КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

автоматизації і інформаційних технологій
(факультет)

інформаційних технологій проектування та прикладної математики
(кафедра)

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО АТЕСТАЦІЙНОЇ ВИПУСКНОЇ РОБОТИ
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТР

на тему: «Моделі і методи системи стрес-тестування веб-серверу з використанням хмарних технологій»

САПАЄВ ВІКТОР ІВАНОВИЧ
(прізвище, ім’я та по батькові студента повністю)

Київ 2023 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

автоматизації і інформаційних технологій
(факультет)

інформаційних технологій проектування та прикладної математики
(кафедра)

ЗАТВЕРДЖУЮ
Завідувач кафедри ІТППМ
д.т.н., професор Терентьєв О.О.

„___” _____2023 року

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО АТЕСТАЦІЙНОЇ ВИПУСКНОЇ РОБОТИ
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТР

на тему: " Моделі і методи системи стрес-тестування веб-серверу з використанням хмарних технологій "

Виконав: студент II-го курсу, групи ІСТм-II

Спеціальності: 126 «Інформаційні системи та технології» .
(шифр і назва напряму підготовки, спеціальності)

Сапаєв В.І. .
(прізвище та ініціали)

Керівник к.т.н., доц. Гончаренко Т.А.
(прізвище та ініціали)

Рецензент к.т.н., доц. Шабала Є.Є.
(прізвище та ініціали)

Київ, 2023 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет: Автоматизації і інформаційних технологій

Кафедра: Інформаційних технологій проектування та прикладної математики

Освітній рівень: “магістр за ОПП”

Спеціальність: 126 “ Інформаційні системи та технології ”

ЗАТВЕРДЖУЮ
Завідувач кафедри ІТППМ
д.т.н., професор Терентьєв О.О.
_____
„___” _____2023 року

З А В Д А Н Н Я
ДО АТЕСТАЦІЙНОЇ ВИПУСКНОЇ РОБОТИ
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МІГІСТРА

Сапаєв Віктор Іванович

1. Тема роботи: “Моделі і методи системи стрес-тестування веб-серверу з використанням хмарних технологій”
   затверджена наказом ректора КНУБА №__ від “__” _____2023 року
2. Керівник роботи: Гончаренко Тетяна Андріївна, кандидат технічних наук, доцент, завідувач кафедри ІТ
3. Строк подання студентом роботи до захисту Грудень 2023 року.
4. Зміст пояснювальної записки за розділами:

Р. 1. Аналітично-теоретичне обґрунтування та розгляд суб’єкту атестаційної роботи
Р. 2. Складання плану робіт по виконанню поставленого завдання
Р. 3. Архітектурне рішення програмного забезпечення
Р. 4. Модель аналізу програмного забезпечення
Р. 5. Алгоритм впровадження програмного забезпечення

5. Графічний матеріал за розділами:

Р. 1. Дерево цілей, алгоритм автотестів
Р. 2. Структурна схема проекту
Р. 3. Клієнт-серверна архітектура, модель “ядра” проекту, інтерфейс
Р. 4. Графіки аналізу роботи програми, діаграми бази даних
Р. 5. Алгоритми мережі агентів

6. Календарний план виконання робіт: а) наукова частина; б) практична частина

Види робіт та їх зміст	Дата виконання
Розділ 1. Аналітичний огляд	Вересень 2023р.
Розділ 2. Формулювання технічного завдання	Жовтень 2023р.
Розділ 3. Розробка архітектури програмного продукту та інтерфейсу	Листопад 2023р
Розділ 4. Системи аналізу та збереження інформації	Грудень 2023р.
Розділ 5. Технічне впровадження програмного продукту	Грудень 2023р.
Остаточне оформлення роботи	Грудень 2023р.
Направлення роботи на рецензування, перевірку на плагіат	Грудень 2023р.
Попередній захист роботи на кафедрі	Грудень 2023р.

7. Консультанти розділів атестаційної випускної роботи

Розділ	Прізвище, ініціали та посада консультанта	Перевірив	
		Дата	Підпис
Прийом програмного продукту	к.т.н. доц. Шабала Є.Є.		

8. Дата видачі завдання 05 Вересня 2023 року.

Керівник _____ Гончаренко Т.А.
(підпис) (прізвище та ініціали)

Студент _____ Сапаєв В.І.
(підпис) (прізвище та ініціали)

РЕЗЮМЕ

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

Студент: Сапаєв Віктор Іванович

Науковий керівник: к.т.н., доц. Гончаренко Тетяна Андріївна

Факультет: Автоматизації і інформаційних технологій

Кафедра: Інформаційних технологій проектування та прикладної математики

Освітній рівень: “магістр за ОПП”

Спеціальність: 126 “ Інформаційні системи та технології ”

Тема роботи: “Моделі і методи системи стрес-тестування веб-серверу з використанням хмарних технологій”

Обсяг роботи. Атестаційна випускна робота магістра складається: розділів 5, стор. 108, таблиць 3, рис. 26, слайдів 6, завдання, анотації, вступу, висновків, списку використаних джерел (42 ресурси) та 4-х додатків.

Актуальність теми. Стрімкий розвиток інформаційних технологій, а саме у сфері надання онлайн послуг, призводить до зростання кількості запитів за період часу до веб-ресурсів. Відповідно до зростання трафіку кінцевих клієнтів, зростають ресурси необхідні для підтримання працездатності веб-серверів. Для моніторингу стресостійкості необхідні значні фінансові вкладення для розробки власного продукту, або затрати на придбання комерційних продуктів.

У вступі проведено критичний аналіз відомих інформаційних технологій та аналітичних засобів, обґрунтовано актуальність теми, сформульовано мету та основні завдання досліджень, показано зв`язок із програмами, планами, темами.

У першому розділі «Аналітичний огляд» проведене теоретичне дослідження предметної області як у сфері методології тестування та й у

сферах автоматизованих підходів до створення програмних продуктів тестування. Також, проведений аналіз існуючих програмних продуктів для тестування стресостійкості серверів та зроблений висновок про надмірну складність, або надмірну вартість комерційних варіантів.

Результатом даного розділу є створення дерева робіт проекту.

Другий розділ формує чітке технічне завдання на базі отриманих результатів аналітичного огляду. Серед них як список робіт так і критерії до кінцевого продукту. Також другий розділ визначив необхідні засоби для створення проекту. У ході написання другого розділу була створена структурна схема програмного продукту.

У третьому розділі відбувається архітектурне планування та розробка функціонального ядра проекту а також інтерфейсів. Також закладується фундамент модульності програмного продукту, необхідна для розширення функціональності програми.

Четвертий розділ присвячений плануванню структури збереження інформації, методам її обробки та аналізу. Розроблена система аналізу результатів проведених тестувань, для надання рекомендацій кінцевому користувачу щодо функціонування об’єкта тестування.

П’ятий розділ присвячений розгортанню проекту у межах комерційного сектору (автоматизоване розгортання), методам автоматизації виконання тестувань та системі розгалуженого тестування з переадресацією запитів через спеціально розроблені сервера.

Ключові слова: хмарні технології, стрес тестування, тестування, автоматизація, інформаційні технології, база даних, сервер.

Keywords: Cloud technologies, stress testing, testing, automation, information technology, database, server.

Якість оформлення випускної роботи. Атестаційна випускна робота магістра оформлена у відповідності до діючих нормативних документів та методичних вказівок для студентів спеціальності 122 «Комп’ютерні науки».

Загальний висновок стосовно роботи та присвоєння авторові освітнього рівня «магістр». Робота виконана на високому рівні, студент продемонстрував високий рівень теоретичної підготовки та сформованих практичних навичок в області сучасних інформаційних технологій.

Рекомендована оцінка. Заслуговує оцінку «відмінно».

Науковий керівник: _____ / к.т.н., доц. Гончаренко Т.А./

(підпис)

Посада, місце роботи: КНУБА, пр-т. Повітрофлотський, 31, доцент, завідувач кафедри інформаційних технологій

« 05 » грудня 2023 р.

АНОТАЦІЯ

Дипломної роботи на здобуття освітнього ступеня “Магістр”

Тема “Моделі і методи системи стрес-тестування веб-серверу з використанням хмарних технологій”

Автор: магістрант Сапаєв В.І.

Науковий керівник: к.т.н., доцент Гончаренко Т.А.

Стрімкий розвиток сучасних інформаційних технологій спричинили зсув від експлуатації локальних мережевих систем до розгалужених глобальних інформаційних мереж. Глобальні мережі (Інтернет) дозволяють великій кількості користувачів одночасно користуватися одним і тим же сервісом (веб-сайтом), що спричиняє надлишкове навантаження на сервер (технічне забезпечення веб-сайту) та зашкоджує нормальній роботів користувачів з веб-сайтом.

Крім цього, у зв’язку зі стрімким розвитком інформаційних технологій хмарного сегменту (мається на увазі веб-сегмент) та стислі терміни розробки виникає загроза персональній інформації користувачів подібних сервісів (під час розробки даних продуктів, допускаються помилки, що дають можливість отримати неавторизований доступ до персональної інформації).

Для запобігання озвучених проблем (захист інформації та стресо-стійкість серверів) витрачаються значні технічні та фінансові ресурси у великих підприємствах. Додатків витрати ресурсів на дослідження даних проблем у середніх та малих підприємствах не є доцільним, що відповідно впливає на якість наданих сервісів.

Ціллю даної роботи є створення автоматизованого програмного продукту з максимально простим інтерфейсом для покриття обраних клієнтом веб-серверів з ціллю верифікації вірності та швидкості відпрацювання даних серверів за заданими параметрами запитів.

Дипломна робота складається зі вступу, п’яти глав, висновку, списку літератури та додатків. У першому розділі проводиться теоретичний огляд предметної області та огляд існуючих програмних рішень поставленої проблеми. Другий розділ описує детальне технічне завдання. У третьому розділі надані аспекти технічної реалізації функціонального ядра продукту та інтерфейсу. Четвертий розділ присвячений реалізації бази даних та математичній моделі аналізу отриманих результатів. У п’ятому розділі розглянуті аспекти автоматизованого запуску та розгортанню програмного продукту. У висновках надані результати проведених досліджень, що дозволяють вирішити проблему з навантаженням серверів та захистити веб-сервери від певних типів несанкціонованого доступу.

Робота містить 3 таблиці, 26 малюнків, 42 літературних джерел та 4 додатки.

Хмарні технології, стрес тестування, тестування, автоматизація, інформаційні технології, база даних, сервер.

ANNOTATION

The rapid development of modern information technology has caused a shift from the operation in the local network systems to distributed global information networks. Global networks (the Internet) allow a large number of users to simultaneously use the same service (website), which causes excessive load on the server (technical maintenance of the website) and prohibits usual exploitation of the web-service.

In addition, cloud segment development (meaning web segment) nd the short terms of development, there is a threat to the personal information of users of a similar services (as a result of fast development, errors may be present in the system that allow unauthorized access to the sensative data).

In order to prevent these problems (information security issues and stress-resistant servers), considerable technical and financial resources are spent in large enterprises. Due to high costs of resolution of these problems, this solution (researches) cannot be applied for smaller companies, and as a result it influences quality of their services.

The purpose of this work is to create an automated software product with the most simple interface to cover the customer-selected web servers for the purpose of verifying the authenticity and speed of the processing of these servers according to the specified query parameters.

The thesis consists of an introduction, five chapters, a conclusion, a list of literature and appendices. The first section provides a theoretical review of the subject area and an overview of existing software solutions to the problem. The second section describes the detailed technical specification. The third section provides aspects of the technical implementation of the functional core of the product and the interface. The fourth section is devoted to the implementation of the database and the mathematical model of the analysis of the results. The fifth section covers the aspects of automated launch and deployment of a software product. The conclusions provide the results of the research conducted to solve the

problem with the load of servers and protect web-servers from certain types of unauthorized access.

Work contains 3 tables, 26 drawings, 42 literary sources and 4 applications.

Cloud technologies, stress testing, testing, automation, information technology, database, server.

ЗМІСТ

ВСТУП

Сучасні програмні продукти (ПП) доволі часто розробляються в стислі терміни і при обмежених бюджетах проектів, що зумовлено методологією розробки Agile, а також терміном життя популярного типу бізнесу як Start Up. **Актуальною проблемою** стало нехтування інформаційною безпекою та стресостійкістю продукту, що зумовлено розробкою у поспіху, що піддає тим самим користувачів таких продуктів на невиправданий ризик.

Наприклад, системи Інтернет-платежів для віртуальних магазинів, як правило, розробляються для кожного магазину окремо, практично «з нуля», використовуючи технічні засоби без урахування ступеня їх захищеності. У результаті [1] та нестабільної роботи систем [2].

Наприклад (загроза втрати частки ринку), у вересні 2001-року John Pescatore, експерт в області Інтернет-безпеки компанії Gartner, провідної консалтингової компанії в галузі інформаційних технологій, рекомендував компаніям і організаціям, які постраждали від вірусів Red Code і Nimda, негайно відмовитися від використання Microsoft Internet Information Server [3]. Подібні рекомендації не могли не позначитися на популярності Веб-серверів на ринку [4]. Слід визнати, що наступні версії Веб-сервера компанії Microsoft, виходили зі зрослим увагою до питань забезпечення інформаційної безпеки [5] [6].

Для **практичного вирішення** даної проблеми, сучасні корпорації та великі підприємства використовують автоматизовані системи тестування та навантаження веб-серверів (або хмарного ПЗ (програмне забезпечення)) [7]. Варто зазначити, що вартість розробки таких індивідуальних систем доволі висока [8], що виключає їхнє використання малими підприємствами та збільшує ризик критичних помилок.

Існує велика кількість умовно-безкоштовних сервісів [9] з допомогою яких можна проводити стрес тестування, проте дані рішення не вирішують спектру різностороннього тестування (наприклад, навантаження з різних

вузлів, що імітує реальну DDOS атаку на сервер), вимагають комплексного налаштування виділеного серверу, продають необхідний функціонал додатково (наприклад, хмарне тестування або функціонал проксі серверів) або мають занадто складу систему налаштування автоматизації або узагалі виконання сценарію (наприклад, JMeter), що вимагає наявність спеціаліста з налаштування подібних систем.

Метою даного дослідження є розробка простого для користувача ПЗ, для тестування складних веб систем з використанням хмарних технологій.

Для досягнення даної цілі необхідно виконати наступні **задачі**:

1. Провести аналіз чинних програмних рішень
2. Розробити архітектуру ПЗ
3. Провести User eXperience (UX) дослідження для створення максимального просто графічного інтерфейсу (GUI)
4. Створити ПЗ та простий спосіб його розвернення на сервері
5. Автоматизації ПЗ з використанням хмарного сервісу

Науковою проблемою даної роботи виступає аналіз та дослідження необхідного функціоналу для малого та середнього бізнесу для автоматизованої підтримки ПЗ у працездатному стані з максимально спрощеним процесом інсталяції та експлуатації розробленого продукту.

Темами дипломної роботи стануть:

1. Розробка архітектуру ПЗ згідно методології ООП
2. UX дослідження користувачів подібних програм
3. Виокремлення найкращих хмарних рішень для подібних автоматизованих систем

Об’єктом дослідження є програмне забезпечення, що цілеспрямоване на роботі з клієнтом, та яке є суб’єктом потенційних стресових та інформаційних навантажень.

Предметом дослідження виступає серверна частина ПЗ та інтерфейс взаємодії з користувачем (у даному ключі – браузером) в умовах тестування відмовостійкості усієї системи.

Зв’язком предмету та об’єкту дослідження є інтерфейс передачі даних від клієнта у систему аналізу та обробки даних, а також інтерфейс виведення інформації клієнту (простіше кажучи, IO системи).

Основними **методами дослідження** для проведення аналізу чинних систем буде порівняння характеристик безкоштовного функціоналу та експериментальне використання програмного забезпечення.

Для побудови архітектури буде використаний стандартний метод індукції та дедукції для максимальної деталізації архітектури проекту.

Для UX застосуємо експериментально-теоретичний метод та анкетне опитування дизайнерів, з метою розробки найпростішого інтерфейсу.

З метою створення програмного продукту буде використана форма формалізації – відображення архітектури за допомогою прикладної мови програмування.

Хмарна частина проекту буде здійснена з допомогою аналізу та часткової формалізації (хід виконання даної задачі залежить від архітектури проекту, та результатів UX дослідження).

У ході виконання дипломної роботи будуть використані наступні **джерела**:

1. “How Google Tests Software”, James A. Whittaker, Jason Arbon, Jeff Carollo
2. “Agile Testing: A Practical Guide for Testers and Agile Teams”, Lisa Crispin, Janet Gregory
3. “The Art of Application Performance Testing: From Strategy to Tools”, Ian Molyneaux
4. “Performance Testing Guidance for Web Applications”, Microsoft corporation
5. “Тестирование дот ком”, Роман Савин
6. “http://developers.loadimpact.com/sdk/”, Load Impact SDKs

Результатом даної дипломної **роботи** стане open source програмний продукт, що може бути використаний у любій програмній системі, що працює з використанням REST клієнт-серверної архітектури “із коробки” (тобто, без додаткових налаштувань), простої структурою та мінімальним інтерфейсом (що дозволить користуватися продуктом без додаткових навичок).

Наукові розробки буде представлена на захисті дипломної роботи **18.06.2018** року.

1 АНАЛІТИЧНИЙ ОГЛЯД

Для проведення максимально ефективного аналізу та обґрунтування предметною області, необхідно детально познайомитися з ключовими поняттям, що будуть використані для виконання дипломної роботи.

Потрібно розглянути такі поняття як (у представленому порядку):

1. Тестування продуктивності комп’ютерних систем (у тому числі типи тестування)
2. Автоматизація процесів тестування та розгортання автоматизованих систем
3. UX дослідження для дизайну графічного інтерфейсу користувача
4. Аналіз чинних джерел та програмних продуктів

1.1 Загальна характеристика тестування продуктивності

З появою Web 2.0 сайти стали інтерактивними - вони навчилися дізнаватися своїх відвідувачів і підлаштовуватися під їхні очікування, а користувачі, у свою чергу, стали вимогливішими до сайтів, чекаючи, що їх будуть розуміти з півслова і адекватний запитам результат буде отримано миттєво. Все це призвело до колосального росту навантаження на веб-сервери і посилення вимог до рівня їх обслуговування - будь-які зволікання чи помилка в роботі сайту призводять до втрати відвідувачів. Працездатність сайту стала критично важливою для більшості сучасних компаній, а ціна прорахунку в оцінці його реальної продуктивності значно збільшилася. Все це робить актуальним тестування навантаження. [10]

Сьогодні використовується два підходи до автоматизації тестування навантаження і створення тестового навантаження. Перший - створення безлічі копій браузерів, що виконують дії користувача в автоматичному режимі. У даному випадку відтворюється сеанс взаємодії з реальним користувачем, що, однак, вимагає великої кількості ресурсів. На одному комп'ютері стандартної архітектури можна запустити не більше сотні

паралельно функціонуючих копій браузера, і для моделювання істотного навантаження необхідно розгорнути мережу комп'ютерів. Другий підхід - емуляція HTTP-запитів, при якому емулюються дії користувача, але можливе створення тисяч віртуальних користувачів на одному комп'ютері.

Для максимальної ефективності у ході виконання дипломної роботи буде емульована гібридна система генерації HTTP-запитів з використанням виділених серверів (хмарні мережі).

Продукти для навантажувального тестування можна розбити на три класи: безкоштовні; корпоративні, що володіють розширеною функціональністю; недорогі, але універсальні. До першої категорії належить такий продукт, як JMeter [11], який, не зважаючи на вільне поширення, може вимагати дорогих фахівців. Крім того, у продукті відсутня підтримка, а процес створення тесту займає багато часу. До другої групи можна віднести продукти HP LoadRunner, IBM Rational Performance Tester [12] і Borland Silk Performer, що дозволяють автоматизувати процес тестування навантаження, прискоривши та значно підвищивши його якість. Наприклад, функціональність LoadRunner задовольнить навіть найвибагливішого тестувальника - в продукті є практично все для проведення різного роду тестування, проте вартість його висока, а освоєння вимагає часу на вивчення всіх можливостей. До третьої групи можна віднести такі продукти, як PureLoad [11], NeoLoad і WAPT, які легко встановлюються і налаштовуються для виконання навантажувального тестування веб-додатків, що працюють під управлінням будь-якої ОС.

З якими особливостями навантажувального тестування в сучасних веб-середовищах доводиться зіткнутися тестувальникам і яким вимогам повинні задовольняти інструменти, що автоматизують процес тестування?

1. Важливим критерієм при тестуванні є використання **динамічних сесій**. Сучасний сайт являє собою багаторівневий динамічний додаток, що складається з мережевого екрану, балансувальника навантаження, кількох веб-серверів і серверів додатків, бази даних. Для того щоб

«оцінювати» своїх відвідувачів, більшості сайтів доводиться підтримувати індивідуальні сесії користувачів і працювати з динамічними даними. Така гнучкість призводить до того, що і тести для перевірки працездатності таких сайтів теж повинні змінюватися динамічно. Це призвело до появи нової методології тестування, керованої даними, яка висуває нові вимоги тестування навантаження.

2. Тест повинен керуватися даними, а це означає, що кожен віртуальний користувач повинен мати у своєму розпорядженні свій набір тестових даних, повинен виглядати для сайту унікальним, відмінним від інших «відвідувачів», повинен вміти працювати з призначеними для нього відповідями сервера.
3. При створенні сучасних веб-сайтів все більшого поширення набуває методологія гнучкої розробки (agile) [13], коли процес розробки ділиться на серію швидких ітерацій, кожна з яких проходить повний цикл - від формування вимог до отримання робочої версії, що реалізує ці вимоги. Оскільки в рамках даної методології зміни в систему вносяться часто, то і навантажувальні тести для створюваного веб-додатку доводиться постійно змінювати. Це вимагає від навантажувального тестування можливості швидкого створення тестового сценарію, або впровадження системи **автопараметризації**
4. Сучасні сайти в разі помилки зазвичай відповідають користувачеві не кодом помилки, а повідомленням про нештатну ситуацію, тому ключові відповіді сервера доводиться перевіряти на коректність виконання. Засіб навантажувального тестування повинно дозволяти тестувальника задавати критерії для перевірки (**валідації**) відповідей сервера.
5. Зазвичай тестування проводиться в середовищі, яка по якомусь критерію відрізняється від робочої (ресурси, засоби балансування навантаження і т.п.), тому поширеною проблемою є невідповідність тестової і робочої систем. Часто через нестачу ресурсів або обмеження програм навантажувального тестування тести проводять на слабкому

обладнанні і моделюють меншу кількість користувачів, ніж передбачається для робочої системи. При цьому частина підсистем - наприклад, що відповідає за динамічне виділення ресурсів залежно від навантаження, балансувальник навантаження і т.п. - можуть взагалі не брати участь в тестуванні. В результаті може сформуватися неадекватна оцінка продуктивності тестованої системи, а частина проблем може бути взагалі не виявлено. Часто при тестуванні не враховується географічний розподіл навантаження, яка зазвичай має великий розкид за швидкостями з'єднання користувачів і часи відгуку. Однак в робочій системі це може призводити до специфічних проблемам - наприклад, повільні користувачі можуть вимагати більше ресурсів сервера (з'єднання відкрито довше, витрачено більший обсяг пам'яті на зберігання даних і т.д.). Всі ці особливості повинні враховуватися при розробці тестів, засоби тестування повинні дозволяти створювати навантаження, еквівалентну робочої, а тестова система по продуктивності повинна бути дорівнює робочої і використовувати ті ж підсистеми. Даний критерій має назву **тестування на «неадекватній» системі**.

6. Сучасні навантаженні (мається на увазі трафік) сайти - це розподілені системи, які часто розташовуються в хмарах. Для тестування подібних сайтів необхідні порівнянні за потужністю ресурси, тому також доцільно проводити **тестування навантаження в хмарах**. [14] Перебуваючи в хмарі, можна проводити тестування як з-за меж периметра веб-сайту, так і зсередини. Це, зокрема, дозволяє визначити вплив брандмауера і балансувальника навантаження на продуктивність системи.
7. Тестування навантаження призначене не тільки для визначення продуктивності системи, але і для знаходження її вузьких місць. Для цього потрібно вміти проводити моніторинг продуктивності тестованої системи. Збір даних про роботу тестованої системи (завантаження

процесора, використання пам'яті і дискової підсистеми, утилізація мережевих інтерфейсів, дані про продуктивність веб-серверів і серверів баз даних) дозволяє знаходити її вузькі місця, що призводять до проблем з продуктивністю. Тобто, ще одним критерієм є **моніторинг продуктивності**.

Сьогодні на ринку є ряд універсальних досить простих засобів навантажувального тестування, що дозволяють більшості розробників і тестувальників створювати надійні сайти. Як приклад можна назвати сімейство продуктів WAPT, можливості яких типові для рішень даного класу. Ці системи дозволяють організувати як локальне тестування при невеликих навантаженнях до 2 тис. віртуальних користувачів, так і розподілене - з моделюванням високих навантажень. Крім того, в сімействі є засоби для тестування в хмарах і з хмар. Продукти можна використовувати для тестування будь-якого веб-додатки, що працює з протоколом HTTP(S), виключаючи деякі застарілі протоколи (наприклад, RTMP).

Для створення профілю віртуального користувача досить виконати всі його дії в обраному браузері, а спеціальний рекордер запише послідовність запитів браузера і параметризує динамічні дані. Для підтримки динамічних сесій є можливість параметризації даних в запиті: заголовок запиту, файли cookie, параметри запиту в рядку і в тілі запиту, що дозволяє створювати унікальні запити. Параметризація проводиться за допомогою набору функцій, що дозволяють обійтися без написання коду, хоча в особливо складних випадках можна використовувати JavaScript. Для реакції на відповіді сервера і для зміни поведінки користувача використовуються оператори циклів, розгалуження, рандомізації, що дозволяють створювати користувачів з відмінним один від одного поведінкою.

Валідації відповідей здійснюються за часом відповіді та по його вмісту. Валідація за часом може бути застосована у випадках, коли тимчасові характеристики відповіді задані заздалегідь. Валідація у вмісті дозволяє за ключовими словами у відповіді сервера визначити помилковість або

правильність виконання. Також завжди є можливість перевірити відповідь вручну з JavaScript.

Для створення розподіленого навантаження існує «робоче місце» для запуску тестів і агентів, що створюють навантаження. Агенти можуть розташовуватися в будь-якому місці - для їх роботи необхідний доступ до тестованого сайту і зв'язок з «робочим місцем» користувача. Все це дозволяє масштабувати і розподіляти навантаження, моделюючи роботу необхідної кількості користувачів. Можна також змінювати індивідуальну швидкість віртуального користувача, що дозволяє імітувати, наприклад, роботу повільних користувачів, що звертаються до додатка з мереж 3G.

У продуктах такого типу повинні бути також кошти розміщення в публічному сервісі, наприклад Amazon. З огляду на, що центральні сервери Amazon розташовуються по всьому світу, це дозволить вибрати найбільш зручне місце для тестування і організації географічно розподіленої навантаження.

Для підвищення якості тестування в системі повинен бути організований моніторинг продуктивності веб-серверів, наприклад через протоколи WMI і SNMP. Під час виконання тесту з допомогою вбудованих лічильників продуктивності можна відстежувати навантаження на процесор, оцінювати використання пам'яті, дискової та мережевих підсистем будь-якого сервера, що входить до складу сайту, а через ODBC можна отримувати специфічні для баз даних лічильники продуктивності.

1.2 Поняття та види стрес-тестування

Стрес-навантаження розподіляється на наступні типи:

- навантажувальний (load)
- стрес (stress)
- тестування стабільності (endurance or soak or stability)
- конфігураційне (configuration)

Можливі два підходи до тестування продуктивності програмного забезпечення [15, pp. 35-38]:

- в термінах робочого навантаження: програмне забезпечення піддається тестуванню в ситуаціях, що відповідають різним сценаріям використання;
- в рамках бета-тестування, коли система випробовується реальними кінцевими користувачами.

1.2.1 Тестування навантаження

Тестування навантаження (англ. Load testing) - підвид тестування продуктивності, збір показників і визначення продуктивності і часу відгуку програмно-технічної системи або пристроїв у відповідь на зовнішній запит з метою встановлення відповідності вимогам, що пред'являються до даної системи (пристрою). [16]

Термін перевірки навантаження використовується різними способами в професійному співтоваристві тестування програмного забезпечення. Тестування навантаження взагалі відноситься до практики моделювання очікуваного використання програмного забезпечення шляхом імітації декількох користувачів, які одночасно отримують доступ до програми. [17] Таким чином, це тестування найбільш актуальне для багатокористувацьких систем; часто будується за допомогою моделі клієнта / сервера, наприклад веб-сервери. Тим не менш, інші типи програмних систем також можуть бути перевірені навантаженням. Наприклад, текстовий процесор або графічний редактор можуть бути змушені читати надзвичайно великий документ; або фінансовий пакет може бути змушений сформувати звіт на основі кількох років даних. Найбільш точне тестування навантаження імітує фактичне використання, на відміну від тестування з використанням теоретичного або аналітичного моделювання.

Ціллю стрес тестування є створення таких умов роботи для об’єкту тестування, за яких максимально близько до реальності імітується поведінка

кінцевого клієнта, а також існує можливість знімати показники ефективності роботи системи, що тестується.

Веб-сервіс з функціональністю кошика покупця розрахований на 100 одночасно працюючих користувачів, які слідують деякого певним сценарієм (задані дії в зазначених пропорціях):

- *25 користувачів переглядають товар і виходять із системи.*
- *25 користувачів додають товар в кошик, оформляють його і виходять із системи.*
- *25 користувачів використовують функцію повернення товару і виходять із системи.*
- *25 користувачів входять в систему і не виявляють ніякої активності.*

У цьому кейсі тест навантаження повинен наслідувати описаний звичайний сценарій взаємодії з веб-службою, щоб забезпечити готовність служби до введення в експлуатацію.

Приклад 1.1 Сценарій тестування навантаження

В ідеалі, вимоги щодо ефективності системи, що складаються та задокументовані на етапі формулювання функціональних вимог системи до основних архітектурних рішень, служать критерієм успіху тестування навантаження. Проте, часто такі вимоги не були чітко узгоджені чи узгоджені взагалі. У такому разі першим навантаженням буде тест на пошукове навантаження і ґрунтується на обґрунтованих припущеннях про очікуване навантаження та споживання ресурсів апаратного забезпечення.

Одна з найкращих практик використання тестування навантаження для вимірювання продуктивності системи - тестування на перших етапах розробки системи. Випробування навантаження на перших етапах готовності архітектурного рішення для визначення його здатності витримувати навантаження називається випробуванням "proof-of-concept".

1.2.2 Основні принципи навантажувального тестування

Нижче розглянуті деякі експериментальні факти, узагальнені принципи, які використовуються при тестуванні продуктивності в цілому і застосовні до будь-якого типу тестування продуктивності (зокрема і до тестування навантаження).

1. Унікальність запитів. Навіть сформувавши реалістичний сценарій роботи з системою на основі статистики її використання, необхідно розуміти, що завжди знайдуться винятки з цього сценарію. У разі прикладу зі сценарієм тестування навантаження це може бути користувач, який звертається до відмінних від всіх інших, унікальних сторінок веб-сервісу.
2. Час відгуку системи. У загальному випадку час відгуку системи підпорядковується функції нормального розподілу. Зокрема, це означає, що, маючи достатню кількість вимірювань, можна визначити ймовірність з якою відгук системи на запит потрапить в той чи інший період часу.
3. Залежність часу відгуку системи від ступеня розподіленості цієї системи. Дисперсія нормального розподілу часу відгуку системи на запит пропорційна відношенню кількості вузлів системи, паралельно обробних такі запити і кількості запитів, що припадають на кожен вузол. Тобто, на розкид значень часу відгуку системи впливає одночасно кількість запитів припадають на кожен вузол системи і сама кількість вузлів, кожен з яких додає деяку випадкову величину затримки при обробці запитів.
4. Розкид часу відгуку системи. З попередніх тверджень можна також зробити висновок, що при досить великій кількості вимірювань величини часу обробки запиту в будь-якій системі, завжди знайдуться запити, час обробки яких перевищує певні у вимогах максимуми; причому, чим більше сумарний час проведення експерименту тим вищі будуть нові максимуми. Цей факт необхідно враховувати при

формуванні вимог до продуктивності системи, а також при проведенні регулярного навантажувального тестування.

5. Точність відтворення профілів навантаження. Необхідна точність відтворення профілів навантаження тим дорожче, чим більше компонентів містить система. Часто неможливо врахувати всі аспекти профілю навантаження для складних систем, тому що чим складніше система, тим більше часу буде витрачено на проектування, програмування і підтримку адекватного профілю навантаження для неї, що не завжди є необхідністю. Оптимальний підхід в даному випадку полягає в балансуванні між вартістю розробки тесту і покриттям функціональності системи, в результаті якого з'являються припущення про вплив на загальну продуктивність тієї чи іншої частини тестованої системи.

1.2.3 Стрес-тестування

Стрес-тестування (англ. Stress Testing) - один з видів тестування програмного забезпечення, яке оцінює надійність і стійкість системи в умовах перевищення меж нормального функціонування. Стрес-тестування особливо необхідне для «критично важливого» ПО, однак також використовується і для решти ПО. Зазвичай стрес-тестування краще виявляє стійкість, доступність і обробку винятків системою під великим навантаженням, ніж те, що вважається коректною поведінкою в нормальних умовах. [18]

Загалом методологія стрес-тестування ґрунтується на видаленні та аналізі ефективності застосування на навантаженнях, що перевищують очікувані на етапі супроводу, і призначені для визначення витривалості або стійкості застосування у разі сплеску діяльність щодо її використання.

Необхідність стрес-тестування диктується наступними факторами [18]:

1. Більшість систем розробляються з припущенням про функціонування в нормальному режимі, і навіть коли дозволяється збільшити навантаження, фактичний обсяг його збільшення не

враховується.

2. У випадку контракту SLA (угода про рівень обслуговування), вартість системної відмови в екстремальних умовах може бути дуже високою.
3. Не завжди можна виявити деякі помилки чи дефекти функціонування системи, використовуючи інші типи тестування.
4. Тестування, зроблене розробником, може бути недостатнім, щоб наслідувати умови, за яких система відмовляється.
5. Переважно, будьте готові до керування екстремальними умовами системи, ніж очікувати його невдачі.

Основні напрямки застосування стрес-тестування:

Загальне дослідження поведінки системи при пікових навантаженнях.

- Аналіз обробки критичних помилок і фатальних (помилкових) ситуацій системою при максимальних навантаженнях.
- Аналіз “вузьких” точок системи або аналіз окремих вузлів системи використовуючи нерівномірне навантаження.
- Аналіз місткості системи.

Пропорційне навантаження

Стрес-тестування можна використовувати як для автономних програм, так і для розподілених систем з архітектурою клієнт-сервер. Примітивним прикладом такого тестування виділеної програми є відкриття файлу розміром 50 мегабайт у програмі “Блокнот”, яка входить до складу операційної системи Windows. Умови стрес-тестування програми зазвичай формуються на основі критичних бізнес-процесів його функціональності, визначених на етапі розробки вимог та аналізу ризику групою, відповідальною за ефективність.

Диспропорційне навантаження

У випадку тестування багаторозподілених розподільних систем

необхідно враховувати не тільки фактичний об'єм навантаження, а й їх частку в загальному обсязі.

Веб-сервіс призначений для обробки і відображення створених користувачем сторінок, кожна з яких може складатися з звичайного тексту і динамічних елементів управління. Відомо, що один користувач створює 1 сторінку в день, яка містить в середньому 1000 символів тексту і одну форму. Відомо також, що до системи йде 1 запит на відображення вихідної сторінки в хвилину. При цьому швидкість відображення сторінки є критичним бізнес-процесом.

Використовуючи описану вище модель поведінки використовують систему користувачів, нескладно змоделювати зміну динаміки навантаження при їх збільшенні. Стрес-тест, в якому взята за основу така модель навантаження не адресує ризиків, пов'язаних з тим, що система перестане задовольняти вимогам продуктивності при зміні сценарію її використання. Наприклад, швидкість відображення сторінки може істотно знизитися, якщо користувачі будуть додавати на неї десятки форм замість однієї.

Приклад 1.2. Стрес-тест сценарій

Використання непропорційного навантаження в тестах навантаження може бути використаним для виявлення вузьких вузлів певних модулів системи.

1.2.4 Тестування стабільності

Тестування стабільності або надійності (Stability / Reliability Testing) - один з видів автоматизованого тестування ПЗ [19], метою якого є перевірка працездатності програми при тривалому тестуванні з очікуваним рівнем навантаження.

Перед виконанням екстремального тесту (пікові навантаження) варто провести аналіз системи за умов передбачених навантажень, тобто у

очікуваній робочій атмосфері. [20] Під час тестування тривалість його поведінки не має першочергового значення, головне завдання - спостерігати за споживанням ресурсів, виявляти витік пам'яті та відстежувати швидкість обробки даних та / або час відгуку програми на початку тест і з плином часу не знижувався. В іншому випадку можливі збої в продукті та перезавантаження системи.

1.3 Процес автоматизації процесу тестування

Ключовим моментом тестування продуктивності є наявність на простота автоматизації процесу тестування, так як автоматизації призводить до зменшення оперативних витрат на персонал та збільшення ефективності підтримки працюючого ПЗ.

Автоматизоване тестування ПЗ (Automation Testing) [21] - це процес перевірки ПЗ, де більшість (або усі) кроки (ініціалізація, виконання, аналізу та презентації результатів) виконуються програмно, без втручання оператору.

1.3.1 Ціль автоматизації

Для того щоб правильно представити цілі автоматизації, необхідно виокремити недоліки та переваги використання автоматизації у процесі розробки ПЗ для тестування продуктивності веб-сервісів.

Переваги автоматизації тестування

- Повторюваність - виключений «людського фактору» за рахунок повного дублювання кожного перебігу програми.
- Швидке виконання – автоматизований тест виконується набагато швидше у порівнянні з мануальною перевіркою.
- Менші витрати на підтримку – підтримка життєздатності автоматизованого коду вимагає менше часу аніж мануальна перевірка.
- Звіти – висновки перебігу тесту складаються автоматично, що виключає потребу у написанні звітів самостійно.

- Виконання без втручання – тести частіше за усе проводяться без оператору, в не робочий час.

Недоліки автоматизації тестування

- Повторюваність – так як тести виконуються одноманітно це включає фактор того, що оператор побачить щось нове, не описане в автоматизованому тесті.
- Витрати на підтримку – чим швидше змінюється додаток тим дорожче стає його підтримка.
- Великі витрати на розробку – написання автоматизованих тестів вимагає значних вкладень – як фінансових так і часових. Мануальне тестування не вимагає такої довготривалої підготовки
- Вартість інструменту для автоматизації – комерційні тести автоматизації вимагають значних фінансових вкладень. Тоді як безкоштовні аналоги часто мають обмежений функціонал.
- Пропуск незначних помилок – якщо авто-тест не запрограмований на пошук певних помилок – він їх буде пропускати, тому більшість незначних помилок залишається непоміченою.

У випадку з розробкою інструменту тестування продуктивності ПЗ ключовими моментами є:

- Відсутність персоналу для підтримки
- Простота в експлуатації
- Постійні репорти, а також аналіз

Згідно з поставленими критеріями автоматизація ПЗ є необхідною.

1.3.2 Об’єкт автоматизації

Зазвичай, автоматизацію використовують для наступних складових багато-модульних систем:

1. Важкодоступні місця в системі (серверні процеси, логування системи, запис в базу даних)
2. Часто використовувана функціональність, ризики від помилок в якій

досить високі.

3. Рутинні операції, такі як перебори даних (форми з великою кількістю введених полів, рутинні POST запити)
4. Валідаційні повідомлення (автоматизувати заповнення полів некоректними даними і перевірку на появу тієї чи іншої валідації)
5. Довгі end-to-end сценарії
6. Перевірка даних, що вимагають точних математичних розрахунків
7. Перевірка правильності пошуку даних

Для більш ефективного використання автоматизації тестування краще розробити окремі тест кейси, що перевірятимуть:

- Базові операції створення / читання / зміни / видалення сутностей (так звані CRUD операції - Create / Read / Update / Delete).
    - Приклад: створення, видалення, перегляд і зміна даних про користувача.
- Типові сценарії використання додатка, або окремі дії.
    - Приклад: користувач заходить на поштовий сайт, гортає листи, переглядає нові, пише і відправляє лист, виходить з сайту. Це так званий end-to-end сценарій, який перевіряє сукупність дій.
- Інтерфейси, файли та інші моменти, незручні для тестування вручну.
    - Приклад: система створює певний xml файл, структуру якого необхідно перевірити.

1.3.3 Спосіб автоматизації

Для вибору способу автоматизації потрібно звернути увагу наскільки добре інструмент для автоматизації розпізнає елементи управління в вашому додатку. Якщо елементи не розпізнаються варто пошукати плагін, або відповідний модуль. Якщо такого немає - від інструменту краще відмовитися.

Потрібно звернути увагу на те скільки часу потрібно на підтримку коду написаному за допомогою обраного інструменту. Для цього варто написати

простий сніпет (приклад коду) який виконує простий тестовий сценарій та перевірити час на іншу подібну реалізацію. Якщо для відновлення працездатності сценарію потрібно перезаписати код цілком, то інструмент не оптимальний, так як реальні сценарії набагато складніше.

І останній момент на який потрібно звернути увагу - наскільки зручний інструмент для написання нових сценаріїв. Скільки потрібно на це часу, наскільки можна структурувати код (підтримка ООП), наскільки код простий, наскільки зручне середовище розробки для рефакторингу (переробки коду).

1.3.4 Три рівня автоматизації тестування

Умовно, тестований додаток можна розбити на 3 рівні:

- Unit Tests Layer
- Functional Tests Layer (Non-UI)
- GUI Tests Layer

Для забезпечення кращої якості продукту, рекомендується автоматизувати всі 3 рівня. Розглянемо більш детально стратегію автоматизації тестування на основі трирівневої моделі:

1. Рівень модульного тестування (Unit Test layer). Під автоматизованими тестами на цьому рівні розуміються Компонентні або Модульні тести написані розробниками. Тестувальникам ніхто не забороняє писати такі тести, які будуть перевіряти код, звичайно ж, якщо їх кваліфікація дозволяє це. Наявність подібних тестів на ранніх стадіях проекту, а також постійне їх поповнення новими тестами вбереже проект від багатьох серйозних проблем.
2. Рівень функціонального тестування (Functional Test Layer non-ui). Як правило не всю бізнес логіку програми можна протестувати через GUI рівень. Це може бути особливістю реалізації, яка ховає бізнес логіку від користувачів. Саме з цієї причини за домовленістю з розробниками, для команди тестування може бути реалізований доступ безпосередньо до

функціонального шару, що дає можливість тестувати безпосередньо бізнес логіку додатка, минаючи призначений для користувача інтерфейс.

3. Рівень тестування через призначений для користувача інтерфейс (GUI Test Layer). На даному рівні є можливість тестувати не тільки інтерфейс користувача, але також і функціональність, виконуючи операції бізнес логіки програми. Дане покриття найкраще відображає стан системи та критичні помилки.

1.3.5 Архітектура автоматичних тестів

Для зручності накладення автоматизованих тестів, на вже наявні тест кейси, структура тестових сценаріїв повинна бути аналогічна структурі тестового випадку.

Отримуємо правило, що кожен тест сценарій повинен мати:

- Precondition
- Steps (Test)
- Post Condition

Перелічимо основні функції сценарію:

- Precondition
    - Ініціалізація додатки (наприклад, відкриття головної сторінки, вхід під тестовим користувачем, перехід в необхідну частину програми та підведення системи до стану придатного для тестування)
    - Ініціалізація тестових даних
- Steps
    - Безпосереднє проведення тесту
    - Занесення даних про результат тесту, з обов'язковим збереженням причин провалу і кроків, за якими проходив тест
- Post Condition
    - Видалення, створених в процесі виконання скрипта,

непотрібних тестових даних

- Коректне завершення роботи програми

Рекомендується також створити загальну бібліотеку по обробці помилок і виняткових ситуацій. наприклад:

- PreConditionException
- TestCaseException
- PostConditionException

1.3.6 Стратегія використання автоматизованих тестів

Щоб автоматизація тестування скоротила час на тестування ПЗ, варто реалізувати наступне:

1. Написанням тестів повинні займатися фахівці з автоматизованого тестування (Software Automation Testers). Після написання, тести передаються команді ручного тестування, яка вже здійснює їх щоденний запуск і аналіз результатів. Тим самим автоматизовані тести також проходять тестування, і в результаті збільшується їх надійність і життєздатність.
2. Написані і налагоджені тести також можуть передаватися команді розробки, для налагодження нових версій.
3. Команді розробки рекомендується здійснювати щоденну збірку, з прогоном всіх написаних тестів на всіх рівнях автоматизації тестування. І тільки після того, як нова версія відповідає критеріям якості, здійснювати установку нової версії на тестову платформу.

Написання і підхід до автоматизації тестування залежить від процесу розробки програми, розбиту на фази:

- Inception phase - вибір інструменту автоматизації, в залежності від якого вирішується чи будуть використовуватися вже готові напрацювання (фреймворки) або ж все буде написано "з нуля".
- Elaboration phase - написання тестів на основну архітектуру (в подальшому ці тести будуть використовуватися для прийому білда -

Build Verification Tests)

- Construction phase - більш детальна автоматизація: критична функціональність, перевірка регресій, end-to-end сценарії
- Transition phase - підготовка тестів до передачі замовнику (якщо це потрібно)

1.4 User Experience

Важливим (проте не тяжким у реалізації) є UX та оптимізація клієнтського інтерфейсу. Перш за все необхідно оглянути базову теорію розробки за методологією UX, далі необхідно провести дослід існуючих інтерфейсів та відповідно до розробленої архітектури, винести функціонал взаємодії з клієнтом у UX-friendly категорії. У даному розділі, будуть розглянуті основні аспекти роботи з UX та дизайну. Подальша розробка GUI продовжиться у розділі з розробкою архітектури програмного продукту.

Дизайн взаємодії з користувачем (UX, UXD, UED чи XD) це кроки спрямовані для підвищення задоволеності користувача за рахунок юзабіліті, доступності та отримання насолоди при роботі з продуктом. [22]

Дизайн містить традиційну взаємодію людина-комп'ютер (HCI) і поширює його методом аналізу всіх аспектів суб’єкту дизайну, як це сприймає користувач. [23]

1.4.1 Візуальний дизайн

Візуальний дизайн - широко відомий як графічний дизайн відображає естетику або вид сприйняття / відчуття зовнішнього інтерфейсу любого функціоналу програмного продукту для користувача. Ціллю візуального дизайну є використання базових засобів сприйняття (колір, форма) для передачі необхідної інформації користувачу. Основні принципи психології та зору людини дають когнітивну точку зору на те, як створювати ефективну комунікацію з користувачем. [24]

1.4.2 Інформаційна архітектура

Інформаційна архітектура - це наука про організацію елементів інтерфейсу для досягнення максимального комфорту користувача. Інформаційним об’єктом у даному контексті є інформація отримана від знань (досвід) так і від даних. Об'єкти відрізняються від сайту до сайту. Те ж саме стосується і метаданих: унікальна інформація про дані та інформацію.

Структуризація, організація і маркування

Структуризація зменшує об’єкти до їх основних будівельних блоків, і потім пов'язуючи їх між собою. Організація включає в себе угруповання цих одиниць унікальним і осмисленим способом. Маркування – позначення певних даних для спрощення контекстного пошуку.

Виявлення та управління

У разі якщо користувачі не можуть знайти потрібну інформацію без поглибленого пошуку та аналізу – інформаційна архітектури є невдалою. Навігацію має бути максимально чітка та проста для сприйняття.

1.4.3 Проектування взаємодії

Існує багато критеріїв до розуміння проектування взаємодії і чи це може задовільнити клієнта. Побудова юзабіліті вимагає детального проектування взаємодії для відігравання центральної ролі у сприянні даних представлених клієнту. Високий попит на покращений досвід взаємодії кінцевих користувачів зробили проектування взаємодії критичним в осмисленні дизайну, який відповідає призначеним для користувача очікуванням, і стандартам останніх шаблонів для користувача інтерфейсів і компонентів. Працюючи, дизайнери взаємодії розглядають декілька понять: [25]

1. Виокремлення шаблонів взаємодії, що найкраще підходять в контексті
2. Використання користувальницьких вимог зібраних в ході

дослідження користувача проектах

3. Функції та інформація, які важливі для користувача
4. Інтерфейсну поведінку, як перетягування, виділення і дії мишею
5. Ефективно повідомляючи сильні сторони системи
6. Створення інтерфейсу інтуїтивно за допомогою побудови можливостей
7. Підтримка узгодженості у всій системі.

На даний момент, роль дизайнерів взаємодії змістилася від того, щоб бути сфокусованим на точному визначенні деталей призначеного для користувача інтерфейсу і передача їх інженерам до ситуації, де зараз дизайнери мають більше свободи в проектуванні контекстних інтерфейсів, які засновані на допомогу задовольнити власні потреби. [26] З цієї причини, дизайн взаємодії з користувачем еволюціонував в міждисциплінарну галузь дизайну, яка включає між-дисципліні технічні аспекти від проектування анімаційного дизайну і мультиплікації до програмування.

1.4.4 Юзабіліті

Юзабіліті є частиною більш широкого терміну "користувальницький досвід" і стосується простоти доступу та / або використання продукту чи веб-сайту. Дизайн непридатний або непридатний для себе; його особливості, а також контекст користувача (те, що користувач хоче зробити з ним), визначає його рівень зручності. [27]

1.4.5 Доступність

Доступність системи – це її легкість досяжності, використання і розуміння. З точки зору дизайну взаємодії з користувачем, вона може також бути пов'язана із загально зрозумілими інформацією і функцією дизайну. Вона сприяє скороченню кривої навченості асоційованої з системою. Доступність часто пов'язана з простотою використання об’єкту дизайну для людей з обмеженими здібностями\можливостями.

1.5 Аналітичний аналіз системи автоматизованого стрес тестування складних систем

До найбільш відомих засобів, здатним виконувати вищенаведені види тестування, відносять [28]:

1. IBM Rational Performance Tester;
2. HP Load Runner;
3. Apache JMeter;
4. SilkPerformer;
5. WAPT.

1.5.1 RPT

Розглянемо кожне з них докладніше: IBM Rational Performance Tester (RPT) [29] - інструмент тестування продуктивності, розробляється підрозділом IBM Rational Software, використовується для верифікації продуктивності і масштабованості веб-ресурсів заснованих на клієнт-серверній архітектурі додатків. Виконання такого роду тестів дозволяє визначити вузькі місця у взаємодії компонентів програми та знайти способи їх можливого усунення.

До основних можливостей і особливостей RPT відносяться:

1. підтримка широкого спектру додатків і протоколів, та- ких як HTTP / HTTPS, SAP, Siebel, SIP, TCP Socket, Citrix, Windows Sockets і ін .;
2. можливість швидкого створення тестів без написання коду і без вимоги навичок програмування;
3. наявність настроюється графічного інтерфейсу користувача;
4. наявність повнофункціонального редактора тестів з деревоподібною структурою, що забезпечує як загальне високорівневе, так і докладне уявлення тестів;
5. автоматизація зміни даних тестування, а також можливість вставляти користувальницький код мовою програмування Java для гнучкого налаштування тесту;

6. автоматизація ідентифікації динамічних характеристик сервера і управління ними;
7. гнучке моделювання різного навантаження;
8. можливість тестування в середовищах ОС Windows, Linux і середовищах, побудованих на основі технологій мейнфреймів;
9. формування звітів в режимі реального часу, що дозволяє негайно виявляти проблеми продуктивності та відтворити розмітку веб-сторінок в форматі HTML в ході тестування;
10. збір і інтеграція даних про серверні ресурсах з даними про продуктивність додатків, які отримуються в режимі реального часу;
11. підтримка операційних систем сімейства Windows і Linux завдяки платформі Eclipse, написаної мовою програмування високого рівня Java;
12. низькі вимоги до обсягу пам'яті і обчислювальної потужності процесора, що забезпечує проведення масштабного тестування в багатокористувацькому середовищі при обмежених апаратних ресурсах;
13. наявність Performance Testing SDK (інструментарій розробника ПЗ), що дозволяє розробнику програмного забезпечення використовувати функції RPT.

1.5.2 HP LoadRunner

HP LoadRunner [30] - це програмний комплекс для автоматизованого тестування навантаження, що розробляється компанією HP. На сьогоднішній день HP LoadRunner займає 77% на ринку автоматизованого тестування навантаження. HP LoadRunner дозволяє емулювати одночасне звернення сотень і тисяч користувачів, щоб вивчити поведінку програми при реальному навантаженню. При цьому збирається інформація про ключових компонентах інфраструктури (веб-сервери, сервери баз даних і т.д.). Результат може бути детально проаналізований, щоб зрозуміти причини конкретної поведінки

системи. Програма має відповідні набори інструментів для проведення тестування. Також до складу HP LoadRunner входить набір інструментів для роботи з використанням різних протоколів з додатком (віддалено, через проксісервер і т.п.).

До основних можливостей і особливостей HP LoadRunner відносяться:

1. підтримка безлічі додатків і протоколів, таких як: HTTP / HTTPS / HTML, TrueClient, SAP, LDAP, Flex, .NET, GUI (UFT), Citrix, RDP, ODBC, COM / DCOM, IMAP, MAPI, POP3, SMTP , FTP, Windows Sockets, CORBA, RMI, JMS, Jacada, Siebel та ін .;
2. можливість інтеграції в інтегровані середовища розробки Microsoft Visual Studio і Eclipse;
3. підтримка операційних систем сімейства Windows і Linux;
4. наявність гнучкого графічного інтерфейсу користувача;
5. підтримка моніторингу широкого спектру параметрів продуктивності;
6. наявність багатих засобів візуалізації, побудови звітів та аналізу даних, зібраних в ході тестування продуктивності;
7. наявність великого набору інструментів написання тестових сценаріїв (скриптів), включаючи можливість запису дій користувача при складанні сценаріїв тестування з можливістю подальшого редагування записаного сценарію;
8. можливість детального аналізу і контролю процесу виконання сценарію з можливістю перегляду всіх проміжних характеристик в режимі реального часу;
9. підтримка локалізації російською мовою;
10. наявність можливості інтеграції з хмарою;
11. управління процесом тестування через мобільний додаток для операційної системи Google Android.

1.5.3 Apache JMeter

Apache JMeter [31] - інструмент для проведення тестування

продуктивності, що є вільним програмним забезпеченням з відкритими вихідним кодом, написаним мовою програмування високого рівня Java і розробляються Apache Software Foundation. Спочатку Apache JMeter розроблявся як засіб тестування продуктивності веб-додатків, згодом список підтримки різних технологій розширився. Він може використовуватися для моделювання великого навантаження на сервері, групі серверів, мережі, щоб протестувати їх максимальну навантажувальну здатність або розкласти загальну продуктивність під різними типами завантаження. Apache JMeter здатний виконувати тестування продуктивності на динамічних і статичних ресурсах.

До основних можливостей і особливостей Apache JMeter відносяться:

1. підтримка широкого переліку додатків / протоколів, таких як: HTTP / HTTPS, SOAP, FTP, JDBC, LDAP, JMS, SMTP, POP3, IMAP, MongoDB, TCP і ін .;
2. підтримка власних команд і сценаріїв тестування
3. підтримка мультіплатформенності завдяки 100% -ної розробці мовою програмування високого рівня Java;
4. підтримка аналізу завантаження мережі;
5. повна підтримка багатопоточності, що дозволяє виконувати паралельну обробку запитів, функцій та інших можливих операцій;
6. наявність ясного і зрозумілого графічного інтерфейсу користувача, що дозволяє швидко створювати і коректувати виконуються тести, а також усувати різні несправності;
7. підтримка кешування і оффлайнові дослідження / відтворення результатів тестування;
8. замінні генератори навантаження, що дає необмежені можливості тестування;
9. підтримка паралельного збору статистичної інформації одночасно декількома модулями аналізу;
10. підтримка різних доповнень для візуалізації та аналізу даних, що

дозволяє гнучко налаштувати систему «під себе»;

11. можливість зміни тесту прямо в процесі роботи, а також можливість гнучкого маніпулювання даними;
12. підтримка великого спектру мов скриптування сценарію тестового випробування (BSF- і JSR223 - сумісні мови).

1.5.4 SilkPerformer

SilkPerformer [32] - інструмент для автоматизованого тестування навантаження різного рівня складності. Інструмент створений компанією Borland, яка в даний час придбана британською компанією Micro Focus. SilkPerformer є потужним і в той же час простим у використанні інструментом навантаження і стрес-тестування корпоративного класу. Візуальний сценарій і можливість тестування декількох прикладних середовищ з тисячами віртуальних користувачів дозволяють ретельно перевірити корпоративні додатки на надійність, продуктивність і масштабованість, перш ніж вони будуть розгорнуті, незалежно від їх розміру і складності. Потужний аналіз першопричин і інструменти управління звітністю допомагають ізолювати проблеми і швидко приймати рішення.

До основних можливостей і особливостей SilkPerformer відносяться:

1. підтримка безлічі додатків / протоколів, таких як: HTTP / HTTPS, TTP (S) / HTML, SNMP, SOAP, FTP, LDAP, MAPI, IMAP, DCOM, SMTP, POP3, SSL, .NET, Citrix, TCP / IP , UDP, Jacada і ін .;
2. створення тестів і прогонів програми за допомогою інтуїтивно зрозумілого інтерфейсу SilkPerformer або з використанням інтегрованого середовища Eclipse;
3. можливість різної візуалізації даних, побудови звітів та аналізу результатів тестування;
4. тестування в широкому діапазоні корпоративних середовищ за допомогою гнучких, спільно використовуваних, мультипротокольних типів віртуальних користувачів;

5. аналіз навантажувальних тестувань в реальному часі, щоб уникнути появи недійсних результатів тестування, що вимагають тривалих повторних прогонів прецедентів тестування;
6. наявність можливості роботи в розподіленої конфігурації, з метою створення більшого навантаження;
7. легко настроюються тести з випадковими даними користувачів - без необхідності написання хоча б одного рядка коду;
8. наявність можливості гнучкого налаштування тестового сценарію з використанням вбудованої мови програмування BDL (Benchmark Description Language).

1.5.5 WAPT

WAPT [33] є інструментом для навантажувального і стресового тестування веб-сайтів і будь-яких додатків, подібних за будовою і архітектурі. Розробляється в новосибірської компанії SoftLogica LLC. Продукт створює навантаження на тестований сайт шляхом емуляції типової активності сотень або навіть тисяч користувачів, що працюють з сайтом одночасно. В процесі тесту сайт поводиться таким же чином, як і при реальному навантаженню того ж рівня, видаючи ті ж параметри продуктивності.

До основних можливостей і особливостей WAPT відносяться:

1. підтримка широкого переліку додатків / протоколів, таких як: HTTP / HTTPS, SOAP, JSON, GWT, .NET, Silverlight, SNMP, WNI і ін .;
2. підтримка локалізації на російську мову;
3. підтримка моніторингу параметрів продуктивності сервера і баз даних;
4. наявність графічного інтерфейсу користувача;
5. можливість роботи в розподіленої конфігурації;
6. наявність коштів побудови звітів і багаті можливості
7. візуалізації та аналізу параметрів продуктивності;
8. можливість перегляду характеристик продуктивності

9. в режимі реального часу;
10. наявність спеціального майстра створення тестового сценарію,
11. що дозволяє спростити період адаптації користувача;
12. підтримка запису дій користувача при складанні сценаріїв тестування з подальшим редагуванням, а також можливість написання скриптів тестового сценарію мовою програмування JavaScript.

1.6 Висновок

Згідно з представленими теоретичними даними, необхідно виокремити наступні ключові поняття (представлені у виді дерева цілей):

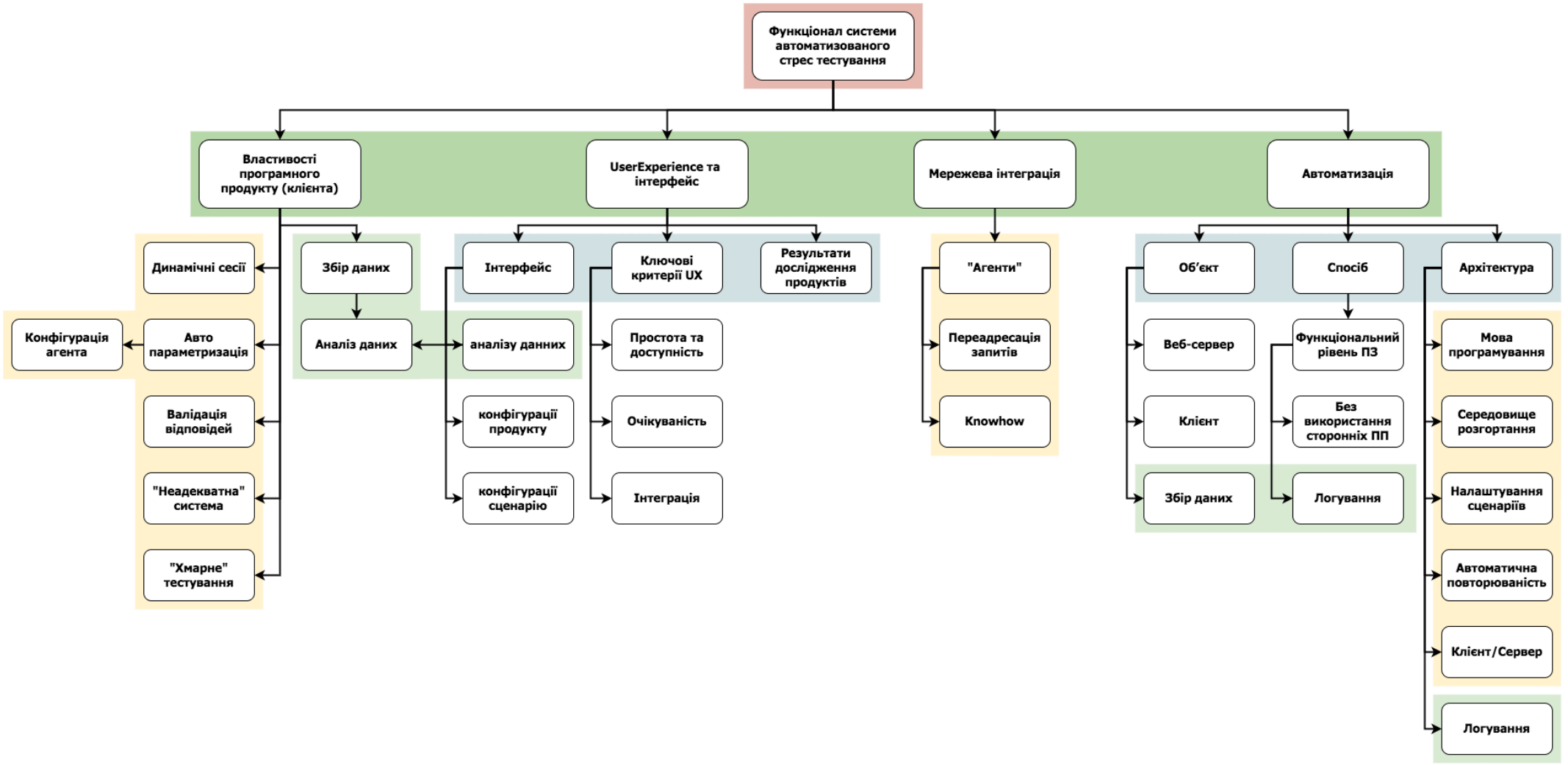

Рисунок 1.1 Ключові поняття необхідні для реалізації автоматизованого ПЗ для тестування стресостійкості веб-ресурсів

Згідно ключових понять представлених цілей, під час розробки плану дипломної роботи, а надалі й архітектури самого ПЗ необхідно врахувати такі поняття як “Властивості ПП”, UX та інтерфейс, інтеграція (хмарний сервіс) та автоматизація продукту. Розглянемо детальніше кожен з пунктів.

Властивостями ПП можна назвати підсистеми, або модулі ПЗ, які будуть виконувати необхідні для ПП тестування навантаження:

- Динамічні сесії дозволять організувати унікальних клієнтів
- Параметризація це по-суті реалізація унікальної поведінки клієнта
    - Конфігурація дозволить керувати процесом автоматичної параметризації
- Валідація відповідей є ключовим моментом для аналізу успішності проходження тестування
- Нестандартна система важливий момент при тестуванні, проте, у розрізі даної роботи, нестандартною системою стане віртуальна машина, з обмеженими розрахунковими потужностями.
- “Хмарне тестування” є модулем підключення клієнта до мережі агентів.

UX частиною роботи стане аналіз існуючих інтерфейсів на предмет задоволенням представлених критеріїв клієнтського інтерфейсу.

Мережевою інтеграцією стане налаштування LAN (Local Area Network) мережі, між низкою віртуальних машин зі штучно зміненими адресами, для симуляції роботи хмарного сервісу. Це важливий момент для зниження собівартості тестування складних систем.

Автоматизація проекту буде проходи за допомогою вбудованих у Java REST API систем та UNIX кронів (детальніше даний процес розглянутий у наступних розділах.

На даному етап діаграма демонструє шаблонну роботу автоматизованого на продукту:

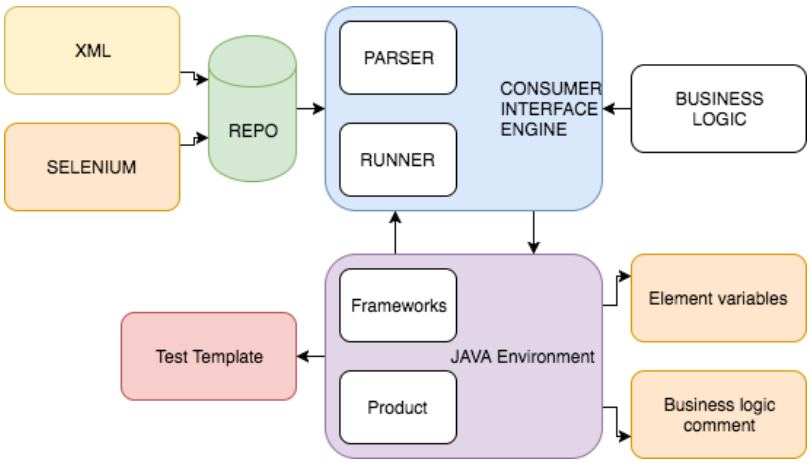

Рисунок 1.2 Система автоматизованого тестування

2 ФОРМУЛЮВАННЯ ТЕХНІЧНОГО ЗАВДАННЯ

У даному розділі формулюється чітка мета та структура виконання дипломної роботи.

2.1 Тема та мета задачі

Темою дипломної роботи є дослідження та аналіз систем тестування програмного забезпечення збудованого та клієнт-серверній архітектурі.

Метою роботи є реалізація автоматизованого програмного продукту, що проводить різностороннє стрес-тестування REST-like програмних продуктів та аналізує отримані результати.

2.2 Вимоги до проекту

2.2.1 Функціональні вимоги до проекту

1. Багатопоточні запити по суб’єкту тестування.
2. Унікальність запитів.
3. Параметризація запитів.
4. Валідація відповідей серверу (з використанням вбудованого аналізатору-парсеру).
5. Мережева інтеграція (модуль, що дозволяє інтегрувати хмарне середовище).
    a. Система агентів-клієнт (агент – інстанс серверу, що працює з командами надісланими з клієнта)
6. Збір та аналіз результатів:
    a. Збереження у локальну базу даних.
    b. Виведення результатів тесту.
    c. Виведення рекомендація.
    d. Хронологічний аналіз уже проведених тестів для цієї ж системи.
7. Автоматизація проекту:
    a. Автоматизоване розгортання проекту.

b. Періодичне виконання тестів без втручання оператору.

2.2.2 Нефункціональні вимоги

1. UX
    a. Простота в експлуатації
    b. Простота у розгортанні проекту
    c. Коротка документація та справка
    d. Доступність конфігурації
    e. Error-safe архітектура
    f. Зрозумілість представлених результатів
2. GUI
    a. Welcome інтерфейс
    b. Інтерфейс конфігурації
    c. Інтерфейс проведення тесту
    d. Інтерфейс аналізу результатів

2.3 Вхідна та вихідна інформація

2.3.1 Вхідний потік даних

Вхідним потоком даних буде текстова відповідь серверу на запити HTTP methods.

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
Content-Length: 88
Content-Type: text/html
Connection: Closed

<html>
<body>
```

```
<h1>Hello, World!</h1>
</body>
</html>
```

Приклад 2.1. Приклад відповіді серверу

2.3.2 Вихідний потік даних

Вихідні потоки даних для користувача:

1. Лог-файли перебігу тесту (текстова інформація);
2. Графік отриманих результатів тесту;
3. Таблиця у базі даних;
4. Текстовий аналіз отриманих результатів.

Вихідним потоком типу клієнт-сервер буде запит до серверу.

```
GET /hello.html HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.google.com
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

Приклад 2.2. Приклад запиту до серверу

2.3.3 Аналіз даних

Аналіз даних буде проводити клієнтська частина у два етапи:

- Первинний аналіз отриманих результатів - аналіз коду відповіді, виділення часу відповіді, відповідність отриманої відповіді до запиту надісланого на сервер;
- Вторинний аналіз проводитиме порівняння теперішнього тесту з попередніми тестами у хронологічному порядку (якщо такі є).

2.4 Етапи проектування

I. Розробка архітектури проекту:
   a. Побудова системної моделі проекту (зв’язок між компонентами

проекту, що включає усі сторонні програмні продукти, агентів та суб’єктів тестування);

b. Детальний розбір кожного з внутрішніх компонентів системи (побудова діаграми класів)

c. Виокремлення необхідних конфігурацій для зовнішніх модулів проекту (спосіб підключення кожного з модулів до проекту, та розбір взаємодії між модулем та проектом по типу response-request)

II. Розробка інтерфейсу:

a. Дослідження уже існуючих інтерфейсів;

b. Дизайн власного інтерфейсу.

III. Математична модель та схеми аналізу отриманих результатів:

a. Виокремлення необхідних критеріїв аналізу та метрик;

b. Створення математичних моделей для аналізу критеріїв згідно обраних метрик;

c. Реалізація механізму оцінки якості проведених тестів та надання рекомендацій;

d. Створення способу репрезентації отриманих результатів.

IV. Розробка моделі агента для створення хмарної мережі маршрутизації запитів клієнта.

V. Автоматизація проекту:

a. Дослідження системи автоматичного розгортування проекту та необхідних сторонніх модулів.

b. Створення системи автоматичної та автономної роботи проекту.

2.5 Структура системи

Структура системи загалом описуватиме найвищий рівень деталізації дипломного проекту зі включенням сторонніх модулів.

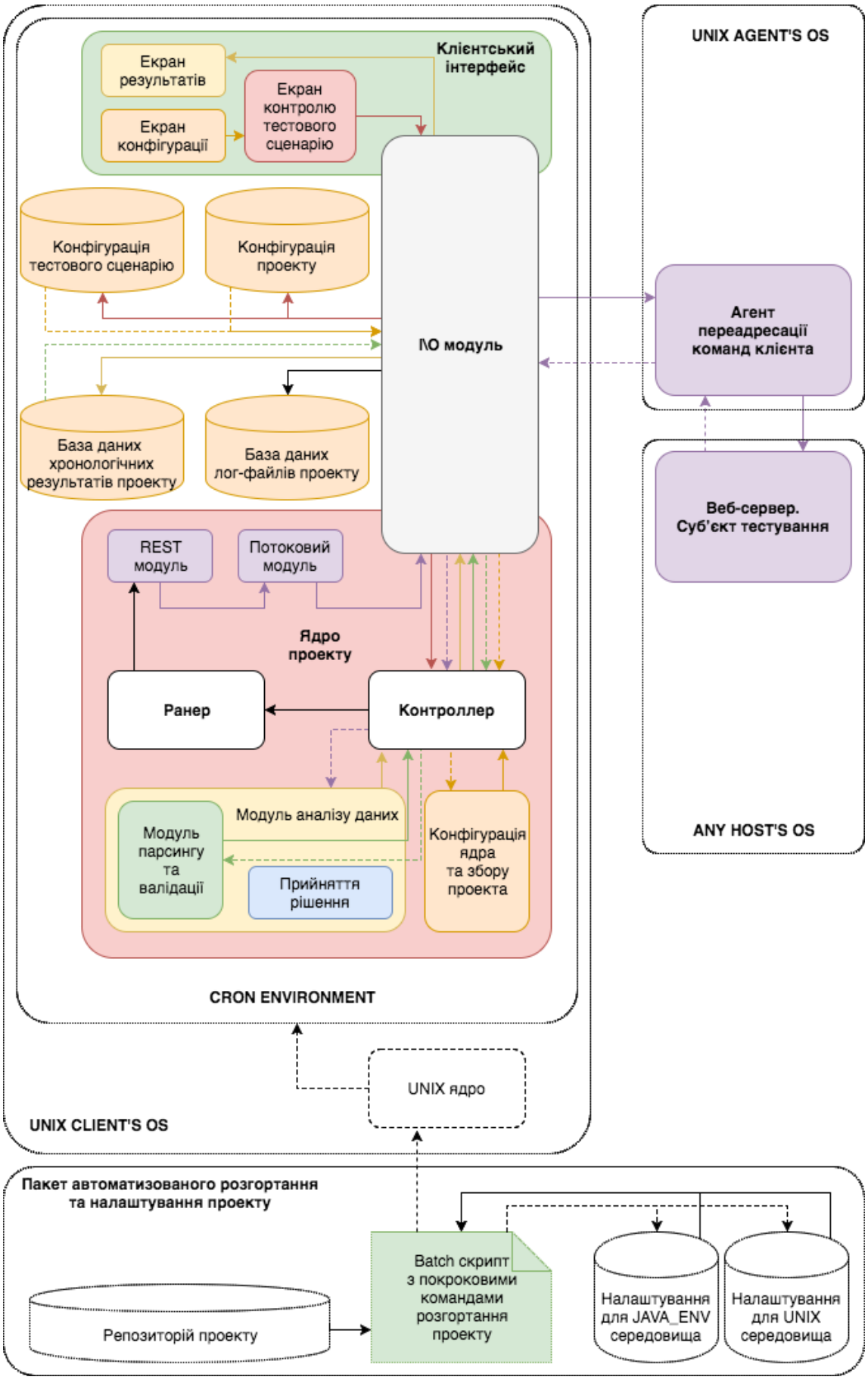

Рисунок 2.1 Спрощена структурна схема проекту

2.6 Сторонні засоби та застосунки

У ході розробки даного проекту будуть використані наступні сторонні засоби:

- Операційна система збудована на базі UNIX ядра. А саме Ubuntu Server 16.04.3 LTS [34]
- Демон для запуску скриптів у UNIX середовщі GNU Crontab daemon [35]
- Середовище розробника Java JDK 9 [36]
- Серверне середовище запуску JAVA Server JRE 9 [36]
- Фреймворк для роботи з REST архітектурою Jersey [37]
- Standalone сервер агенту Jetty [38]
- База даних для збереження уже виконаних тестів проекту MySQL [39]

2.7 Висновок

Згідно результатів даного розділу, очевидно, що проект охоплює велику низку сучасних технологій (особливо, веб-архітектурних проектів), що зумовлює низку питань щодо інтеграції усіх проектів в одному середовищі.

Варто виокремити середовище розробки на базі UNIX системи як саме зручне для створення як клієнтського застосунку так і аналогу “хмарної” мережі виконання запитів (так званих агентів).

Важливою частиною роботи стануть UX дослідження (що не є характерним етапом розробки серверного застосунку) а також математичний аналіз отриманих результатів з системою прийняття рішень (однією з задач роботи є виведення рекомендацій клієнту, щодо отриманих результатів тестувань). Побудова даної системи вимагатиме математичних розрахунків та алгоритмізації системи аналізу даних.

Середовище збереження інформації було розбите на два різних під-модулі:

1. ООП база даних (або NoSQL), що оптимізована для роботи з великою кількістю однотипної текстової інформації, та
2. Реляційна база даних для спрощення роботи з системою аналізу отриманих даних.

Завершення проекту буде створення простої системи розгортання проекту, з допомогою якої, сам проект можна буде використовувати на невеликих підприємствах без залучення спеціалістів.

3 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМНОГО ПРОДУКТУ ТА ІНТЕРФЕЙСУ

У даному розділі проводиться ґрунтовний аналіз шляхів реалізації поставлених завдань та мети проекту, вибір найкращих (з точки зору продуктивності, простоти реалізації та підтримки) варіантів.

Відповідно до сформованих етапів проектування, у даному розділі розпочнеться детальний огляд по формування кожного з компонентів проекту (побудова діаграми класів, схем послідовностей та передачі даних).

Також, буде проведено дослідження клієнтського інтерфейсу з використанням метода експертних оцінок для визначення необхідного функціоналу з прямим доступом до інтерфейсу, функціоналу, що буде застосований через поліморфний інтерфейс та функціоналу який буде повністю інкапсульований.

3.1 Архітектура ядра проекту

Основою програмного продукту є інтерфейс взаємодії з виділеним веб-сервером з використанням стандартної бібліотеки JAVA NIO [40].

Звичайна схема взаємодії клієнта сервера виглядає наступним чином:

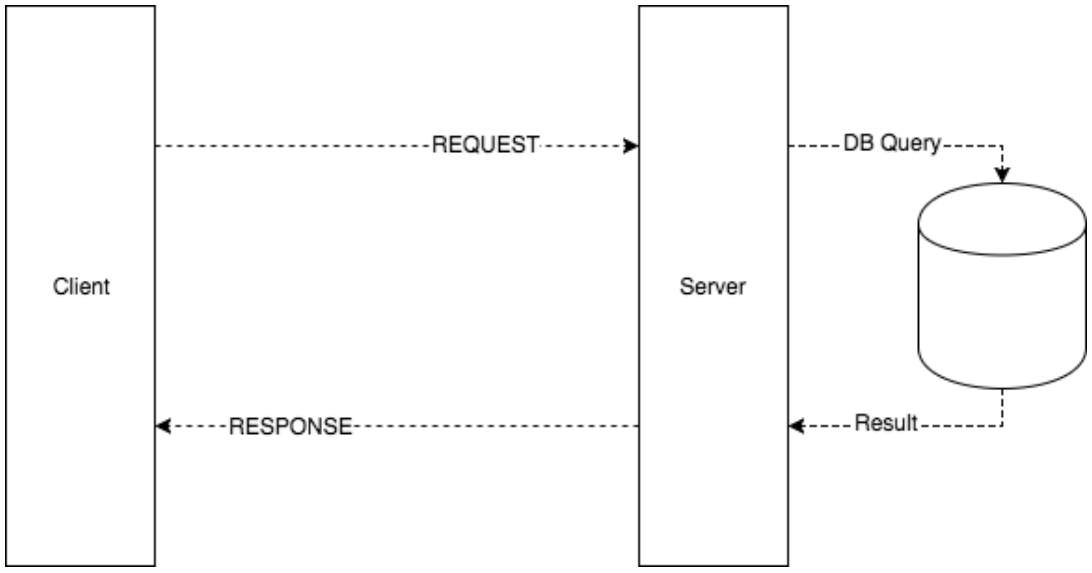

Рисунок 3.1 Однопотокова схема взаємодії клієнта-сервера

Клієнт (браузера, або додаток який працює по HTTP протоколу) надсилає запит спеціального формату на відомий інтернет адрес (IP адрес або доменне ім’я), очікуючи певну відповідь (у залежності від сформованого запиту). У залежності від конфігурації серверу, на серверній частині відбувається запит до внутрішньої АПІ та\або бази даних, формується відповідь та відправляється клієнту.

Вузькою частиною даної схеми завжди є час обробки серверу, так як при масштабування схеми, клієнт може надсилати безліч запитів серверу, і сервер, для їх обробки, буде використовувати обмежені ресурси. Тоді як клієнт (або клієнти) в теорії можуть мати необмежені ресурси.

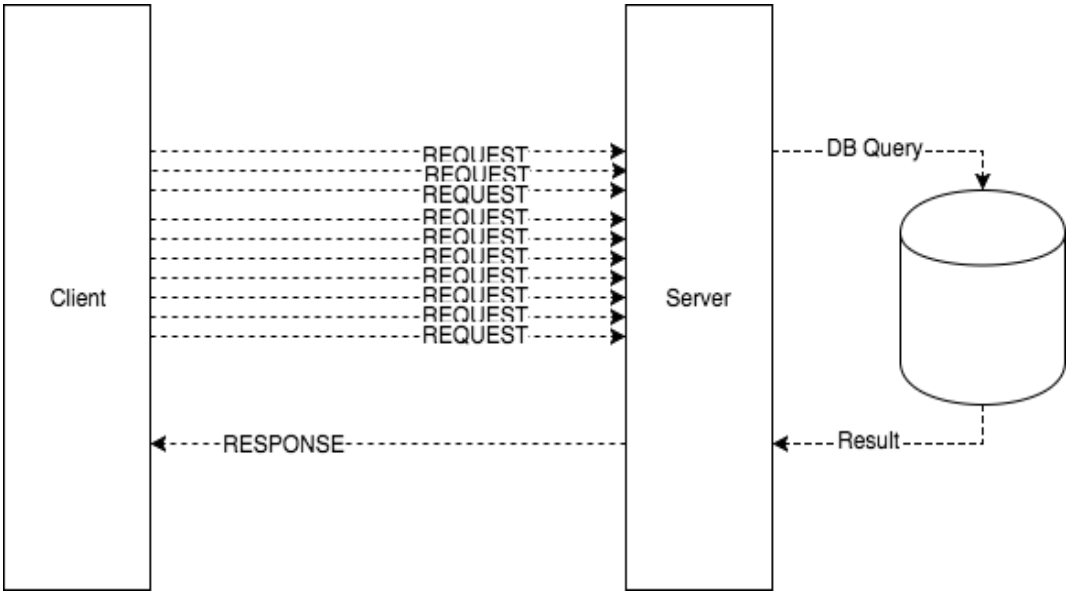

Рисунок 3.2 Багатопотокова клієнт-серверна взаємодія

Для боротьби з даним явищем (як навмисним (DDOS атаки) так і випадковими (витік памя’ті) була створена система кешування: одному клієнта для одного й того ж запиту формується на клієнтській стороні уже готова відповідь, яка перевіряється браузером, та віддається не через запит до серверу, а з локального сховища. А також, були розроблені системи “відсікання” клієнтів, які за певний період часу створюють значну кількість запитів.

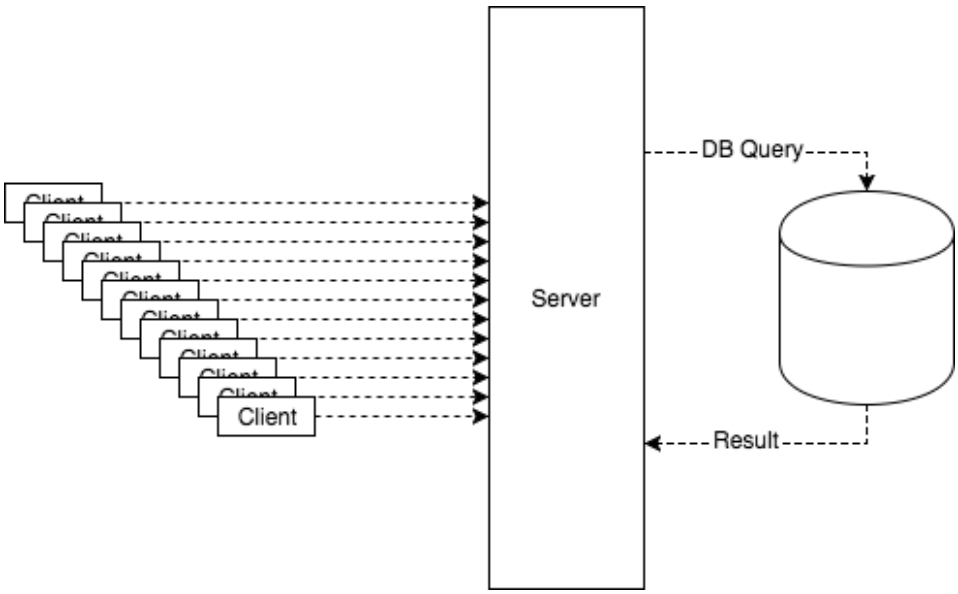

Рисунок 3.3 Мультипоточні запити від декількох клієнтів до серверу

На схемі вище зображена структура взаємодії декількох клієнтів (у даному випадку, мається на увазі унікальних клієнтів, у яких різний цифровий підпис, що у свою чергу, є поєднанням локального адресу, підпису пристрою використаного для доступу до серверу, а також наявності локальних ідентифікаторів(cookies, sessions, local db)) з сервером. У даній моделі, сервер не має змоги кешування, і тому має обробити усі запити одночасно. Дана модель найнебезпечніша для відкритих систем (мається на увазі доступ), тому саме ця модель буде основою механізму тестування веб серверу. Єдиною зміною, буде створення системи клієнт – багато клієнтів – сервер:

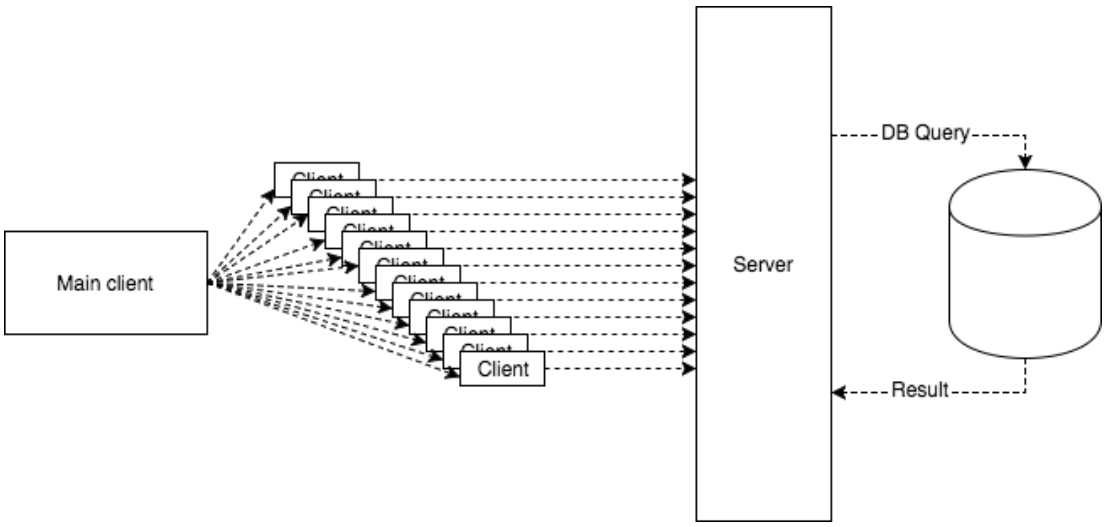

Рисунок 3.4 Клієнт - багато клієнтів – сервер

З допомогою даної моделі буде використана мінімальна кількість ресурсів самого клієнта (системи клієнт – клієнти) та максимально ефективно протестована сервера частина.

Відповідно до цього, програмний продукт складається з двох частина: самого клієнта, задачею якого є створення запитів та розподіленням їх у системі, а також з “імітатора” клієнта, задачею якого переадресація запитів та прийом відповідей.

Отже:

- Клієнт – це прикладний програмний продукт з GUI, що створює набір запитів до виділеного серверу.
- Агент – це демон (прикладна програма без GUI), що приймає налаштування від клієнта, маршрутизує запити до серверу, отримує відповідь та передає відповідь назад клієнту.

Клієнт може працювати як напряму, так і з агентами. Агент являється налаштовуваним маршрутизатором і не приймає ніякої участі у функціональних процесах системи.

3.1.1 Спрощена модель “ядра”

Створення запитів до серверу відбувається за принципом створення потоків, кожен з яких паралельно один до одного проходить ітерації з запитами, тобто, 2 потоки по 3 ітерації кожен, у сумі проведуть 6 запитів, при цьому, паралельно будуть виконуватися 2 з’єднання за одиницю часу.

Структурно, схема створення потоку виглядати наступним чином:

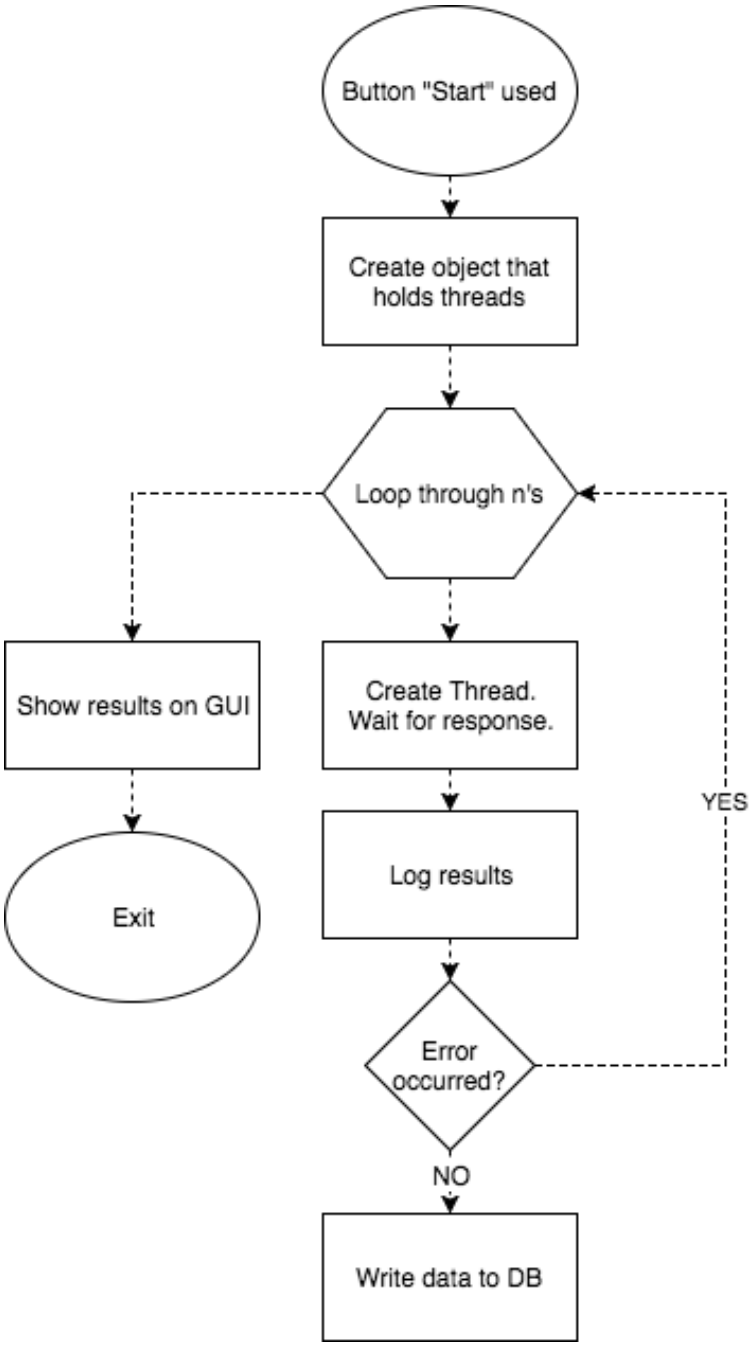

Рисунок 3.5 Алгоритм створення потоків

Алгоритм описує процес запуску програми з заданими параметрами, створення необхідних об’єктів та запуск у циклі кожного з потоків. Далі у паралельному середовищі відбувається запис необхідних даних у базу даних (за умови відсутності внутрішніх помилок) та очікування завершення усіх потоків. Якщо усі потоки завершили своє виконання – вивести результат на графічний інтерфейс.

Робота потоку відображена наступною схемою:

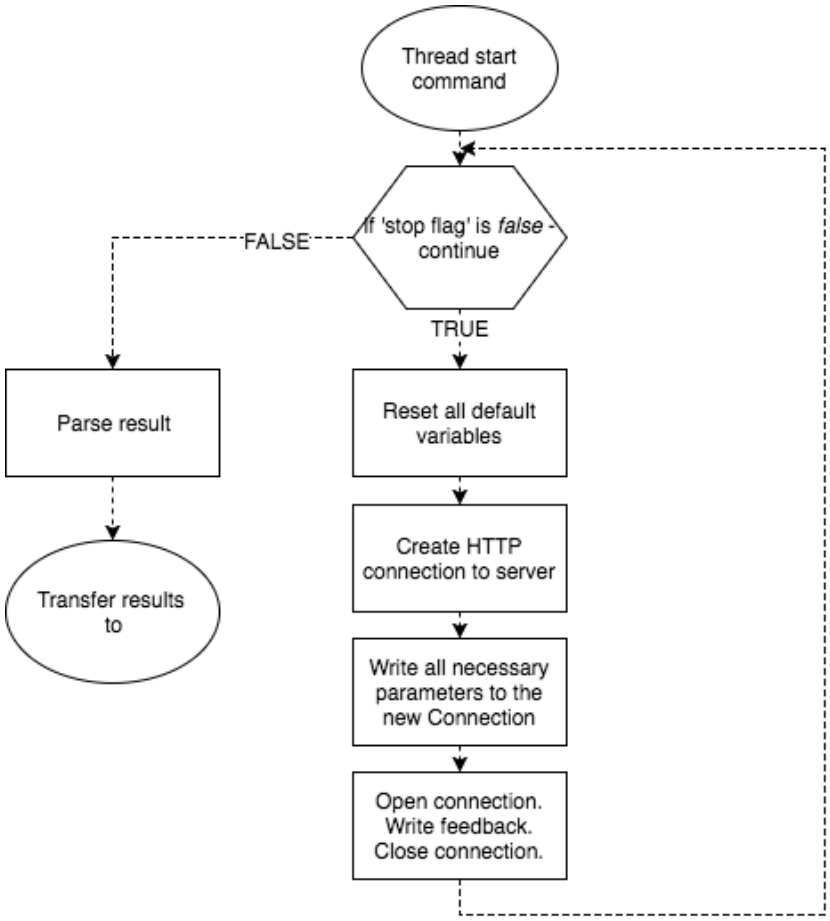

Рисунок 3.6 Робота потоку

Робота окремого потоку не відрізняється від роботи любого HTTP клієнта (за різниці роботи у циклі). Одразу після запуску потоку, створюється цикл з ітерацій, у кожній з яких створюється з’єднання з сервером, записуються параметри та відкривається сесія. Далі результат оброблюється (валідується) та передається виконавчому коду.

3.1.2 Формування запиту та відповіді

Перед реалізацією інтерфейсу роботи з сервером, необхідно скласти запит та дізнатися відповідь серверу, щоб відповідно правильно “спілкуватися” по HTTP протоколу.

Запит завжди складається з заголовків (мета інформація про запит) та з тіла запиту.

Відповідь аналогічно складається з заголовків та тіла.

Запит або відповідь не обов’язково мають тіло – пакет може складатися лише з заголовків.

Відповідно до прикладів запиту та відповіді представлених вище, необхідні параметри (які задаються клієнтом) це:

- URL адреса ресурсу, що працює по HTTP протоколу;
- HTTP Метод запиту

Інформація, що цікавить нас з відповіді серверу:

- Код відповіді
- Тіло відповіді
- Статус сесії

HTTP Методи

- OPTIONS

Повертає методи HTTP, які підтримуються сервером. Цей метод може служити для визначення можливостей веб-сервера.

- GET

Запрошує вміст вказаного ресурсу. Запитаний ресурс може приймати параметри (наприклад, пошукова система може приймати як параметр шуканий рядок). Вони передаються в рядку URI (наприклад: http://www.example.net/resource?param1=value1&param2=value2). Згідно зі стандартом HTTP, запити типу GET вважаються ідемпотентними — багатократне повторення одного і того ж запиту GET повинне приводити до однакових результатів (за умови, що сам ресурс не змінився за час між

запитами). Це дозволяє кешувати відповіді на запити GET. Якщо назва ресурсу не вказана (у URI наявні лише схема та доменне ім'я), то веб-сервер повертає індекс директорії веб-сервера.

- HEAD

Аналогічний методу GET, за винятком того, що у відповіді сервера відсутнє тіло. Це корисно для витягання мета-інформації, заданої в заголовках відповіді, без пересилання всього вмісту. Зокрема, клієнт чи проксі, перевіривши заголовок Last-Modified: (останній час модифікації), таким чином може переконатися, що сторінка на сервері не змінилася від часу попереднього запиту.

- POST

Передає призначені для користувача дані (наприклад, з HTML-форми) заданому ресурсу. Наприклад, в блогах відвідувачі зазвичай можуть вводити свої коментарі до записів в HTML-форму, після чого вони передаються серверу методом POST, і він поміщає їх на сторінку. При цьому передані дані (у прикладі з блогами — текст коментаря) включаються в тіло запиту. На відміну від методу GET, метод POST не вважається ідемпотентним, тобто багатократне повторення одних і тих же запитів POST може повертати різні результати (наприклад, після кожного відправлення коментаря з'являтиметься одна копія цього коментаря).

- PUT

Завантажує вказаний ресурс на сервер.

- PATCH

Завантажує певну частину ресурсу на сервер.

- DELETE

Видаляє вказаний ресурс.

- TRACE

Повертає отриманий запит так, що клієнт може побачити, що проміжні сервери додають або змінюють в запиті.

- CONNECT

Для використання разом з проксі-серверами, які можуть динамічно перемикатися в тунельний режим SSL.

З даних методів найчастіше використовуються GET та POST, проте, для повноцінності тестування будуть реалізовані ще такі методи як DELETE, PUT, HEAD та OPTIONS. Методи PATCH та TRACE використовуються у спеціалізованих ресурсах, тому не мають функціональної цінності для стрес тестування.

Коди відповідей серверу

1xx - інформаційний - запит прийнятий, продовжуй процес

2xx - успіх - дія була успішно передана, зрозуміла, та прийнята

3xx - перенаправлення - наступні дії мають бути успішно виконані для реалізації запиту

4xx - помилка клієнта - запит містить синтаксичні помилки або не може бути виконаний

5xx - помилка серверу - сервер не зміг виконати правильно сформований запит

Найтиповіші статуси:

200 OK — запит виконаний успішно;

301 Moved Permanently — ресурс переміщено

403 Forbidden — доступ до запитаного ресурсу заборонений;

404 Not Found — запитаний ресурс не знайдений.

503 Service Unavailable — сервіс недоступний

3.1.3 Технічна реалізація

Для технічної реалізації даного функціоналу досить створити один клас, задачею якого бути потоком (тобто, мати можливість паралельного виконання зі своїми ж копіями), отримувати параметри передані клієнтом з інтерфейсу, створювати з’єднання з сервером, створювати та закривати сесію з сервером та відправляти результати назад клієнту (контролер).

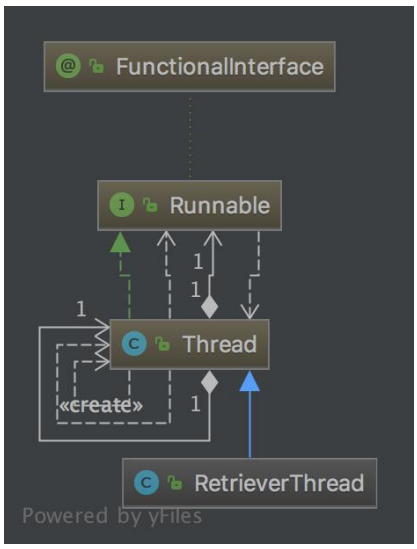

Рисунок 3.7 Діаграма зв’язку RetrieverThread з бібліотекою Java SE8

Вхідними параметрами даного класу є:

- Адрес серверу
- Назви параметрів запиту
- Атрибути параметрів запиту
- HTTP Метод.

Вихідні параметри:

- Час виконання запиту (зв’язок з сервером)
- Статус потоку (необхідний для регулювання контролером робочих потоків)
- Назва потоку та номер ітерації (що формує унікальний номер кожного запиту, необхідний для аналізу даних)
- Відповідь серверу (що включає в себе заголовки пакету та тіло)

3.2 UX дослідження клієнтського інтерфейсу

Для максимально ефективного графічного інтерфейсу необхідно провести спрощене дослідження. Для проведення даного дослідження була сформована мінімальна вибірка по фокус групі та проведений метод експертних оцінок згідно якого визначяться найважливіші критерії графічного інтерфейсу користувача.

Для проведення потрібно сформувати критерії:

- Інтерфейс налаштувань програми (мається на увазі, багатоцільовий інтерфейс з гнучкого налаштування кожного аспекту програми);
- Налаштування сценарію (тут – можливість налаштовувати поведінку програми як клієнта)
- Загальна структурна чіткість та зрозумілість інтерфейсу (простота в експлуатації)
- Повноцінний дизайн інтерфейсу (тут загальна завершеність інтерфейсу, як то значки, іконки і тд)
- Візуалізація (ступінь виведення ходу виконання програми, графіки, логи)

3.2.1 Матриця індивідуальних переваг експерта

Фокус група складається з 9 осіб. Серед них розробник, професійний дизайнер, керівник проекту (сфера інформаційних технологій), розробник веб-застосунків, адміністратор розподілених хмарних систем, офісний адміністратор та три офісних працівника, які не працюють з серверними ресурсами безпосередньо.

Кожен з експертів пройшов індивідуальне опитування згідно наступної матриці індивідуальних переваг:

Таблиця 3.1 Приклад заповнення матриці індивідуальних переваг експертом №1

	Global Settings	**Scenarious Settings**	**Comprehension**	**Design**	**Visualization**	**Sum**	**R**
Global Settings		0	0	1	1	2	3
Scenarious Settings	1		0.5	1	1	3.5	1
Comprehension	1	0.5		1	0	2.5	2
Design	0	0	0		1	1	4.5
Visualization	0	0	1	0		1	4.5

Дана матриця оцінює переваги експерта між критеріями, рахує загальну перевагу кожного критерію, та проводить рангування критеріїв.

Матриці інших експертів можна знайти у додатках.

3.2.2 Матриця групових переваг експертів

Матриця групових переваг виглядає наступним чином:

Таблиця 3.2 Матриця групових переваг з критеріями узгодження

Experts	Procedures				
	Global Settings	**Scenarious Settings**	**Comprehension**	**Design**	**Visualization**
	w_1	w_2	w_3	w_4	w_5
1	3	1	2	4.5	4.5
2	4	2.5	1	5	2.5
3	4	2	1	5	3
4	4	3	1	4	2
5	5	2	1	4	3
6	4	2	1	5	3
7	4	3	1	5	2
8	4	1.5	1.5	5	3
9	4	2.5	1	5	2.5
ΣR_{wj}	36	19.5	10.5	42.5	25.5
R_{group}	4.000	2.167	1.167	4.722	2.833
R'_{group}	4.000	2.000	1.000	5.000	3.000
D_i	0.25	0.4375	0.125	0.1944	0.5625
σ_i	0.5	0.661	0.354	0.441	0.75
v_i, %	12.5	30.528	30.305	9.338	26.471

Опис полів даної таблиці:

- 1-9 – ранги по критеріям кожного з експертів
- ΣR_{wj} – сума рангів (необхідна для подальших розрахунків)
- R_{group} – групова думка (середнє арифметичне, що показує узагальнену думку про певний критерій)

- R'_{group} – округлена групова думка
- D_i – дисперсія по критеріям (показує “розкиданість” оцінок)
- σ_i – середнє арифметичне відхилення (вказує на віддаленість критерію від середнього значення)
- v_i, – коефіцієнт варіації (вказує на ступінь “довіри” до оцінки експертів). Так як коефіцієнт варіації < 33% - оцінки експертів скоординовані.

Наступним етап дослідження є розрахунок коефіцієнту кореляції рангу Кендала (*3.1*). У даному опитуванні, коефіцієнт Кендала вкаже на узгодженість думок експертів, та чи потрібно проводити повторне групове опитування. Згідно попередніх результатів за коефіцієнтом варіації, коефіцієнт Кендала має лежати у межах 0.7-1 одиниць.

$$W = \frac{12S}{m^2(n^3 - n) - m\sum_{j=1}^{m} T_j} \tag{3.1}$$

, де

S – узагальнена дисперсія (*3.2*):

$$T_j = \sum(t_i^3 - t_i), \tag{3.2}$$

T_j – кількість однакових рангів у *j*-у рядку, та зафіксованому j-у експерту.

Згідно розрахунків, $W = 0.8051$, що впадає у рамки 0.7-1 та означає, що думка експертів скоординована

Наступним етапом буде розрахування ваговий критеріїв, згідно яких буде відданий пріоритет або функціональному інтерфейсом наповненню, або UX, або дизайну.

Таблиця 3.3 Вагові коефіцієнти

	Global Settings	**Scenarios Settings**	**Comprehension**	**Design**	**Visualization**	**Sum**
R'_{group}	4.000	2.000	1.000	5.000	3.000	
C	0.400	0.800	1.000	0.200	0.600	3.000
w_j	0.133	0.267	0.333	0.067	0.200	1.000

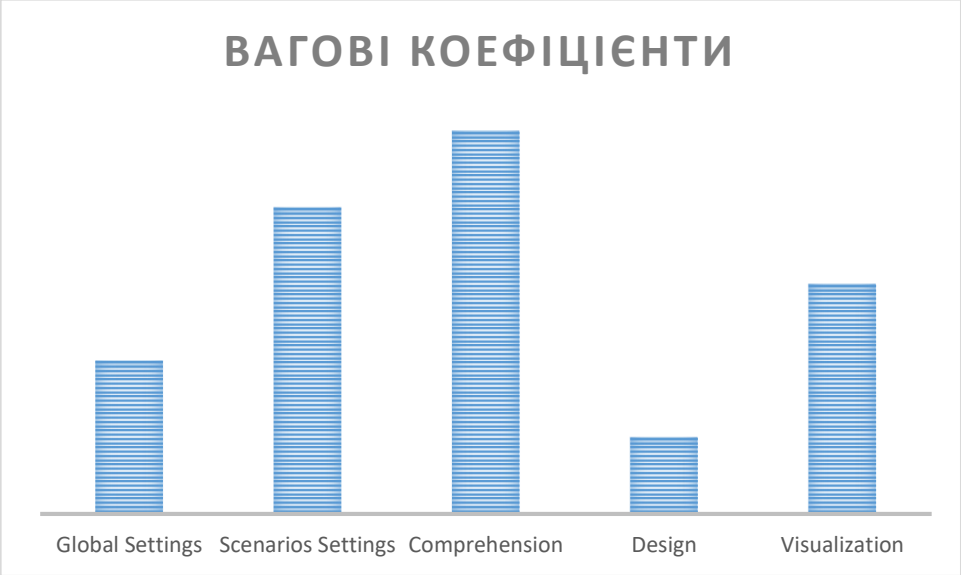

Рисунок 3.8 Графічне відображення вагових коефіцієнтів

Відповідно до вагових коефіцієнтів, перевагу у дизайні варто віддати досвіду користувача (простота та зрозумілість інтерфейсу) та налаштуванням тестового сценарії (що саме має тестувати).

3.3 Дизайн інтерфейсу

У ході розробки інтерфейсу були використані базові принципи проектування:

- Інтерфейс має бути максимально простим (не зважаючи на ціль програми)
- Для кожного з пунктів існує справка («хмарки» з підказками)
- Фокус користувача має бути зосередженим на основному вікні, з яким він працює.
- Кількість налаштувань (елементів інтерфейсу, у сенсі взаємодії з

програмною частиною) має бути достатньою (мається на увазі, винести лише основні показники які змінюються часто, тоді як інші налаштування перенести у файли конфігурації)

Відповідно до даних критеріїв, та попередніх оцінок експертів, було вирішено винести усі налаштування та груповані елементи в окремі категорії, відійти від дизайну «одне вікно – весь інтерфейс» для спрощення сприйняття клієнтського інтерфейсу.

3.3.1 Головний екран запуску

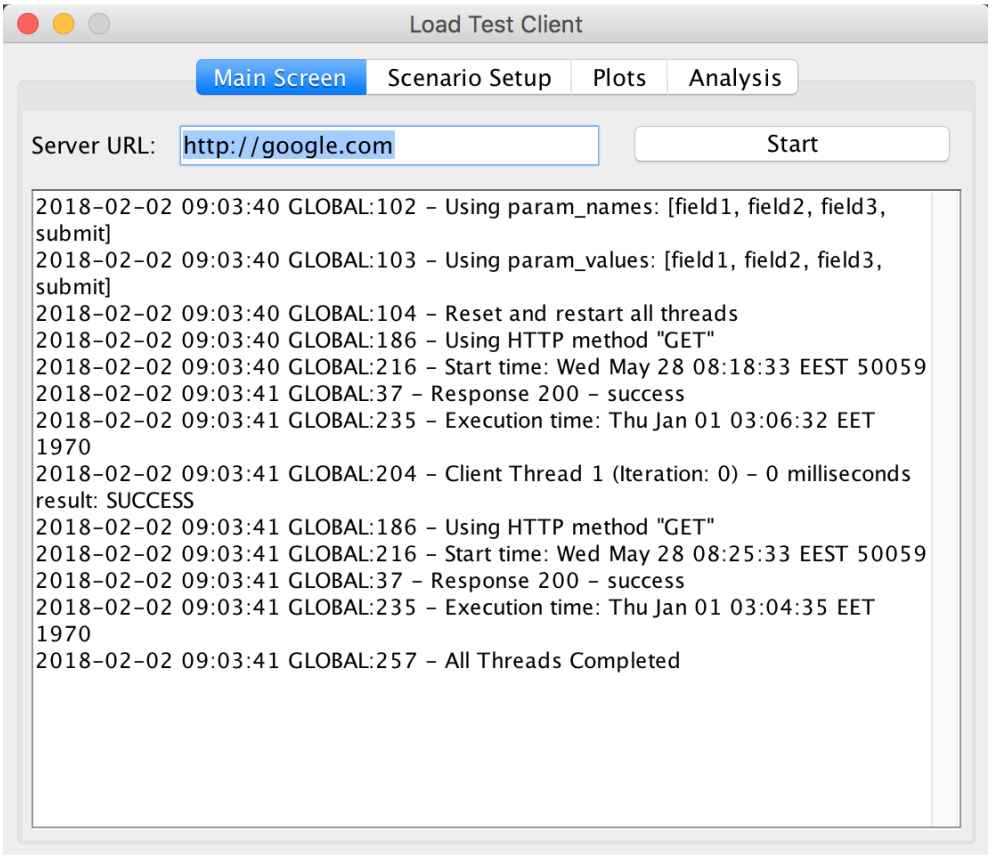

Рисунок 3.9 Головний екран програми

Головний екран програми, максимально простий для користувача, та максимально інформативний водночас. Основним налаштування для програми є адрес веб-серверу з яким має працювати програма, тому цей адрес винесений в основний екран, так як це єдине налаштування яке має бути змінено для функціонування програми.

Правіше від поля адресу розташована кнопка “Start” яка відповідно запускає програму.

Усю іншу частину інтерфейсу займає термінал виведення процесу проходження тестування (також помилки, попередження та дебаг інформація, якщо така підключена).

Даний термінал виводить максимально стислий та спрощений варіант інформації (логів), тоді як повний варіант записується у файл.

3.3.2 Екран контролю тестового сценарію

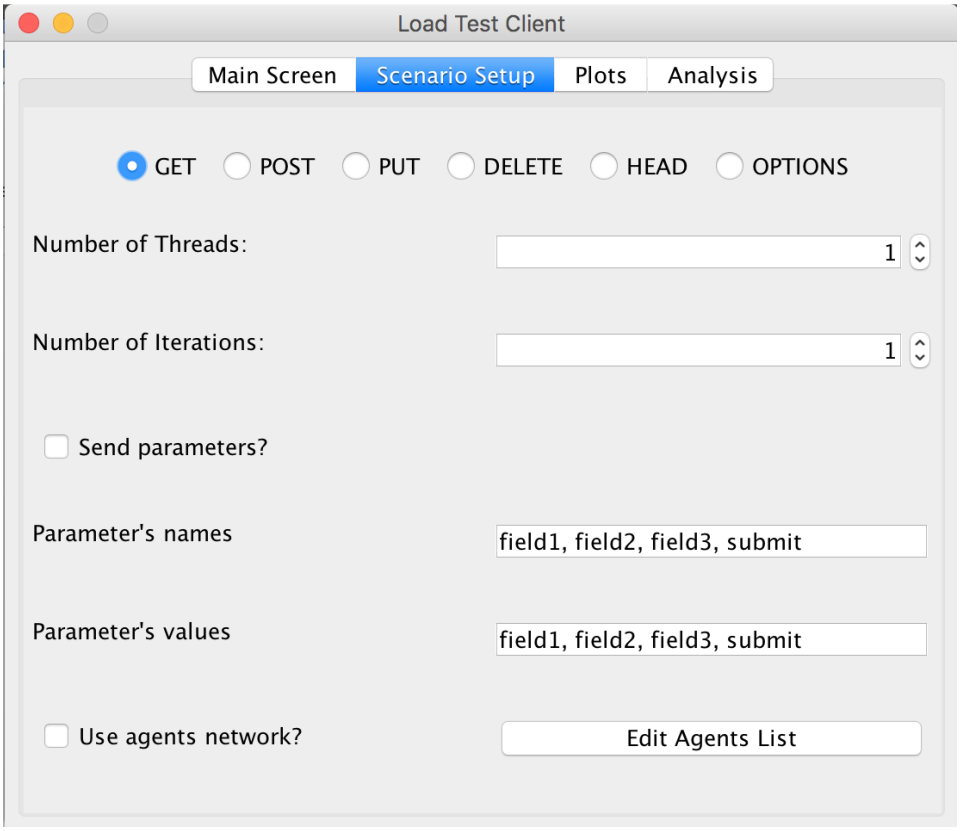

Рисунок 3.10 Екран контроль тестового сценарію

Даний екран потрібен для налаштування того, як програма буде взаємодіяти з сервером.

- Група радіо-кнопок представляє собою HTTP Методи, відповідно до яких буде сформований пакети відправлений на веб-сервер.
- Кількість потоків – кількість віртуальних клієнтів, що будуть працювати з сервером.

- Кількість ітерацій – кількість разів, які віртуальний клієнт буде надсилати запит на сервер.
- Назви веб параметрів – назви атрибутів, що прийме сервер (потрібні для сайтів з реєстрацією, для методів як оновлюють чи видаляють інформацію з серверу).
- Значення веб параметрів – мають відповідати іменам, та містять у собі команди для серверу.

3.4 Екран виведення графічної інформації

Для спрощення розуміння того, що робить програма, був створенний розділ виведення графічного представлення результатів тестування.

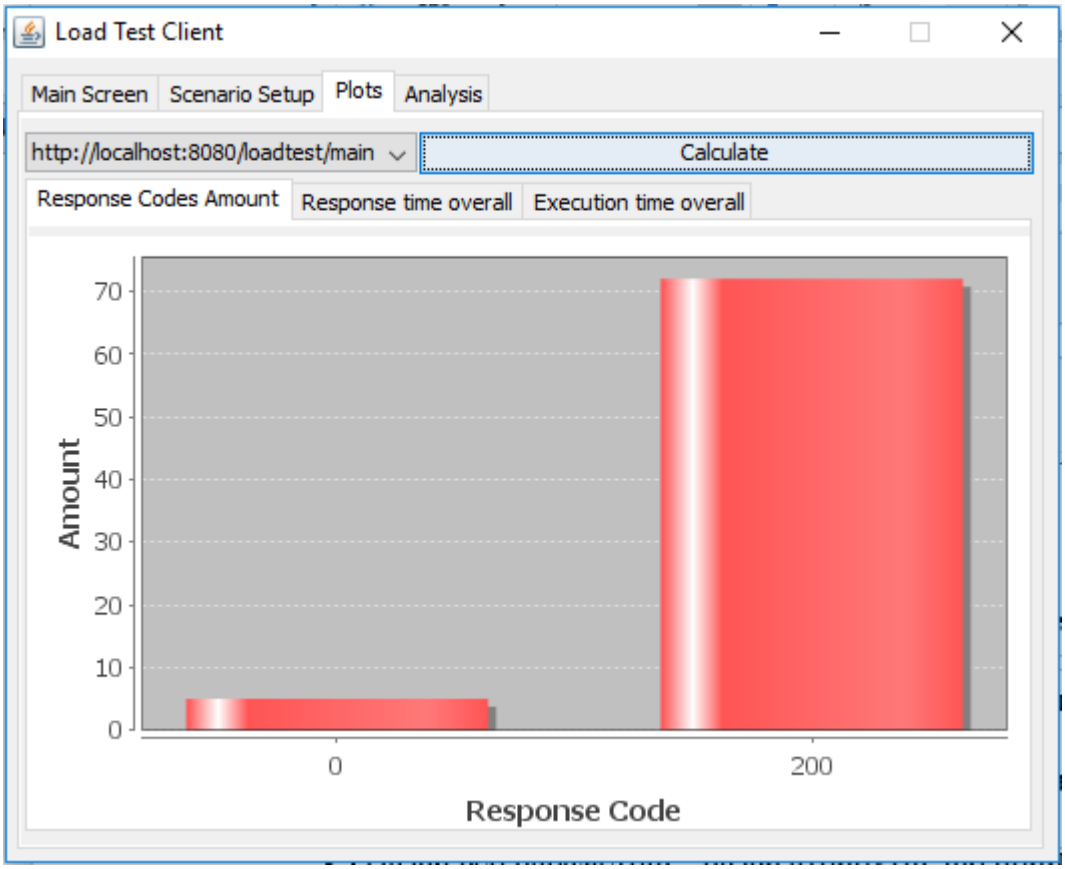

Рисунок 3.11 Панель роботи з графічними результатами тестування

Панель складається з вибору адресу сервера який є у базі (та який є предметом аналізу), кнопки “Calculate” яка розраховує та відображає графіки, вибором типу графіка а також самою областю виведення графіку.

Кожен з типів представлених графіків буде розглянуто детальніше у наступному розділі.

3.4.1 Аналіз результатів

Графіки представлені програмою не завжди інформативні, тому що сервер може почати «кешувати» інформацію, відсікати клієнтів і тд.

Відповідно до цього, було вирішено створити окрему панель для математичного аналізу отриманих результатів, та виведення рекомендацій щодо отриманих результатів.

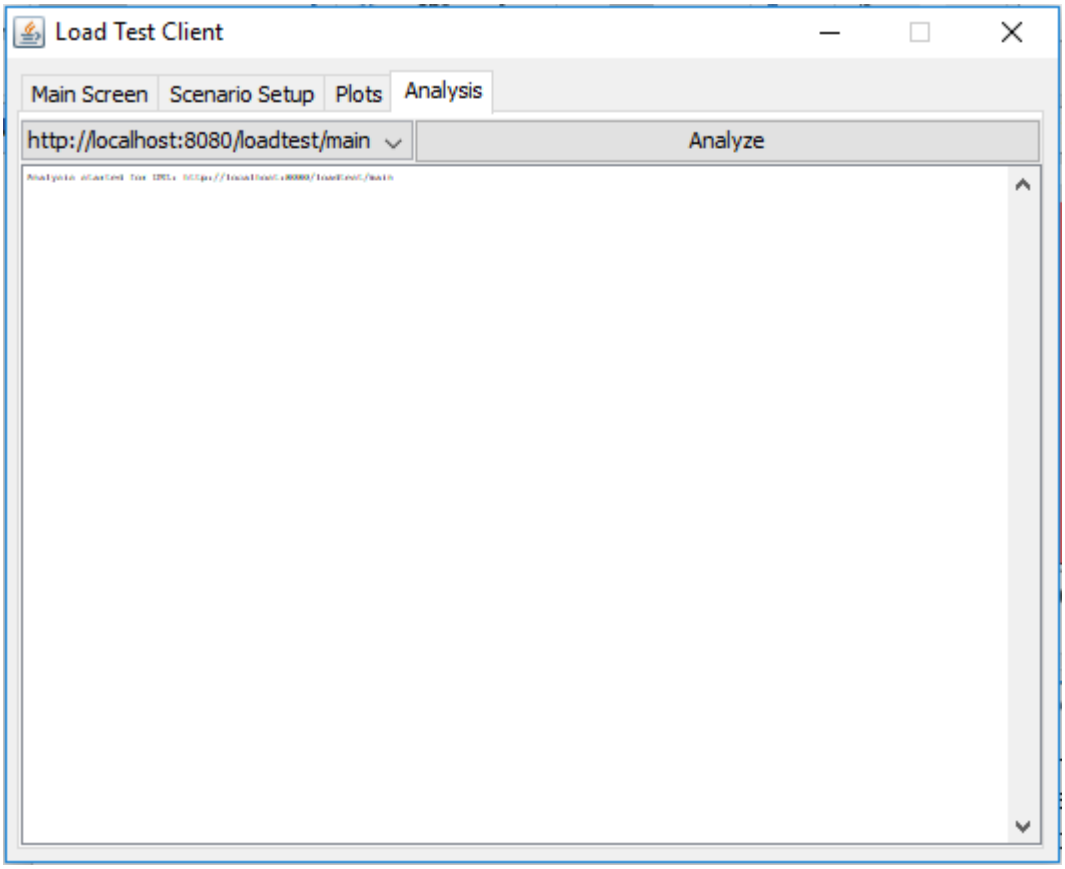

Рисунок 3.12 Інтерфейс аналізу

Інтерфейс складається з вибору об’єкта (серверу), кнопки “Analyze” та термінального вікна, що виведе рекомендації клієнту відповідно до отриманих результатів.

Детальніше механізм аналізу описаний у наступному розділі.

3.5 Висновок

Створення «ядра» проекту та графічного інтерфейсу є ключовими моментами у написанні клієнт-орієнтованої програми. Клієнта завжди цікавить що дасть даний продукт та наскільки тяжко з ним працювати. Відповідно до даних вимог і було написано «ядро» проекту, для роботи з веб серверами та визначені основні вимоги від клієнта (тобто, налаштування необхідні для роботи програми).

Отримавши основні критерії для інтерфейсу, та провівши експертне опитування, був визначений пріоритет для кожного з критеріїв інтерфейсу. Відповідно до результатів фокус групи, потенційних клієнтів не цікавить дизайн програми. Основний пріоритет – це зрозумілість інтерфейсу та виведення інформації, так як продукт не націлений на спеціалістів у даній сфері, а на рядкових інженерів та навіть простих офісних працівників.

Маючи результати опитування був розроблений максимально спрощений та інформативний інтерфейс для роботи клієнта. Особливу увагу отримали інтерфейси з виведенням результатів тестування, так як не розуміючи цілей та висновків які робить програма клієнт не зможе достовірно трактувати результати виконання тестових сценаріїв, та як висновок – розуміти що відбувається з розподіленою мережею серверів.

4 СИСТЕМИ АНАЛІЗУ ТА ЗБЕРЕЖЕННЯ ІНФОРМАЦІЇ

Однім з пунктів дипломного проекту є створення системи аналізу серверу. Для створення системи аналізу даних, перш за все необхідно реалізувати систему збереження етапів проведення тестування, використаних налаштувань та результатів перебігу тестових сценаріїв, на основі яких буде проведений аналіз та виведені результати.

4.1 Формування критеріїв оцінювання

Відповідно до загально прийнятих метрик та критеріїв оцінювання веб систем (детальніше представлено у теоретичному розділі про стрес тестування) для аналізу ефективності роботи стрес систем з допомогою інструментів навантаження використовуються наступні метрики:

- Метрики віртуальних користувачів
    - Мінімальна кількість активних потоків
    - Максимальна кількість активних потоків
    - Середня кількість активних потоків
    - Запущені потоки
    - Виконані потоки
- Метрики по відповіді серверу
    - Кількість успішних відповідей до усіх запитів
    - Навантаження серверу на одиницю часу (кількість запитів за секунду)
    - Мінімальний час відповіді серверу
    - Максимальний час відповіді серверу
    - Середній час відповіді серверу
    - Відсоток успішних відповідей серверу
    - Кількість неуспішних відповідей серверу
    - Мінімальний час відповіді для неуспішної відповіді
    - Максимальний час відповіді для неуспішної відповіді

- Середній час відповіді для неуспішної відповіді
- Загальна кількість відповідей від серверу

- Тип відповіді серверу
  - Помилкова відповідь (5** коди)
  - Відсутність відповіді (сервер недоступний, або не може обробити запит)

Більшість даних метрик використовуються для внутрішніх розрахунків системи та не мають практичної цінності для спеціаліста, який не працює з тестуванням виділених систем.

Для максимальної візуалізації та ефективного відображення дійсного стану систем, були обрані лише певні метрики для проведення аналізу системи (веб-серверу) та графіки які описані далі по тексту.

4.2 Графічні критерії оцінювання

Для максимально простої та інформативної візуалізації були використані наступні графіки (та відповідно потрібні метрики).

4.2.1 Кількість успішних та неуспішних відповідей

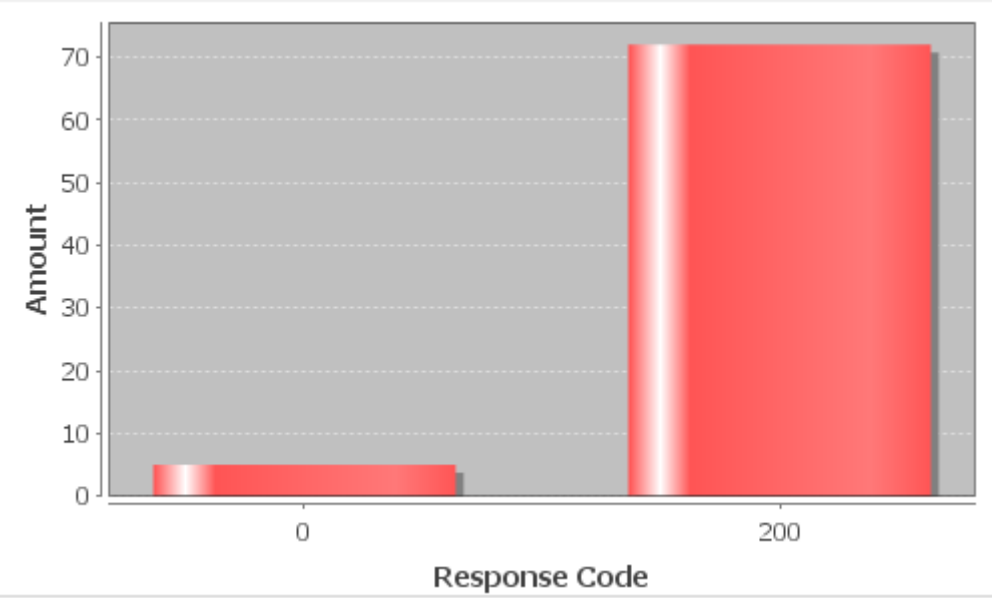

Рисунок 4.1 Графік кількості кодів відповідей

На даному графіку представлені агреговані результати усіх тестових сценаріїв по обраному серверу та коди відповідей які прийшли від даного серверу.

Для аналізу даного графіку потрібно лише знати основні критерії кодів відповідей серверу (детальніше у теоретичній частині дипломної роботи).

На малюнку представленому вище, присутні лише два коди відповідей (для спрощення прикладу): 72 успішних виконання (код 200) а також 5 неуспішних виконань (код 0). За даної метрики, варто зрозуміти причину виникнення коду 0 – якщо причиною стало хибне налаштування системи, або відсутність інтернет зв'язку – присутність даного коду не є критичною. Проте, якщо повна відсутність зв'язку з сервером є результатом періодичних збоїв у роботі серверу – варто звернутися до системних адміністраторів.

Дану дилему з кодом помилки 0 можна вирішити за допомогою проведення детального аналізу серверу у наступному розділі.

4.2.2 Час відповіді серверу відповідно до кількості запитів

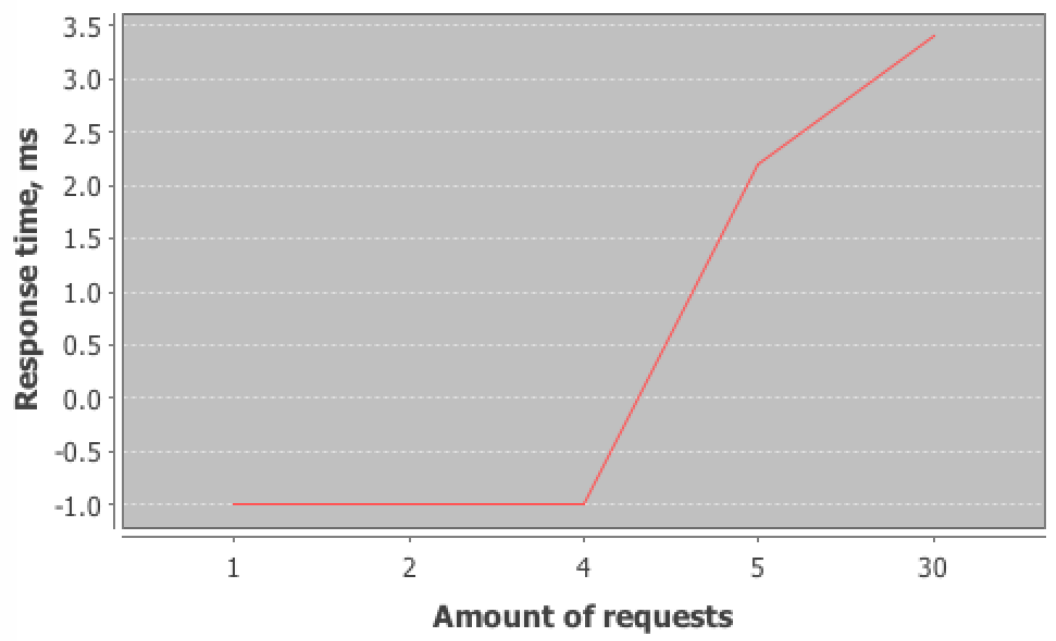

Рисунок 4.2 Відповідь серверу у залежності від кількості запитів

Графік часу відповіді до кількості запитів - найважливіша метрика роботи серверу для бізнесу (мається на увазі концепція трьох дев'яток 99.9% часу роботи серверу; дана метрика важлива для пошукових систем). Даний графік простий у представленні: агрегується середня кількість запитів за тестовий сценарій та вираховується середній час відповіді серверу.

Проте, для правильного аналізу даного графіку потрібно розуміти роботу серверу, або використовувати систему агентів (другий варіант варто застосовувати у випадку роботи зі спеціалістами, які не мають справ з розподіленими веб серверами).

Відповідно до логіки роботи серверу на HTTP протоколу, час відповіді серверу має лінійно залежати від кількості запитів за секунду. Проте, існують наступні фактори:

- Швидкість інтернету
- Робота в одній мережі
- Швидкість запитів до бази
- Локальне кешування
- Кешування на сервері

Якщо проаналізувати графік вище, до видно, що сервер гірше справляється зі спонтанними запитами невеликої кількості клієнтів, що виконують запити одночасно, ніж з одним клієнтом що виконує багато запитів одночасно.

Як результат, при великій кількості запитів, сервер починає віддавати «кешовану» інформацію, а не проводити внутрішній запит, так як запит відбувається одним і тим же клієнтом.

Даний графік являється більш інформативний після використання системи «вірутальних» клієнтів розподіленим по іншим серверам, так як об'єкт тестування не зможе віддавати кешовану версію запиту «різним» клієнтам.

4.2.3 Час роботи тестових сценаріїв

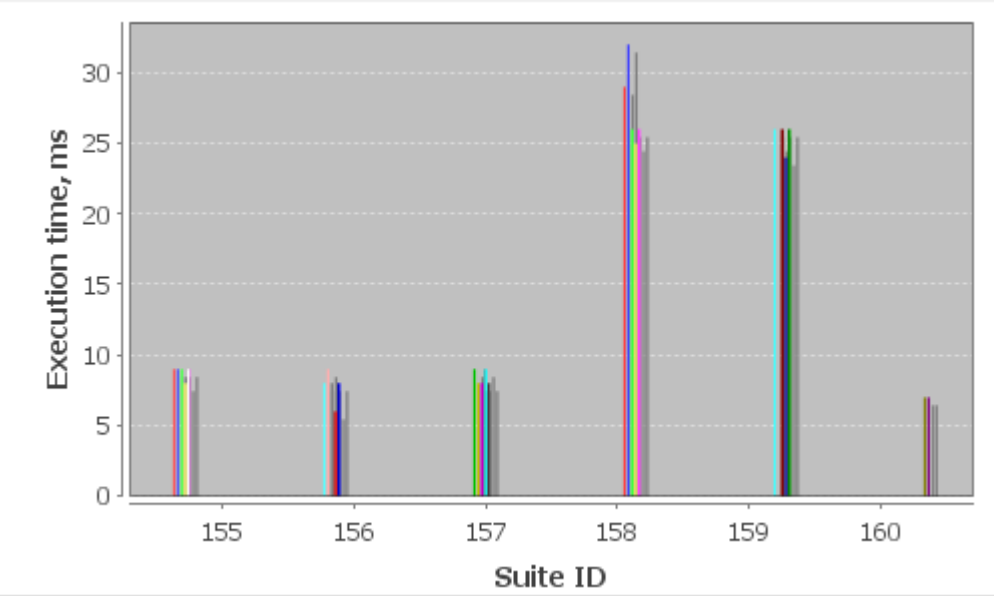

Рисунок 4.3 Час роботи тестових сценаріїв

Дана метрика візуалізує роботу серверу відповідно до хронології запуску тестових сценаріїв. Зважаючи на представлені метрики, при перебігу сценаріїв 158 та 159 сервер виконував запити значно довше (у середньому на 15 мс) ніж під час стандартного тестування.

Причиною такої роботи стало виконання збереження інформації на сервер (HTTP метод “POST” з параметрами), що вимагає передачі параметрів, збереження, та формування відповіді, на відміну від стандартного потоку формування відповіді.

4.3 Система збереження даних

Важливою складовою роботи програмного продукту є аналіз та оцінка отриманих результатів. Дана функція можлива лише за наявності структурованої системи збереження інформації.

У даній програмі існують два потоки збереження:

- Логування (файлова не структурована система збереження подібна до об’єктно орієнтованих баз даних)
- Структурована база даних (реляційна база даних, описана системою ORM)

4.3.1 Логування

Логування програмного продукту виконано за допомогою системи Apache Log4j 2, за виконання відкритої ліцензії Apache License Version 2.0, від грудня 2004.

Принцип логування простий – у точках програми, які виконують важливі функції та потенційно можуть призвести до помилок, або при розрахунку візуальної інформації, підключається спеціальний модуль, який усі потоки вводу-виводу перелаштовує у систему логування, яка у своє чергу проводить фільтрацію та збереження отриманої текстової інформації (відповідно до попередньо заданих налаштувань).

На даний момент, програма містить у собі 3 потоки виводу інформації (та 1 потік вводу):

- Потоком вводу інформації є системний вивід текстової інформації під час виконання (Runtime). У середовищі Java це System.*
- Потоки виведення інформації:
    - File name="FILE"

      Приймає усі рівні фільтрації, та записує у файл. Необхідний для перегляду повного перебігу сценарію, та виявлення помилок системи після формування готового пакету продукта.
    - Console name="STDOUT"

      Виведення в консоль розробника. Необхідний лише на рівні відладки, або під час запуску програми у консольному режимі.
    - CustomAppender name="CustomAppender"

      Виведення інформації у клієнтську консоль (на даний момент, лише критичні повідомлення та інформація про перебіг тестування).

Програма має 5 рівнів фільтрування:

- DEBUG – відладка для розробника. Найнижчий рівень.
- INFO – інформація про перебіг виконання коду.
- WARN – повідомлення про не стандартну але передбачену роботу коду (наприклад, якщо сервер не відповідає, або якщо база даних не доступна).
- ERROR – помилка роботи програми та\або серверу. Подальший перебіг даного функціоналу зупинений, проте робота іншого функціоналу продовжується.
- FATAL – системний збій. Робота програмного продукту зупиняється повністю. Не збережена інформація втрачена.

4.3.2 Структура бази даних

База даних для роботи даної програми містить усього дві таблиці заповнених індексами. Прямий доступ до бази даних не передбачався.

Побудована база даних на основі типів даних MySQL, проте, формування нового DDL скрипту на іншу базу даних не є проблемою, тому що використовувався універсальний синтаксис SQL 2.0

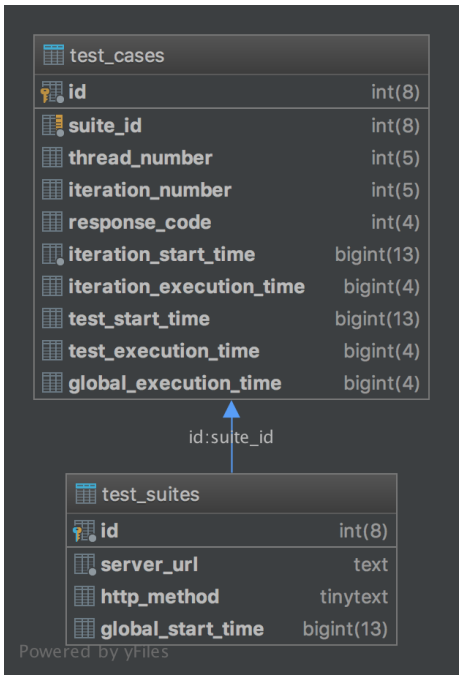

Рисунок 4.4 Діаграма структури бази даних

Таблиця test_suites

Дана таблиця містить у собі наступні поля:

- Id – унікальний ідентифікаційний номер, що використовується як зовнішній ключ для таблиці з тестовими кейсами
- Server_url – адрес об’єкта тестування. Також використовується для вибору клієнтом серверу під час аналізу
- http_method – метод, який, був використаний під час аналізу
- global_start_time – час запуску тестового сценарії (unix timestamp).

Таблиця test_cases

Поля таблиці test_cases:

- Id – унікальний номер потрібний для дотримання структурованості бази даних
- Suite_id – зовнішній ключ до таблиці test_suites
- Thread_number – номер потоку виконання (клієнта)
- Iteration_number – кількість запитів від одного клієнта
- Response_code – код відповіді на кожен з запитів
- Iteration_start_time – початок даної ітерації (одиниці запиту)
- Iteration_execution_time – час проведення запиту (час відповіді серверу)
- Test_start_time – початок даного циклу потоку (початок роботи клієнта)
- Test_execution_time – час виконання даного циклу запиту(час роботи клієнта)
- Global_execution_time – загальний час виконання потоку (час останнього потоку в сценарії є повним часом виконання тестового сценарію).

4.3.3 ORM опис бази даних

ORM (англ. Object-relational mapping) — технологія програмування, яка зв'язує бази даних з концепціями об'єктно-орієнтованих мов програмування, створюючи «віртуальну об'єктну базу даних». [41]

У об'єктно-орієнтованому програмуванні об'єкти в програмі представляють об'єкти з реального світу. Як приклад можна розглянути адресну книгу, яка містить список людей разом з кількома телефонами і кількома адресами. В термінах об'єктно-орієнтованого програмування вони представлятимуться об'єктами класу «Людина», які міститимуть наступний список полів: ім'я, список (або масив) телефонів і список адрес.

Суть проблеми полягає в перетворенні таких об'єктів у форму, в якій вони можуть бути збережені у файлах або базах даних, і які легко можуть бути витягнуті в подальшому, зі збереженням властивостей об'єктів і відносин між ними. Ці об'єкти називають «постійними». Історично існує кілька підходів до рішення цієї задачі.

Для опису кожної з баз даних були створені наступні об’єкти:

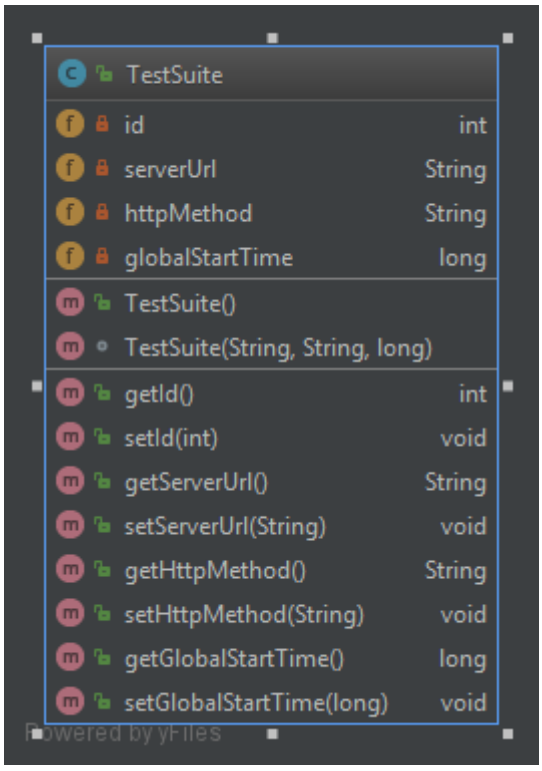

Рисунок 4.5 Об'єкт таблиці test_suites

Змінні даного об’єкту повторюють поля таблиці test_suites. Даний об’єкт також містить конструктор (що створює представлення даної таблиці у коді), гетери (дають доступ зчитувати кожне поле) та сетери (дають доступ змінювати кожне поле).

Аналогічно описаний об’єкт для таблиці test_cases:

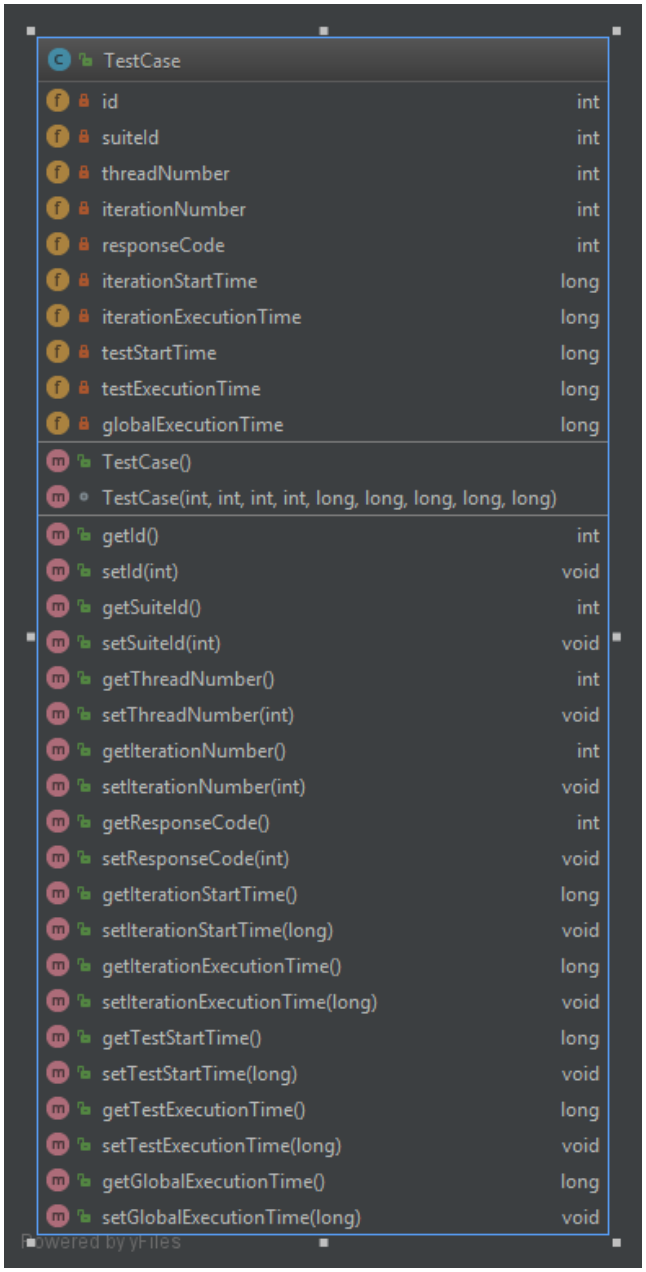

Рисунок 4.6 Об'єкт таблиці test_cases

4.4 Валідація

Важливим пунктом роботи з веб сервером та його відповіддю, є опрацювання результатів (відповідей серверу). Для цього був створений окремий модуль, який приймає відповідь серверу, перевіряє її у відповідності

до заздалегідь описаних критеріїв

Даний клас працює наступним чином: на вхід клас отримує посилання на об’єкт зв’язку з сервером який містить у собі усі заголовки та тіло відповіді. Наступним кроком є опрацювання коду відповіді: перевірити чи код відповідає заздалегідь визначеним (таким як 200, 500, 504, 302, 301 та ін), для яких існує чіткий опис та які найчастіше зустрічаються у реальних веб системах. Якщо код знайдено у словнику – передати інформацію назад головному контролеру, та записати її у базу даних.

Якщо даного коду не знайдено – перевірити його на групу кодів (2**, 4**) та видати клієнту відвідну рекомендацію звернутися до ширшого словника.

Також даний клас опрацьовує тіло відповіді серверу, та за необхідності (відповідно до конфігурації) відображає його клієнту.

4.5 Система конфігурації проекту

Конфігурація проекту закладається у 2-х файлах, а також на графічному екрані.

Файл config.properties відповідає за налаштування за впромовчанням проекту, а також тестового сценарію (необхідний для автоматизованого запуску перевірки систем), а також містить у собі більше налаштувань поведінки клієнта (перехід за покликаннями, тим текстової інформації і тд).

Файл hibernate.cfg.xml відповідає за налаштування зв’язку з базою даних, та являється обов’язковим для налаштування під час розгортання проекту, у випадку, якщо робота модуля аналізу необхідна.

Графічний інтерфейс користувача дозволяє налаштовувати конфігурацію теперішнього тестового сценарію, та перезаписується з файлу config.properties після перезапуску графічного інтерфейсу.

4.6 Висновок

Система аналізу та збереження інформації є половиною дипломного проекту, так як однією з основних цілей є відображення отриманих результатів тестування, та аналіз цих результатів з виведенням зрозумілої для користувача інформації.

Важливою частиною є роботи з базою даних через ORM модель, так як це дозволяє розробляти продукт з високим рівнем програмування згідно моделі ООП, так як ORM дозволяє працювати з кожною таблицею як з окремим об’єктом, а не виконувати окремі не інкапсульовані запити напряму до бази даних.

Аналіз результатів за самого користувача знижує «поріг входження» клієнтів у програмний продукт, що дозволяє користуватися даним продуктом звичайним технічним працівникам, що не мають знань у сегментів веб серверів та інтернет потоків.

5 ТЕХНІЧНЕ ВПРОВАДЖЕННЯ ПРОГРАМНОГО ПРОДУКТУ

Даний розділ присвячений розгортанню проекту та проведенню автоматизованих тестувань без участі користувача, та без налаштувань кожного з тестових сценаріїв.

Даний програмний продукт може працювати без системи «віртуальних клієнтів» (агентів), проте, для вірності та максимальної реалістичності тестування, дану систему варто розгорнути хоча б на 3-х інших серверах. Виділяти окремі сервери під агентів не потрібно – варто лише відкрити один UDP порт 7201 який буде використаний для спілкування з агентом, та порт 80, 443 (HTTP та HTTPS відповідно) для маршрутизації запитів від клієнта до серверу.

На наступній схемі жовті блоки позначають алгоритми який виконуються клієнтом, зеленим кольором позначається частина алгоритму за яку відповідає контролер, який приймає інформацію від клієнта, створює проксі-сервер для зв’язку з переданим URL, та передає інформацію даному проксі серверу. Проксі сервер створює сокет-зв’язок з об’єктом тестування, отримує відповідь (або очікує на її відсутність), та передає зв’язок клієнту, який уже опрацьовує отриманий об’єкт зв’язку.

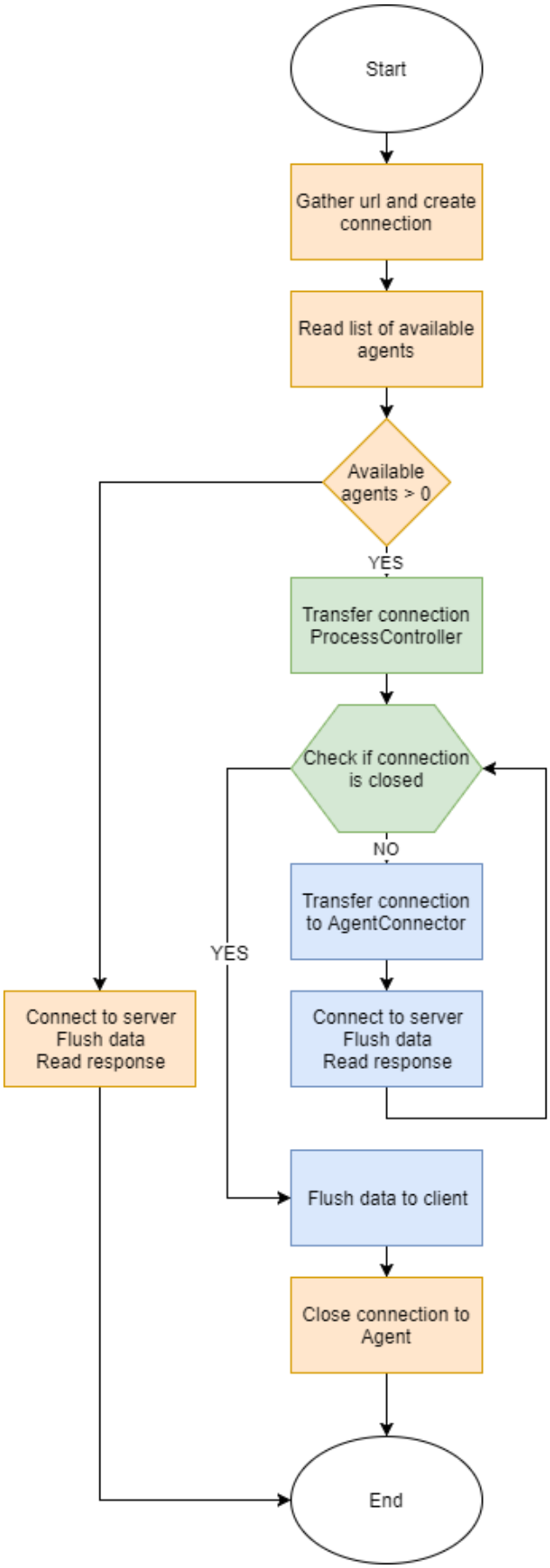

Рисунок 5.1 Алгоритм роботи агента як частини клієнтської програми

5.1 Структура агенту переадресації

Агент представляє собою два класи: один з них це аналог проксі серверу з налаштовуваним інтерфейсом зв'язку з сервером; інший це контролер, який завжди запущений та очікує команд від клієнта. Обидва класи працюють на сокет з'єднаннях.

Контролер агенту очікує налаштування від клієнта, створює необхідний проксі сервер (якщо такий ще не існує), та переадресує з'єднання до самого агенту, тоді як агент отримує налаштування, конфігурується та створює сам зв'язок з об'єктом. Робота даного продукту представлена на наступні схемі:

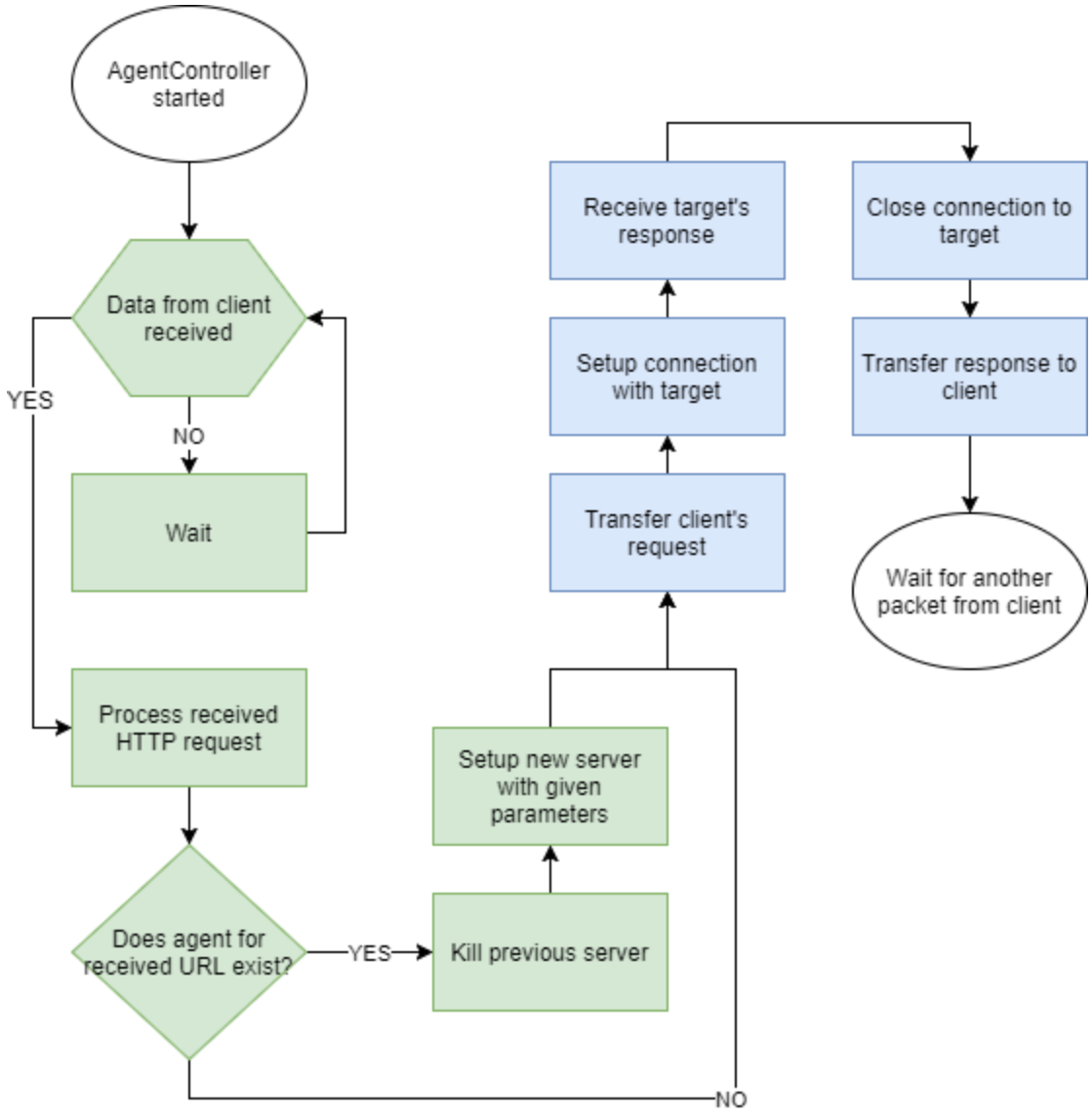

Рисунок 5.2 Алгоритм роботи "агенту"

5.2 Пакет автоматизованого налаштування та розгортання проекту

Пакет автоматизованого налаштування та розгортання проекту – це архів з усіма необхідними модулями для роботи даного програмного продукту. Розрахований даний пакет на NIX системи, з підтримкою bash скриптингу.

Даний пакет має включати у собі:

- JRE SE 8
- Loadtest.jar (запакований проект)
- Файли конфігурації
- MySQL Server (опціональна інсталяція)

У сам пакет інсталяції безпосередньо архівовані лише запакований програмний продукт, файли конфігурації та bash скрипт необхідний для розгортання проекту.

5.2.1 Алгоритм bash скрипту

Скрипт написаний для розпакування проекту виконується у наступній послідовності:

1. Зчитуються параметри передані користувачем
2. Перевіряється наявність параметру для інсталяції бази даних
    a. Якщо так – завантажити та встановити MySQL
    b. Якщо існує доступ до mysql консолі – створити базу даних loadtest з таблицями необхідної структури
    c. Інакше вивести DDL скрипт з інструкціями для введення користувачем або адміністратором особисто
3. Перевірити наявність JRE
    a. Якщо ні – завантажити та встановити
4. Перевірити наявність JAVA_ENV глобальних змінних
    a. Якщо ні – добавити у системний конфігураційний файл змінних

5. Розпакувати loadtest.jar у виконавчу директорію - /usr/local/bin/loadtest
6. Розпакувати файли конфігурації у виконавчу директорію /usr/local/bin/loadtest (дані файли розміщені саме у виконавчу директорію для спрощення керування програмою та для потенційно мульти-платформеності)
7. Запропонувати користувачу корегування файлів налаштувань
    a. Якщо так - відкрити у системному редакторі за промовчанням файл конфігурації бази даних та глобальної конфігурації програми.
8. Запропонувати користувачу налаштувати автоматичний запуск програмного продукту (за умови, що користувач виконав попередній пункт)

Приклад bash скрипту та файлів конфігурації знаходяться у додатках дипломної роботи.

5.2.2 CRON запуск проекту

Для автоматизованої перевірки серверу, та для автоматизації самого проекту, буде використана команда cron-job яка присутня на UNIX-подібних операційних системах. Дана команда та тип операційних систем був обраний через їх часте використання на серверах, тобто, дані машини працюють 24 години на добу.

Так як основні налаштування відбуваються через графічний інтерфейс, тоді як cron здатний працювати лише з термінальними командами, клас view рівня був модифікований для отримання команд з термінального вікна, та запуск без графічного інтерфейсу. У такому режимі роботи, більшість налаштувань потрібно вносити у конфігураційний файл, або передавати через атрибути запуску. Список атрибутів та файл конфігурації представлені у додатках до дипломної роботи.

Для створення команди, потрібно відкрити термінальне вікно, ввести команду по редагування списку так званих кронів та добавити новий запис наступного формату:

```
0 0 * * * java -jar /home/mkyong/crawler/webcrawler.jar param1 param2
```

Приклад 5.1 Приклад cron команди для запуску автоматизованого тесту один раз на день о 00.00

5.3 Висновок

Автоматизоване розгортання та налаштування проекту спрощує систему установки проекту, а також рівень входження користувача у роботу з програмним продуктом.

Розгортання проекту дозволяє користувачам, які не працюють з розподіленими веб системами, просто та без зайвих втрат часу скористатися програмою для перевірки стресостійкості та доступності вузлів веб серверів, тоді як автоматизоване використання продукту дозволяє в автоматизованому режимі перевіряти роботу серверів, без присутності користувача, за заздалегідь визначеними налаштуваннями.

Системам агентів у свою чергу розширює можливість програмного продукту, дозволяючи тестувати об’єкт з допомогою розподіленої системи запитів з різних мережевих вузлів. Дані агенти представляють максимально наближені до реальності дані про роботу об’єкту тестування.

6 ЗАГАЛЬНІ ВИСНОВКИ

Використання програмного продукту здатного в автоматизованому режимі перевіряти цілісність системи веб серверів з простою системою керування та налаштування – допоможе малим та середнім підприємствам своєчасно та ефективно попереджувати серверні або тунельні проблеми пов’язані з роботою веб систем.

Основними критеріями дипломної роботи є:

- Простота у використанні програмного продукту, для зниження рівня входження користувача у середовище стрес тестування
- Широкий функціонал та сфера тестування HTTP зв’язку
- Автоматизована робота
- Система аналізу отриманих результатів, та, як результат, попередження потенційно проблемних систем

Кожен з даних пунктів був досягнений шляхом аналізу існуючих рішень та теоретичних практик.

У рамках дипломної роботи були отримані наступні теоретичні та практичні результати (далі відповідно за розділами):

1. Аналітичний огляд існуючих програмних продуктів та загальних методологій мануального та автоматизованого тестування дозволили скласти сучасну картину у сфері автоматизованого стрес тестування. На даний момент, більшість продуктів є надзвичайно складні у експлуатації з високим рівнем входження експлуатанта, або є комерційними і розроблюються для великих підприємств.
2. Відповідно до отриманих результатів аналітичного огляду, була сформована технічна задача, у рамках якого потрібно було виконати наступні цілі:
    a. Багатопоточне тестування веб-серверу
    b. Розподілене тестування (аналог хмарного сервісу)
    c. Автоматизація роботи програмного продукту

Також, були сформовані наступні критерії до кінцевого продукту:

a. Простота у експлуатації
    i. Графічний інтерфейс
    ii. Налаштування за промовчанням
    iii. Інструкції по експлуатації
b. “Легкість” продукту для операційної системи

3. Хід розробки програмного продукту був розділений на три етапи. Першим етапом було створення функціональну основу проекту, яка відправляє запити до серверу, читає відповідь, та оброблює її у відповідності до заданих цілей тестування. Також, частиною даного етапу було створення графічного інтерфейсу для роботи з функціональним ядром. У ході розробки інтерфейсу, був проведений математичний аналіз (експертна оцінка) можливих варіантів графічного інтерфейсу для спрощення роботи кінцевого користувача з програмою.
4. Другим етапом розробки було створення системи збереження отриманих результатів у двох потоках:
    a. Реляційна база даних
    b. База з логами перебігу роботи програми

    Відповідно до отриманих та збережених даних, проект отримав модуль аналізу результатів. Серед них:
    a. Графічне представлення збережених результатів зв’язку з об’єктом тестування
    b. Система аналізу відповідно до обробки хронології роботи програми з певним веб-ресурсом (виконана у рамках математичної моделі побудованої на вагових критеріях)
5. Останнім етапом розробки проекту стала автоматизація розгортання проекту на операційній системі UNIX. Також, у рамках останнього етапу був створений скрипт для

автоматизованої роботи проекту без участі оператора. Важливою частиною третього етапу стало створення системи агентів, завданням якої є отримання запитів від клієнта (у даному ключі – програмного продукту), маршрутизація цих запитів до серверу, отримання результатів, та передання цих результатів назад клієнту.

Даний програмний продукт рекомендований для застосування у любих компаніях з розподіленими системами. Наприклад, продукт в автоматизованому режимі може перевіряти життєздатність внутрішніх локальних серверів консалтингової компанії, або періодично навантажувати роботів клієнтські (на яких працюють клієнти) сервери мережі провайдерів хостингових послуг.

Будь яке підприємство яке має навіть один сервер у своєму розпорядженні може використовувати даний продукт для своєчасної перевірки та усунення потенційно проблемних вузлів мережі.

Подальший розвиток програмного продукту, та досліджень у даній сфері можливий у декількох напрямках. Перш за все, можливо створення повноцінної системи моніторингу за розподіленою системою веб серверів. Ще одним потенційним напрямком є створення системи тестування безпеки, серверу, яка буде відправляти різноманітні типи запитів та параметрів на сервер, та аналізувати отримані результати, з метою отримання приватної інформації серверу.

7 СПИСОК ЛІТЕРАТУРИ

1. Голдовский І. Безопасность платежей в Интернете. Санкт Петербург: 2002.
2. Ньюман П. The Crash of the AT&T Network in 1990 [Електроний ресурс] // PhWorld: [сайт]. [2008]. URL: http:// www.phworld.org/history/attcrash.htm
3. Колаковський Н. HP TouchPad Needs 6 to 8 Weeks for Additional Shipments [Електроний ресурс] // eWeek: [сайт]. [07]. URL: http://www.eweek.com/mobile/hp-touchpad-needs-6-to-8-weeks-for-additional-shipments
4. Netcraft Ltd. Web Server Survey [Електроний ресурс] // NetCraft: [сайт]. [2017]. URL: https://news.netcraft.com/archives/ category/web-server-survey/
5. Говард М. IIS6 vs Apache2 Security Defects [Електроний ресурс] // Microsoft.com: [сайт]. [2004]. URL: https:// blogs.msdn.microsoft.com/michael_howard/2004/10/15/iis6-vs-apache2-security-defects/
6. Говард М. Follow-up on IIS6 and Apache Security [Електроний ресурс] // Microsoft.com: [сайт]. [2004]. URL: https:// blogs.msdn.microsoft.com/michael_howard/2004/10/18/follow-up-on-iis6-and-apache-security/
7. Google Inc. Google Testing Blog [Електроний ресурс] // GoogleBlog.com: [сайт]. URL: https://testing.googleblog.com/
8. Дончeнко В. Карьера в IT: должность QA Automation engineer [Електроний ресурс] // DOU: [сайт]. [18]. URL: https:// dou.ua/lenta/articles/qa-automation-engineer-position/
9. softwaretestinghelp.com. Comprehensive Load Testing Tools

List [Електроний ресурс] // Softwaretestinghelp: [сайт]. [2017]. URL: http://www.softwaretestinghelp.com/performance-testing-tools-load-testing-tools/

10. Леонов С. Тестирование производительности веб-приложений // Открытые системы.СУБД. 2014. URL: Открытые системы.СУБД. (дата звернення: 7.10.2017).
11. Рогов С., Намиот Д. Тестирование производительности Web-серверов // Открытые системы.СУБД. 2002. URL: http:// www.osp.ru/os/2002/12/182266 (дата звернення: 20.10.2014).
12. Можаєв П. Средства автоматизированного тестирования // Открытые системы.СУБД. 2009. URL: http://www.osp.ru/os/2009/ 03/8161608 (дата звернення: 20.10.2014).
13. Дубова Н. Платформы разработки // Открытые системы.СУБД. 2006. URL: http://www.osp.ru/os/2006/01/380740 (дата звернення: 20.10.2014).
14. Єфімцева Н. Тестирование облачных сервисов // Открытые системы.СУБД. 2012. URL: http://www.osp.ru/os/2012/05/13016242 (дата звернення: 20.10.2014).
15. Брадтке Р. ISTQB 100 Success Secrets - ISTQB Foundation Certification Software Testing the ISTQB Certified Software Tester 100 Most Asked Questions. Emereo Publishing, 2008. 170 pp.
16. Microsoft Corporation. Performance Testing Guidance for Web Applications. ThriftBooks - Green Earth, 2007. 288 pp.
17. Байон Е. Performance Testing with Jmeter 2.9. ООО «Книга по Требованию», 2013. 138 pp.
18. Криспін Л., Грегорі Д. Гибкое тестирование: практическое руководство для тестировщиков ПО и гибких команд. Москва: «Вильямс», 2010. 464 pp.

19. Дохі Т., Накавага Т., та Стохастік Т. Reliability and Maintenance Modeling: Essays in Honor of Professor Shunji Osaki on his 70th Birthday. London: Springer, 2013. 360 pp.

20. Спілнер А., Лінз Т., Роснер Т., та Вінтер М. Software Testing Practice: Test Management: A Study Guide for the Certified Tester Exam ISTQB Advanced Level. Rocky Nook, 2007. 339 pp.

21. Колава А., Худжинга Д. Automated Defect Prevention: Best Practices in Software Management. Wiley-IEEE Computer Society Press, 2007.

22. Куджала С., "UX Curve: A method for evaluating long-term user experience," // Interacting With Computers, Vol. 23, No. 5, 2011.

23. IBM Design. Design in motion [Електроний ресурс] // IBM Design: [сайт]. [2015]. URL: http://www-01.ibm.com/software/ucd/designconcepts/whatisUXD.html (дата звернення: 11.10.2017).

24. webstyleguide.com. Visual Design Web Style Guide 3 [Електроний ресурс] // Web Style Guide: [сайт]. [2015]. URL: http://webstyleguide.com/wsg3/7-page-design/3-visual-design.html (дата звернення: 11.10.2017).

25. Псомас С., "The Five Competencies of User Experience Design," // UX Matters, Листопад 2007.

26. Лонвгрен Д. Interaction Design - brief intro // Interaction Design. URL: http://www.interaction-design.org/encyclopedia/interaction_design.html

27. Маркус А. Design, User Experience, and Usability: Design Discourse. Springer, 2015. 672 pp.

28. Темичев А., Файзрахманов Р., "Аналитический обзор средств автоматизации тестирования производительности применительно к системам мониторинга," // Вестник ПНИПУ,

Vol. 15, 2015.

29. IBM. Справочные материалы по Rational Performance Tester [Електроний ресурс] // Сайт компании IBM: [сайт]. URL: http:// www.ibm.com/de- veloperworks/downloads/r/rpt/ (дата звернення: 5.11.2014).

30. HP Ltd. Справочные материалы по программе LoadRunner [Електроний ресурс] // Сайт компании HP: [сайт]. URL: http:// www8.hp.com/us/en/ software-solutions/loadrunner-load-testing/ (дата звернення: 5.11.2014).

31. Apache Foundation. Справочные материалы по программе JMeter [Електроний ресурс] // Сайт Apache Software Foundation: [сайт]. URL: https://jmeter.apache.org/ index.html (дата звернення: 5.11.2014).

32. Borland. Справочные материалы по программе SilkPerformer [Електроний ресурс] // Сайт Micro Focus: [сайт]. URL: http:// www.borland.com/Products/ Software-Testing/Performance-Testing/ Silk-Performer/Features (дата звернення: 5.11.2014).

33. SoftLogica Inc. Справочные материалы по программе WAPT [Електроний ресурс] // Сайт SoftLogica Inc: [сайт]. URL: http:// www.loadtestingtool.com (дата звернення: 5.11.2014).

34. Canonical Ltd. Ubuntu 16.04.3 LTS (Xenial Xerus) [Електроний ресурс] // Ubuntu.com: [сайт]. URL: https:// wiki.ubuntu.com/XenialXerus/ ReleaseNotes?_ga=2.17724272.895013929.1508044866-1858907825.1508044866 (дата звернення: 15.10.2017).

35. Віксі П. MAN crontab [Електроний ресурс] // OpenNet.ru: [сайт]. URL: http://www.opennet.ru/ man.shtml?topic=crontab&category=5 (дата звернення: 15.10.2017).

36. Oracle. Java SE Development Kit 9 [Електроний ресурс] // Oracle.com: [сайт]. URL: http://www.oracle.com/technetwork/java/javase/downloads/jdk9-downloads-3848520.html (дата звернення: 15.10.2017).

37. Вогель Л. REST with Java (JAX-RS) using Jersey [Електроний ресурс] // Jersey: [сайт]. [2017]. URL: http://www.vogella.com/tutorials/REST/article.html (дата звернення: 15.10.2017).

38. The Eclipse Foundation. Eclipse.com [Електроний ресурс] // Jetty HTTP Server: [сайт]. URL: https://www.eclipse.org/jetty/ (дата звернення: 15.10.2017).

39. Oracle. Why MySQL? [Електроний ресурс] // MySQL TM: [сайт]. URL: https://www.mysql.com/why-mysql/ (дата звернення: 15.10.2017).

40. Oracle. Java Docs (Oracle) [Електроний ресурс] // java.nio (Java SE7): [сайт]. URL: https://docs.oracle.com/javase/7/docs/api/java/nio/package-summary.html

41. Венерс Б. Object-Relational Mappings // Artima Developer. 2003. URL: http://www.artima.com/intv/abstract3.html (дата звернення: 2.12.2017).

42. MongoDB, Inc. What is MongoDB? [Електроний ресурс] // MongoDB: [сайт]. URL: https://www.mongodb.com/what-is-mongodb (дата звернення: 15.10.2017).

Додаток А. Матриці індивідуальних переваг експертів

Експерт 2

	Global Settings	Scenarious Settings	Comprehension	Design	Visualization	Sum	R
Global Settings		0	0	1	0	1	4
Scenarious Settings	0		0	1	1	2	2.5
Comprehension	1	1		1	0	3	1
Design	0	0	0		0	0	5
Visualization	1	0	1	0		2	2.5

Експерт 3

	Global Settings	Scenarious Settings	Comprehension	Design	Visualization	Sum	R
Global Settings		0	0	1	0	1	4
Scenarious Settings	1		0	1	1	3	2
Comprehension	1	1		1	1	4	1
Design	0	0	0		0	0	5
Visualization	1	0	0	1		2	3

Експерт 4

	Global Settings	Scenarious Settings	Comprehension	Design	Visualization	Sum	R
Global Settings		0	0	0	0	0	5
Scenarious Settings	1		0	1	0	2	3
Comprehension	1	1		1	1	4	1
Design	1	0	0		0	1	4
Visualization	1	1	0	1		3	2

Експерт 5

	Global Settings	Scenarious Settings	Comprehension	Design	Visualization	Sum	R
Global Settings		0	0	0	0	0	5
Scenarious Settings	1		0	1	1	3	2
Comprehension	1	1		1	1	4	1
Design	1	0	0		0	1	4
Visualization	0	1	0	1		2	3

Експорт 6

	Global Settings	Scenarious Settings	Comprehension	Design	Visualization	Sum	R
Global Settings		0	0	1	0	1	4
Scenarious Settings	1		0	1	1	3	2
Comprehension	1	1		1	1	4	1
Design	0	0	0		0	0	5
Visualization	1	0	0	1		2	3

Експорт 7

	Global Settings	Scenarious Settings	Comprehension	Design	Visualization	Sum	R
Global Settings		0	0	1	0	1	4
Scenarious Settings	1		0	1	0	2	3
Comprehension	1	1		1	1	4	1
Design	0	0	0		0	0	5
Visualization	1	1	0	1		3	2

Експорт 8

	Global Settings	Scenarious Settings	Comprehension	Design	Visualization	Sum	R
Global Settings		0	0	1	0	1	4
Scenarious Settings	1		0.5	1	1	3.5	1.5
Comprehension	1	0.5		1	1	3.5	1.5
Design	0	0	0		0	0	5
Visualization	1	0	0	1		2	3

Експорт 9

	Global Settings	Scenarious Settings	Comprehension	Design	Visualization	Sum	R
Global Settings		1	0	1	1	3	1
Scenarious Settings	1		0.5	1	0	2.5	2.5
Comprehension	1	0.5		1	0	2.5	2.5
Design	0	0	0		1	1	5
Visualization	0	1	1	0		2	4

Додаток Б. Bash скрипти та DDL для розгортання проекту

```
#!/bin/bash
echo "Installation script for loadtest software"

echo "Provide administrative password to continue"
sudo apt-get update

# Check if Java is there. If not - install
output=$(file `which java`)
is_java_installed=false
string_is_java_installed="Java is not installed"
while read -r line
  do
        if [[ $line = *"architecture"* ]]; then
                string_is_java_installed="Java is installed"
                is_java_installed=true
        fi
  done <<< "$output"

echo "$string_is_java_installed"
echo

if [[ $is_java_installed = false ]]; then
        sudo apt-get install default-jdk #JAVA_ENV is installed automatically
fi

# Install MySQL if clients want
echo "WARNING! If you have password protected MySQL server, please, manually flash loadtest.sql"
echo
read -p "Do you want to install MySQL? [y - Yes, n - No] " -n 1 -r
echo
if [[ $REPLY =~ ^[Yy]$ ]]; then
   /bin/bash ./mysql.sh #call it from the same directory
fi

# Change config
read -p "Do you want to edit loadtest configuration file? [y - Yes, n - No] " -n 1 -r
echo
if [[ $REPLY =~ ^[Yy]$ ]]; then
   nano "config.properties"
fi

# Change workdir for app
workdir="/usr/local/bin/loadtest/"
read -p "Do you want to change default workdir ("$workdir")? [y - Yes, n - No] " -n 1 -r
echo
if [[ $REPLY =~ ^[Yy]$ ]]; then
   read workdir
   echo "Workdir change to "$workdir""
fi

# Copying loadtest-client.jar
echo "Copying loadtest-client.jar to $workdir"
cp loadtest-client.jar "$workdir"
cp config.properties "$workdir"
echo “Installation complete”
echo “Refer to manual.txt for launching”
```

Приклад 7.1 Bash крипт для розгортання проекту

```
#!/bin/bash
echo "Starting MySQL installation"
sudo apt-get install mysql-server
read -p "Do you want to securely setup MySQL? " -n 1 -r
echo
if [[ $REPLY =~ ^[Yy]$ ]]
   mysql_secure_installation
fi
echo "Provide root password for MySQL server"
cat loadtest.sql | mysql -u root -p
```

Приклад 7.2 Bash скрипт для інсталяції MySQL серверу та створення бази

```
CREATE DATABASE `loadtest`
  CHARACTER SET utf8
  COLLATE utf8_bin;
GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, ALTER, INDEX ON `loadtest`.*
TO 'loadtest-user'@'localhost'
IDENTIFIED BY 'password';
CREATE TABLE test_cases
(
  id                    INT(8) AUTO_INCREMENT
    PRIMARY KEY,
  suite_id              INT(8)     NOT NULL,
  thread_number         INT(5)     NULL,
  iteration_number      INT(5)     NULL,
  response_code         INT(4)     NULL,
  iteration_start_time  BIGINT(13) NOT NULL,
  iteration_execution_time BIGINT(4)  NULL,
  test_start_time       BIGINT(13) NULL,
  test_execution_time   BIGINT(4)  NULL,
  global_execution_time BIGINT(4)  NULL
);

CREATE INDEX idx_test_cases_suite_id
  ON test_cases (suite_id);

CREATE TABLE test_suites
(
  id              INT(8) AUTO_INCREMENT
    PRIMARY KEY,
  server_url      TEXT      NOT NULL,
  http_method     TINYTEXT  NULL,
  global_start_time BIGINT(13) NULL
);
```

Приклад 7.3 DDL для створення бази та таблиць для програми

Додаток В. Файл конфігурації проекту

```
# Scenario Setup Config
PARAM_NAMES = field1, field2, field3, submit
PARAM_VALUES = value1, value2, value3, submit
URL = http://google.com
COUNT_THREAD = 1
COUNT_ITERATIONS = 1
HTTP_METHOD = GET
FOLLOW_REDIRECTS = true
REQUEST_PROPERTY_CONTENT = content-type
REQUEST_PROPERTY_APP = application/x-www-form-urlencoded

# DB Connection config
# 1 - MYSQL, 2 - ORACLE, 3 - POSTGRES
HIB_DIALECT = 1
# Omit connection protocol (everything till '//')
DB_HOST = 192.168.56.101
DB_PORT = 3306
DB_NAME = loadtest
# Use URL parameters separator "&"
DB_PARAMS = useSSL=false&test=test
DB_USERNAME = root
DB_PASSWORD = password
# Timeout for DB connection in seconds
DB_TIMEOUT = 10
```

Приклад 7.4 Приклад файлу конфігурації проекту за промовчанням

Додаток Г. Інструкція по розгортанню проекту

Loadtest System - програмний продукт розроблений для магістерської роботи
Завданням програми є навантажувальне тестування веб-серверів по HTTP(S) протоколу.

В інсталяційний пакет входять наступні файли:
- loadtest-client.jar - програма,
- agents/
-- loadtest-agent.jar - агент адресації,
-- loadtest-agentmanger.jar - менеджер адресації агентів,
-- process-builder.jar - створювач процесів у середовищі *nix
- install.sh - bash скрипт інсталяції
- mysql.sh - bash скрипт для інсталяції mysql серверу
- loadtest.sql - DDL скрипт для розгортування бази даних та створення таблиць
- config.properties - файл конфігурації проекту за промовчанням

===== Інсталяція =====
Для розгортання проекту запустіть скрипт install.sh:
OSX: відкрийте термінал та виконайте команду "sh install.sh"
LINUX: відкрийте термінал та виконайте команду "chmod +x install.sh && ./install.sh"
WINDOWS:
0. Перенесіть проект (усі файли) у зручну для вас директорію
1. Встановіть JAVA SE 8+ за інструкціями звідси - https://www.java.com/en/download/help/download_options.xml
2. Встановіть MySQL Server за інструкціями звідси - https://dev.mysql.com/downloads/installer/
3. За необхідності налаштуйте файл config.properties
3.1 Змініть DB_HOST на адрес MySQL серверу (вкажіть localhost якщо сервер був встановлений самостійно)
3.2 Змініть DB_USERNAME та DB_PASSWORD у випадку, якщо ви вказали інші паролі під час інсталяції бази loadtest

===== Запуск =====
Для запуску проекту потрібно виконати команду java -jar loadtest-client.jar
Дана команда запустить графічний інтерфейс для роботи з програмою
У випадку якщо необхідно запустити програму з додатковими параметрами скористайтеся атрибутом -? (приклад java -jar loadtest-client.jar -?)

===== Система агентів =====
Для роботи з системою агентів необхідно виконати наступні пункти:
1. Перенесіть папку (з усіми файлами) agent на сервер, що слугуватиме агентом
2. Запустіть менеджер агентів наступною командою java -jar loadtest-agentmanager.jar 7901 7902 80 loadtest-agent.jar process-builder.jar
2.1 Параметри 7901 та 7902 відповідають за порт для менеджера (для конфігурації серверу) та порт для самого агенту (через який відправляться запити).
Дані порти можна змінювати відповідно до ваших вимог
2.2 loadtest-agent.jar process-builder.jar - назви .jar пакетів за промовчанням. Можна змінювати у відповідності до справжніх імен відповідних файлів.
3. Додайте адреса агентів у файл agent.txt що розташований у тій же папца що і файл loadtest-client.jar
3.1 Приклад запису 127.0.0.1:7901:7902
127.0.0.1 - IP адреса серверу з агентами
7901 - порт менеджера
7902 - порт агента
4. Запустіть loadtest-client.jar з використанням системи агентів
4.1 Під час роботи з графічним інтерфейсом встановіть галку "Use Agents?" для використання системи агентів
4.2 Під час роботи з термінальним інтерфейсом добавте параметр -a для запуску мережі агентів

Кузьмінський Олег Вікторович, ФАІТ, ІУСТ-61, КНУБА, Київ, 2018 рік

Приклад 7.5 Інструкція програми