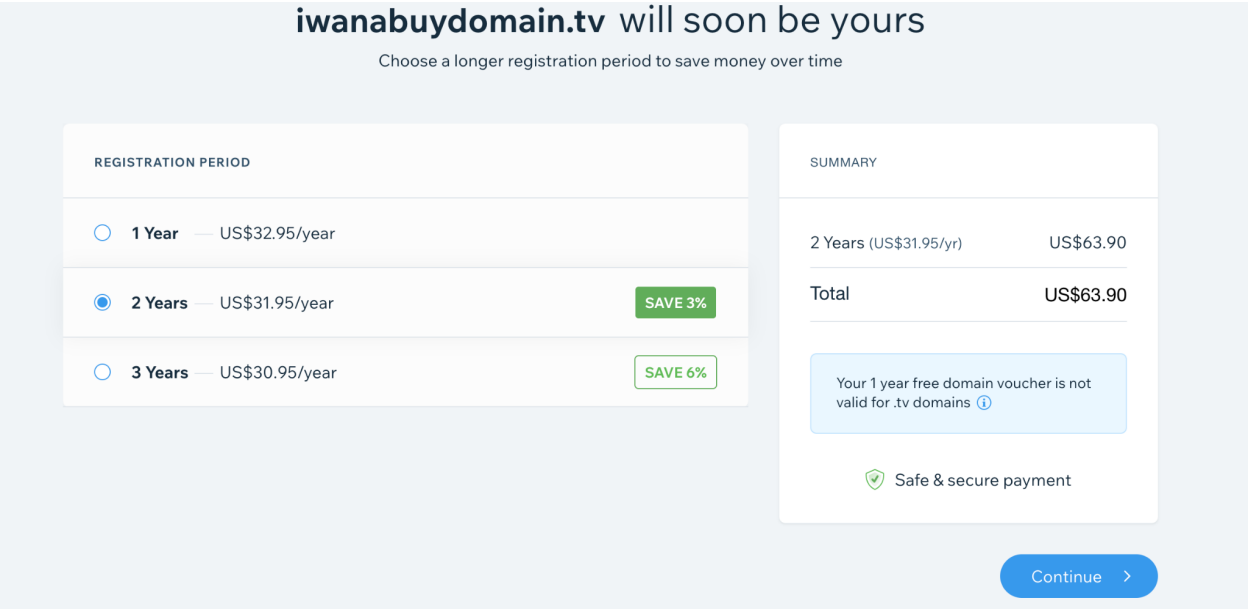


Рисунок 3.10 - Флоу покупки домену

Тепер відкриємо програму Charles і подивимося які запити були на сервер і що він відповів. Все це показано на рисунку 3.11

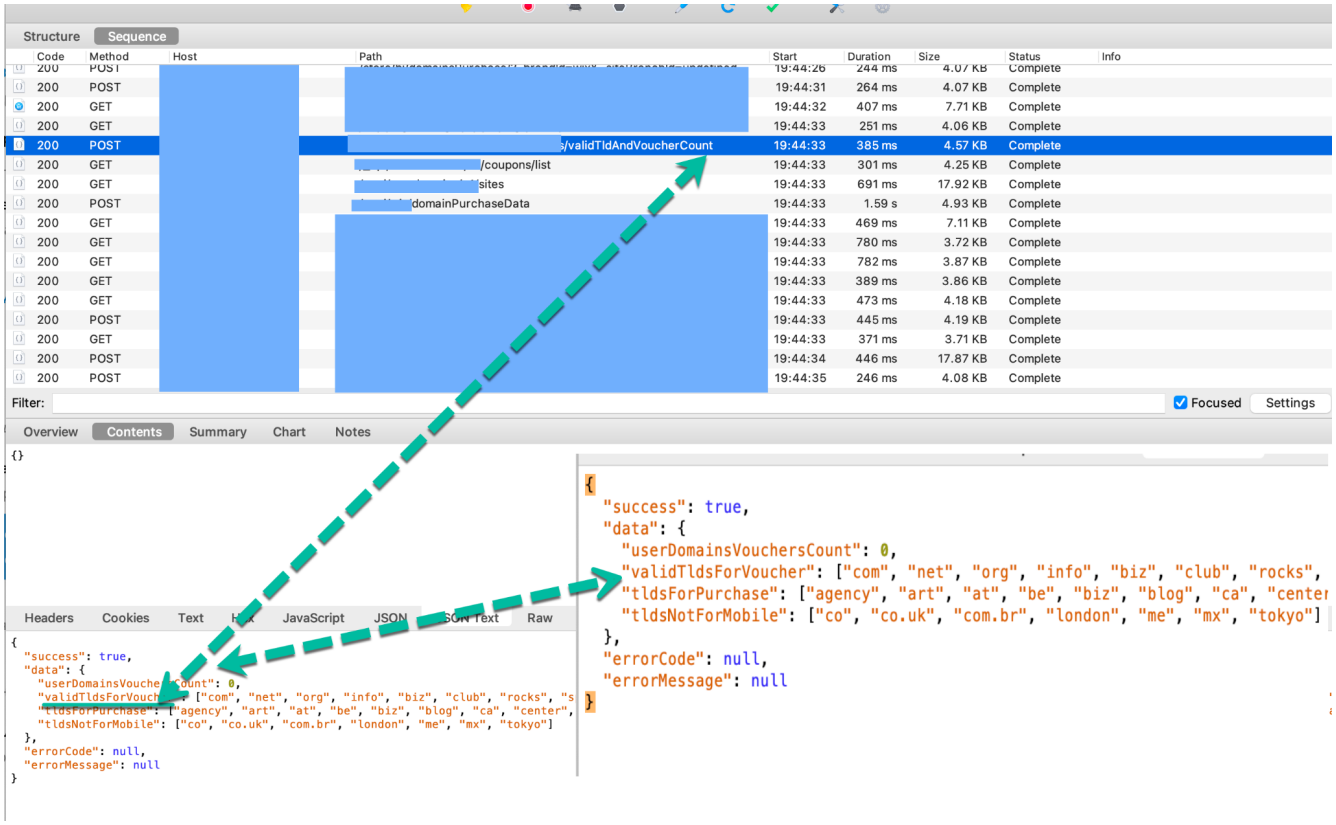


Рисунок 3.11 - Список Запитів на сервер під час покупки домену

Отже ми бачимо декілька запитів:

- coupon/list - тут ми знаходимо список купонів які є в користувача
- ValidTldAndVoucherCount - доменні обмеження
- sites - список сайтів користувача
- domainPurchaseData - дані щодо домену

Відкриваючи запит ValidTldAndVoucherCount ми в респонсі бачимо інформацію щодо доменів які можна купувати з мобільного телефону, які можна взагалі купувати та які можна купувати використовуючи наш скидочний купон. Бачимо що нашого домену з розширенням '.tv' немає в списку доменів на дозвіл використання ваучера. Тому це гарний кейс перевірити загрозу покупки будь-якого домену з ваучером. Отже натискаємо правою кнопкою на нашому запиті та натискаємо 'Breakpoints' що означає що при наступній відправці цього реквесту наша програма його перехватить і в нас буде час його підмінити та пустити далі на сервер, рисунок 3.12.

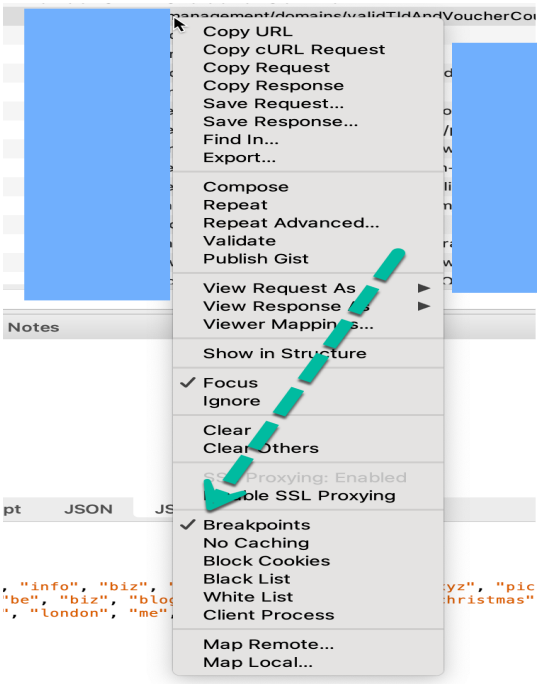


Рисунок 3.12 - Вибір запиту для перехоплення

Перезапускаємо запит на покупку домену та перехоплюємо респонс від сервера і просто вписуємо наше розширення домену в список доменів які дозволяються для покупки з ваучером та натискаємо ‘Execute’ в нижньому правому кутку, рисунок 3.13

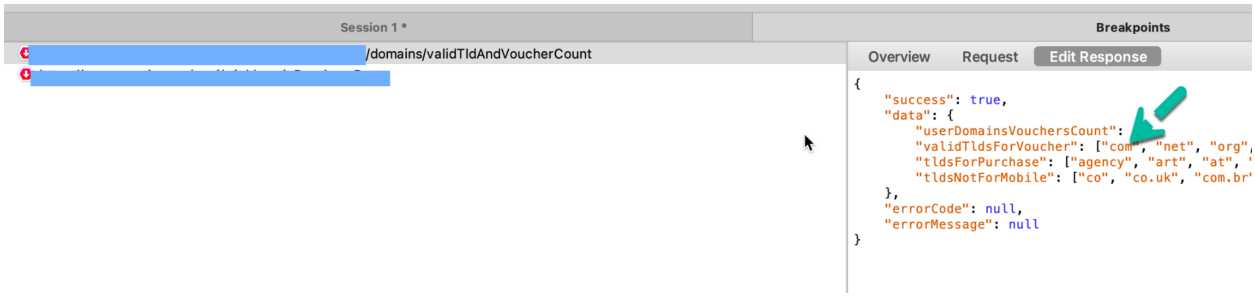


Рисунок 3.13 - Редагування тіла респонсу

Отже в результаті ми отримуємо наш домен з можливістю покупки з купоном безкоштовно як показано на рисунку 3.14.

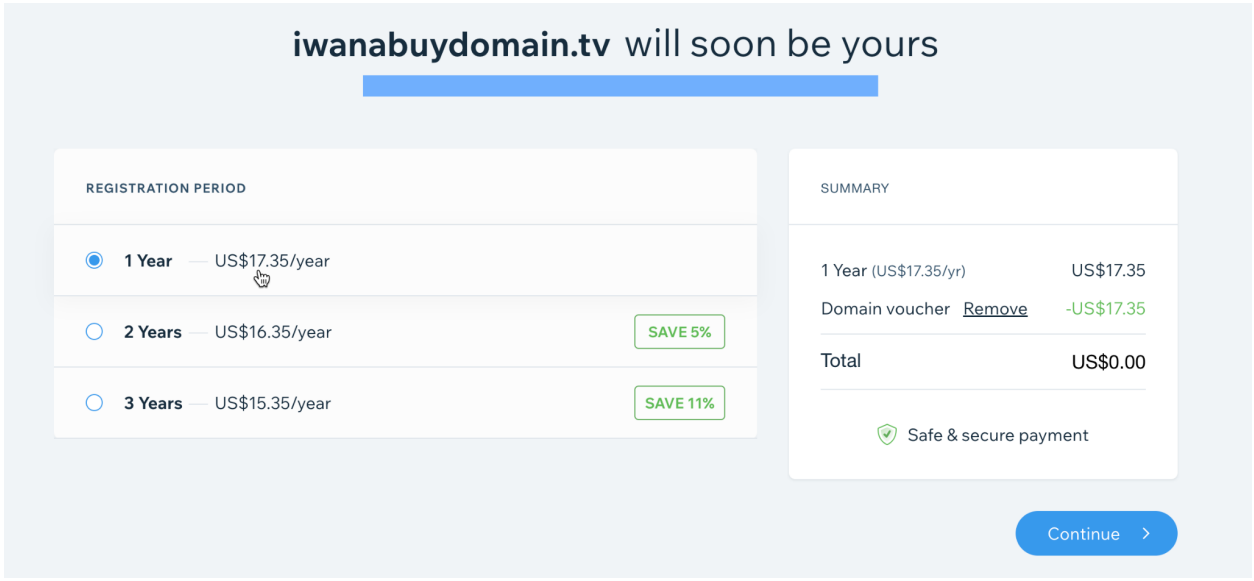


Рисунок 3.14 - Наш домен з використанням купону

Як бачимо даний спосіб дуже дієвий і може принести чимало проблем бізнесу. Результати тестування підміни відповідей від сервера переконливо підтверджують, що використання цього методу є надзвичайно

рекомендованим для тестувальників. Надійність системи у виявленні та ефективній обробці спроб маніпуляції відповідями свідчить про важливість включення цього виду тестування в розклад роботи тестувальної групи.

Методика тестування підміни відповідей виявляє потенційні вразливості та допомагає забезпечити високий рівень стійкості системи до атак. З урахуванням постійних змін у сфері кібербезпеки, використання цього методу стає важливою практикою для забезпечення безпеки програмного забезпечення.

Такий підхід до тестування варто впроваджувати в робочі процеси тестувальних команд, оскільки він не лише допомагає виявляти можливі ризики, але й сприяє постійному вдосконаленню безпеки системи в цілому.

Таким способом можна перевіряти і помилки системи - відправляючи неправильні дані і дивлячись як система реагує на них.

Сценарій 2: Перевірки аутентифікації JWT token

Ще один з дуже важливих аспектів в роботі тестувальника це перевірка аутентифікації[12]. Для цього нам знадобиться програма Postman так як там дуже зручно створювати нові запити та зберегти їх в колекції. Перш ніж ми почнемо, давайте спочатку розберемося що таке JWT та як він працює.

1.Спочатку користувач входить на сервер аутентифікації за допомогою аутентифікаційного ключа (це може бути пара логін/пароль, або ключ Facebook, або ключ Google, або ключ від іншого облікового запису).

2.Потім сервер аутентифікації створює JWT і надсилає його користувачеві.

3.Коли користувач робить запит до API додатка, він додає до нього отриманий раніше JWT.

4.Коли користувач робить API запит, додаток може перевірити за переданим разом з запитом JWT, чи є користувач тим, за кого себе видає. Схема роботи показано на рисунку 3.15

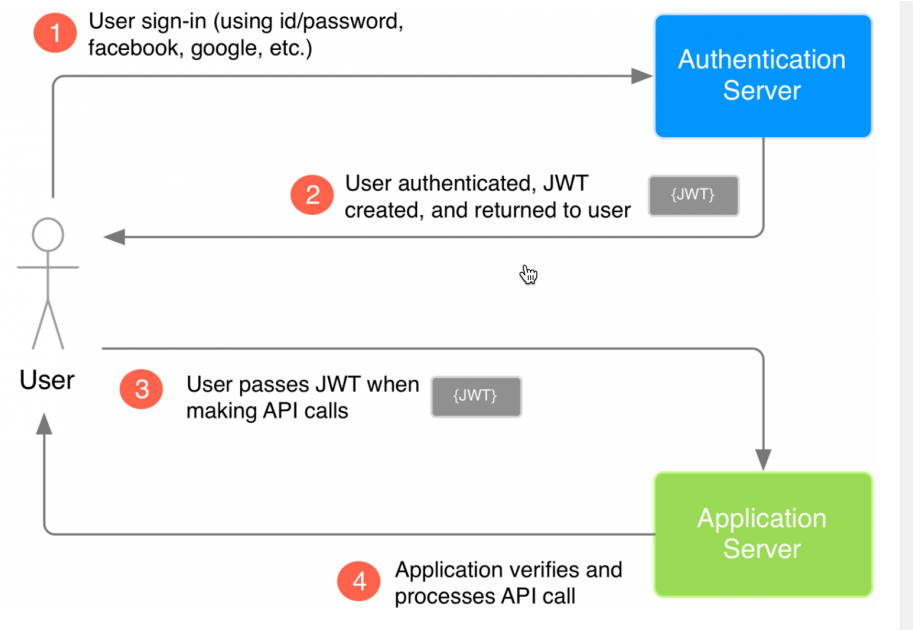


Рисунок 3.15 - Принцип роботи JWT

Отже дуже важливо перевірити такі сценарії:

- коли JWT заекспайрився(тобто час дії закінчився)
- коли JWT немає в запиті
- не валідний JWT

Уявімо що ми тестуємо інтеграцію платіжного провайдера з нашим сервісом і нам потрібно перевірити чи валідує ці сценарії провайдер.

Для початку нам потрібно генерувати собі JWT коли його час дії вже вийшов. Отже для генерації нам потрібно спочатку за допомогою алгоритму SHA-256 згенерувати собі хеш і вставити тіло яке ми будемо відправляти провайдеру як це показано на рисунку 3.16

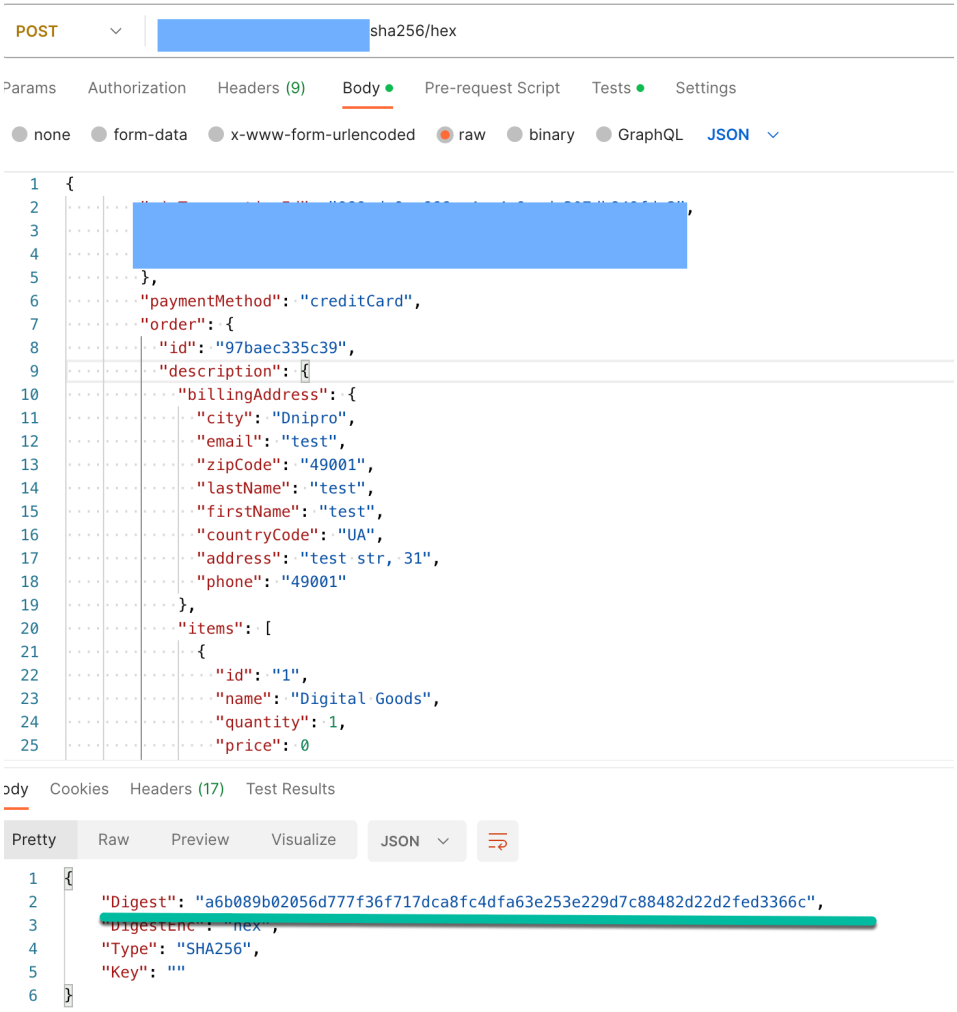


Рисунок 3.16 - Генерація хеш суми з тіла запиту

Далі нам потрібно маючи вже хеш суму згенерувати наш токен JWT[13]. З рисунку 3.17 ми бачимо що в поле hashedPayload ми підставляємо наш Хеш з попереднього запиту, а в поле expiration_in_mills ставимо значення 0, що означає що наш токен вже заекспайрився. Результатом ми отримуємо наш токен який тепер можно вставити в запит до провайдера

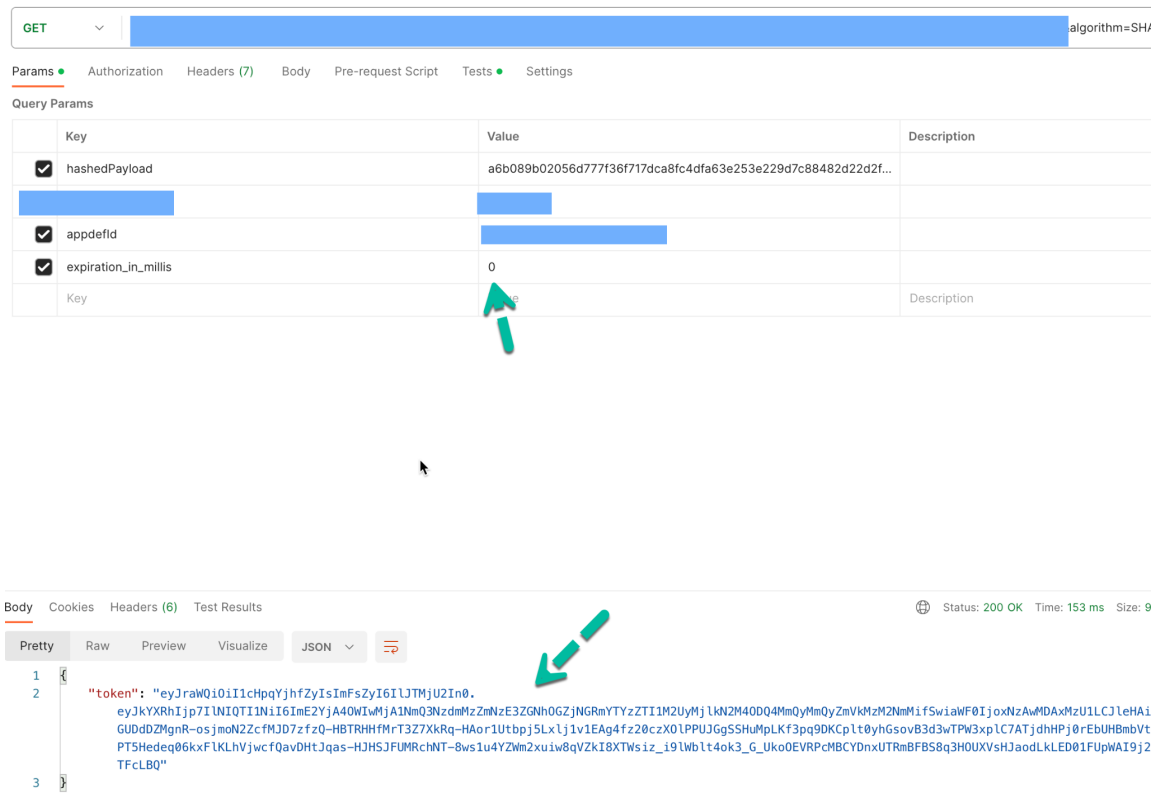


Рисунок 3.17 - Генерація JWT

Додаємо в секцію Headers наш токен та відправляємо наш запит на провайдер. Якщо провайдер опрацює та відповість нам створеною транзакцією це будет загрозою для нас так як хакери можуть скористатися тим що провайдер не валідує підпис і наприклад використовувати цей запит для підбору CVV до карток користувачів що принесе бізнесу і компанії в цілому чи малих проблем. Детально запит показано на рисунку 3.18. Ми бачимо що провайдер опрацював наш токен і перевірив що від заекспайрився, тому відповів нам помилкою що є добре.

Аналогічно перевіряємо випадки з не валідним(передаємо тіло не те що очікує провайдер) JWT або коли його не має у запиті (просто не передаємо).

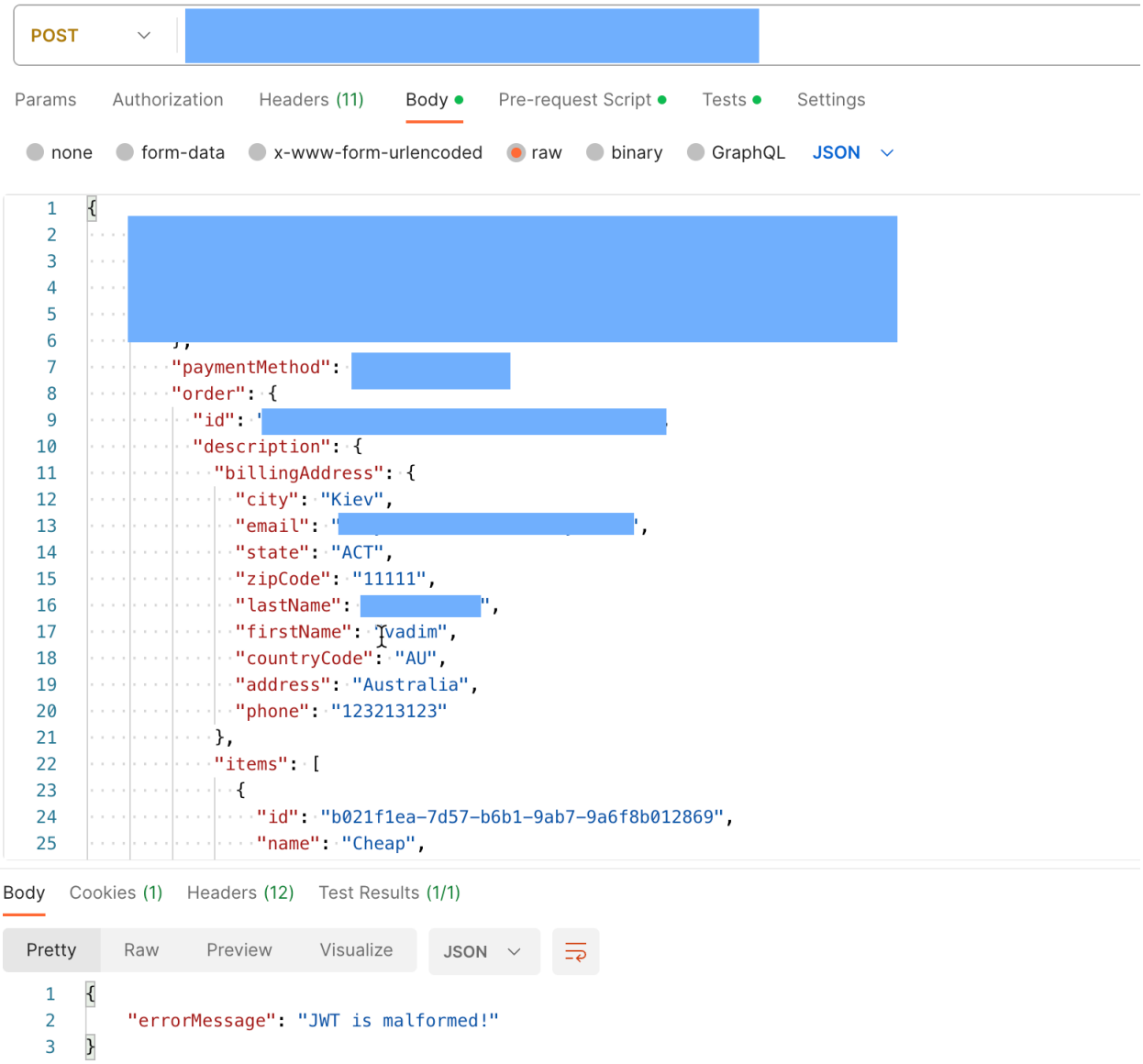


Рисунок 3.18 - Запит з JWT токеном час якого вийшов

Саме ці перевірки забезпечують надійність роботи інтеграції та нашої системи в цілому тому дуже важливо акцентувати увагу на перевірках з авторизацією. Можна перевіряти багато різних сценаріїв які може не покривати ваша система. Саме тому тут потрібна особлива увага і кмітливість під час створення тестових сценаріїв.

Сценарій 3: Тестування навантаження (Load Testing)

Перш за все тестувальнику потрібно добре розуміти продукт та систему в цілому. Для тестування навантаження потрібно знайти в системі найбільш трудно затратну операцію і навантажити саме цей ендпоінт для перевірки. Також можна грузити окремі ендпоінти щоб бути впевненими що система буде працювати під навантаженням добре[14].

Отже як приклад візьмемо систему платежів з провайдером. В цій системі найбільш затратне флоу це оплата. Давайте подивимося що відбувається коли покупець натискає кнопку купити товар. Рисунок 3.19.

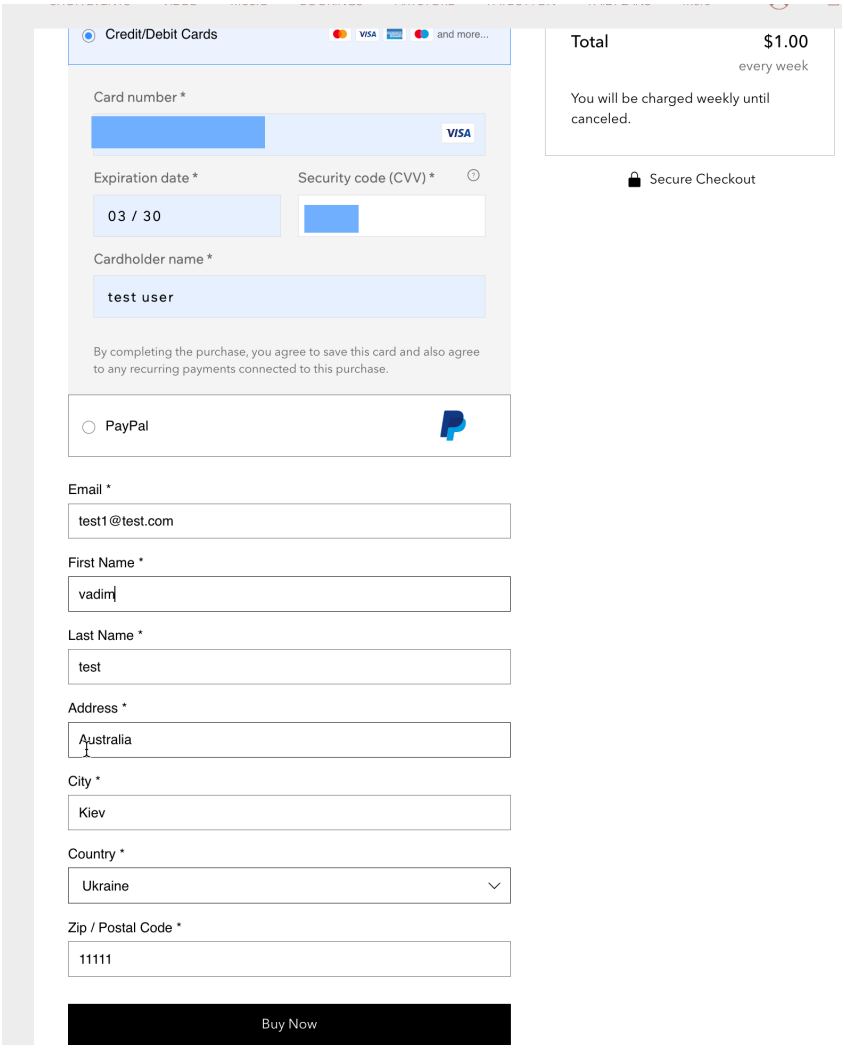


Рисунок 3.19 - Покупець заповнив форму оплати

Отже покупець заповни всі дані які треба для оплати і далі натискає кнопку оплатити товар. В цей час ми відкриваємо Chrome-Network та дивимося які запити йдуть, рисунок 3.20.

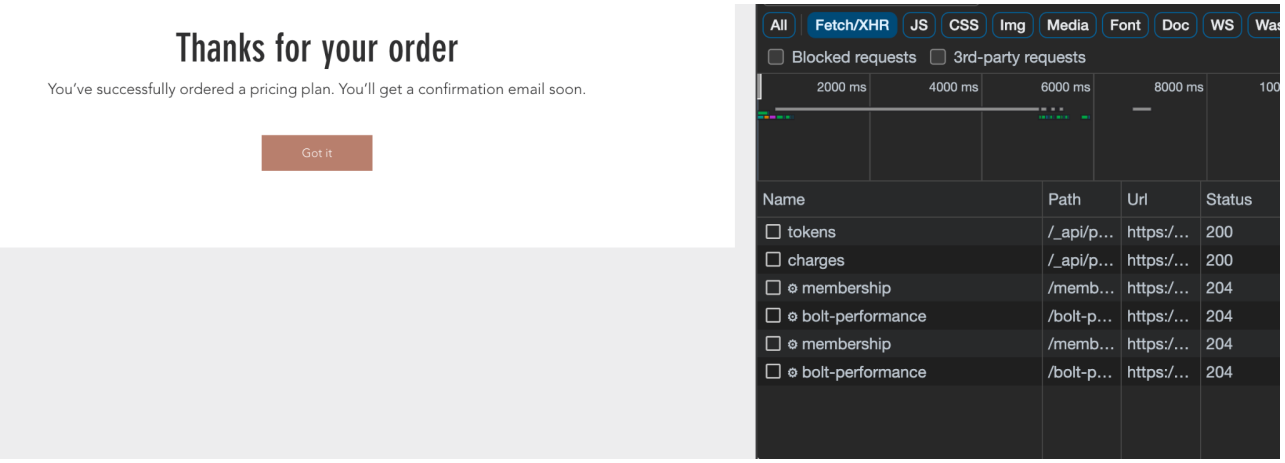


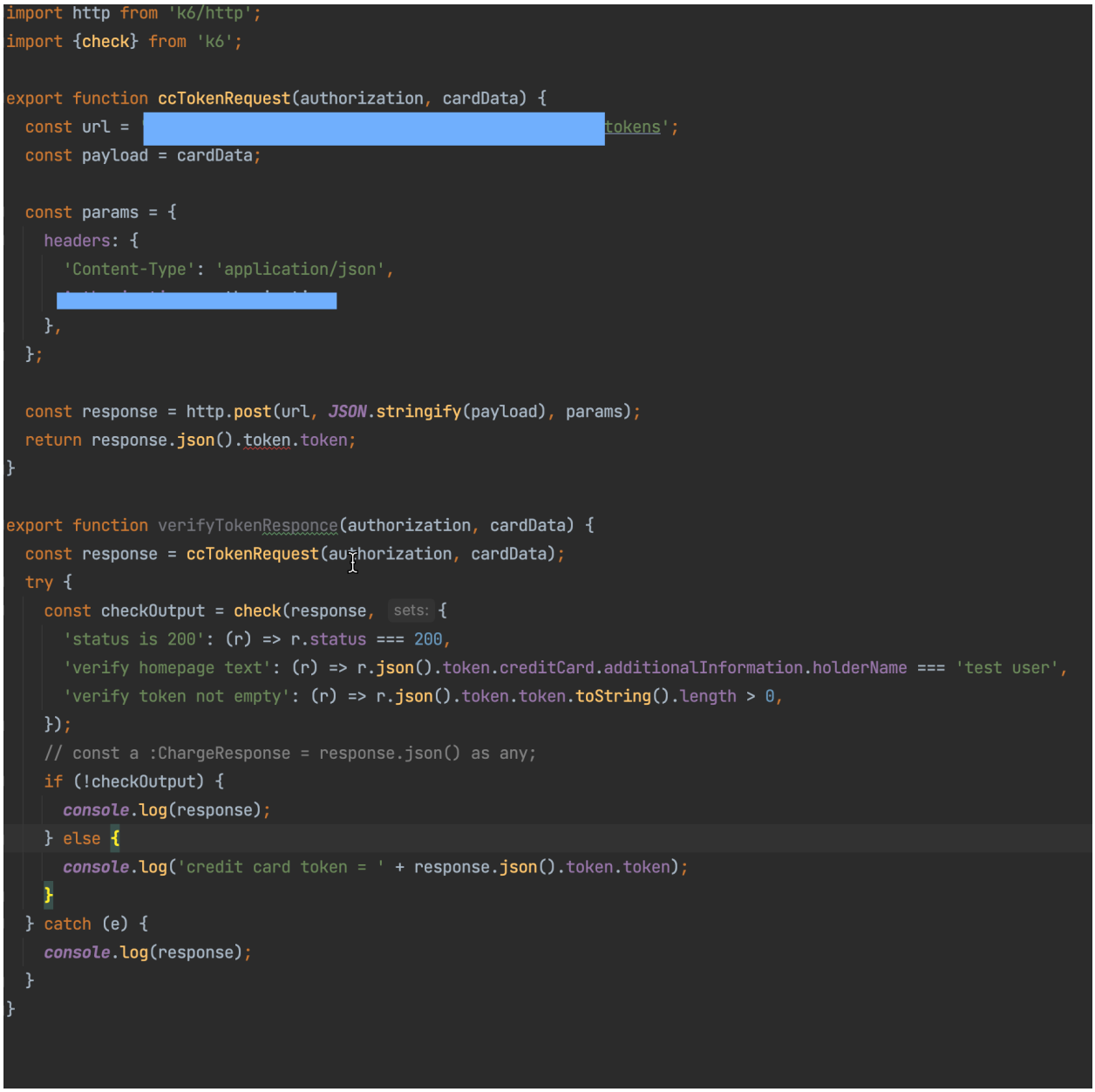
Рисунок 3.20 - Список запитів під час оплати

Отже ми бачимо що після натискання на кнопку оплатити в нас йде запит на токен картки а потім йде запит на саму оплату до провайдера а далі вже йдуть біай івенти. Тому ми обираємо цілю навантажити саме оплату адже цей ендпоінт покриє і токенизацію картки і запити до провайдера та їх обробку.

В програмі Intellij Idea ми будемо писати скрипт для навантаження. Для цього ми створимо директорію **loadTesting** та в середині створимо дві під дерикторії[15-16]:

- **helper** - тут ми будемо описувати сам запит до сервера з його тілом
- **runner** - тут ми будемо визивати нашу функцію що знаходиться в helper та передавати всі необхідні параметри для запуску

отже в хелпері ми створимо два файли окремо під token та під чардж. Давайте розберемо код файлу token на рисунку 3.21.



```
import http from 'k6/http';
import {check} from 'k6';

export function ccTokenRequest(authorization, cardData) {
  const url = '                                                tokens';
  const payload = cardData;

  const params = {
    headers: {
      'Content-Type': 'application/json',
                                          
    },
  };

  const response = http.post(url, JSON.stringify(payload), params);
  return response.json().token.token;
}

export function verifyTokenResponce(authorization, cardData) {
  const response = ccTokenRequest(authorization, cardData);
  try {
    const checkOutput = check(response, sets: {
      'status is 200': (r) => r.status === 200,
      'verify homepage text': (r) => r.json().token.creditCard.additionalInformation.holderName === 'test user',
      'verify token not empty': (r) => r.json().token.token.toString().length > 0,
    });
    // const a :ChargeResponse = response.json() as any;
    if (!checkOutput) {
      console.log(response);
    } else {
      console.log('credit card token = ' + response.json().token.token);
    }
  } catch (e) {
    console.log(response);
  }
}
```

Рисунок 3.21 - Код файлу token

Отже ми бачимо що ми написали функцію **ccTokenRequest** яка приймає на вхід авторизацію та карткові дані і результатом її буде повернено token який нам потрібен буде в наступному запиті на саму оплату. Також в нас є функція **verifyTokenResponse** вона буде застосовуватися у випадку якщо ми вирішимо нагрузити тільки запит на токен.

Далі переходимо до самого файлу charge рисунок 3.22.

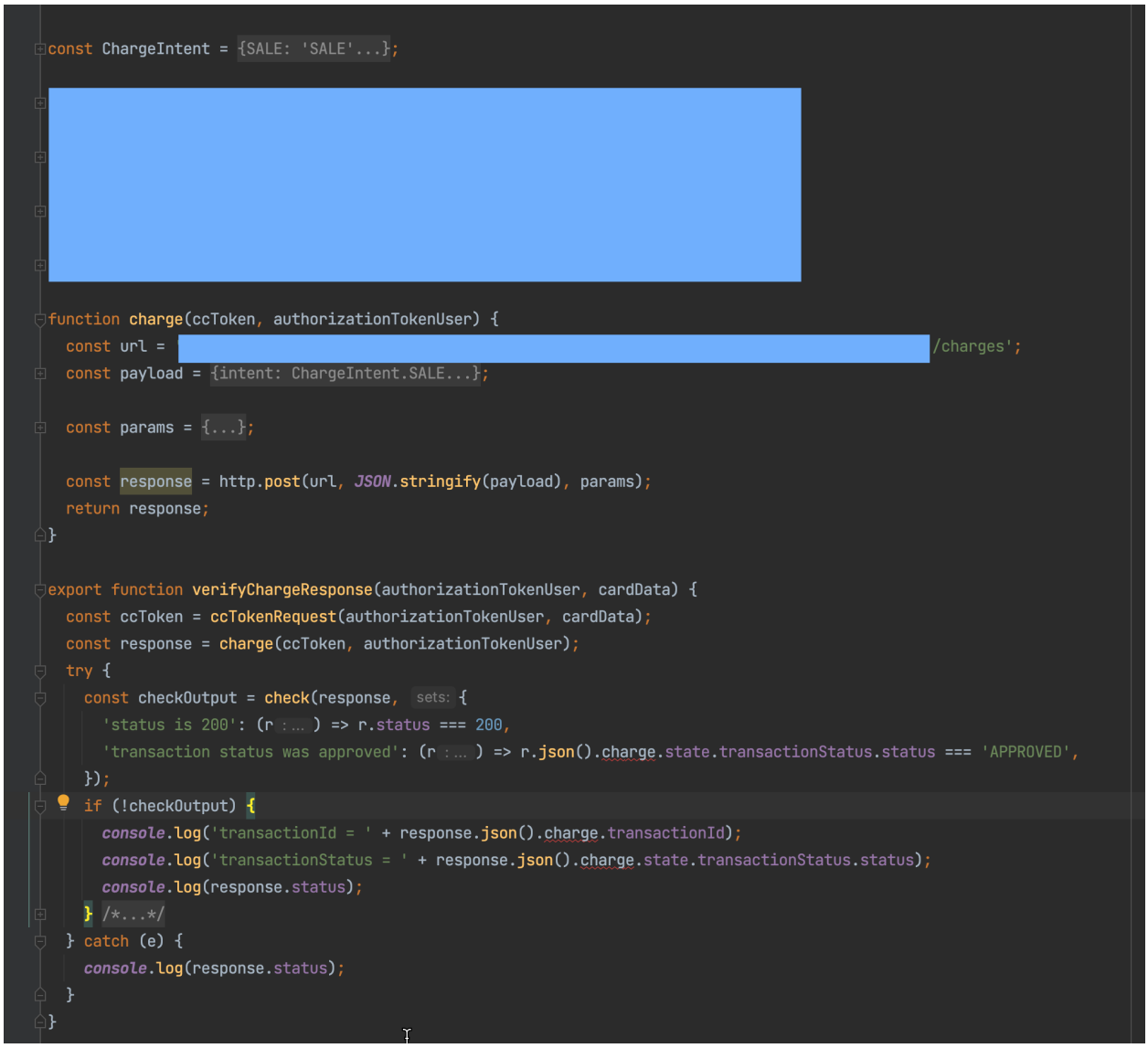


Рисунок 3.22 - Код файлу charge

Тут ми створюємо функцію charge в яку передаєм наш токен який потрібен для запиту та авторизацію. В тілі запиту ми створили константу об'єкт в яку поклали всю інформацію що заповнював юзер для оплати. Далі ми написали функцію яку і будем викликати в нашому файлі runner, **verifyChargeResponse**. Як бачимо ми тут викликаємо спочатку нашу функцію з отримання токену і вже потім наступним колом викликаємо наш запит на оплату в який кладемо наш токен картки. Далі в самому **try&catch** ми

валідуємо наш результат та виводимо інформацію яка нас цікавить якщо сталася помилка.

Тепер коли в нас все є для запуску переходимо в директорію runner та створюємо там файл **createTransaction** в якому задаємо параметри для нашої нагрузки, та викликаємо нашу функцію **verifyChargeResponse** рисунок 3.23.

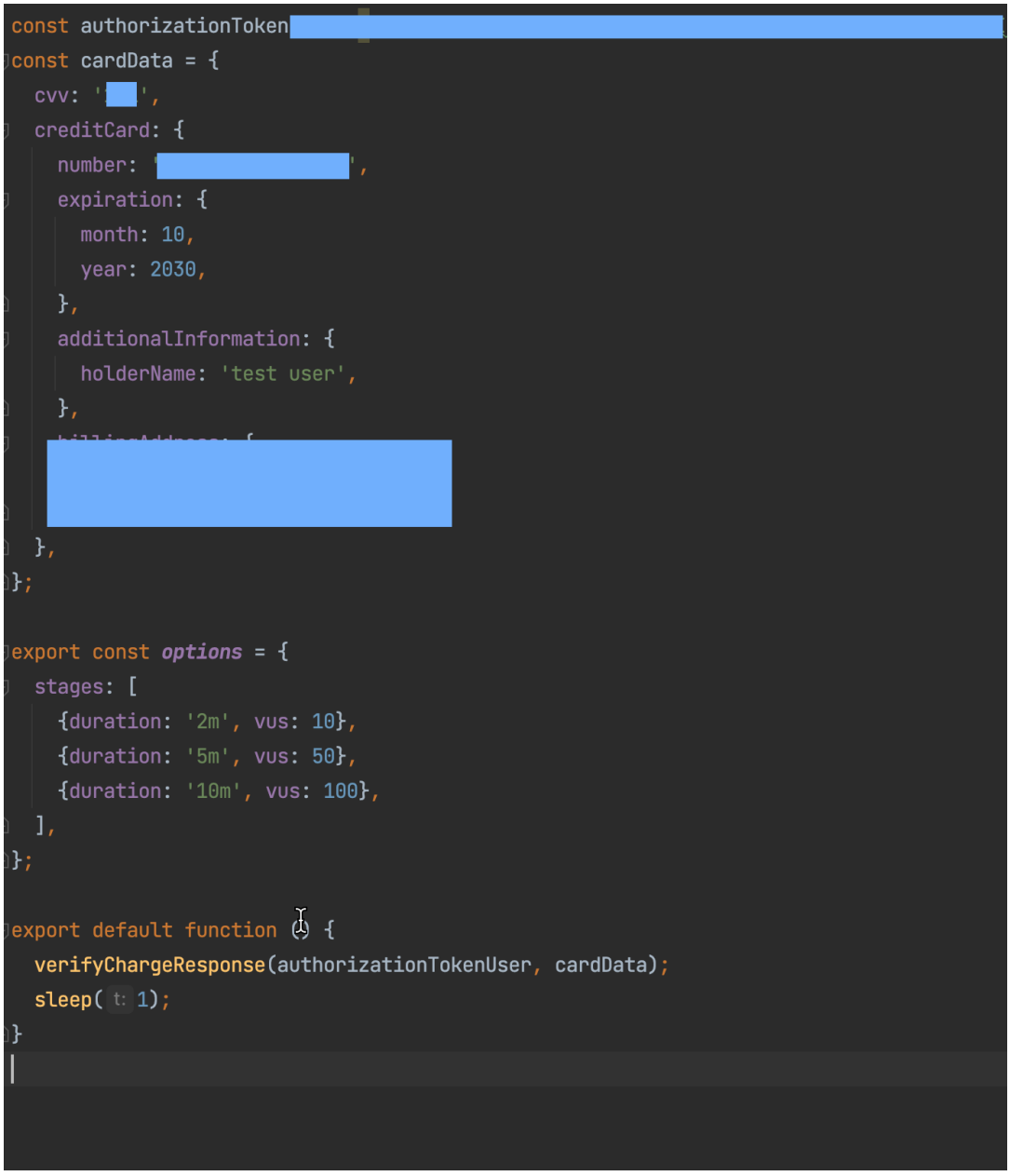


Рисунок 3.23 - Код файлу createTransaction

Отже згідно документації K6, **default function()** служить для запуску нашого тесту навантаження, тому саме в неї ми передаємо виклик нашої функції та прокидуємо токен та кредитні дані. В змінній **options** ми описуємо як буде проходити навантаження тобто першим етапом в нас буде йти паралельно 10 запитів (**vus: 10**) на протязі 2 хвилин (**duration: '2m'**) а далі буде збільшуватися до 100 запитів паралельно кожну 1 секунду (**sleep(1)**).

Отже після запуску нашого тесту командою **k6 run createTransaction.ts** в терміналі програми ми маємо результат навантаження на рисунку 3.24.

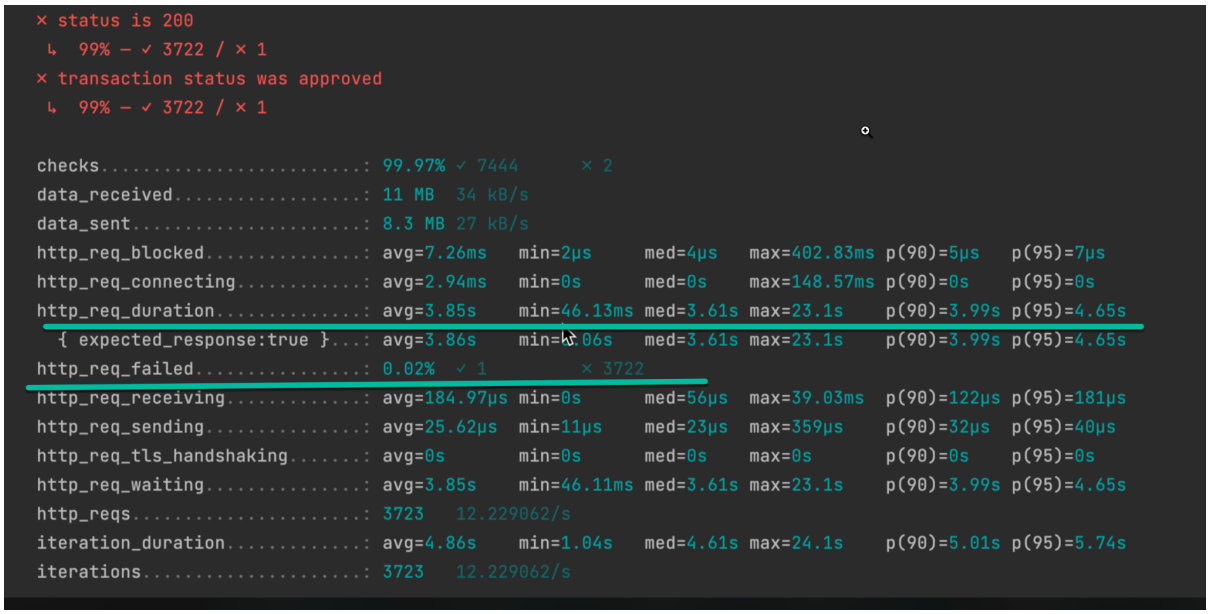


Рисунок 3.24 - Результат тесту навантаження

Як бачимо в цілому система навантаження витримує дуже добре. З помилок в нас було тільки одна помилка. Щодо покращення то якщо подивитися на колонку http_req_duration ми бачимо що максимальний час відповіді доходив до 23 секунд а це не дуже добре і отже є куди покращувати вашу систему.

3.3 Аналіз результатів тестування та розробка рекомендацій з застосування

1. Підміна відповідей від серверу за допомогою програми Charles: Під час тестування виявлено потенційні уразливості в системі, пов'язані із можливістю підміни відповідей від серверу за допомогою програми Charles. Детальний аналіз результатів вказує на необхідність зміцнення механізмів додаткової валідації на сервері, а також впровадження заходів для захисту від середнього типу атак.

Рекомендації:

Розширити використання шифрування для захисту передачі даних між клієнтом і сервером.

Запровадити механізми контролю даних і на клієнті і на сервері для виявлення можливих модифікацій відповідей.

Запровадити механізми виявлення та блокування ненормальної активності.

Поглиблювати знання тестувальника в програмі Charles

2. Тестування JWT за допомогою програми Postman:

Аналіз результатів тестування JWT вказує на стабільність та надійність механізмів генерації та перевірки токенів але це потрібно перевіряти для кожного продукту.

Рекомендації:

Обов'язково перевіряти варіанти з JWT які ми розглядали в прикладі.

Також продумувати інші види атак використовуючи JWT іншого юзера.

3. Тестування навантаження за допомогою бібліотеки K6:

Результати тестування навантаження свідчать про високу стабільність та ефективність системи під впливом різних навантажень. Виявлені бажані показники продуктивності та ефективності відповіді системи під час навантаження.

Рекомендації:

Продовжити моніторинг продуктивності системи при великих навантаженнях для визначення можливих точок перевантаження.

Удосконалити механізми автоматичного масштабування для забезпечення стійкості системи в умовах великих навантажень.

Постійно Аналізувати Систему та знаходити слабкі місця для подальшого їх покращення.

ВИСНОВОК

Отже ми вияснили що мануальне тестування програмного забезпечення визначається як критична складова процесу розробки. Цей метод використовується для виявлення помилок, вразливостей та можливих проблем, які можуть стати точкою входу для атак ззовні. Завдяки мануальному тестуванню розробники можуть бути впевнені в безпеці свого продукту перед випуском його на ринок.

Але, незважаючи на важливу роль, яку мануальне тестування відіграє в процесі розробки програмного забезпечення, саме через цю важливість воно може стати об'єктом кібератак та атак на інформаційну безпеку. Процес тестування може виявити вразливості, які, якщо залишити без відповідної уваги, стануть точкою доступу для зловмисників, порушуючи конфіденційність і цілісність даних та наносячи значний збиток організаціям та користувачам.

Саме тому ми з вами розглянули 3 приклади як тестувальник може покращити покриття продукту застосовуючи перевірки безпеки та знаходячи баги продукту до його релізу.

Усі ці аргументи свідчать про те, що мануальні тестувальники повинні активно розвивати свої знання та навички в галузі кібербезпеки. Це вимагає постійного навчання, оновлення та вдосконалення своєї експертизи. Такий підхід допомагає забезпечити високий рівень безпеки та надійності програмного забезпечення та відповідає вимогам сучасного кіберпростору. На завершення, важливо відзначити, що в сучасному світі програмне забезпечення може бути безпечним та функціональним лише завдяки спільним зусиллям та співпраці між різними групами фахівців, включаючи мануальних тестувальників, експертів з кібербезпеки та розробників.

СПИСОК ЛІТЕРАТУРИ

1. Задачі та обов'язки QA інженера та їх роль на проекті URL: https://dou.ua/lenta/articles/qa-engineer-position/
2. Життєвий цикл багу URL: https://ppt-online.org/71150
3. Різниця між QA автоматизатором та QA мануальним URL: https://dou.ua/forums/topic/41115/
4. Хто такий QA в ігровій інустрії та його вплив на проект URL: https://gamedev.dou.ua/articles/qa-engineer-in-gamedev/
5. Testing Computer Software, by Cem Kaner, Hung Q. Nguyen, Jack Falk 78p.
6. Відеоурок на тему підходи тестування та види тестування URL: https://www.youtube.com/watch?v=_bRyOvcMPxY
7. Структурна діаграма з більшістю видами тестування URL: https://coggle.it/diagram/We-qF3iayQABgwGL/t/testing
8. Rapid Testing, by Robert Culbertson, Chris Brown, Gary Cobb 134p.
9. Foundations of Software Testing ISTQB Certification, 4th edition 37p.
10. Налаштування програми Charles та взаємодія з нею в тестуванні URL: https://training.qatestlab.com/blog/technical-articles/use-charles-tool-in-testing/
11. Налаштування бібліотеки K6 та її повна документація URL: https://k6.io/docs/get-started/installation/
12. Foundations of Software Testing by Dorothy Graham and Erik van Veenendaal 231p.
13. Effective Software Test Automation by Kanglin Li 119p.
14. Agile Testing: A Practical Guide for Testers and Agile Teams by Lisa Crispin and Janet Gregory 32p.

15. Selenium WebDriver: From Foundations to Framework by Yujun Liang and Alex Collins 10p.
16. Appium Essentials by Manoj Hans 14p.