МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ВИПУСКНА РОБОТА

на тему:

«Застосування методів EDM для розробки системи підтримки рішень»

Завідувач
випускаючої кафедри **Довбиш А.С.**

Керівник роботи **Кузіков Б.О.**

Студентка групи ІН.м– 92 **Петренко А.М.**

СУМИ 2020

Сумський державний університет
(назва вузу)

Факультет *ЕлІТ* Кафедра *Комп'ютерних наук*

Спеціальність *«Інформатика»*

Затверджую:
зав.кафедрою _____
_____
"_____" _____ 20___р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТОВІ

Петренко Анні Михайлівні
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Застосування методів EDM для розробки системи підтримки рішень*

затверджую наказом по інституту від "____" _____ 20 ___ р. № _____

2. Термін здачі студентом закінченого проекту (роботи) _____

3. Вхідні данні до проекту (роботи) *лог-файли із активністю студентів на електронній навчальній системі СумДУ - https://mix.sumdu.edu.ua*

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Огляд алгоритмів та методів специфічних Education Data Mining та повязаних із визначенням продуктивності сервера; 2) аналіз проблеми, постановка задачі і формування завдань для реалізації; 3) вибір середовища, мови та алгоритму для розробки моделі; 5) розроблення алгоритмів попередньої обробки вхідних даних; 6) реалізація моделі для підтримки рішень щодо навантаженості сервера; 7) аналіз результатів.*

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

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
1.	*Аналіз проблеми. Постановка задачі дослідження*		
2.	*Огляд відомих методів вирішення*		
3.	*Обрання методу вирішення*		
4.	*Розробка моделі для підтримки рішень щодо навантаженості сервера*		
5.	*Оформлення пояснювальної записки до дипломної роботи*		

Студент – дипломник _____
(підпис)

Керівник проекту _____
(підпис)

РЕФЕРАТ

Записка: 83 стор., 10 рис., 1 табл., 2 додатки, 24 літературні джерела.

Об'єкт дослідження — Дистанційний навчальний ресурс https://mix.sumdu.edu.ua.

Мета роботи — розробити модель, яка б допомогла адміністратору електронної системи прийняти рішення щодо навантаженості сервера та прогнозувала б максимальну кількість унікальних користувачів та запитів при якій сервер буде неспроможний обробляти запити клієнтів., що включає в себе попередній аналіз літератури за темою аналізу продуктивності інформаційних систем та аналіз лог-файлу з даними студентів, розробку алгоритмів обробки даних, реалізацію графіків та виведення результату щодо масштабованості електронного ресурсу.

Методи дослідження — в процесі виконання реалізації моделі було застосовано методи категорії прогнозування, закон Літтла та перевірка статистичної гіпотези щодо рівності середніх значень.

Результати — проведено аналіз вхідних даних, літератури, обрані методи, алгоритм виконання поставленої задачі, реалізовані наглядні графіки, що показують масштабованість електронного ресурсу, розроблено модель для допомоги адміністратору ресурса прийняти рішення щодо його продуктивності.

EDUCATION DATA MINING, АНАЛІЗ ДАНИХ, JAVASCRIPT, ZINGCHART, ПРОГНОЗУВАННЯ, ЗАКОН ЛІТТЛА, СТАТИСТИЧНІ ГІПОТЕЗИ, DATA-MINING

ЗМІСТ

ВСТУП

У час інтернет-технологій безліч аспектів життя переноситься в мережу, що прискорює темпи інформаційного розвитку суспільства і долає географічні бар'єри. Освіта не стає в цьому виключенням.

Дистанційна освіта стала дуже популярною з того моменту, як інтернет з'явився для широкої верстви населення. Такий вид освіти відкрив кардинальні можливості для розвитку громадян з віддалених населених пунктів або ж людей, що мають важкий робочий графік, проте є бажання навчитися чомусь новому. На початку такий вид навчання сприймався лише для підготовки до екзаменів або ж як додатковий спосіб для отримання нових знань. Сьогодні люди мають можливість пройти повноцінні дистанційні курси та програми для підвищення їх кваліфікації і вибрати вони можуть навіть престижні вузи різних країн і при цьому знаходитись у іншій точці світу.

Дистанційне навчання – це така форма навчання, де використовуються комп'ютерні та телекомунікаційні технології, які на різних етапах навчання забезпечують взаємодію викладачів та студентів, що є інтерактивною. Також сприяють самостійній роботі студента з матеріалами, що знаходяться в інформаційній мережі. [6]

З теперішньою світовою ситуацією, де дистанційне навчання стало обов'язковим і вже протягом року воно є в такому режимі, всі навчальні заклади просто вимушені перенести навчання до електронної системи. Навчання, поведінка студентів та дедлайни не змінилися, тому як зазвичай завдання виконуються в самий останній момент, створюючи велику нагрузку на сервер. Часто навіть трапляється ситуація, коли сервер не витримує і перестає працювати, створюючи неможливим роботу з навчальною електронною системою як студентам так і викладачам. Не є винятком і електронна система дистанційного навчання СумДУ - https://mix.sumdu.edu.ua.

У бакалаврській роботі було покладено перший крок задля того, щоб відстежувати поведінку студентів методами EDM. У даній роботі буде продовжуватися застосування методів EDM.

Результатом цієї роботи має бути модель, що зможе давати інформацію користувачу про те, скільки користувачів може працювати з електронною системою та передбачати коли сервер перестане обробляти запити. Для того, щоб візуально результат був зрозумілим користувачу, буде реалізовано графік аналізу масштабованості.

Актуальністю теми є те, що завдяки реалізованій моделі користувач зможе отримати інформацію щодо завантаження сервера та, якщо воно надто високе, прийняти рішення щодо попередження його падіння. Це рішення зробить можливим безперервну роботу студентів та викладачів з електронною системою.

Виходячи з актуальності проблеми, необхідне впровадження механізмів отримання підтримки прийняття рішення щодо роботи сервера електронної системи навчання СумДУ шляхом запровадження моделі, що зможе розрахувати кількість користувачів при якій сервер буде неспроможний обробляти запити клієнтів та вивести графік, що наглядно продемонструє масштабованість сервера.

Для досягнення поставленої мети сформульовано наступні задачі:

1. Провести аналіз літератури за темою аналізу продуктивності інформаційних систем.
2. Вибрати середовище, алгоритм та мову для розробки моделі та реалізування графіку аналізу масштабованості.
3. Розробити алгоритми попередньої обробки вхідних даних.
4. За допомогою вибраного алгоритму реалізувати модель, що зможе розрахувати кількість користувачів при якій сервер буде неспроможний обробляти запити клієнтів.
5. Протестувати та розрахувати точність розробленої моделі.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Аналіз проблеми

Впровадження інтерактивних навчаьлних систем – це одна з сучасних тенденцій розвитку електронного навчання. Взаємодія студента з такою системою породжує надзвичайно великий масив даних, що може бути використаний для поліпшення та корегування навчального процесу. Для того, щоб вирішити таку задачу можна звернутися до методів сучасної науки Educational Data Mining [3]. EDM – це новочесна міждисциплінарна наука, яка має на меті розробку методів за допомогою яких можна дослідити дані навчального процесу. Ця наука розроблює та використовує алгоритми для того, щоб покращити освітні результати і витлумачити навчальні стратегії для прийняття рішень щодо навчального процесу.

Розроблені EDM алгоритми мають застосування для вилучення знань з даних освітніх ресурсів і вивчення параметрів, які можуть зробити продуктивність більшою. Перш за все вивчення почалося в класі, що не є дистанційним і грунтувалося на конструктивістських, поведінкових, когнітивних моделях. Моделі поведінки вивчають зміни в поведінці студента длятого, щоб оцінити результат навчання. Когнітивні моделі спираються на активну участь вчителя у самому процесі навчання. У моделях конструктивістських все навпаки – студентам необхідно самим навчатися доступним предметам та отримувати знання. [3]

Бейкер і Яцеф виділили такі наступні важливі чотири цілі EDM [3, 4]:

1. Спрогнозувати поведінку учнів методом вдосконалення моделей студентів. Моделювання має на меті охарактеризувати та класифікувати характеристики або стани студента, що, об’єднуючись, складають мотивацію, метапізнання, ставлення та знання студента.

2. Виявити або ж вдосконалити моделі структури доменних знань. Наприклад, тут мають місце моделі, що можуть пояснити взаємозвязок знань у конкретній області, та концептуальні моделі навчальних матеріалів.
3. Виявити та вивчити найбільш ефективну педагогічну підтримку щодо навчання студентів, що може бути зроблена за допомогою систем для навчання.
4. Встановлення емпіричних даних, що зможуть підтвердити або сформулювати рамки, педагогічні теорії та явища для того, щоб визначити основні впливові компоненти навчання, що зможуть посприяти розробці кращих систем навчання.

Безперебійна робота додатків та сервісів напряму залежить від нагрузки, що надходить на віртуальний сервер. Збої в його роботі можуть визвати різні причини – від різко збільшеної кількості відвідуваності та запитів до сервера до масових кібератак.

Зниження ефективності роботи ресурсів на віртуальному сервері не завжди означає, що треба купляти більш потужну віртуальну машину. Для того, щоб вирішити проблему, треба виявити основні причини перенавантаження віртуального сервера. Зазвичай виправити такі помилки можна без додаткових трат на нове обладнання.

Навантаження на сервер – це кількісна оцінка характеристик ресурсів хостингу, які мають застосування під час виконання дійсних задач. Іншими словами – це процент навантаження ресурсів сервера – процесора, оперативної пам’яті та дискового простору.[8]

Навантаження може бути двох видів:

1. На базу даних, що викликано важкими SQL-запитами , відсутністю оптимізації та неправильні налаштування конфігураційного файлу.
2. На веб-сервер, що викликано збільшенням відвідуваності інтернет-ресурсу, що знаходиться на віртуальному сервері.

Якщо є доступ до сервера, можна виконати його моніторинг, як часто й роблять. Моніторинг –збір та обробка інформації щодо завантаженості за певний проміжок часу з різних облікових записів ресурсів сервера. Моніторинг на початковому рівні дозволяє попередити розвиток негативних наслідків на невирішуваних проблем.

Для цього перш за все треба або оптимізувати акаунт, який створює підвищене навантаження, або ж розширити його ресурсну базу.

Для аналізу мережевої активності використовується утиліта atop. Вона записується в лог подій, в якому можна знайти процес, що призводить до перенавантаження сервера. Після того, як утиліта була встановлена, потрібно запустити команду: sudo atop 1. В терміналі відобразиться інформація, що буде розділена на дві секції. У першій – основні дані (загруженість CPU, RAM та диску), а у другій – дані про вибраний процес. Потім за допомогою команди sudo atop -r є можливість отримати інформацію щодо завантаженості сервера.

Якщо процентне співвідношення еквівалентно до 100%, то це значить, що присутня проблема в операціях вводу/виводу або викристанні самого серверу.

Діагностика серверу застосовується для того, щоб виявити проблемне програмне забезпечення, яке породжує високе навантаження на сервер. Існує безліч додатків, що можуть допомогти виконати такий аналіз. Їх можна розподілити на дві категорії:

1. Прості – показують час завантаження веб-сторінки.
2. Важкі – можуть імітувати підключення з різних місць та робити DOS-атаки на додаток, що тестується.

Прикладами таких додатків можуть бути [10]:

1. Locust (масштабований інструмент для тестування завантаженості, що написаний на Python; допомагає детально оцінити продуктивність серверної частини ресурсу).

2. Host-Tracker (інструмент, що допомагає виконати тестування сервера на надмірне навантаження, одночасно підключаючить з 90 точок по усій планеті).
3. OpManager (за допомогою цього інструмента можна виконувати проактивний моніторинг стану серверів, мережі, та маршрутизаторів).
4. WebLOAD (універсальний сервіс для моніторингу, який дозволяє перевірити всі сторінки додатку та висвітлити час, за який завантажується кожна із сторінок).
5. LoadImpact (виконує тестове навантаження на сервер, використовуючи 50 одночасних підключень, що відкривають одночасно до 20-и сторінок)
6. Apache JMeter – Java-додаток з відкритим доступом до коду, що призначений для тестування навантаження не тільки веб-додатків, а й їх окремих компонентів, FTP-серверів, баз даних та мережі.

Однак, ні утиліта atop, ні інші додатки не можуть бути використані для розробки моделі, що зможе розрахувати кількість користувачів при якій сервер буде неспроможний обробляти запити клієнтів. Причина у вхідних даних для розробки моделі.

Apache JMeter – інструмент для перевірки того, скільки трафіку може підтримувати сервер. Але даний інструмент не зможе відповісти на питання «Скільки користувачів зможе витримати сервер», що являється ключовим в розробці моделі.

Основні причини отримання навантаженого сайту [9]:

1. Помилка в системі кешування сайту, що спричинила собою створення більшої кількості сторінок і в результаті при кожному зверненні пошукової системи створюється навантаження на сайт і сервер.

2. Сталося порушення в структурі бази даних, яку використовує сайт.

Раптові відключення електроживлення комп'ютера, дії комп'ютерних вірусів і шкідливих програм іншого типу, значні помилки користувача або помилки в програмному забезпеченні, а також дії зловмисників можуть привести до пошкодження внутрішньої структури файлів бази даних певного сайту.

3. Інші причини: зріст відвідуваності ресурсу.

Компанія Яндекс тестує продуктивність в рамках Continuous Integration. Продукти цієї компанії мають величезне навантаження. На один сервіс є тисячі хостів з сотнями тисячами запитів за секунду і відповідати на запити клієнтів вони мають постійно, за долю секунди. Тестують продуктивність та оцінюють вплив кода на сервіс дуже ретельно після навіть незначних змін в ньому.

Яндекс виділив такі важливі показники сервера:

1. Request per second (RPS) – кількість запитів в секунду, що зможе витримати сервер та не впасти.
2. Time per request – зміст сайту має відображатися якомога швидше, щоб користувачу цього сайту не набридло чекати. Якщо на один запит система відповідає занадто довго, то користувач може і не побачити частину важливої для нього інформації.
3. Resident set size (RSS) – скільки програма використовує пам’яті. Якщо сервіс використає всю пам’ять, то про відмовостійкість не буде йти мови.

Щоб оцінити скільки запитів в секунду витримає сервер, Яндекс направляє в нього поток запитів (однотипні, штучно створені). Але найточнішими будуть ті виміри, які є зібраними з реального пулу запитів від користувачів за якийсь інтервал часу.

Ємність можна виміряти двома методами.

- Відкрита модель навантаження (стрес-тестування)

Суть методу в наступному.

Зробити "користувачів", детальніше кілька потоків, які будуть відправляти певні запити в систему.

Навантаження треба давати не постійне, а нарощувати або ж навіть подавати хвилями. Тоді це наблизить результат до реального життя. Нарощуємо RPS і знаходимо таку точку, в якій сервіс перестане відповідати. Таким чином можна знайти межі роботи системи.

Для розрахунку кількості користувачів можна скористатися формулою Літтла.

У теорії масового обслуговування, детальніше у розділі теорії ймовірностей, законом Літтла називають теорему, що є сформульованою американським вченим Джоном Літтлом [11]:

Довгострокова середня кількість L вимог в стаціонарній системі дорівнює довгостроковій середній інтенсивності λ вхідного потоку, помноженої на середній час W перебування заявки в цій системі. Алгебраїчно можна позначити, $L = \lambda W$.

Сформулювавши іншими словами, при заданій інтенсивності вхідного потоку час в системі пропорційний кількості заявок в цій же системі. Хоча результат і виглядає інтуїтивно зрозумілим, він яскравий, так як добре виражено зв'язок не опосередкований розподілом надходження, розподілом обслуговування, порядком обслуговування або якимись іншими сторонніми характеристиками.

Закон можна застосувати до будь-яких систем, зокрема, до підсистем.

- Закрита модель навантаження (тестування навантаження)

Суть методу в наступному.

Треба взяти фіксовану кількість "користувачів". Наступний крок - налаштувати так, щоб вхідна черга, що відповідає конфігурації сервісу, була завжди повна.

При цьому необхідно зробити число потоків більше, ніж є ліміт черги, немає сенсу, так як ми будемо наштовхуватися на це ж число, а інші запити будуть відкинуті сервером з помилкою 5xx. Треба дивитися на кількість запитів в секунду, що конструкція зможе видати.

Схема, наведена вище, в загальному випадку не є схожою на певний реальний потік запитів від користувачів, але вона може допомогти показати існуючу поведінку системи при якій буде максимальне навантаження. Також вона допоможе оцінити пропускну здатність системи на поточний момент.

Для значної кількості систем (де критичний ресурс не має відношення до обробки з'єднань) результат буде таким самим. При цьому ж у закритій моделі шум менше. Причина цього – система знаходиться в необхідній для нас області навантаження під час вього часу тестування.

Яндекс при тестуванні свого сервісу використовує закриту модель. В результаті видається кількість запитів в секунду, що сервіс зміг видати. Застосовує ця компанія додаток Яндекс.Танк.

Важливо відстежувати, щоб сервіс не деградував в часі. Маленькі просадки можна й не побачити, але вони можуть накопичуватися, і на великому часовому проміжку ці показники можуть просісти. На рисунку 1.1. можна побачити графік, за якими Яндекс дивиться на RPS. Він показує відносне значення при кожній новій зміні, її номер і можливість подивитися звідки було відведено реліз.

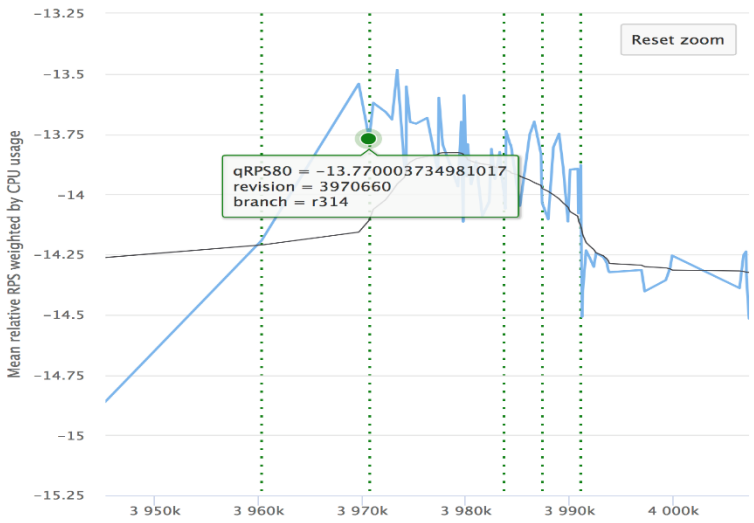

Рисунок 1.1 – Результат RPS, отриманий командою Яндекс

1.2 Огляд методів EDM

1.2.1 Методи категорії прогнозування

Прогнозування – це один із найвідоміших категорій EDM методів в огляді Baker & Yacef. Мета цієї категорії прогнозування заключається у тому, щоб видати один такий аспект даних (вихідна прогнозована змінна) із певної комбінації других аспектів даних по значенням (незалежних змінних або ж предикторів).[7]

Якщо змінна, що є вихідною, отримує безперервні значення, то у цьому випадку ми працюємо безпосередньо з регресією – це пошук залежностей між вхідними та вихідними змінними. У тому випадку, якщо вихідна змінна отримує певний дискретний кінцевий набір значень (або ж класів), тоді говоритиметься про необхідність класифікації – це те, коли у залежності від значень змінних, що входять, змінна вихідна стосується до такого чи другого класу. [7]

Прогнозування передбачає розмічену частину із всіх даних. Це означає, що для цих даних існують значення вхідних змінних та відповідні цим даним значення змінної, що створюється на виході.

Після навчання на такому наборі даних, алгоритм класифікації або ж регресії дозволяє передбачити значення вихідної перемінної для немаркованих (нових) даних.

Так як приклад, беручи за основу дані щодо активностей користувачів в електронній системі, час виконання їх запитів, можна спочатку розрахувати навантаження на сервер та кількість користувачів, що сервер зможе витримати, а потім спрогнозувати те ж навантаження для вже реальної системи з більшою кількістю користувачів.

1.2.2 Виявлення структури

Такі алгоритми, як structure discovery, по-іншому, це алгоритми виявлення структури, беруть за мету виявлення в отриманих даних структуру, до того ж без використання довільних апріорних уявлень про цю структуру. Найбільш відома група схожих алгоритмів - це алгоритми кластеризації.

Кластеризацією можна назвати логічне продовження ідеї класифікації. Особливістю кластеризації є те, що класи об'єктів вхідних даних на початку не є виявленими, при тому що під час класифікації до того як перейти до реалізації моделі треба представики конкретну підмножину даних.

Результат кластеризації - відокремлення множини об'єктів на певні групи схожих у довільному сенсі об'єктів.[7]

1.2.3 Виявлення взаємовідносин

Метою relationship mining, що значить виявлення взаємовідносин, полягає в тому, щоб висвітлити взаємозв'язок усіх змінних в наборі даних із великою кількістю цих змінних.

Так до прикладу, можна визначити змінні, що найбільш сильно пов'язані з тією змінною, яка нас і цікавить, або в якої з пар змінних, що пов'язані між собою зв'язок сильніший, ніж в усіх інших.

Найчастіше виявлення взаємовідносин в Education Data Mining має використання у формі пошуку асоціативних або ж сполучних правил (назва - association rule mining) або у формі пошуку послідовних шаблонів (назва - sequential pattern mining).

Мета пошуку у сполучних правил – це знаходження правила, що приймає вигляд "if-then", що відповідає на те, що у випадку якщо (if) одні змінні мають довільне безлічі значень, то тоді (then) інша змінна отримає якесь конкретне значення.

Таким чином можна привести приклад: якщо в транзакції існує набір товарів А, то можна зробити певний висновок, що в цій же самій транзакції мусить бути набір товарів В.

Подальший пошук сполучних правил – це пошук послідовних шаблонів. Він призначений для визначення взаємовідносин між проявами послідовних в часі подій (таким чином, якщо якийсь користувач системи придбав товар А, то колись згодом він придбає і товар В, тому що був зацікавлений у товарах хоч раз). [7]

1.2.4 Методи, специфічні для EDM

Наряду із традиційними звичними для аналізу даних методами, в науці EDM використовуються також і специфічні методи, такі як: відкриття із використанням моделей (Discovery with Models) і перегонка даних для подальшого прийняття рішень користувачем, адміністратором (Distillation of Data for Human Judgment).

Відкриття за допомогою моделі передбачає, що реалізація прогнозованої моделі це не конкретна мета роботи, а лише проміжна ланка у рішенні іншої проблеми, що знаходиться поряд із основною.

Перегонка даних для подальшого прийняття рішень людиною має таке визначення: дані необхідно перетворити та представити у такому вигляді, щоб користувач зміг їх сприймати.

Методи специфічні для EDM націлені на використання певних переваг візуального способу для подання інформації, що була попередньо оброблена методами аналізу даних (як приклад, для зменшення розмірності даних).

1.3 Закон Літтла

Щоб обчислити пропускну здатність можна скористатися законом Літтла - простою, але дуже корисною мірою, яку можна застосовувати дуже широко. Розглянемо простий чорний ящик, зображений на рисунку 1.2.

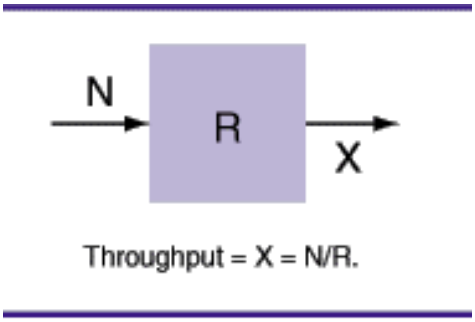

Рисунок 1.2 – Закон Літтла

Закон Літтла говорить, що якщо в цьому вікні міститься в середньому N користувачів, і середній користувач проводить у ньому R секунд, то пропускна здатність X цього поля приблизно X = N/R. [15]

Нехай в систему масового обслуговування надходять N заявок протягом відрізка часу [0, T]. Деяка частина заявок за час T будуть обслужені і покинуть систему, інша частина може залишатися в системі. на момент часу T. Нехай кожна i-я заявка до моменту часу T перебувала в системі проміжок часу wi. Сумарний час S, проведений усіма заявками в системі обчислюється так [12]:

$$S = \sum_{i=1}^{N} w_i \quad (1.1)$$

Середній час знаходження однієї заявки в системі за проміжок часу від 0 до T є відношенням сумарного часу S до загальної кількості заявок N, що надійшли в систему за цей час: W = S / N. Звідки S = W N.

З іншого боку, сумарний час S можна обчислити іншим способом.

У кожен момент часу τ на проміжку часу [0, T] в системі знаходиться кінцеве число n (τ) заявок. Очевидно $0 \leq n(\tau) \leq N$. Функція n (τ) не є безперервною і має k точок розриву на [0, T], де $k \leq 2N$. Нехай τj і τj + 1 - сусідні точки безперервності, і τj + 1 - τj = tj. Можна припустити, що 0 = τ0, T = τk + 1. Нехай nj - число заявок, що знаходяться в системі на проміжку часу tj. Тоді сумарний час, проведений заявками в системі за час від 0 до T, буде

$$S = \sum_{j=0}^{k} n_j t_j \quad (1.2)$$

Середнє число L заявок в системі можна обчислити, розділивши сумарний час, проведений всіма заявками в системі, на проміжок часу T.

$$L = \frac{\sum_{j=0}^{k} n_j t_j}{\sum_{j=0}^{k} t_j} = \frac{S}{T} = \frac{WN}{T} \quad (1.3)$$

Позначимо за λ середнє число заявок, що прийшло в систему за одиницю часу: λ = N / T. Ця величина λ називається інтенсивністю (або потужністю) вхідного потоку. У підсумку ми отримали формулу Літтла L = λW

$$L = \lambda W \quad (1.4)$$

Таким чином, середнє число заявок, що знаходяться в системі, дорівнює середньому числу заявок, що надходять в систему за одиницю часу, помноженому на середній час, що проводиться однією заявкою в системі.

Якщо система масового обслуговування стаціонарна, тобто, середнє число заявок, що надходить в систему за одиницю часу і середній час обслуговування кожної заявки не змінюється, і при цьому продуктивність системи перевершує потужність вхідного потоку, то середнє число заявок, що знаходяться в системі, і середній час перебування заявки в системі не залежить від проміжку часу T.

1.4 Поняття capacity та scalability.

Кількість користувачів складно прогнозована величина.

Необхідно зрозуміти, чи впорається електронна система в дні надактивності користувачів. Для цього необхідно оцінити запас можливостей для безперебійної і коректної роботи.

Capacity – максимальна кількість, що може бути витримана або отримана.Тестування потенційних можливостей (capacity testing) – є підвидом тестування масштабованості. Якщо в тестування масштабованості основне питання, яке ставиться: наскільки добре справляється система зі зростаючою кількістю користувачів (навантаження). То в даному випадку питання звучить наступним чином: скільки користувачів (з яким навантаженням) може працювати з системою при цьому час відгуку і інші параметри продуктивності повинні знаходиться в межах допустимих значень? Даний вид тестування дозволяє визначити стратегію масштабування і взагалі зрозуміти чи варто масштабувати систему. [13]

Масштабованість (scalability) - це властивість системи обробляти все більший обсяг роботи, додаючи ресурси до системи. [14]

Тут кількість користувачів або генераторів навантаження (N) збільшується за фіксованою конфігурацією обладнання. У цьому випадку кількість користувачів виступає як незалежна змінна, тоді як конфігурація процесора залишається фіксованою протягом діапазону вимірювань навантаження користувача. Це найпоширеніша ситуація в середовищах тестування навантаження, де використовуються такі інструменти, як LoadRunner або Apache JMeter.

Система називається масштабованою, якщо вона здатна збільшувати продуктивність пропорційно додатковим ресурсам.

Масштабованість можна оцінити через відношення приросту продуктивності системи до приросту використовуваних ресурсів. Чим ближче цей показник до одиниці, тим краще.

Також під масштабованістю розуміється можливість нарощування додатковив ресурсів без структурних змін центрального вузла системи.

В системі з поганою масштабованістю додавання ресурсів призводить лише до незначного підвищення продуктивності, а з деякого «порогового» моменту додавання ресурсів не дає ніякого позитивного ефекту.

1.5 Статистичні гіпотези та гіпотеза щодо рівності середніх значень

Гіпотезу щодо закону розподілу статистичної сукупності або щодо числових параметрів розподілів, що є відомими, називають статистичною.[24]

Можна виділити дві головні статистичні гіпотези:

1. Щодо закону розподілу.
2. Щодо числових характеристик двох розподілів.

У цих двох гіпотезах розповідається про генеральні сукупності та висуваються такі гіпотези на підставі аналізу відбірних даних. Це є розповсюдженою схемою, однак, вона не є єдиною та й істують інші статистичні гіпотези.

Гіпотезу, що є розглянутою вище, називають нульовою та позначають через H0. В основному, це є найбільш очевидною та правдоподібною гіпотезою, однак, на противагу до неї постійно береться до уваги альтернативна або ж конкуруюча гіпотеза H1.

Так як нульова гіпотеза є висунутою на підставі даних із вибірки, то вона може бути правильною або ж неправильною. Таким чином через це вона підлягає статистичній перевірці.

Перевірку треба проводити використовуючи статистичні критерії, що можна назвати випадковими спеціальними величинами, які можуть приймати будь-які дійсні значення.

У результаті перевірки нульову гіпотезу або приймають, або відкидають на користь конкуруючої. При цьому може бути таке, що допускаються помилки двох типів.

Суть помилки першого роду: гіпотеза H0 буде відкинутою, хоча вона може виявитися правильною. Для виділення ймовірності допущення такої помилки віднесли параметр – рівень значущості – та позначили його знаком α.

Суть помилки другого роду: гіпотеза H0 буде значитись прийнятою, але насправді вона може виявитися неправильною. Буквою β позначають ймовірність отримати помилку такого роду.

Значення 1-β називають потужністю критерію – це ймовірність відторгнення неправильної гіпотези. Рівень значущості треба задавати самостійно та найбільш часто вибираються три значення: $\alpha = 0,1$; $\alpha = 0,05$ та $\alpha = 0,01$.

Треба правильно вибрати рівень значущості тому що при зменшенні ймовірності α (відторгнення правильної гіпотези) зростає ймовірність β (прийняття неправильної гіпотези). Тому в самому початку потрібно правильно вибрати співвідношення ймовірностей α та β. До того ж враховується тяжкість наслідків, що можуть зробити помилка першого та другого рівнів.

Усі статистичні гіпотези поділяються на два види:

1. Гіпотеза щодо розподілу статистичних сукупностей.
2. Гіпотеза щодо генеральної середньої нормального розподілу.
3. Гіпотеза щодо рівностей генеральних середніх кількох розподілів.
4. Гіпотеза щодо генеральної дисперсії нормального розподілу.
5. Гіпотеза щодо рівності двох нормальних розподілів генеральних дисперсій.
6. Гіпотеза щодо ймовірностей подій.

У даній роботі необхідно створити модель, яка б допомогла адміністратору електронної системи прийняти рішення щодо навантаженості сервера та прогнозувала кількість користувачів при якій сервер буде неспроможний обробляти запити клієнтів.

Розберемо більш детально гіпотезу щодо рівностей генеральних середніх кількох розподілів тому що саме вона буде необхідною при розробці моделі та представлення кінцевого результату щодо навантаженості сервера.

Для розробки моделі добре підійде формула для перевірки нульової гіпотези у випадку, якщо вибіркі незалежні та генеральні сукупності розподілені нормально та відомі їх дисперсії σ_x^2, σ_y^2.

У цьому випадку користуються статистичним критерієм, що наданий у формулі нижче.

$$Z = \frac{X-Y}{\sqrt{\frac{\sigma_x^2}{n}+\frac{\sigma_y^2}{m}}}, \tag{1.5}$$

Де X, Y – це випадкові значення вибіркових середніх.

Критична область однозначно визначається критичним значенням $z_{\text{кр}}$, що можна знайти із співвідношення для односторонньої області нижче, що є формулою Лапласа. У додатку А можна побачити таблицю значень функції Лапласа.

$$\Phi(z_{\text{кр}}) = \frac{1-2\alpha}{2}, \tag{1.6}$$

де $\Phi(z_{\text{кр}})$ – функція Лапласа, α – рівень значущості.

Знаходження $z_{\text{кр}}$ – критичне значення – це значення якоїсь випадкової величини Z, що є залежним від вибраного рівня значущості α та необовязково від інших параметрів. Критичне значення може визначати критичну область, що буває правосторонньою, лівосторонньою та двусторонньою. На рисунку 1.3 зображено варіанти критичних областей.

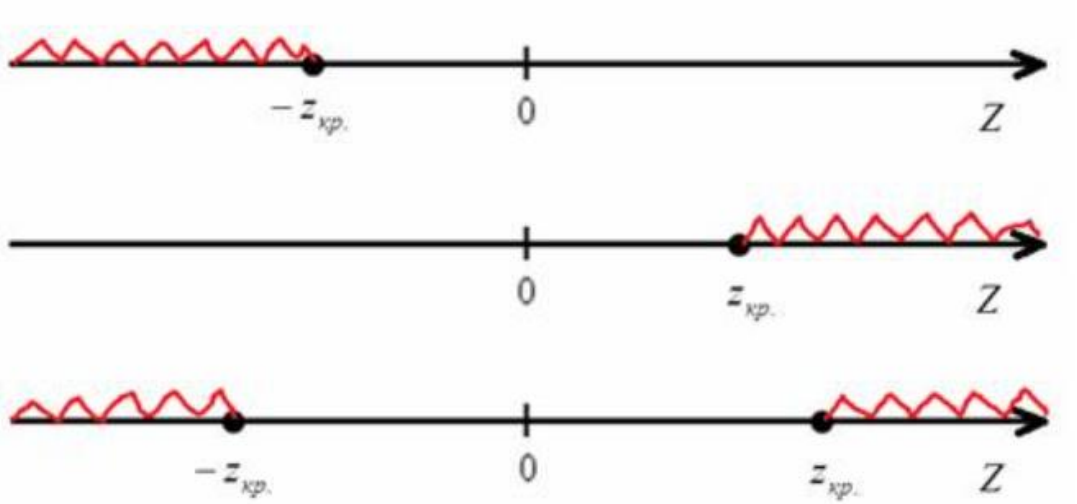

Рисунок 1.3 – Критичні області

Критична область – це зона, де відторгається нульова гіпотеза. Незаштрихована область – це зона, де гіпотеза приймається.

Наступним кроком треба розрахувати $z_{\text{набл}}$ за формулою 1.5, що наведена вище. Якщо це число не потрапляє у критичну область, то нульова гіпотеза приймається на рівні значущості α. У даному випадку із ймовірністю α був ризик не прийняти правильну гіпотезу.

Проте, не варто думати, що це 100 відсотків правильний результат тому що ймовірність β (прийняття невірної гіпотези) все ж існує.

Якщо знайдене число потрапляє в критичну область, то нульова гіпотеза не приймається на рівні значущості α. При цьому варто приймати до уваги те, якщо, наприклад, α = 0,05, то приблизно в 5 випадках із 100 є варіант відторгнути правильну гіпотезу (тобто зробимо помилку 1-го роду).

Підсумуючи вище написане, що стосується гіпотези щодо рівності середніх значень, якщо $z_{\text{набл}}$ не потрапляє у критичну зону, то гіпотеза H0: X = У приймається на рівні значущості α. Якщо ж навпаки потрапляє у критичну зону, то нульова гіпотеза не приймається у користь гіпотези H1, що є альтернативною.

1.6 Постановка задачі

У даній роботі необхідно створити модель, яка б допомогла адміністратору електронної системи прийняти рішення щодо навантаженості сервера та прогнозувала кількість користувачів при якій сервер буде неспроможний обробляти запити клієнтів.

Проблеми в реалізації такої моделі в тому, що не зрозуміло як вимірювати максимальну кількість користувачів, бо немає доступу до серверу електронного ресурсу та вхідні дані зберігаються тільки у вигляді лог-файлів.

У результаті роботи моделі потрібно відобразити вхідний лог-файл у вигляді зручного та на основі нього продемонструвати графічно навантаження на сервер по RPM та кількості користувачів.

Завдяки реалізації цієї моделі користувач зможе проаналізувати навантаження на сервер та також отримає максимальну кількість користувачів, що сервер здатний витримати у такому ж режимі, як і з досліженою кількістю запитів. Як наслідок користувач зможе прийняти рішення щодо роботи серверу та знеможливити його падіння для комфортної взаємодії студентів та викладачів з електронною системою.

Предметом дослідження було обрано запити до освітнього ресурса дистанційного навчання СумДУ, що відображають в собі лог-файл активності за деякий проміжок часу з певною кількістю параметрів.

Дані з вхідного лог-файлу, що можуть бути використані при розробці моделі, що допоможе в прийнятті рішення щодо навантаженості сервера:

1. Час, в який запит було відправлено. Наприклад, [2020-10-01T11:43:44.503779 #58894].
2. Час, за який запит виконано та його статус. Наприклад, Completed 200 OK in 2ms або Found in 190ms.
3. Ідентифікатор користувача. Наприклад, USER:12694.
4. Час роботи БД. Наприклад, ActiveRecord: 30.4ms.

Для реалізації моделі треба взяти вхідний лог-файл активності в електронній навчальній системі користувачів за деякий проміжок часу. Для тестування моделі можна буде вивантажити новий актуальний лог-файл.

2 ВИБІР МЕТОДУ ВИРІШЕННЯ

2.1 Вибір середовища та мови для розроблення моделі

Для того, щоб вирішити якою має бути модель, треба розібратися що таке десктопний додаток, а що таке веб-орієнтований додаток і який варіант найбільш прийнятний для вирішення задачі щодо розробки моделі, що зможе розрахувати кількість користувачів при якій сервер буде неспроможний обробляти запити клієнтів та вивести графік, що наглядно продемонструє масштабованість сервера.

Десктопна програма - клієнтське програмне забезпечення, що реалізує Windows Forms інтерфейс. Додаток інсталюється на робочу станцію користувача і запускається локально або запускається віддалено. Допускається варіант запуску такого додатка з використанням введення URL адреси в браузері, але від цього веб-клієнтом не стає, також як і завдяки запуску за допомогою різного роду емуляторів.[16]

Веб-клієнт - клієнтське програмне забезпечення, що представляє собою браузер і використовує http / https протоколи. Додаток не вимагає інсталяції або завантаження програмних модулів на робочу станцію користувача. Допускається установка додаткових загальносистемних бібліотек і використання захищених мережевих протоколів, і від цього десктопних додатком не стає.[16]

Що кожен з користувачів, власників системи, архітекторів і співробітників служб безпеки чекає від програмного продукту і клієнтського додатка:

1. Функціональність, простота використання і швидкодія;
2. можливість кастомізації і стилізації;
3. мобільність і легкість оновлення;
4. масштабованість і кроссплатформенність;
5. безпеку і надійність.

У кожній другій документації (якщо не частіше) в розділі технічних вимог можна помітити вимоги до наявності веб-клієнта або веб-доступу.

Як правило, обгрунтування такого запиту і, власне, плюси веб-орієнтованих додатків можна виділити наступні:

1. Мобільність (є можливість увійти в систему з будь-якого комп'ютера, що має доступ до інтернету і є до нього підключеним);
2. легкість розгортання та оновлення (перевстановлення програмних модулів не є необхідним та обов’язковим на комп’ютерах користувачів);
3. простота створення тестового і продуктового середовища (на сервері додатків розгорнуто два веб-додатки до однієї БД, таким чином, тестування нових версій програмного забезпечення окремими групами користувачів стає зручним і порівняно «безпечним», так як завжди можна повернутися до діючої версії системи звернувшись до неї за іншою адресою).

Безпека і надійність - дуже серйозні питання. Деякі організації принципово не хочуть і не пропонують можливість роботи в корпоративних системах за межами свого домену. Необхідність застосування засобів криптографічного захисту інформації та електронного підпису вже давно нікому доводити не треба. Для того, щоб використовувати такі технології, необхідно звертатися до зовнішніх бібліотек, і варто зважати на те, що більшість веб-додатків мають певні обмеження. Стабільність роботи самих браузерів також є потенційно вузьким місцем, і проблема в тому, що вплинути на це розробник бізнес-додатків може не завжди. Офлайн робота, власне кажучи, частіше і простіше реалізуються, якщо використовувати десктопні додатки.

Можливість змінити колір, іконки, логотипи, шрифти базових інтерфейсів - гарний бонус для клієнта, користувача. Десктопні програми можуть надавати можливість застосування колірної теми, налаштування окремих інтерфейсних елементів, але веб-додатки, застосовуючи каскадні таблиці стилів, з цим справляються явно краще. Можливість кастомізації визначається ступенем розвитку самого програмного продукту і тип клієнтської програми не має ролі.

Функціональність - найважливіша вимога до будь-якого програмного продукту.

Здавна вважається, що десктопні програми є більш ергономічними і функціональними. Якщо намагатися розробляти веб-клієнт з самого початкового рівня, то так воно і буде. Але з роками були розроблені цілі інтерфейсні бібліотеки, що дозволяють створювати надможливе:

1. Ієрархічні списки з якими можна переміщувати колонки і накладати фільтри;
2. функції drag & drop до будь-яких елементів з довільними візуальними ефектами;
3. ділова графіка і інтерактивні панелі;
4. додавання аудіо і відео програвачів;
5. обробка будь-яких подій.

Сучасний користувач комп'ютера більше часу проводить в браузері, ніж за роботою над десктопними додатками. І перший варіант роботи складніше йому не здається. Проте змога масштабувати в браузері, деяким категоріям користувачів, приносить значну користь.

Розробляючи веб-додатки з дотриманням стандартів можна сподіватися, що програмне забезпечення буде коректно працювати у всіх браузерах, хоча б, в першій п'ятірці.

З врахунком всіх плюсів веб-додатків було прийнято рішення реалізовувати модель саме так. До того ж, так як треба реалізувати її для електронної системи навчання СумДУ, що і є веб-орієнтованою системою, для адміністратора цієї системи буде зручно користуватися розробленою моделлю саме як компонентом. Адміністратор зможе вписати необхідний проміжок часу, за який треба проаналізувати навантаження сервера та просто натиснути кнопку, щоб отримати графік, що наглядно продемонструє масштабованість сервера та дасть зрозуміти при якій максимальній кількості користувачів в такому ж режимі сервер не витримає та перестане працювати.

Так як було вирішено додати компонент у якості розробленої моделі до вже існуючого веб-додатку, електронної системи навчання СумДУ, повстає питання щодо вибору мови для розробки моделі.

Є декілька найпопулярніших мов для веб-програмування, розглянемо та виберебо найоптимальнішу для реалізації веб-компоненти.

PHP - це скриптова мова, яка використовується для того, щоб швидко створити динамічні веб-сторінки. Це добрий вибір для frontend і backend розробників, він знаходиться за такими веб-гігантами, як Facebook і WordPress. PHP дозволяє швидко і легко розширювати веб-додатки і запускати веб-сайти з повторюваними серверними завданнями (наприклад, оновлювати новинні стрічки). Він має відкритий вихідний код і має гарний рейтинг серед компаній-початківців, медіа-агентств та електронної комерції - таких людей, які постійно займаються рекрутингом нових веб-розробників.[17]

Є наступний перелік переваг, які роблять широко застосовуваною мову PHP для розробки, що орієнтована на веб [18]:

1. розробка за допомогою PHP дає багато можливостей. При належному рівні володіння мовою програмування, за допомогою шаблонізатора можна створювати не тільки сценарії для веб-додатків, але й повноцінні програми. Існують і такі рішення, що дозволяють створювати на PHP мобільні додатки;
2. вивчення PHP не вимагає багато часу;
3. кроссплатформенність;
4. підтримка веб-серверів;
5. безкоштовне розповсюдження. Можливо, PHP не був би таким популярним для створення web-додатків, якби не був безкоштовно розповсюдженим так як і більшість інструментів для роботи з ним. Аналоги, які, в основному, можуть виконати ту ж саму роботу, зазвичай коштують дорожче;
6. має достатню різноплановість для web-розробки;

7. наявність навчальних матеріалів;
8. безперервний розвиток.

Проте, є й невеликий мінус: обробка статичного файлу html відбувається швидше, ніж інтерпретуємого файлу php. Різниця за часом на клієнті може становити близько однієї секунди. Якщо ж даних для обробки буде багато, то час обробки буде ще більшим.

SQL – це дуже важлива частина веб-розробки, що дозволяє отримувати конкретні дані з великих, наповнених, складних баз даних. Він користується серед великих компаній дуже часто. До списку таких компаній входить і Microsoft, тому це є розумним вибором для будь-якого розробника з високими амбіціями або ж є необхідність працювати з базами даних на регулярній основі.[17]

Проте, для реалізації моделі щодо підтримки рішень не є необхідністю створювати таблиці в базі даних, а простіше оброблювати вхідні дані «на льоту». Причина в тому, що вхідних значень велика кількість і в реалізації моделі вони будуть застосовуватись тільки невелику кількість разів. Видаляти дані з бази даних кожен раз перед новим аналізом навантаженості сервера – неправильний вибір. Накоплювати ці дані на сервері – також неправильний варіант, від цього сервер буде ще повільніше їх оброблювати.

JavaScript - мова інтерфейсу, що широко використовується для створення та розробки веб-сайтів, ігор та настільних додатків. Ця мова має застосування у всіх браузерах і може співпрацювати з такими програмами, що не розміщені в Інтернеті. Він підтримує як функціональні, так і об'єктно-орієнтовані стилі програмування, і в основному, це підхід до створення зручних додатків, інтерфейсів для фінального користувача. До того ж, вони виглядають добре.[17]

Javascript здобув популярність не просто так, а завдяки своїм безперечним перевагам, таким як [19]:

1. Незамінність для веб-розробки. Підтримка скриптів усіма популярними браузерами; повномасштабна інтеграція з серверною частиною та версткою сторінок (HTML + CSS).
2. Швидкість роботи та значна продуктивність. Javascript дозволяє частково обробляти веб-сторінки на комп'ютерах користувача без надсилання запитів до сервера. Це значно економить час і трафік, а також зменшує навантаження на сервер.
3. Потужна інфраструктура (екосистема). На початку, у перші 10 років цього не було. Пізніше кількість готових рішень у відкритому доступі зросла на стільки, що працювати з Javascript і його фреймворками стало зручно і із задоволенням.
4. Раціональність і простота застосування. Просте завдання є можливість вирішити за 5 хвилин, до того ж не треба робити зайву додаткову роботу. Для складних завдань є безліч варіантів вирішення і можна підібрати кращий та адаптувати.
5. Зручність для фінального користувача інтерфейсів. Заповнення форм, вибір дій, активація кнопок, перевірки введення, реагування на наведення або ж кліки миші. Це дає дуже гарний рівень юзабіліті для користувача.
6. Легкість освоєння мови.

Як будь-яка мова програмування, Javascript має й певні недоліки [19]:

1. Типізація, що є нестрогою та вільне трактування. Мова може ігнорувати видимі нестиковки. Має місце різна інтерпретація даних. Немає можливості раннього виявлення помилок. Всі недоліки виявляються вже на етапі роботи.
2. Немає підтримки віддаленого доступу. Тому мова не можна використовувати для мережевих додатків.

3. Доступність для зловмисників. У вільну скриптову мову найпростіше вбудувати фрагмент шкідливого коду, який може нашкодити користувачеві. Однак, з антивірусом шкоди не буде.

Треба відзначити, що розробники Javascript активно покращують мову, усуваючи багато недоліків. Вузьких місць стає все менше. Браузери постійно вдосконалюють роботу з JS та вихід HTML 5 дав новий поштовх до розширення можливостей скриптів.

Після етапу порівняння, однозначно можна сказати, що найліпшою мовою для розробки моделі буде Javascript з поєднанням із HTML + CSS. Вхідних даних буде велика кількість і потрібно буде їх оброблювати швидко. Також для роботи з даними будуть використані масиви. З цим також Javascript впорається на відмінно. Для обробки вхідних даних не треба створювати таблиці в базі даних та, помістивши всі значення туди, працювати з ними. Обробка вхідних даних має відбуватися «на льоту» і з цим Javascript теж впорається. Також ця мова програмування підтримує безліч корисних бібліотек у кількість яких входять такі, що допоможуть створити красиві та структуровані графіки для аналізу масштабованості системи.

2.2 Бібліотека ZingChart

ZingChart - корисний інструмент, що дозволяє створити інтерактивні та адаптивні графіки для будь-яких потреб. Ця бібліотека є швидкою та гнучкою. Вона дозволяє продуктивно працювати з великою кількістю даних та ефективно генерувати графіки зі значними обсягами даних.

Основні можливості можна виділити:

1. Підтримує більше 30 типів графіків і діаграм.
2. Повністю кастомізована, з настройками оформлення як в CSS.
3. Сумісна з jQuery, Angular, Node.js, PHP та ін.
4. Робота з даними в режимі реального часу, надшвидкий рендеринг наборів даних довільного обсягу.

5. Дані можна завантажувати через JSON, JS об'єкти, CSV, AJAX, JSON, MySQL.
6. Детальна і досить зручна документація по API.
7. Два види підтримки: безкоштовна і преміум через довідковий центр ZingChart, чат, Stack Overflow та електронну пошту.

Компанії, що користуються бібліотекою: Microsoft, Boeing, Apple, Adobe, Cisco, Alcatel, Google і ін.

Брендована ліцензія забезпечує повний доступ до бібліотеки ZingChart безкоштовно. Для комерційного використання потрібно отримати платну ліцензію (коштує від $199).

Для того, щоб швидко підключити ZingChart бібліотеку, треба додати CDN лінку, наприклад [20]:

```
<!DOCTYPE html>
<html>
 <head>
<!—Далі знаходиться лінка на скрипт для доступу до zingchart-->
<script src="https://cdn.zingchart.com/zingchart.min.js"></script>
</head>
<body>
<!—Далі знаходиться блок для розміщення графіку -->
<div id="ChartMY"></div>
</body>
</html>
```

Візуалізація діаграми здійснюється за допомогою методу zingchart.render ({...}). Мінімальними вимогами до цього методу є id та властивості даних. Наприклад [20]:

```
// Метод візуалізації
zingchart.render({
  id: 'ChartMY',
```

```
        data: myConfig,
});
```

Також за допомогою атрибутів треба додати функції до цієї бібліотеки. Мінімальними атрибутами для відображення діаграми є атрибути типу (наприклад, type: 'bar') та серії (сюди входять дані, що будуть відображатись на графіку).

Для графічного відображення навантаженості сервера, як варіант, можна використати діаграму, як на рисунку 2.1, підключивши бібліотеку ZingChart додаванням CDN лінки в блок <head> та за допомогою Javascript з поєднанням із HTML + CSS візуалізувати графік.

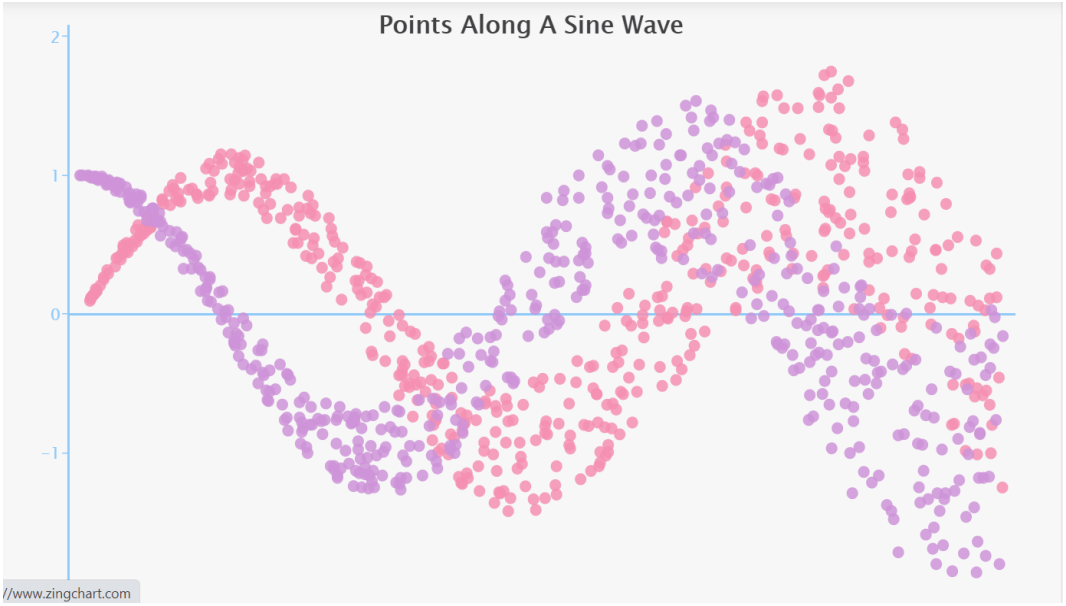

Рисунок 2.1 – Графік з бібліотеки ZingChart

2.3 Представлення даних в JSON

JSON - текстовий формат даних, що є наступним за синтаксисом об'єкта JavaScript. Цей формат даних можна застосовувати без поєднання із JavaScript навіть незважаючи на те, що вони по буквенному синтаксису дуже схожі. Також велика кількість середовищ програмування мають можливість аналізувати та генерувати JSON. [21]

JSON існує як рядок, який є необхідним щоб передати дані по мережі. Спочатку його треба перетворити у власний об'єкт JavaScript, якщо треба отримати доступ до даних. JavaScript створює глобальний об'єкт JSON, для якого присутні методи для перетворення між ними.

Об'єкт JSON є можливість зберегти у власному файлі, що в загальному є текстовим файлом з розширенням .json і MIME type application / json. [21]

JavaScript надає наступні методи:

1. JSON.stringify щоб перетворити об'єкти в JSON.
2. JSON.parse щоб перетворити JSON знову в об'єкт.

JSON підтримує наступні типи даних:

1. Об'єкти {...}.
2. Масиви [...].
3. Примітиви:
    1) числа;
    2) рядки;
    3) логічні значення true / false;
    4) null.

Формат представлення даних JSON має такі переваги:

1. зручні і швидкі в роботі методи, які призначені для конвертації або ж парсинга рядків JSON в об'єкт JavaScript і навпаки;
2. проста і зрозуміла структура даних;
3. дуже маленький розмір у порівнянні з іншими форматами даних (наприклад XML). Це пояснюється тим, що формат JSON містить форматування, що є мінімально можливим. Це значить, що для того, щоб його написати, треба використовувати лише кілька спеціальних знаків. Це є значною перевагою, тому що дані представлені в форматі JSON будуть завантажуватися більш швидко, ніж, тоді, коли вони були б представлені в якихось інших форматах.

Робота з даними JSON після парсинга здійснюється як зі звичайним об'єктом JavaScript.

Для перебору елементів в об'єкті можна використовувати цикл for..in:

```
for (key in person) {
  if (person.hasOwnProperty (key)) {
    // ключ = key
    // значення = person [key]
    console.log ("Ключ =" + key);
    console.log ("Значення =" + person [key]);
  } // якщо об'єкт person має ключ (якщо у person є властивість key)
} // перенабрати всі ключі (властивості) в об'єкті
```

2.4 Методи роботи з парсингом тексту в JavaScript

Для того, щоб зчитати з файлу та зберегти зчитані дані у перемінні в мові JavaScript, достатньо скористатися FileReader.

API спеціально розроблений так, щоб бути схожим на XMLHttpRequest, так як обидва, в загальному є методом завантаження даних із ресурсів, що є поза браузером (зовнішні). Читання робиться асинхронно, для того щоб не блокувати браузер.

Існує декілька форматів, в які FileReader може представляти дані з файлу, треба задати формат тоді, коли файл відкривається для читання. Щоб здійснити читання можна скористатися викликом одного з наступних методів [23]:

1. readAsText () – метод надсилає вміст файлу як plain text
2. readAsBinaryString () – метод надсилає вмістовність файлу у вигляді рядка закодованих даних у двійковій системі (можна замість нього скористатися методом readAsArrayBuffer ())
3. readAsArrayBuffer () - метод надсилає вмістовність файлу у вигляді ArrayBuffer (пасує для даних двійкових, наприклад, зображення)
4. readAsDataURL () - метод надсилає вмістовність файлу як data URL

Кожен з перелічених методів ініціює читання файлу і є схожим на метод send () об'єкта XHR, що ініціює HTTP запити. Таким чином, перш ніж почати роботу, необхідно встановити onload - обробник завантаження подій.

Обробник подій onload викликається тоді, коли файл вдалось успішно прочитати в той час, як обробник OnError викликається, якщо файл взагалі не був прочитаний з якихось причин. Через event.target об'єкт типу FileReader буде доступний всередині обробника події. Якщо читання даних буде мати успішний результат, в поле result, буде записано вміст файлу, якщо ж неуспішний - інформація про помилки.

Дані, що були прочитані з файлу можна помістити в JavaScript у вигляді масивів.

Масивом називається впорядкована колекція значень. Значення в масиві – елементи і кожен такий елемент має свою числову позицію в масиві, яка називається індексом. В мові JavaScript масиви є нетипізованими: елементи масиву можуть бути будь-якого типу. Особливістю є те, що різні елементи одного і того ж масиву можуть бути з різними типами. До масиву можуть включатися об'єкти або ж інші масиви. Це дозволяє створювати масиви масивів або масиви об'єктів, тобто складні структури даних.

Відлік індексів масивів в мові JavaScript розпочинається з нуля. Для масивів застосовуються 32-бітові цілі числа - найперший елемент масиву є з індексом 0. Масиви в JavaScript можна назвати динамічними тому що вони мають можливість збільшуватися та зменшуватися в розмірах, якщо це є необхідним; немає потреби оголошувати фіксовані розміри масивів при їх створенні або повторно розподіляти пам'ять тоді, коли треба змінити їх розміри.

Масиви ж в мові JavaScript - це спеціалізована форма об'єктів, а індекси, що є цілими числами, цих масивів виступають не просто іменами властивостей.

Для того, щоб мати доступ до елементів масиву, треба скористатися оператором []. Зліва від дужок повинно бути посилання на масив. Усередині дужок має перебувати довільний вираз, що повертає невід'ємне ціле значення.

Цей синтаксис придатний як для читання, так і для запису значення елемента масиву.

Особливість масивів полягає в тому, що при використанні імен властивостей, які є невід'ємними цілими числами, масиви автоматично визначають значення властивості length. Наприклад, вище був створений масив arr з єдиним елементом. Потім були привласнені значення його елементів з індексами 1, 2 і 3. В результаті цих операцій значення властивості length масиву змінилося і стало рівним 4.

Всі такі індекси називаються іменами властивостей, але тільки властивості з іменами, що представлені цілими числами є індексами. Всі масиви є об'єктами, тому можна додавати до них властивості з довільними іменами.

Привласнення значення новим індексам - найпростіший спосіб, щоб додати елементи в масив. Для того, щоб додати один або більше елементів в кінець певного масиву можна використати інший метод - push ():

```
var arr = []; // Створення порожнього масиву
arr.push ('zero'); // Додавання значення в кінець
arr.push ('one', 2); // Додавання двох значень в кінець масиву
```

Додати елемент в кінець масиву можна й іншим способом - присвоєнням значення елементу arr [arr.length]. Для того, щоб вставити елемент в початок масиву можна застосувати метод unshift(). При використанні цього способу, елементи, що існують в масиві, переміщаються на інші позиції, що мають більші індекси.

Для того, щоб видалити елементи з масиву (як звичайні властивості об'єктів), можна скористатися оператором delete.

Справжні багатовимірні масиви JavaScript не підтримує, але дозволяє гарно імітувати їх за допомогою масивів з масивів. Для того, щоб мати доступ до елементів даних в масиві з масивів треба двічі використати оператор [].

Метод .parse() об'єкта JSON у JavaScript – це глобальна функція, яка призначена для аналізу, або ж парсингу у форматі JSON.

Метод .parse(), якщо необхідно, перетворює і повертає значення, що були отримані після етапу аналізу. Таким чином можна отримати елементарне значення, що було отримано у результаті аналізу переданого рядку в форматі JSON. Також можна отримати об'єкт або ж масив. [22]

Для того, щоб перетворити елементарні значення, об'єкти, або масиви в рядок в форматі JSON можна скористатися глобальною функцією JSON.stringify(). За допомогою цієї функції серіалізуются об'єкт, елементарне значення або масив.[22]

Синтаксис є наступним:

JSON.parse(text);

JSON.parse(text, reviver);

text - рядок

reviver - функція

У таблиці 2.1 можна побачити значення параметрів синтаксису JSON.stringify().

Таблиця 2.1 – Значення параметрів JSON.stringify()

Параметр	Опис
text	Рядок, що аналізується у форматі JSON. Це є обов'язковим параметром.
reviver	Функція, яка призначена для перетворення значення, що було отримано в результаті аналізу перед тим як повернутися з методу. Це є додатковим параметром. У результаті виконання функції буде передано декілька аргументів: Першим аргументом є ім'я властивості об'єкта, або ж індекс масиву у вигляді рядка. Другим аргументом є елементарне значення властивості об'єкта, або ж елемента масиву.

	Значення, яке функція повертає, буде новим значенням заданої властивості. Якщо вона поверне таке саме значення, яке було надано в другому аргументі, то значення цієї властивості не буде замінене. Якщо значення, що було повернене функцією - undefined, або ж функція взагалі не поверне ніякого значення, то ця властивість буде видалена з об'єкту або ж масиву перед тим як метод JSON.parse () поверне його. Функція викликається як метод об'єкта або масиву, що містить елементарне значення, за винятком того випадку, коли рядок, що аналізується є представленим як елементарне значення, а не об'єкт або масиву. В такому випадку елементарне значення збережеться у новоствореному об'єкті в властивості з іменем, що є порожнім рядком, і така функція буде викликана до цього новоствореного об'єкта з нового рядка в першому аргументі, а в другому буде розташовано елементарне значення.

2.5 Представлення кінцевого результату

Після того, як користувач, а в нашому випадку, адміністратор електронної системи навчання СумДУ, завантажить файл з логами, за допомогою JavaScript дані з файлу будуть розпарсені та відображені за допомогою бібліотеки ZingChart у вигляді графіків.

Всього графіків буде декілька.

Перший буде відображати залежність кількості оброблених сервером пакетів від середньої швидкості цієї обробки. Це допоможе зрозуміти наскільки електронний ресурс, що досліджується, є масштабованим. Одна точка буде відповідати кількості пакетів, оброблених за 5 хвилин. Цей параметр можна буде змінити за необхідністю, якщо треба буде мати більшу кількість точок.

Проте, при збільшенні кількості точок можна буде бачити велику розбіжніть між ними, тобто з’явиться так званий шум. Таким чином, якщо всі точки будуть знаходитись на графіку у вигляді однієї горизонтальної лінії, то це буде означати, що система працює добре і вона є добре масшабованою. Якщо ж точки будуть розміщені хаотично або будуть розташовані у вигляді зростаючої лінії, то це вже буде вказувати на те, що є проблеми та затримки при паралельній обробці запитів. Така система буде називатись погано масштабованою і адміністратору треба буде замислитись над виправленням помилок, закупівлі додаткового обладнання або ж оптимізації коду для обробки даних в електронній системі.

Другий графік буде відображати залежність середньої швидкості обробки запитів від кількості унікальних користувачів, що відправляли запити до електронної системи. У цьому графіку одна точка буде відповідати певній кількості унікальних користувачів за той же самий проміжок часу, як і в першому графіку – 5 хвилин, до середньої швидкості обробки запитів цих унікальних користувачів.

У результаті адміністратор буде бачити два графіки, що описані вище, за допомогою яких він може мати змогу проаналізувати навантаженість на сервер.

Середнє число L заявок в системі можна обчислити, розділивши сумарний час, проведений всіма заявками в системі, на проміжок часу T.

$$L = \frac{\sum_{j=0}^{k} n_j t_j}{\sum_{j=0}^{k} t_j} = \frac{S}{T} = \frac{WN}{T} \quad (2.1)$$

Якщо привести цю формулу до такої, що може бути використана під час проектування моделі (зважаючи на те, що логи розділені для аналізу по 5 хвилин), отримаємо:

$$RP5M = \frac{300000}{T} * W, \text{де} \quad (2.2)$$

300000 – кількість мс в 5 хвилинах,

T – сумарний час виконання запитів сервером за 5 хвилин,

W – кількість пакетів, що обробляється за 5 хвилин.

В результаті отримаємо максимальну кількість пакетів, що зможе бути оброблена сервером за 5 хвилин у такому режимі.

Для того, щоб показати навантаженість сервера, візьмемо середню кількість пакетів для кожного із режиму навантаження з першого графіку (відображає залежність кількості оброблених сервером пакетів від середньой швидкості цієї обробки). Перша точка – середня швидкість обробки пакетів при низькому навантаженні, друга – при типовому навантаженні, а третя – при високому навантаженні сервера. Отримані значення максимальної кількості пакетів, що зможуть бути оброблені сервером, будуть відображені у заключному результаті щодо навантаженості.

Таким чином можна розрахувати максимальну кількість користувачів, що сервер може бути здатним витримати у такому ж режимі, що й аналізується. Візьмемо точку із найбільшою сумарною швидкістю обробки запитів для якоїсь кількості унікальних користувачів, що задавали ці запити за 5 хвилин. Та за допомогою формули Літтла (формула 2.1) буде розраховано максимальну кількість унікальних користувачів. Замість кількості оброблених пакетів буде братися кількість унікальних користувачів. Цей результат буде також виведено у результаті щодо навантаженості для подальшого аналізу фінальним користувачем.

Також буде представлена на першому та другому графіках лінія тренду – лінія із середніми значеннями швидкості обробки запитів, щоб наглядно побачити, що середня швидкість обробки запитів збільшується або ні у залежності від кількості оброблених пакетів або ж від кількості унікальних користувачів, що надсилають запити у систему. Якщо масштабованість сервера добра, то лінія тренду буде прямою горизонтальною.

Для того, щоб довести, що середні значення швидкості обробки запитів рівні для низького, типового (середнього) та підвищеного навантаження для першого графіку, всі точки будуть розподілені на 3 області (одна точка відповідає кількості пакетів, оброблених за 5 хвилин).

Перша область – низьке навантаження на сервер, друга – типове, третя – високе навантаження.

Потім для кожної з областей буде знайдено дисперсію, загальну кількість точок, середнє навантаження.

У результаті за допомогою гіпотези щодо рівності середніх значень (формула 1.5) буде доведено, що середні навантаження рівні у трьох областях, або ж навпаки – різняться. Результат доведення гіпотези буде також виведено для фінального користувача, щоб розуміти масштабованість сервера. Якщо середні значення низького, типового та високого навантаження будуть різнитися, тобто теорема про середні значення буде відхиленою, то навчальна система буде вважатися немасштабованою.

Фінальний результат буде у вигляді текстового повідомлення для фінального користувача під графіком. Таким чином він буде містити в собі інформацію щодо доведення гіпотези про середні значення, результатом якої є висновок масштабована система чи ні, та максимальна кількість пакетів та користувачів, що сервер здатний витримати у такому ж режимі, що було в лог-файлі з даними, які аналізувалися.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Підключення вхідного файлу, нормалізація даних для подальшої роботи

Як вже було висвітлено у постановці задачі, необхідно брати в аналіз для реалізації моделі вхідний лог-файл з даними щодо активності студентів на навчальному ресурсі СумДУ. Для того, щоб реалізувати модель, буде застосовуватися JavaScript та фінальний результат можна буде побачити у веб-браузері. Це буде дуже зручно для фінального користувача – адміністратора. Такий веб-орієнтований додаток можна встроїти у панель адміністратора навчального ресурса для комфортного аналізу навантаженості сервера.

Для того, щоб користувач мав можливість додати лог-файл для аналізу у розроблений веб-додаток, була реалізована кнопка для вибору файла, як зображено на рисунку 3.1 та подальше завантаження файлу. Загалом, якщо впровадити рішення у навчальний ресурс, можна отримувати дані для аналізу не через вивантажений заздалегідь лог-файл, а шляхом впровадження декількох полів для вводу дати та запиту на сервер для отримання тих самих даних із активністю користувачів.

Для реалізації завантаження файлу було додано наступний синтаксис:

<input type="file" id="infile">.

За допомогою нього при натисканні на кнопку, відкривається вікно у якому можна вказати на файл, що користувач бажає відкрити.

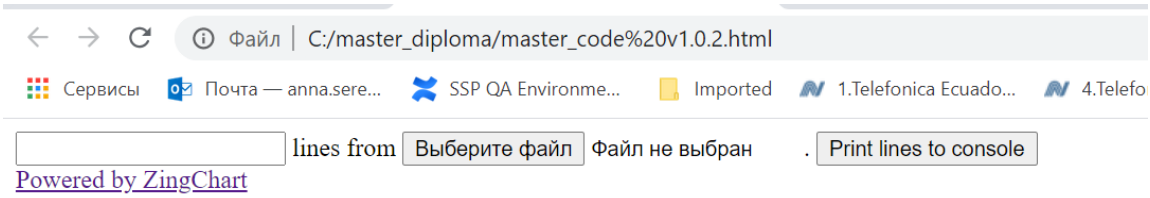

Рисунок 3.1 – Початкова сторінка веб-додатку

Також на рисунку 3.1 було додано кнопку для друку логів до консолі. Це може знадобитися адміністратору, щоб спостерігати за працездатністю додатку.

Таке рішення було додано за допомогою наступного синтаксису:

<input type="button" id="start" value="Calculation">

Отже, вхідний лог-файл було зімпортовано для подальшого його парсингу та роботи з даними у JavaScript за допомогою наступної функції.

```
document.getElementById('start').onclick = function(){
   var file = document.getElementById('infile').files[0];
   if (!file) {
   console.log('No file selected.');
   return;
   }
```

Для подальшого зчитування файлу було створено функцію function readSomeLines(file, maxlines, forEachLine, onComplete), що містить в собі FileReader. Цей об’єкт дозволяє читати веб-додаткам те, що знаходиться у файлах або ж у буферах даних, що зберігаються у користувача на комп’ютері, асинхронно. Для цього треба скористатися об’єктом File. За допомогою нього задамо лог-файл для читання.

Об’єкти File отримуються за допомогою об’єкта FileList, який як результат повертається після вибору користувачем файлів із використанням елементу <input>.

FileReader, у свою чергу, має такі обробники подій, що були задіяні в написанні зчитувача з лог-файлу:

1. FileReader.onload – обробник, що спрацьовує, якщо операція читання з файлу прошла успішно, використовується для обробника load. У цьому обробнику було реалізовано розбиття файлу на окремі лінії та на порції у розмірності 50 кб. Без цієї реалізації файли з великою кількістю даних просто не зчитувалися та, відповідно, не оброблювалися.
2. FileReader.onerror – обробник, що спрацьовує тоді, коли при читанні файлу виникає помилка, використовується для обробника error.

Для того, щоб зрозуміти, що всі дані було прочитано з вхідного файлу, було створено фінкцію seek().

Є два варіанти, коли зчитування файлу зупиниться. По-перше, тоді, коли максимальна кількість ліній, що була задана, буде вже прочитаною. По-друге, якщо файл вже було повністю зчитано.

Тому, фінкція seek() необхідна для того, щоб перевірити чи треба закінчувати читати файл та, якщо ні – продовжити зчитування наступних 50 кб вхідних даних або меншої кількості, що залишилась в кінці лог-файлу. Якщо зупиняється зчитування вхідного лог-файлу, то буде запущено функцію onComplete(), що відобразить результат кінцевому користувачу після опрацювання вхідних даних.

Отримування необхідних даних із вхідного файлу та подальша їх обробка виконується функцією function readSomeLines(file, maxlines, forEachLine, onComplete) та відбувається методом парсингу.

Так, наприклад, із лог-файлу необхідно отримати наступні значення, що використовуються при розробці моделі, що допоможе в прийнятті рішення щодо навантаженості сервера:

1. Час, в який запит було відправлено. Наприклад, [2020-10-01T11:43:44.503779 #58894].
2. Час, за який запит виконано та його статус. Наприклад, Completed 200 OK in 2ms або Found in 190ms.
3. Ідентифікатор користувача. Наприклад, USER:12694.
4. Час роботи БД. Наприклад, ActiveRecord: 30.4ms.

Тому за допомогою парсингу було отримано дані вище.

Для прикладу, для того щоб розрахувати кількість відправлених до серверу запитів та сумарну швидкість обробки запитів, було застосовано parseInt() та вирізання зі строки.

```
if (line.match("Completed")) {
request_times_counter += parseInt(line.substring(line.indexOf('in') + 3, line.indexOf('ms')));
```

total_times_counter += parseInt(line.substring(line.indexOf('in') + 3, line.indexOf('ms')));

}

Застосована функція parseInt() оброблює строку у якості аргумента та повертає число ціле згідно до вказаного основою системи числення.

Синтаксис цієї функції наступний:

parseInt(string, radix).

Параметри:

1. string – це значення, що буде оброблюватися функцією, пропуски на початку лінії не враховуються та не оброблюються;
2. radix – це ціле число, що знаходиться в діапазоні між 2 та 36 та представляє собою основу системи числення строки string; цей параметр є необов'язковим.

Метод substring () у свою чергу повертає підрядок рядка, що знаходяться між двома індексами, або від одного індексу і до самого кінця рядка.

Синтаксис цього методу наступний:

str.substring (indexA [, indexB]).

Параметри:

1. indexA – це ціле число, що знаходиться у діапазоні від 0 до довжини рядка, що визначає зміщення в рядку першого символу, котрий буде включений в підстроку, що повернеться після вирізання;
2. indexB – це ціле число, що знаходиться у діапазоні від 0 до довжини рядка та знаходить зміщення в рядку першого символу, що не буде включений в підстроку, що повернеться; цей параметр є необов'язковим.

Таким чином із рядка лог-файлу I, [2020-10-01T11:43:44.563683 #10642] INFO -- : Completed 200 OK in 2ms (Views: 0.2ms | ActiveRecord: 0.0ms) вилучено значення швидкості обробки запиту – 200.

Для отримання дати було також застосовано метод substring():

var datetime_str = line.substring(4,30);

Із прикладу вище за допомогою методу отримаємо дату 2020-10-01T11:43:44.563683. Для отримання дати у форматі, з яким можна буде працювати у JavaScript, було створено новий об’єкт типу Date:

var datetime = new Date(datetime_str);

Потім за допомогою методів getMinutes() та getHours() отримуємо години та хвилини відповідно.

var minute = datetime.getMinutes();

hour = datetime.getHours();

Ці дані будуть далі аналізуватися та розроблюватиметься графік для аналізу масштабованості навчальної системи.

3.2 Підключення бібліотеки ZingChart

Для того, щоб підключити ZingChart бібліотеку, було додано CDN лінку в блок <head> таким чином:

<script src="https://cdn.zingchart.com/zingchart.min.js"></script>.

Наступним кроком було задавання розмірів для майбутнього графіку, обрано для відображення всі 100% для висоти та ширини. Це було реалізовано наступним чином у блоку <head>:

```
<style>
    html,
    body,
    #myChart {
      height: 100%;
      width: 100%;
    }
  </style>.
```

Для того, щоб видалити спеціальний водний знак ZingChart бібліотеки необхідно придбати ліцензію та записати отриманий код у перемінну ZC.LICENSE для кожної сторінки із графіком. Наприклад,

ZC.LICENSE = ["569d52cefae586f634c54f86dc99e6a9"];

Також за допомогою атрибутів наступним кроком буде додавання функції до ZingChart бібліотеки. Мінімальними атрибутами для відображення діаграми є атрибути типу (наприклад, type: 'bar') та серії (сюди входять дані, що будуть відображатись на графіку).

Для реалізації графіку для поточного завдання буде ідеальним відображення графіку зі змішаним типом, type: 'mixed'.

3.3 Реалізація сторінки та графіків для аналізу навантаженості

Як вже було вказано у частині з вибором методу вирішення, треба реалізувати два графіки.

Перший графік буде відображати залежність кількості оброблених сервером пакетів від середньої швидкості цієї обробки. Це допоможе зрозуміти на скільки електронний ресурс, що досліджується, є масштабованим. Одна точка буде відповідати кількості пакетів, оброблених за 5 хвилин.

Другий графік буде відображати залежність середньої швидкості обробки запитів від кількості унікальних користувачів, що відправляли запити до електронної системи. У цьому графіку одна точка буде відповідати певній кількості унікальних користувачів за той же самий проміжок часу, як і в першому графіку – 5 хвилин, до середньої швидкості обробки запитів цих унікальних користувачів.

Для того, щоб реалізувати розділення даних на частини по 5 хвилин, у функції readSomeLines(file, maxlines, function(line) було реалізовано поділ та розрахунок середньої швидкості обробки даних за кожні 5 хвилин та записано до масивів значення середньої швидкості, кількості оброблених пакетів за цей проміжок часу та конкретну годину в яку входять 5 хвилин для подальшого відображення кольорової точки.

У випадку із другим графіком у функції readSomeLines(file, maxlines, function(line) відбувся розподіл даних по 5 хвилин і в результаті для кожної частини до масивів було записано кількість унікальних користувачів, що

надсилали запити в систему, середня швидкість обробки цих запитів та конкретна година, в яку входить частина даних.

```
user_points[points_counter] = unique_users_set.size;
time_points[points_counter] = request_times_counter / requests_per_period_counter;
hour_points[points_counter] = hour;
```

Візуалізація діаграми здійснюється за допомогою методу zingchart.render ({...}). Мінімальними вимогами до цього методу є id та властивості даних. Було реалізовано діаграму таким чином:

```
zingchart.render({
id: 'myChart',
data: myConfig,
height: 400,
width: "100%"
});
```

Для діаграми було реалізовано повномірне масштабування: висота та ширина на 100%.

```
#myChart {
height: 100%;
width: 100%;
}
```

У змінній myConfig прописано властивості, параметри діаграми, такі як: легенда графіку та її параметри, розміщення на надписи по x та y, розмір точок та ліній на графіку.

Для того, щоб виділити точки певним кольором (у випадку з розроблюваною моделлю кожний час у добі має певний колір) для кращого розуміння у який час доби найбільше навантаження на сервер, до змінної myConfig також було додано параметр "series" та заповнено для кожної години певний колір. Таким чином, для години 01:00 отримано:

{

"type": "scatter",

"values": color1,

"text":"01:00",

}

Тип візулізації даних розсіюваний ("type": "scatter") показує взаємозв’язок між різними змінними. Ці дані відображаються шляхом розміщення різних точок даних між осями x та y. По суті, кожна з цих точок даних виглядає такою, що є розсіяною навколо графіка, та цим і надає цій візуалізації даних таку назву.

Так як було для кожного кольору прописано значення, тип та час, бібліотека ZingChart автоматично призначила унікальний колір для конкретної години.

Також для змінної myConfig прописані параметри для лінії тренду:

{

"type": "line",

"values": approx,

"text":"trend",

}

Значення змінної approx визначається за допомогою звернення до наступної функції: var approx = findLineByLeastSquares(rps_points, time_points).

Саме функція function findLineByLeastSquares(values_x, values_y) визначає лінію тренду для першого та другого графіків. Детальний код для реалізації пошуку середньої лінії буде надано в частині з додатками.

Після виклику функції onComplete() в консоль для адміністратара після прочитання всього лог-файлу виведеться текст «Read all lines» та буде підраховано кількість всіх пакетів або ж унікальних користувачів за кожний із періодів 5-и хвилин, середню швидкість обробки запитів від унікальних користувачів або швидкість обробки пакетів та конкретну годину в яку входить 5-хвилинний проміжок часу.

Для адміністратора всі дані записано в лог додаток у консолі на випадок, якщо йому вони знадобляться для аналізу.

У функції onComplete() викликається інша функція calculation() в якій і буде відбуватися вся аналітична частина. У цій функції буде якраз обробка отриманих даних із лог-файлу та запис точок на графік. Детальніше буде висвітлено у наступній частині.

Також у функції onComplete() викликається не менш важлива функція printGraph(), що дозволяє вивести всі отримані точки на діаграму у відповідних кольорах для розуміння години доби під час якої надсилались запити до інформаційної системи.

Після отримання всіх точок у масиві, за допомогою звернення до функції findLineByLeastSquares(rps_points, time_points), кожну змінну, що буде відповідати за окремий колір, оголошуємо масивом даних, як показано нижче для двох кольорів:

```
var color1 = [];
var color2 = [];
```

Після цього перебираємо всі точки у масиві, користуючись звичайним циклом for. Результат можна побачити нижче:

```
for (j = 0; j < rps_points.length; j++) {
x = rps_points[j];
y = time_points[j];
```

Наступний крок – всередині цикла for реалізуємо switch та для кожної години кожної точки, (hour_points[points_counter] = hour) при цьому points_counter рівний для змінних x, y та hour_points, методом push записуємо точку у відповідний масив, що відповідає за певний колір. Реалізація switch у скороченому форматі (для декількох кольорів) відображена нижче:

```
switch (hour_points[j]) {
                case 1:
                color1.push([x,y]);
```

```
                    break;
                    case 2:
                    color2.push([x,y]);
                    break;
                   ....
                    default:
                    color24.push([x,y]);
}
```

Метод push є дуже корисним для додавання нових значень у масив.

Він додає елементи, починаючи із поточної довжини length і повертає нову, збільшену довжину масиву.

Конструкція switch порівнює вирази для випадків, перевірених всередині неї, а потім виконує відповідні інструкції, що знаходяться всередині описання.

3.4 Застосування статистичної гіпотези та формули Літтла для реалізації моделі

Першим кроком задля застосування статистичної гіпотези про рівність середніх значень до отриманих даних із лог-файлу треба віднайти ці середні значення.

Як вже було вказано, для того щоб аналізувати та віднайти середні значення, треба розподілити дані на частини. У нашому випадку можна ввести поняття трьох режимів навантаження – низьке, середнє та високе.

Для того, щоб ввести поняття низького, типового та високого навантаження було реалізовано розбиття всіх точок на три частини. Відштовхувалась від середньої точки. Додавши 16,999 відсотків точок справа від середньої, було отримано поріг для типового навантаження, віднявши 16,999 відсотків усіх точок, що є зліва, отримала точку, що розпочинає входження до типового навантаження. Всі інші точки, що залишилися зліва увійшли до низького навантаження, всі, що справа – до високого.

У функції onComplete() викликається функція calculation() для обробки даних із лог-файлу.

Першим чином було віднайдено середню точку, а потім розбито всі дані на три частини, як це описано вище. У коді це виглядає наступним чином:

```
var rps_avr = rsp_summ / rps_points.length;
console.log('rps_avr ', rps_avr);
var low_line = rps_avr - (rps_avr * 0.16666);
var high_line = rps_avr + (rps_avr * 0.16666);
```

Так отримано такі три частини:

1. Низьке навантаження: точки від 0 до low_line по координаті x.
2. Середнє навантаження: точки від low_line до high_line.
3. Високе навантаження: точки від high_line до останньої.

Для розробки моделі добре підійде формула для перевірки нульової гіпотези у випадку, якщо вибіркі незалежні та генеральні сукупності розподілені нормально та відомі їх дисперсії σ_x^2, σ_y^2.

Для знаходження середніх значень по У для кожного із режиму навантаження, було знайдено загальну кількість точок для кожної із частин за допомогою звиклого вже циклу for.

```
for (i = 0; i < rps_points.length; i++) {
    if (rps_points[i] <= low_line) time_length1++;
    if (rps_points[i] > low_line && rps_points[i] < high_line) time_length2++;
    if (rps_points[i] >= high_line) time_length3++;
}
```

Наступним кроком було розрахувати суму всіх середніх швидкостей обробки запитів для кожної точки певної частини та віднайти серед них середнє значення.

```
var time_avr1 = time_summ1 / time_length1;
var time_avr2 = time_summ2 / time_length2;
var time_avr3 = time_summ3 / time_length3;
```

За допомогою формули 1.5 розраховуємо статистичний критерій. Так як частин маємо 3, порівняємо середні значення низького навантаження та типового, а потім типового навантаження та високого для того, щоб зрозуміти чи підтверджується статистична гіпотеза чи ні.

Значення X та У – це середні значення першої та другої частини. Це значення змінних time_avr1, time_avr2 та time_avr3.

У формулі також застосовується значення дисперсії. Отримати це значення можна за допомогою формули 3.1, що наведена нижче.

$$D(x) = M(x - M(x))^2, \quad (3.1)$$

Де M(x) – це математичне очікування випадкової величини x.

Таким чином, були отримані дисперсії:

var dispersion1 = Mx2_1 - Math.pow(Mx_1, 2);

var dispersion2 = Mx2_2 - Math.pow(Mx_2, 2);

var dispersion3 = Mx2_3 - Math.pow(Mx_3, 2);

І в результаті було знайдено значення змінної Z наближене.

var Z1 = (time_avr2 - time_avr1) / (Math.sqrt(dispersion1/time_length1 + dispersion2/time_length2));

var Z2 = (time_avr3 - time_avr2) / (Math.sqrt(dispersion2/time_length2 + dispersion3/time_length3));

Далі змінна Zkr була задана рівною 2.33 тому що було задано ймовірність отримати помилку першого рівня – рівень значущості – α = 0,01. Отже, за формулою 1.6 (функція Лапласа) було розраховано значення Z критичне.

На рисунку 1.3 можна згадати, що таке критичні області. Критична область – це зона, де відторгається нульова гіпотеза. Незаштрихована область – це зона, де гіпотеза приймається. Тобто, якщо отримане значення Z1 або Z2 < значення Zkr, то гіпотеза приймається, а це буде означати, що значвення середніх статистично рівні.

Для того, щоб вивести користувачу, що система є масштабованою, необхідно щоб гіпотеза була прийнятою для двох значень: Z1 та Z2.

```
if (Z1 < Zkr && Z1 < Zkr) console.log('The system is scalable');
else console.log('The system isn\'t scalable');
```

Для того, щоб показати навантаженість сервера, візьмемо середню кількість пакетів для кожного із режиму навантаження з першого графіку (відображає залежність кількості оброблених сервером пакетів від середньой швидкості цієї обробки). Перша точка – середня швидкість обробки пакетів при низькому навантаженні, друга – при типовому навантаженні, а третя – при високому навантаженні сервера.

Для кожної із цих точок за формулою Літтла (формула 1.3) розраховано максимальну кількість пакетів для кожного із режиму навантаження. Після спростування було отримано такий результат імплементації:

```
var requests_result1 = 300000 / time_avr1;
var requests_result2 = 300000 / time_avr2;
var requests_result3 = 300000 / time_avr3;
```

У результаті для користувача буде виведено максимальне значення.

Те саме було розраховано і для другого графіку та у результаті виведено результат користувачу на сторінку.

```
result_str += "For low load: ";
result_str += Math.floor(requests_result1).toString().substr(0,8);
```

Метод, що був застосований, Math.floor () - округлення вниз. Округлює аргумент до найближчого меншого цілого.

3.5 Тестування моделі

Задля тестування моделі було вивантажено великий лог-файл (розміром 5,5 гб) із електронної системи дистанційного навчання СумДУ - https://mix.sumdu.edu.ua. У цьому вхідному файлі зібрано активність користувачів електронної системи за вересень-жовтень 2020-го року. З таким надвеликим набором даних можна буде отримати більш детальний результат щодо продуктивності інформаційної системи.

Всього графіків є декілька. Розглянемо результат першого графіку.

Перший графік має відображати залежність кількості оброблених сервером пакетів від середньої швидкості цієї обробки. Це допоможе зрозуміти наскільки електронний ресурс, що досліджується, є масштабованим. Одна точка буде відповідати кількості пакетів, оброблених за 5 хвилин. Цей параметр можна буде змінити за необхідністю, якщо треба буде мати більшу кількість точок. Проте, такої необхідності немає, у отриманій кількості точок великий так званий шум не помічається.

Таким чином, якщо всі точки будуть знаходитись на графіку у вигляді однієї горизонтальної лінії, то це буде означати, що система працює добре і вона є добре масшабованою.

Якщо ж точки будуть розміщені хаотично або будуть розташовані у вигляді зростаючої лінії, то це вже буде вказувати на те, що є проблеми та затримки при паралельній обробці запитів. Така система буде називатись погано масштабованою і адміністратору треба буде замислитись над виправленням помилок, закупівлі додаткового обладнання або ж оптимізації коду для обробки даних в електронній системі.

Завантаживши отриманий код, маємо наступний результат, що представлено на Рисунку 3.2.

На Рисунку 3.2 розташовано точки по всьому графіку у різних кольорах. Кожний колір відповідає за певну годину дня. Інформацію по годинах можна побачити справа в легенді графіку. Серед точок можна зустріти велику кількість розфарбованих у блідно-жовтий колір. Це означає, що при низькому завантаженні сервера велика кількість користувачів відправляє запити приблизно о 23-й годині.

На цьому графіку було додано лінію середніх значень – лінія тренду. Розфарбована вона у зелений колір та є необхідною для застосування статистичної гіпотези щодо рівності середніх значень.

Рисунок 3.2 – Графік залежності кількості пакетів від середньої швидкості їх обробки

Для того, щоб ввести поняття низького, типового та високого навантаження було реалізовано розбиття всіх точок на три частини. Відштовхувалась від середньої точки. Додавши 16,999 відсотків точок справа від середньої, було отримано поріг для типового навантаження, віднявши 16,999 відсотків усіх точок, що є зліва, отримала точку, що розпочинає входження до типового навантаження. Всі інші точки, що залишилися зліва увійшли до низького навантаження, всі, що справа – до високого.

У результаті, отримано такі відрізки точок для різного навантаження на електронну систему:

1. Низьке навантаження: 0...934.222089046074.
2. Типове навантаження: 934.222089046074...1307.892987743889.
3. Високе навантаження: 1307.892987743889...5944.

Також отримано такі середні значення середньої швидкості обробки пакетів для різного навантаження на електронну систему:

1. Низьке навантаження: 58.698877333863315 ms.

2. Типове навантаження: 93.28105776489852 ms.
3. Високе навантаження: 89.3001226515463 ms.

Значення середнього навантаження також виведено для фінального користувача розробленої моделі та представлено вгорі над отриманим першим графіком.

Для високого навантаження по отриманому графіку видно, що майже всі точки знаходяться поряд із відрізком типового навантаження, а лише невелика кількість розташована значно далеко. Це говорить про те, що у якийсь час було відправлено велику кількість пакетів та схожа поведінка була досить рідкою і сервер зміг обробити всі запити. Якщо б таке навантаження тривало довго, то сервер міг би впасти, знеможлививши роботу користувачам із ним.

У результаті перевірки статистичної гіпотези щодо рівності середніх значень із ймовірністю отримати помилку першого рівня – рівень значущості – $\alpha = 0{,}01$ було виведено результат щодо того, що система не є масштабованою. Можна побачити, що середні значення трьох видів навантаження не є рівними та й лінія середніх спрямована вгору, що також стверджує про погану масштабованість серверу.

Також у висновку для фінального користувача можна побачити результат із максимальною кількістю пакетів, що сервер зможе опрацювати в режимі з різними видами навантажень. Це число було отримано за допомогою формули Літтла (формула 2.2).

Отже, в результаті можна виділити таку максимальну кількість пакетів, що сервер здатен витримати: 5110 запитів.

Число вище було отримано із аналізу максимальних значень для трьох виділених режимів навантаження: низике, типове та високе. Максимальне число із отриманих – 5110 запитів і є резельтатом.

Також у логи для адміністратора було додано деякі інші параметри та їх значення, що зображено на Рисунку 3.3 та Рисунку 3.4.

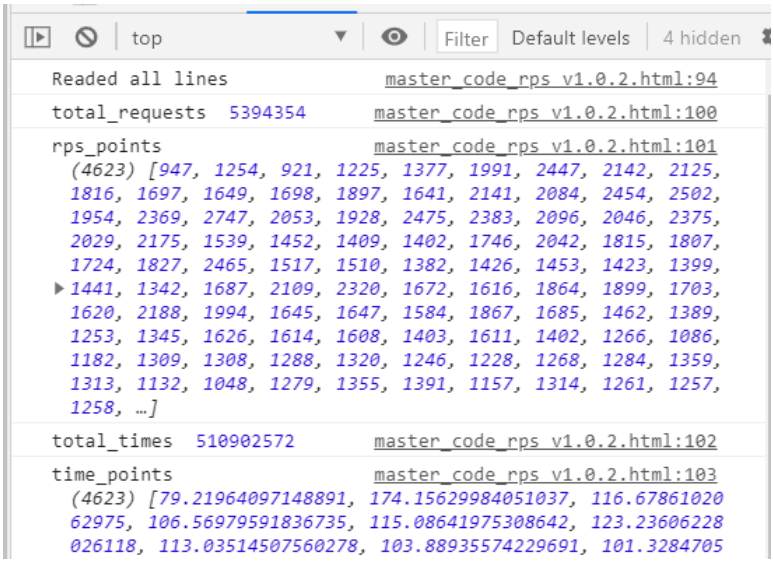

Рисунок 3.3 – Дані в логах для графіку залежності кількості пакетів від середньої швидкості їх обробки

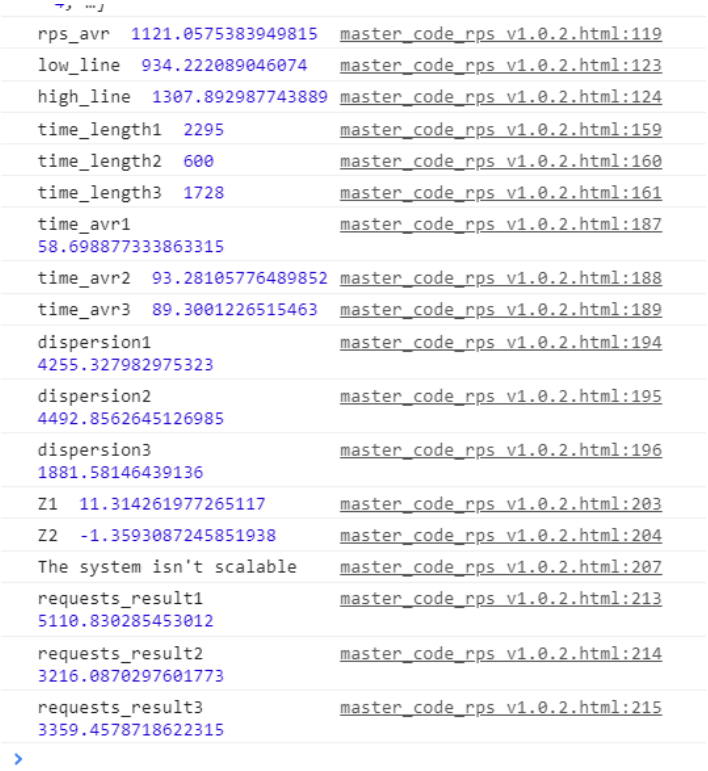

Рисунок 3.4 – Дані в логах для графіку залежності кількості пакетів від середньої швидкості їх обробки

Другий графік має відображати залежність середньої швидкості обробки запитів від кількості унікальних користувачів, що відправляли запити до електронної системи.

У цьому графіку одна точка має відповідати певній кількості унікальних користувачів за той же самий проміжок часу, як і в першому графіку – 5 хвилин, до середньої швидкості обробки запитів цих унікальних користувачів.

Завантаживши отриманий код, маємо наступний результат, що представлено на Рисунку 3.5. На графіку розташовано точки по всій діаграмі у різних кольорах так як і на першому графіку. Серед точок можна зустріти велику кількість розфарбованих у блідно-жовтий колір для режиму низького навантаження. Це означає, що у такому режимі роботи сервера незначна кількість унікальних користувачів відправляє запити приблизно о 23-й годині. Також можна побачити, що значна кількість унікальних користувачів вибирає обідню годину дня для роботи із навчальним електронним сервісом.

Для аналізу навантаженості сервісу було так само як і для першого графіку введено понять низького, типового та високого навантаження.

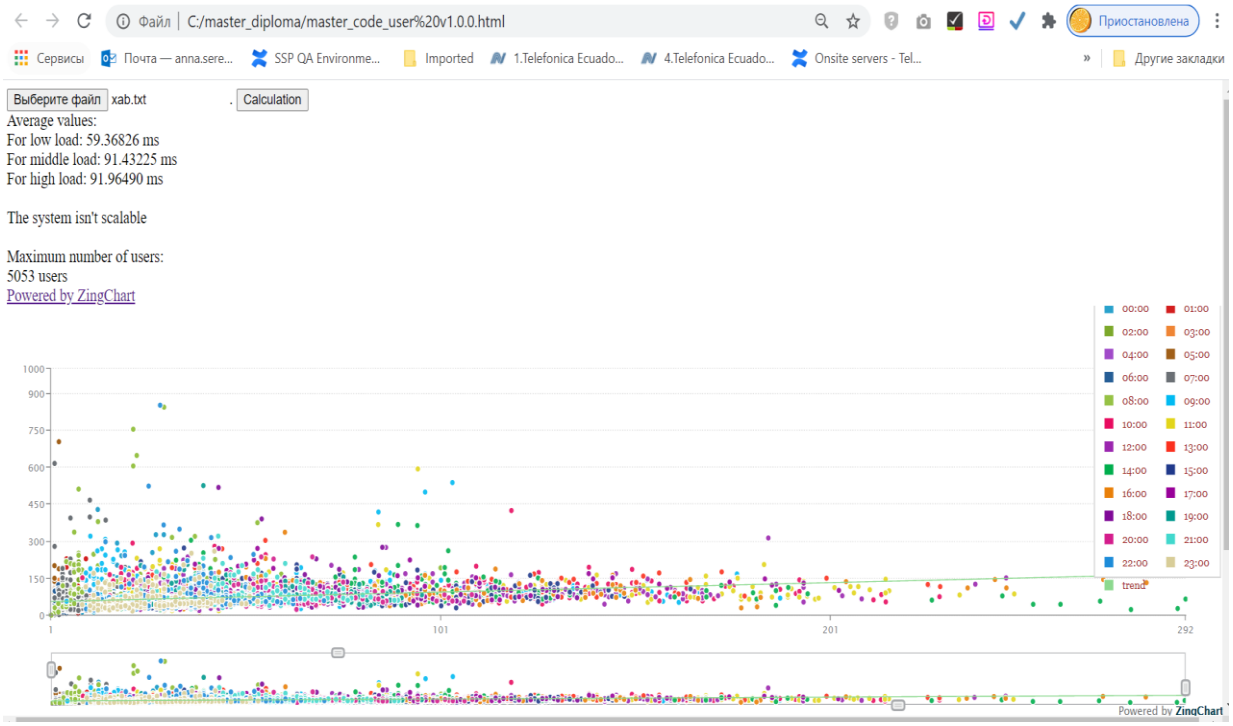

Рисунок 3.5 – Графік залежності кількості унікальних користувачів від середньої швидкості обробки запитів

У результаті, отримано такі відрізки точок для різного навантаження на електронну систему (розподіл по кількості унікальних користувачів):

4. Низьке навантаження: 0... 38.89788.

5. Типове навантаження: 38.89788...54.456288.
6. Високе навантаження: 54.456288...292.

Також отримано такі середні значення середньої швидкості обробки пакетів для різного навантаження на електронну систему при різній унікальній кількості користувачів:

4. Низьке навантаження: 59.36826 ms.
5. Типове навантаження: 91.43225 ms.
6. Високе навантаження: 91.96490 ms.

Значення середнього навантаження також виведено для фінального користувача розробленої моделі та представлено вгорі над отриманим другим графіком.

У результаті перевірки статистичної гіпотези щодо рівності середніх значень із ймовірністю отримати помилку першого рівня – рівень значущості – $\alpha = 0{,}01$ було виведено результат щодо того, що система не є масштабованою. Тому зі збільшенням кількості унікальних користувачів, що відправляють запити до системи, збільшується швидкість обробки запитів. Можна побачити, що середні значення трьох видів навантаження не є рівними та й лінія середніх спрямована вгору, що також стверджує про погану масштабованість серверу.

Також у висновку для фінального користувача можна побачити результат із максимальною кількістю унікальних користувачів для яких сервер зможе опрацьовувати запити у різних режимах навантаження. Це число було також, як і для першого графіку, отримане за допомогою формули Літтла (формула 2.2).

Отже, в результаті можна виділити таку максимальну кількість унікальних користувачів для яких сервер зможе опрацьовувати запити у різних режимах навантаження:

1. Низьке навантаження: 5053 користувача.
2. Типове навантаження: 3281 користувачів.
3. Високе навантаження: 3262 користувача.

Фінальне значення максимальної кількості користувачів – 5053.

Рисунок 3.6 – Дані в логах для графіку залежності кількості унікальних користувачів від середньої швидкості обробки запитів

Також у логи для адміністратора було додано деякі інші параметри та їх значення, що зображено на Рисунку 3.6.

У результаті адміністратор бачить два графіки, що продемонстровані вище, за допомогою яких він може мати змогу проаналізувати навантаженість на сервер для подальшого усунення падіння електронної системи.

ВИСНОВКИ

Дана робота присвячена проектуванню моделі, яка б допомогла адміністратору електронної системи прийняти рішення щодо продуктивності сервера та прогнозувала б максимальну кількість унікальних користувачів та запитів при якій сервер буде неспроможний обробляти запити клієнтів.

У ході постановки задачі було розглянуто вхідні дані, задачу та результат, який очікується після реалізації моделі. У ході проектування було визначено, що необхідні вхідні дані треба занести до масивів у JavaScript для подальшої роботи з ними, використати бібліотеку ZingChart у JS для представлення двох діаграм.

Перший відображає залежність кількості оброблених сервером пакетів від середньої швидкості цієї обробки. Це допоможе зрозуміти наскільки електронний ресурс, що досліджується, є масштабованим. Одна точка буде відповідати кількості пакетів, оброблених за 5 хвилин.

Другий графік відображає залежність середньої швидкості обробки запитів від кількості унікальних користувачів, що відправляли запити до електронної системи. У цьому графіку одна точка буде відповідати певній кількості унікальних користувачів за той же самий проміжок часу, як і в першому графіку – 5 хвилин, до середньої швидкості обробки запитів цих унікальних користувачів.

У результаті адміністратор бачить два графіки, що описані вище, за допомогою яких він може мати змогу проаналізувати навантаженість на сервер.

Реалізація знаходження масштабованості сервера була здійснена за допомогою статистичної гіпотези щодо середніх значень.

Для того щоб аналізувати та віднайти середні значення, треба розподілити дані на частини. У нашому випадку можна ввести поняття трьох режимів навантаження – низьке, середнє та високе.

Таким чином, щоб ввести поняття низького, типового та високого навантаження було реалізовано розбиття всіх точок на три частини. Відштовхувалась від середньої точки.

Додавши 16,999 відсотків точок справа від середньої, було отримано поріг для типового навантаження, віднявши 16,999 відсотків усіх точок, що є зліва, отримала точку, що розпочинає входження до типового навантаження. Всі інші точки, що залишилися зліва увійшли до низького навантаження, всі, що справа – до високого.

Для знаходження максимальної кількості унікальних користувачів та запитів при якій сервер буде неспроможний обробляти запити клієнтів було застосовано формулу Літтла. Компанія Яндекс для оцінки навантаженості свого сервера користувалась саме такою формулою.

Задля тестування моделі було вивантажено великий лог-файл (розміром 5,5 гб) із електронної системи дистанційного навчання СумДУ - https://mix.sumdu.edu.ua. У цьому вхідному файлі зібрано активність користувачів електронної системи за вересень-жовтень 2020-го року. З таким надвеликим набором даних отримано більш детальний результат щодо продуктивності інформаційної системи.

У ході тестування отримано по результатам двох графіків, що система не є масштабованою. Та в результаті можна виділити таку максимальну кількість пакетів, що сервер здатен витримати у різних режимах навантаження:

1. Низьке навантаження: 5110 запитів.
2. Типове навантаження: 3216 запитів.
3. Високе навантаження: 3359 запитів.

Також можна виділити таку максимальну кількість унікальних користувачів для яких сервер зможе опрацьовувати запити у різних режимах навантаження:

1. Низьке навантаження: 5053 коритувача.
2. Типове навантаження: 3281 користувачів.
3. Високе навантаження: 3262 користувача.

СПИСОК ЛІТЕРАТУРИ

1. Silva C. Educational Data Mining: a literature review [Електронний ресурс] / C. Silva, J. Fonseca. – 2017. – Режим доступу до ресурсу: https://www.researchgate.net/publication/313066371_Educational_Data_Mining_A_Literature_Review.
2. Tomkins S. Predicting Post-Test Performance from Online Student Behavior: A High School MOOC Case Study [Електронний ресурс] / S. Tomkins, A. Ramesh, L. Getoor – Режим доступу до ресурсу: http://travellingscholar.com/papers/tomkins-edm16.pdf.
3. Silva C. Educational Data Mining: a literature review [Електронний ресурс] / C. Silva, J. Fonseca. – 2017. – Режим доступу до ресурсу: https://www.researchgate.net/publication/313066371_Educational_Data_Mining_A_Literature_Review.
4. Tetsuya J. Why is Educational Data Mining important in the research? [Електронний ресурс] / Jesse Tetsuya. – 2019. – Режим доступу до ресурсу: https://towardsdatascience.com/why-is-educational-data-mining-important-in-the-research-e78ed1a17908.
5. Машинное обучение (Machine Learning) [Електронний ресурс] – Режим доступу до ресурсу: http://statsoft.ru/home/textbook/modules/stmachlearn.html.
6. Дистанційна освіта [Електронний ресурс] – Режим доступу до ресурсу: http://vnz.org.ua/dystantsijna-osvita/pro.
7. Baker R. Educational Data Mining and Learning Analytics [Електронний ресурс] / R. Baker, G. Siemens. – 2013. – Режим доступу до ресурсу: http://www.columbia.edu/~rsb2162/BakerSiemensHandbook2013.pdf.
8. Причины и способы борьбы с чрезмерной нагрузкой на сервер [Електронний ресурс] – Режим доступу до ресурсу: https://eternalhost.net/blog/hosting/nagruzka-na-server

9. Причини створення навантаження на сервері скриптами сайту [Електронний ресурс] – Режим доступу до ресурсу: https://billing.hostpro.ru/index.php?rp=/knowledgebase/283/
10. Нагрузочное тестирование веб-сервера [Електронний ресурс] – Режим доступу до ресурсу: http://web.kpi.kharkov.ua/otp/ru/nagruzochnoe-testirovanie-veb-servera/
11. Закон Литтла [Електронний ресурс] – Режим доступу до ресурсу: https://kartaslov.ru/карта-знаний/Закон+Литтла
12. Закон Литтла [Електронний ресурс] – Режим доступу до ресурсу: http://class469.apmath.spbu.ru:8080/queue_files/texts/little_ru.pdf
13. Нагрузочное тестирование vs Тестирование производительности [Електронний ресурс] / Максим Рогожников. – 2018. – Режим доступу до ресурсу: https://www.performance-lab.ru/blog/load-testing/testirovanie-proizvoditelnosti
14. Scalability for Dummies — Short Notes collection [Електронний ресурс] / Arun Prakash. – 2020. – Режим доступу до ресурсу: https://medium.com/@reach2arunprakash/
15. Building Scalable and High-Performance Java Web Applications Using J2EE Technology [Електронний ресурс] / Greg Barish. – 2002. – Режим доступу до ресурсу: https://www.informit.com/articles/article.aspx?p=26942&seqNum=18
16. Десктопное приложение или веб-клиент – вот в чем вопрос! [Електронний ресурс] / Вячеслав Лазарев. – 2013. – Режим доступу до ресурсу: https://ecm-journal.ru/post/Desktopnoe-prilozhenie-ili-veb-klient-vot-v-chem-vopros.aspx
17. Языки программирования и технологии для веб-разработки [Електронний ресурс] / Івашкевич. – 2017. – Режим доступу до ресурсу: https://webshake.ru/post/yazyki-programmirovaniya-dlya-veb-razrabotki
18. Чем так хорош язык веб-разработки PHP [Електронний ресурс] / – 2018. – Режим доступу до ресурсу: https://webformyself.com/chem-tak-xorosh-yazyk-veb-razrabotki-php/

19. Язык программирования Javascript: особенности и преимущества [Електронний ресурс] / – 2020. – Режим доступу до ресурсу: https://vc.ru/hr/145461-yazyk-programmirovaniya-javascript-osobennosti-i-preimushchestva
20. Your First JavaScript Chart – Режим доступу до ресурсу: https://www.zingchart.com/docs/getting-started/your-first-javascript-chart
21. Формат JSON, метод toJSON / – 2019. – Режим доступу до ресурсу: https://learn.javascript.ru/json
22. JavaScript метод JSON.parse() – Режим доступу до ресурсу: https://basicweb.ru/javascript/js_json_parse.php
23. Работа с файлами в JavaScript, Часть 2: FileReader / – 2012. – Режим доступу до ресурсу: https://xdan.ru/working-with-files-in-javascript-part-2-filereader.html
24. Проверка статистических гипотез / Емелин Александр – Режим доступу до ресурсу: http://mathprofi.ru/proverka_statisticheskih_gipotez.html

ДОДАТКИ

Додаток А

Таблиця значень функції Лапласа

Таблица значений функции $\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_0^x e^{-z^2/2}\, dz$

x	$\Phi(x)$	x	$\Phi(x)$	x	$\Phi(x)$	x	$\Phi(x)$
0,00	0,0000	0,32	0,1255	0,64	0,2389	0,96	0,3315
0,01	0,0040	0,33	0,1293	0,65	0,2422	0,97	0,3340
0,02	0,0080	0,34	0,1331	0,66	0,2454	0,98	0,3365
0,03	0,0120	0,35	0,1368	0,67	0,2486	0,99	0,3389
0,04	0,0160	0,36	0,1406	0,68	0,2517	1,00	0,3413
0,05	0,0199	0,37	0,1443	0,69	0,2549	1,01	0,3438
0,06	0,0239	0,38	0,1480	0,70	0,2580	1,02	0,3461
0,07	0,0279	0,39	0,1517	0,71	0,2611	1,03	0,3485
0,08	0,0319	0,40	0,1554	0,72	0,2642	1,04	0,3508
0,09	0,0359	0,41	0,1591	0,73	0,2673	1,05	0,3531
0,10	0,0398	0,42	0,1628	0,74	0,2703	1,06	0,3554
0,11	0,0438	0,43	0,1664	0,75	0,2734	1,07	0,3577
0,12	0,0478	0,44	0,1700	0,76	0,2764	1,08	0,3599
0,13	0,0517	0,45	0,1736	0,77	0,2794	1,09	0,3621
0,14	0,0557	0,46	0,1772	0,78	0,2823	1,10	0,3643
0,15	0,0596	0,47	0,1808	0,79	0,2852	1,11	0,3665
0,16	0,0636	0,48	0,1844	0,80	0,2881	1,12	0,3686
0,17	0,0675	0,49	0,1879	0,81	0,2910	1,13	0,3708
0,18	0,0714	0,50	0,1915	0,82	0,2939	1,14	0,3729
0,19	0,0753	0,51	0,1950	0,83	0,2967	1,15	0,3749
0,20	0,0793	0,52	0,1985	0,84	0,2995	1,16	0,3770
0,21	0,0832	0,53	0,2019	0,85	0,3023	1,17	0,3790
0,22	0,0871	0,54	0,2054	0,86	0,3051	1,18	0,3810
0,23	0,0910	0,55	0,2088	0,87	0,3078	1,19	0,3830
0,24	0,0948	0,56	0,2123	0,88	0,3106	1,20	0,3849
0,25	0,0987	0,57	0,2157	0,89	0,3133	1,21	0,3869
0,26	0,1026	0,58	0,2190	0,90	0,3159	1,22	0,3883
0,27	0,1064	0,59	0,2224	0,91	0,3186	1,23	0,3907
0,28	0,1103	0,60	0,2257	0,92	0,3212	1,24	0,3925
0,29	0,1141	0,61	0,2291	0,93	0,3238	1,25	0,3944
0,30	0,1179	0,62	0,2324	0,94	0,3264		
0,31	0,1217	0,63	0,2357	0,95	0,3289		

Продолжение

x	$\Phi(x)$	x	$\Phi(x)$	x	$\Phi(x)$	x	$\Phi(x)$
1,26	0,3962	1,59	0,4441	1,92	0,4726	2,50	0,4938
1,27	0,3980	1,60	0,4452	1,93	0,4732	2,52	0,4941
1,28	0,3997	1,61	0,4463	1,94	0,4738	2,54	0,4945
1,29	0,4015	1,62	0,4474	1,95	0,4744	2,56	0,4948
1,30	0,4032	1,63	0,4484	1,96	0,4750	2,58	0,4951
1,31	0,4049	1,64	0,4495	1,97	0,4756	2,60	0,4953
1,32	0,4066	1,65	0,4505	1,98	0,4761	2,62	0,4956
1,33	0,4082	1,66	0,4515	1,99	0,4767	2,64	0,4959
1,34	0,4099	1,67	0,4525	2,00	0,4772	2,66	0,4961
1,35	0,4115	1,68	0,4535	2,02	0,4783	2,68	0,4963
1,36	0,4131	1,69	0,4545	2,04	0,4793	2,70	0,4965
1,37	0,4147	1,70	0,4554	2,06	0,4803	2,72	0,4967
1,38	0,4162	1,71	0,4564	2,08	0,4812	2,74	0,4969
1,39	0,4177	1,72	0,4573	2,10	0,4821	2,76	0,4971
1,40	0,4192	1,73	0,4582	2,12	0,4830	2,78	0,4973
1,41	0,4207	1,74	0,4591	2,14	0,4838	2,80	0,4974
1,42	0,4222	1,75	0,4599	2,16	0,4846	2,82	0,4976
1,43	0,4236	1,76	0,4608	2,18	0,4854	2,84	0,4977
1,44	0,4251	1,77	0,4616	2,20	0,4861	2,86	0,4979
1,45	0,4265	1,78	0,4625	2,22	0,4868	2,88	0,4980
1,46	0,4279	1,79	0,4633	2,24	0,4875	2,90	0,4981
1,47	0,4292	1,80	0,4641	2,26	0,4881	2,92	0,4982
1,48	0,4306	1,81	0,4649	2,28	0,4887	2,94	0,4984
1,49	0,4319	1,82	0,4656	2,30	0,4893	2,96	0,4985
1,50	0,4332	1,83	0,4664	2,32	0,4898	2,98	0,4986
1,51	0,4345	1,84	0,4671	2,34	0,4904	3,00	0,49865
1,52	0,4357	1,85	0,4678	2,36	0,4909	3,20	0,49931
1,53	0,4370	1,86	0,4686	2,38	0,4913	3,40	0,49966
1,54	0,4382	1,87	0,4693	2,40	0,4918	3,60	0,499841
1,55	0,4394	1,88	0,4699	2,42	0,4922	3,80	0,499928
1,56	0,4406	1,89	0,4706	2,44	0,4927	4,00	0,499968
1,57	0,4418	1,90	0,4713	2,46	0,4931	4,50	0,499997
1,58	0,4429	1,91	0,4719	2,48	0,4934	5,00	0,499999

Додаток В

Реалізація першого графіку для аналізу навантаженості електронної системи

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Master diploma - RPS</title>
  <script src="https://cdn.zingchart.com/zingchart.min.js"></script>
  <style>
    html,
    body,
    #myChart {
      height: 100%;
      width: 100%;
```

```
    }
  </style>
</head>
<body>
  <input type="file" id="infile">.
  <input type="button" id="start" value="Calculation">
  <div id="calc_result"></div>
  <div id="myChart" class="chart--container">
    <a class="zc-ref" href="https://www.zingchart.com/">Powered by ZingChart</a>
  </div>
  <script>
    var rps_points = new Array();
    var time_points = new Array();
    var hour_points = new Array();
    var zero_jump_flag = true;
    var start_loop = true;
    var start_minute = 0;
    var total_requests_counter = 0;
    var total_times_counter = 0;
    var requests_per_period_counter = 0;
    var request_times_counter = 0;
    var points_counter = 0;
    var hour = 0;
    document.getElementById('start').onclick = function() {
      var file = document.getElementById('infile').files[0];
      if (!file) {
        console.log('No file selected.');
        return;
      }
      var linemax = Number.MAX_VALUE;
      var lineno = 1;
      readSomeLines(file, linemax, function(line) {
        if (line.match("Completed")) {
          requests_per_period_counter++;
          request_times_counter += parseInt(line.substring(line.indexOf('in') + 3, line.indexOf('ms')));
          total_requests_counter++;
          total_times_counter += parseInt(line.substring(line.indexOf('in') + 3, line.indexOf('ms')));
          var datetime_str = line.substring(4,30);
          var datetime = new Date(datetime_str);
          var minute = datetime.getMinutes();
          hour = datetime.getHours();
```

```
                if (start_loop) {
                    start_minute = minute;
                    start_loop = false;
                }
                if (minute == 0 && zero_jump_flag) {
                    start_minute -= 60;
                    zero_jump_flag = false;
                }
                if ((minute - start_minute) >= 5) {
                    start_minute = minute;
                    rps_points[points_counter] = requests_per_period_counter;
                    time_points[points_counter] = request_times_counter /
requests_per_period_counter;
                    hour_points[points_counter] = hour;
                    if (!(rps_points[points_counter] > 6000 || time_points[points_counter]
> 1000)) {
                        points_counter++;
                    }
                    requests_per_period_counter = 0;
                    request_times_counter = 0;

                    zero_jump_flag = true;
                }
            }
        }, function onComplete() {
            console.log('Readed all lines');
            rps_points[points_counter] = requests_per_period_counter; //last frame
            time_points[points_counter] = request_times_counter /
requests_per_period_counter;   //last frame
            hour_points[points_counter] = hour;
            console.log('total_requests ', total_requests_counter);
            console.log('rps_points ', rps_points);
            console.log('total_times ', total_times_counter);
            console.log('time_points ', time_points);
            calculation();
            printGraph();
        });
    };
    function calculation() {
        var rsp_summ = 0;
        var i = 0;
        for (i = 0; i < rps_points.length; i++) {
            rsp_summ += rps_points[i];
```

```
}
var rps_avr = rsp_summ / rps_points.length;
console.log('rps_avr ', rps_avr);
var low_line = rps_avr - (rps_avr * 0.16666);
var high_line = rps_avr + (rps_avr * 0.16666);
console.log('low_line ', low_line);
console.log('high_line ', high_line);
var time_length1 = 0;
var time_length2 = 0;
var time_length3 = 0;
var Mx2_1 = 0;
var Mx2_2 = 0;
var Mx2_3 = 0;
var Mx_1 = 0;
var Mx_2 = 0;
var Mx_3 = 0;
var time_summ1 = 0;
var time_summ2 = 0;
var time_summ3 = 0;
var max_point1 = 0;
var max_point2 = 0;
var max_point3 = 0;
var T1 = 0;
var T2 = 0;
var T3 = 0;
var W1 = 0;
var W2 = 0;
var W3 = 0;
for (i = 0; i < rps_points.length; i++) {
   if (rps_points[i] <= low_line) time_length1++;
   if (rps_points[i] > low_line && rps_points[i] < high_line) time_length2++;
   if (rps_points[i] >= high_line) time_length3++;
}
console.log('time_length1 ', time_length1);
console.log('time_length2 ', time_length2);
console.log('time_length3 ', time_length3);
for (i = 0; i < rps_points.length; i++) {
   if (rps_points[i] <= low_line) {
      Mx2_1 += Math.pow(time_points[i], 2) / time_length1;
      Mx_1 += time_points[i] / time_length1;
      time_summ1 += time_points[i];
      if (max_point1 < time_points[i]) max_point1 = time_points[i];
   }
   if (rps_points[i] > low_line && rps_points[i] < high_line) {
```

```
                Mx2_2 += Math.pow(time_points[i], 2) / time_length2;
                Mx_2 += time_points[i] / time_length2;
                time_summ2 += time_points[i];
                if (max_point2 < time_points[i]) max_point2 = time_points[i];
            }
            if (rps_points[i] >= high_line) {
                Mx2_3 += Math.pow(time_points[i], 2) / time_length3;
                Mx_3 += time_points[i] / time_length3;
                time_summ3 += time_points[i];
                if (max_point3 < time_points[i]) max_point3 = time_points[i];
            }
        }
        var time_avr1 = time_summ1 / time_length1;
        var time_avr2 = time_summ2 / time_length2;
        var time_avr3 = time_summ3 / time_length3;
        console.log('time_avr1 ', time_avr1);
        console.log('time_avr2 ', time_avr2);
        console.log('time_avr3 ', time_avr3);
        var dispersion1 = Mx2_1 - Math.pow(Mx_1, 2);
        var dispersion2 = Mx2_2 - Math.pow(Mx_2, 2);
        var dispersion3 = Mx2_3 - Math.pow(Mx_3, 2);
        console.log('dispersion1 ', dispersion1);
        console.log('dispersion2 ', dispersion2);
        console.log('dispersion3 ', dispersion3);
        var Zkr = 2.33;
        var Z1 = (time_avr2 - time_avr1) / (Math.sqrt(dispersion1/time_length1 +
dispersion2/time_length2));
        var Z2 = (time_avr3 - time_avr2) / (Math.sqrt(dispersion2/time_length2 +
dispersion3/time_length3));
        console.log('Z1 ', Z1);
        console.log('Z2 ', Z2);
        if (Z1 < Zkr && Z1 < Zkr) console.log('The system is scalable');
        else console.log('The system isn\'t scalable');
        var requests_result1 = 300000 / time_avr1;
        var requests_result2 = 300000 / time_avr2;
        var requests_result3 = 300000 / time_avr3;
        console.log('requests_result1 ', requests_result1);
        console.log('requests_result2 ', requests_result2);
        console.log('requests_result3 ', requests_result3);
        var result_str = "Average values:<br>";
        result_str += "For low load: ";
        result_str += time_avr1.toString().substr(0,8);
        result_str += " ms<br>";
        result_str += "For middle load: ";
```

```
            result_str += time_avr2.toString().substr(0,8);
            result_str += " ms<br>";
            result_str += "For high load: ";
            result_str += time_avr3.toString().substr(0,8);
            result_str += " ms<br><br>";
            if (Z1 < Zkr && Z1 < Zkr) result_str += "The system is scalable<br><br>";
            else result_str += "The system isn\'t scalable<br><br>";
            result_str += "Maximum number of requests:<br> ";
            result_str += "For low load: ";
            result_str += Math.floor(requests_result1).toString().substr(0,8);
            result_str += " requests<br>";
            result_str += "For middle load: ";
            result_str += Math.floor(requests_result2).toString().substr(0,8);
            result_str += " requests<br>";
            result_str += "For high load: ";
            result_str += Math.floor(requests_result3).toString().substr(0,8);
            result_str += " requests<br><br>";
            document.getElementById("calc_result").insertAdjacentHTML(    'beforeend',
result_str );
        }
        function printGraph() {
            var approx = findLineByLeastSquares(rps_points, time_points);
            var color1 = [];
            var color2 = [];
            var color3 = [];
            var color4 = [];
            var color5 = [];
            var color6 = [];
            var color7 = [];
            var color8 = [];
            var color9 = [];
            var color10 = [];
            var color11 = [];
            var color12 = [];
            var color13 = [];
            var color14 = [];
            var color15 = [];
            var color16 = [];
            var color17 = [];
            var color18 = [];
            var color19 = [];
            var color20 = [];
            var color21 = [];
            var color22 = [];
```

```
var color23 = [];
var color24 = [];
    var x = 0;
    var y = 0;
    var j = 0;
    for (j = 0; j < rps_points.length; j++) {
        x = rps_points[j];
        y = time_points[j];
        switch (hour_points[j]) {
            case 1:
            color1.push([x,y]);
            break;
            case 2:
            color2.push([x,y]);
            break;
            case 3:
            color3.push([x,y]);
            break;
            case 4:
            color4.push([x,y]);
            break;
            case 5:
            color5.push([x,y]);
            break;
            case 6:
            color6.push([x,y]);
            break;
            case 7:
            color7.push([x,y]);
            break;
            case 8:
            color8.push([x,y]);
            break;
            case 9:
            color9.push([x,y]);
            break;
            case 10:
            color10.push([x,y]);
            break;
            case 11:
            color11.push([x,y]);
            break;
            case 12:
            color12.push([x,y]);
```

```
                    break;
                    case 13:
                    color13.push([x,y]);
                    break;
                    case 14:
                    color14.push([x,y]);
                    break;
                    case 15:
                    color15.push([x,y]);
                    break;
                    case 16:
                    color16.push([x,y]);
                    break;
                    case 17:
                    color17.push([x,y]);
                    break;
                    case 18:
                    color18.push([x,y]);
                    break;
                    case 19:
                    color19.push([x,y]);
                    break;
                    case 20:
                    color20.push([x,y]);
                    break;
                    case 21:
                    color21.push([x,y]);
                    break;
                    case 22:
                    color22.push([x,y]);
                    break;
                    case 23:
                    color23.push([x,y]);
                    break;
                    case 0:
                    color24.push([x,y]);
                    break;
                    default:
                    color24.push([x,y]);
                }
            }
        ZC.LICENSE = ["569d52cefae586f634c54f86dc99e6a9",
"b55b025e438fa8a98e32482b5f768ff5"];
```

```
var myConfig = {
   "type": "mixed",
   "preview":{
   },
   "legend": {
      "layout": "12x2",
      "position": "100% -20%",
      "item": {
      "font-color": "brown",
      "font-family": "Georgia"
      }
   },
   "scale-x": {
      "zooming":true,
      "item":{
      "font-size":10
      },
   },
   "scale-y": {
      "zooming":true,
      "values": "0:1000:50",
      "guide":{
      "line-style":"dotted"
      },
      "item":{
      "font-size":10
      },
   },
   "plot": {
      "line-width":1,
      "marker":{
      "size":3,
      "border-width":1,
      },
      "tooltip":{
      "visible":false
      }
   },
   "crosshair-x":{
      "plot-label":{
      "text":"%v"
      },
      "scale-label":{
```

```
    "visible":false
    }
},
"crosshair-y":{
    "type": "multiple",
    "scale-label":{
    "visible":false
    }
},
"series": [
    {
        "type": "scatter",
        "values": color24,
        "text":"00:00",
    },
    {
        "type": "scatter",
        "values": color1,
        "text":"01:00",
    },
    {
        "type": "scatter",
        "values": color2,
        "text":"02:00",
    },
    {
        "type": "scatter",
        "values": color3,
        "text":"03:00",
    },
    {
        "type": "scatter",
        "values": color4,
        "text":"04:00",
    },
    {
        "type": "scatter",
        "values": color5,
        "text":"05:00",
    },
    {
        "type": "scatter",
        "values": color6,
        "text":"06:00",
```

```
},
{
   "type": "scatter",
   "values": color7,
   "text":"07:00",
},
{
   "type": "scatter",
   "values": color8,
   "text":"08:00",
},
{
   "type": "scatter",
   "values": color9,
   "text":"09:00",
},
{
   "type": "scatter",
   "values": color10,
   "text":"10:00",
},
{
   "type": "scatter",
   "values": color11,
   "text":"11:00",
},
{
   "type": "scatter",
   "values": color12,
   "text":"12:00",
},
{
   "type": "scatter",
   "values": color13,
   "text":"13:00",
},
{
   "type": "scatter",
   "values": color14,
   "text":"14:00",
},
{
   "type": "scatter",
   "values": color15,
```

```
    "text":"15:00",
},
{
    "type": "scatter",
    "values": color16,
    "text":"16:00",
},
{
    "type": "scatter",
    "values": color17,
    "text":"17:00",
},
{
    "type": "scatter",
    "values": color18,
    "text":"18:00",
},
{
    "type": "scatter",
    "values": color19,
    "text":"19:00",
},
{
    "type": "scatter",
    "values": color20,
    "text":"20:00",
},
{
    "type": "scatter",
    "values": color21,
    "text":"21:00",
},
{
    "type": "scatter",
    "values": color22,
    "text":"22:00",
},
{
    "type": "scatter",
    "values": color23,
    "text":"23:00",
},
{
    "type": "line",
```

```
                "values": approx,
                "text":"trend",
            }]};
        zingchart.render({
        id: 'myChart',
        data: myConfig,
        height: 400,
        width: "100%"
        });
    };
    function findLineByLeastSquares(x_vals, y_vals) {
        var sum_x = 0;
        var sum_y = 0;
        var xsum_y = 0;
        var xsum_x = 0;
        var count = 0;
        var x = 0;
        var y = 0;
        var values_length = x_vals.length;
        if (values_length != y_vals.length) {
            throw new Error('The parameters x_vals and y_vals need to have same size!');
        }
        if (values_length === 0) {
            return [ [], [] ];
        }
        for (let i = 0; i< values_length; i++) {
            x = x_vals[i];
            y = y_vals[i];
            sum_x+= x;
            sum_y+= y;
            xsum_x += x*x;
            xsum_y += x*y;
            count++;
        }
        var m = (count*xsum_y - sum_x*sum_y) / (count*xsum_x - sum_x*sum_x);
        var b = (sum_y/count) - (m*sum_x)/count;
        var result_values = [];
        for (let i = 0; i < values_length; i++) {
            x = x_vals[i];
            y = x * m + b;
            result_values.push([x,y]);
        }
        return result_values;
```

```
        }
        function readSomeLines(file, linemax, forEachLine, onComplete) {
            var CHUNK_SIZE = 50000;
            var decoder = new TextDec();
            var offset = 0;
            var linecnt = 0;
            var numline = 0;
            var res = '';
            var fr = new FileReader();
            fr.onload = function() {
                res += decoder.decode(fr.result, {stream: true});
                var lines = res.split('\n');
                res = lines.pop();
                linecnt += lines.length;
                if (linecnt > linemax) {
                    lines.length -= linecnt - linemax;
                    linecnt = linemax;
                }
                for (var i = 0; i < lines.length; ++i) {
                    forEachLine(lines[i] + '\n');
                }
                offset += CHUNK_SIZE;
                seek();
            };
            fr.onerror = function() {
                onComplete(fr.error);
            };
            seek();
            function seek() {
                if (linecnt === linemax) {
                    onComplete(); // Done.
                    return;
                }
                if (offset !== 0 && offset >= file.size) {
                    forEachLine(res);
                    onComplete();
                    return;
                }
                var slice = file.slice(offset, offset + CHUNK_SIZE);
                fr.readAsArrayBuffer(slice);
            }
        }
    </script>
</body>
```

```
</html>
```