Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Програмний засіб безпечного конфігурування кешування динамічного вмісту Веб-сторінок»

Виконав: студент 2 курсу, групи 2КІ-20м
напряму підготовки (спеціальності)
123 — «Комп'ютерна інженерія»
_____Курко В.С.

Керівник: доц. каф. ОТ

_____Азарова А.О.

«____» _____ 2021 р.

Опонент: к.т.н., доц., зав. каф МБІС

_____Карпінець В.В.

«____» _____ 2021 р.

Допущено до захисту
Завідувач кафедри ОТ
д.т.н., проф. Азаров О.Д.
«____» _____ 2021 р.

Вінниця ВНТУ - 2021

Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інформаційних технологій та комп'ютерної інженерії_____
Кафедра _____обчислювальної техніки_____
Освітньо-кваліфікаційний рівень _____магістр_____
Спеціальність 123 — «Комп'ютерна інженерія»_____
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
обчислювальної техніки
_____ проф., д.т.н. О.Д. Азаров

«___»_____2021 р.

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Курко Владиславу Сергійовичу

1 Тема роботи «Програмний засіб безпкечного конфігурування кешування динамічного вмісту веб-сторінок»
керівник роботи Азарова Анжеліка Олексіївна, к.т.н., професор,
затверджені наказом вищого навчального закладу від 24.09.2021 р. №227

2 Строк подання студентом роботи 15.12.2021 р.

3 Вихідні дані до роботи: процесор, не гірший ніж Intel Core i3, ОЗП не менше 4 Гб, точка доступу до мережі інтернет, підписка на хмарні сервіси Azure, веб-сторінки з динамічним контентом

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити). Вступ. Огляд і аналіз існуючих методів та засобів кешування вмісту веб-сторінок. Розробка алгоритму роботи програмного засобу. Розробка програми безпечного конфігурування кешування динамічного вмісту веб-сторінок. Розрахунок економічної доцільності створення програмного засобу.

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): Особливості побудови програмного засобу. Алгоритм модуля роботи на налаштування, Алгоритм роботи модуля перевірки кешу. Лістинг роботи програми.

6 Консультанти розділів роботи представлені в таблиці 1.

Таблиця 1 — Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1,2,3	Азарова О.Д., к.т.н., професор		
4	Лесько О. Й., к.е.н., професор		

7 Дата видачі завдання 07.09.2021 р.

Календарний план наведено в таблиці 2.

Таблиця 2 — Календарний план

№	Назва етапів виконання магістерської роботи	Строк виконання етапів роботи	Примітка
1	Постановка задачі роботи	07.09.21	
2	Огляд існуючих методів кешування динамічного вмісту веб-сторінок	08.09-09.09.21	
3	Аналіз та вибір методів кешуванняя динамічного вмісту веб-сторінок	10.09-18.09.21	
4	Розробка програмного засобу безпечного конфігування кешування динамічного вмісту веб-сторінок	19.09-01.10.21	
5	Розробка алгоритму кешування динамічного вмісту веб-сторінок	12.10-22.10.21	
6	Розробка програми ешування динамічного вмісту веб-сторінок	22.10-31.10.21	
7	Розробка інтерфейсу та тестування програми кешування динамічного вмісту веб-сторінок	01.11-10.11.21	
8	Підготовка матеріалів та опис розробки засобів ешування динамічного вмісту веб-сторінок	11.11-16.11.21	
9	Розрахунок економічної частини роботи	17.11-30.11.21	
10	Оформлення пояснювальної записки та ілюстративного матеріалу	01.12-06.12.21	
11	Аналіз виконання роботи, висновки, додатки	07.12-06.12.21	
12	Перевірка якості виконання магістерської роботи та усунення недоліків	15.12.21	

Студент _____ Курко В.С.
Керівник роботи _____ Азарова А.О.

АНОТАЦІЯ

УДК 004.42

Курко В.С. Програмний засіб безпечного конфігурування кешування динамічного вмісту Веб-сторінок. Магістерська кваліфікаційна робота зі спеціальності 123 – комп'ютерна інженерія, освітня програма – комп'ютерна інженерія. Вінниця: ВНТУ, 2021. 92 с.

На укр.мові. Бібліогр.: 52 назв; рис.: 37 ; табл. 8.

В магістерській кваліфікаційній роботі розроблено програмний засіб для безпечного конфігурування кешування динамічного вмісту веб-сторінок, метою якого є спрощення роботи користувача в процесі конфігурування динамічного контенту, який може бути закешований, зменшення ризику кешування персональних даних, збільшення відсотку закешованих сторінок, які містять динамічний контент, зниження навантаження на основний сервер, та пришвидшення завантаження сторінок для кінцевого користувача.

Ключові слова: веб-сайт, веб-сервіс, кешування, динамічне кешування, Azure, хмарні технології.

ANNOTATION

Kurko V.S. Software for securely configuring dynamic content caching of Web pages. Master's thesis in specialty 123 – computer engineering, educational program – computer engineering. Vinnytsia: VNTU, 2021 – 92 p.

In Ukrainian language. Bibliografer: 52 titles; fig .: 37; tabl. 8.

The master's thesis developed a software for secure configuration of caching dynamic content of web pages, which aims to simplify the user's process of configuring dynamic content that can be cached, reduce the risk of caching personal data, increase the percentage of cached pages containing dynamic content. reduce the load on the main server, and speed up the loading of pages for the end user.

Key words: website, web service, caching, dynamic caching, Azure, cloud technologies.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧКИ

Web — система доступу до пов'язаних між собою документів на різних комп'ютерах, підключених до Інтернету

БД — база даних

ПЗ — програмне забезпечення

HTML — HyperText Markup Language

ESI — Edge Side Includes

CDN — Content Delivery Network

HTTP — HyperText Transfer Protocol

CSS — Cascading Style Sheets

AMP — Accelerated Mobile Pages

SEO — Search Engine Optimization

MVC — Model-View-Controller

API — Application Programming Interface

CMS — Content management system

PHP — Hypertext Preprocessor

ВСТУП

Зі збільшенням кількості користувачів глобальної мережі Інтернет, проблема доставки об'ємного контенту стає все більш актуальною. Виникає необхідність пошуку методів оптимізації процесів оброблення запитів та доставки вмісту веб-сторінок. Особливо це важливо для контенту, який необхідно водночас роздати великій кількості користувачів. Виникають нові технології, такі як балансування трафіку, розподілені мережі доставки контенту, кешування та ін. Разом із цим, повною мірою така проблема залишається не вирішеною.

Одним із продуктивних підходів для оптимізації процесу оброблення та доставки контенту є пошук способу кешування динамічно змінюваних даних. Цінність динамічного кешування полягає в економії витрат на хостинг та сервер. Багато CDN [1] стверджують, що пропонують таку можливість, але на сьогоднішній день немає жодного рішення, яке б надало можливість зробити процес кешування динамічного вмісту веб-сторінок дійсно доступним [2].

Динамічність означає, що вміст, який регулярно змінюється, є персоналізованим для деякої підмножини користувачів або, навіть, унікальним для кожного користувача веб-сайту. Насправді динамічне кешування зазвичай відноситься до кешування HTML-документа, який є будівельним блоком усієї веб-сторінки і може змінюватися досить часто. Особливо це стосується веб-сайтів електронної комерції або медіа-сайтів, які регулярно оновлюють вміст сторінок, щоб відображати останні статті, пропоновані товари чи ціни. Динамічний вміст не включає в себе інформацію про обліковий запис та персональні дані користувача, що часто відображається у верхньому куті веб-сторінки або в його особистому кабінеті – цей вміст не слід кешувати. [3]

Кешування HTML-документа приводить до більшої швидкості, ніж кешування статичних об’єктів, таких як зображення, оскільки HTML-документ – це перше, що потрібно створити та надіслати сервером-джерелом веб-сайту після

його підключення до браузера. Він містить інструкції про те, як зобразити веб-сторінку та де знайти пов’язані файли, такі як рисунки, таблиці стилів CSS та коди JavaScript. Якщо 1000 відвідувачів відвідують веб-сайт протягом 1 хвилини, HTML-документ потрібно буде генерувати вихідним сервером 1000 разів. Через цей стрес на вихідних серверах менеджерам веб-сайтів необхідно планувати максимальний обсяг трафіку, який, на їхню думку, вони матимуть у будь-який час. Це означає купівлю серверів для пікових періодів трафіку, навіть якщо ці піки досягаються лише в 1% випадків [4].

Документ HTML є основою всієї веб-сторінки, тому багато веб-сайтів вважають, що надто ризиковано кешувати його. Це пояснюється тим, що існуючі нині методи конфігурації кешу не дозволяють належним чином налаштовувати виняткове кешування динамічного контенту та перевірити і протестувати конфігурацію кешу.

Отже, актуальність дослідження зумовлюється потребою розв’язку задачі надання динамічності вмісту веб-сторінок, що максимально відповідає вимогам користувача. Зокрема, щоб забезпечити максимальний комфорт користувача під час користування веб-сайтом, необхідно надати йому інформацію про товар, послуги сайту та ін. згідно з його потребами.

Об’єктом дослідження є процес конфігурування кешування динамічного вмісту веб-сторінок.

Предметом дослідження є методи і алгоритми кешування динамічного вмісту веб-сторінок.

Мета роботи — підвищення рівня відповідності динамічного вмісту веб-сторінок вимогам користувача засобами конфігурування кешування динамічного вмісту веб-сторінок.

Для досягнення цієї мети було поставлено та розв’язано такі задачі:

— проаналізовано методи і засоби кешування динамічного вмісту веб сторінок;

— обґрунтовано вибір найбільш ефективної технології конфігурування кешування веб-сторінок;

— розроблено метод кешування динамічного вмісту веб-сторінок із можливістю безпечного конфігурування;

— запропоновано алгоритм кешування динамічного вмісту веб-сторінок;

— обрано інструментарій для розроблення програмного продукту;

— перевірити ефективність створеного програмного додатку;

— дослідити економічну доцільність створеного програмного додатку.

Наукова новизна роботи полягає в удосконаленні методу конфігурації кешування динамічного вмісту веб-сторінок, що на основі безпечного конфігурування дозволяє підвищити відповідність контенту вимогам користувача.

Практична цінність роботи полягає у створенні програмного продукту, що уможливлює зростання відповідності контенту вимогам користувача у різних сферах, зокрема, в комп’ютерних системах маркетингу, рекламному бізнесі, торгівлі та ін.

Методами досліджень, використаними у магістерській роботі, є об'єктно-орієнтовані методи проектування, що є базовими для кешування динамічного вмісту веб-сторінок; методи теорії обчислювальних систем і теорії масового обслуговування для побудови математичних моделей, які застосовуються для створення програмного забезпечення конфігурування кешування динамічного вмісту веб-сторінок.

За результатами дослідження що були виконані в магістерській кваліфікаційній роботі, подано тези доповіді на Всеукраїнську науково-практичну інтернет-конференцію «Молодь в науці: дослідження, проблеми, перспективи (МН-2022)», Вінниця, 2022.

1 АНАЛІЗ МЕТОДІВ ТА ЗАСОБІВ КЕШУВАННЯ ВМІСТУ ВЕБ-СТОРІНОК

1.1 Статичний та динамічний вміст веб-сторінок

Статичний контент — це будь-який файл, який зберігається на сервері, і той самий щоразу, коли він доставляється користувачам. HTML-файли та зображення є прикладами такого змісту. Статичний вміст схожий на газету: щойно виходить номер газети, він містить ті самі статті та фотографії для кожного, хто візьме в руки примірник, незалежно від того, які нові події відбудуться протягом дня [6].

Динамічний вміст — це вміст, який змінюється залежно від факторів, характерних для користувача, таких як час відвідування, місцезнаходження та пристрій. Динамічна веб-сторінка не буде виглядати однаково для всіх, і вона може змінюватися під час взаємодії користувачів із нею — наприклад, якби газета могла переписати себе, коли її хтось читає. Це робить веб-сторінки більш персоналізованими та більш інтерактивними. Порівняння статичного та динамічного контенту зображене на рис. 1.1.

Рисунок 1.1 — Порівняння статичного та динамічного контенту [6]

Сучасний веб-сайт новин — хороший приклад динамічного вмісту: на відміну від газет, статті оновлюються протягом дня, а на домашній сторінці можуть бути різні заголовки залежно від місцезнаходження відвідувача сайту або статусу входу. Сторінки соціальних медіа є ще одним прикладом: стрічка

новин Facebook виглядає абсолютно по різному для кожного користувача, і користувачі можуть взаємодіяти з вмістом, щоб змінити його.

Динамічні веб-сторінки не зберігаються у вигляді статичних файлів HTML. Натомість сценарії на стороні сервера генерують HTML-файл у відповідь на події, такі як взаємодія користувачів або вхід користувача, і надсилають HTML-файл у веб-браузер. Оскільки динамічний вміст створюється на стороні сервера, він зазвичай подається з вихідних серверів, а не з кешу.

Довгий час динамічний вміст вважався некешованим. Але нові технології дозволяють веб-сайтам подавати динамічний вміст із кешу, значно скорочуючи затримку, зберігаючи при цьому взаємодію з користувачами.

Звичайний процес веб-кешування — це збереження у кеші копії статичного файлу, такого як зображення. При подачі вмісту сторінки, картинка зберігається в кеш користувача і швидше доставляється наступного разу. Браузери та мережі доставки контенту (CDN) можуть кешувати статичний вміст протягом певного періоду часу і подавати його користувачам до тих пір, поки вміст запитуватиметься. Це можливо, оскільки статичний вміст не змінюється з плином часу; один і той же файл може надходити користувачам знову і знову [6].

Динамічний вміст генерується сценаріями, які змінюють вміст сторінки. Запускаючи сценарії в кеші CDN, а не на віддаленому вихідному сервері, динамічний вміст можна генерувати та доставляти з кешу. Таким чином, динамічний вміст, по суті, "кешується" і не повинен обслуговуватися весь шлях від початку, зменшуючи час відповіді на запити клієнта та прискорюючи динамічні веб-сторінки.

Інший підхід до прискорення динамічних веб-сторінок полягає у стисненні динамічного вмісту, створеного вихідним сервером, і його доставки максимально швидко та ефективно. При динамічному стисненні вміст все ще надходить із вихідного сервера, а не з кешу, але сформовані HTML-файли значно зменшуються, щоб вони могли швидше досягти клієнтського пристрою.

Часто велика кількість вмісту на динамічній веб-сторінці залишається послідовною для всіх користувачів, і лише певні елементи на сторінці є динамічними. Це означає, що значна частина HTML-коду дублюється в кожній динамічній копії сторінки.

Щоб вирішити цю неефективність, ряд компаній працювали разом над розробкою Edge Side Includes (ESI) [7, 8], мови розмітки, яка визначає, де динамічний вміст відображається на веб-сторінці. (ESI використовується на деяких CDN, але ще не прийнятий W3C, організацією, що керує веб-стандартами.)

Вміст із тегом ESI завантажується з іншого місця, а решту вмісту веб-сторінки можна кешувати. Якщо лише частина веб-сторінки генерується динамічно, а решта кешується, веб-сторінка завантажуватиметься набагато швидше, ніж якби всю сторінку потрібно було створити для кожного користувача.

1.2 Кешування та класифікація кешів

«Вебкеш» (або «кеш HTTP») — інформаційна технологія для тимчасового зберігання (кешування) веб-документів і медіа контенту задля зменшення серверних затримок. Система веб-кешу зберігає копії документів, що проходять через неї. Подальші запити можуть бути виконані з кешу за певних умов. Система веб-кешу може посилатися або на програмно-апаратний комплекс, або на комп'ютерну програму [9].

Продуктивність веб-сайтів і додатків можна значно підвищити за рахунок повторного використання раніше отриманих ресурсів. Веб-кеші скорочують затримку і знижують мережевий трафік, зменшуючи тим самим час, необхідне для відображення ресурсів. Використовуючи HTTP-кешування, сайти стають більш чуйними.

Основною перевагою кешування є прискорення роботи веб-сторінок. А більш швидкі веб-сторінки покращують якість користування, а це означає, що відвідувачі веб-сайту будуть щасливішими. Численні дослідження показали, що

користувачі відвідують більше сторінок веб-сайту, коли він завантажується швидше.

Для медіа-компанії це може означати більше статей та оголошень, а для сайту електронної комерції це означає, що клієнти переглядають більше продуктів. Насправді навіть було показано, що швидші веб-сторінки призводять до збільшення конверсій та доходу, а затримка веб-сторінки лише на 1 секунду призводить до середньої втрати доходу на 7% [10, 11]. Більш швидкі веб-сторінки також означають, що пошукові системи більш сприятливо ставляться до вашого веб-сайту, покращуючи його цінність SEO і означаючи, що більше людей знаходять ваш сайт.

Під час відвідування користувачем веб-сторінки він використовує веб-браузер, щоб запитувати цю сторінку із сервера веб-сайту. На сервері зберігаються всі файли, необхідні для збирання цієї веб-сторінки, включаючи документи HTML, зображення, текст, стилі тощо.

У середньому браузер надсилає більше 100 запитів до сервера веб-сайту для створення повної веб-сторінки.

Без будь-якого типу кешування, коли користувач відвідує цю сторінку, він повторює ці запити заново. І кожна інша особа, яка відвідує цю веб-сторінку, робить такі ж запити. Якщо до сторінки одночасно звертається багато людей, сервер сповільнюється і потребує більше часу, щоб доставити веб-сторінку всім.

Кешування вирішує цю проблему, зберігаючи копію зібраної веб-сторінки в кількох різних місцях. Ця копія тимчасово зберігається десь, крім сервера веб-сайту, тому браузеру не потрібно повертатися туди щоразу, коли він завантажуєте ту саму сторінку.

Розглянемо два основні місця кешування та їх роботу.

Один із способів кешування вмісту — безпосередньо на жорсткому диску персонального комп’ютера. Веб-браузери роблять це автоматично для веб-сторінок, які відвідують користувачі, тому їм не потрібно повертатися на сервер веб-сайту, щоб знову завантажити кожен окремий елемент. [12, 13].

Наприклад, логотип веб-сайту часто повторюється на кожній веб-сторінці. Якщо цей логотип є у кеші браузера, браузеру не потрібно повторно завантажувати його для кожної сторінки на веб-сайті, який відвідує користувач.

Багатьом користувачам знайомі фрази «очистити кеш» або «очистити дані веб-переглядача» – це одна з перших речей, яку інженери просять зробити при усуненні несправностей, коли веб-сторінка відображається неправильно, або відображає внесені оновлення.

Очищення кешу видаляє всі файли, збережені на комп'ютері, змушуючи браузер повернутися на сервер веб-сайту та завантажити «чисту» копію веб-сторінки [14].

Іноді веб-сторінки також кешуються на стороні сервера веб-сайту, а не на персональному комп'ютері. Коли веб-сайт встановлює кеш поверх на стороні свого сервера, він зберігає копії відповідних файлів та інструкцій у цьому кеші.

Коли браузер запитує дані з сервера веб-сайту, він спочатку потрапляє в кеш, і якщо в кеші є недавня копія веб-сторінки, яку браузер намагається запросити, кеш доставляє зібраний вміст безпосередньо у веб-браузер.

Веб-сайт може контролювати, як часто потрібно оновлювати їх кешований вміст: якщо це перший відвідувач веб-сторінки після того, як кеш-вміст застарів, кеш знову збиратиме нову версію веб-сторінки з сервера, а потім доставлятиме цей вміст та зберігатиме його до тих пір, поки ця нова версія також не застаріє (рис. 1.2) [15].

Часто кешовані файли однакові для всіх користувачів і не змінюються часто. Вони можуть включати:

— статичні зображення;
— логотипи та активи бренду;
— таблиці стилів;
— файли Javascript, які не змінюються;
— завантажувані файли або інший вміст.

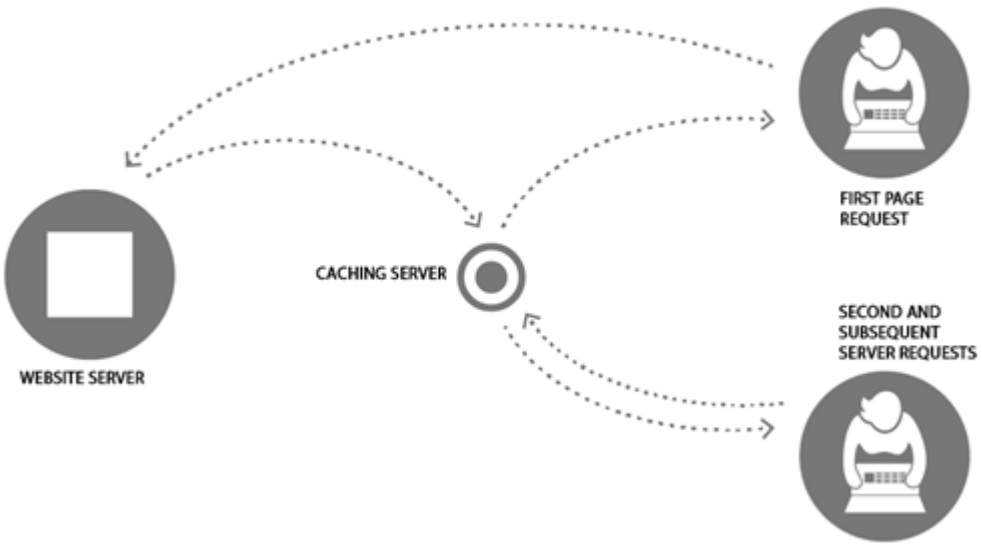


Рисунок 1.2 — Схема роботи кешу, налаштованого на сервері веб-сайту [16]

Файли, які можна кешувати, але їх рідко кешують:

— повні сторінки HTML;

— файли Javascript або інший код, який змінюється частіше.

Файли, які не слід кешувати, включають: спеціальні дані користувача, такі як інформація про обліковий запис, яка відрізняється для кожного відвідувача та будь-які конфіденційні дані, такі як інформація про банківські послуги або кредитні картки [17].

Техніка кешування полягає в збереженні копії отриманого ресурсу для повернення цієї копії у відповідь на подальші запити. Запит на ресурс, вже наявний в веб-кеші, перехоплюється, і замість звернення до вихідного сервера виконується завантаження копії з кеша. Таким чином знижується навантаження на сервер, якому не доводиться самому обслугову-вати всіх клієнтів, і підвищується продуктивність – кеш ближче до клієнта і ресурс передається швидше. Кешування є основним джерелом підвищення продуктивності веб-сайтів. Однак, кеш треба правильно конфігурувати: ресурси рідко залишаються незмінними, так що копію потрібно зберігати тільки до того моменту, як ресурс змінився, але не довше [18, 19].

Існує кілька видів кешей, які можна розділити на дві основні категорії: приватні кеши і кеши спільного використання. У кешах спільного використання

(shared cache) зберігаються копії, які можуть направлятися різним користувачам. Приватний кеш (private cache) призначений для окремого користувача. Порівняння цих двох типів кешу зображене на рис. 1.3.

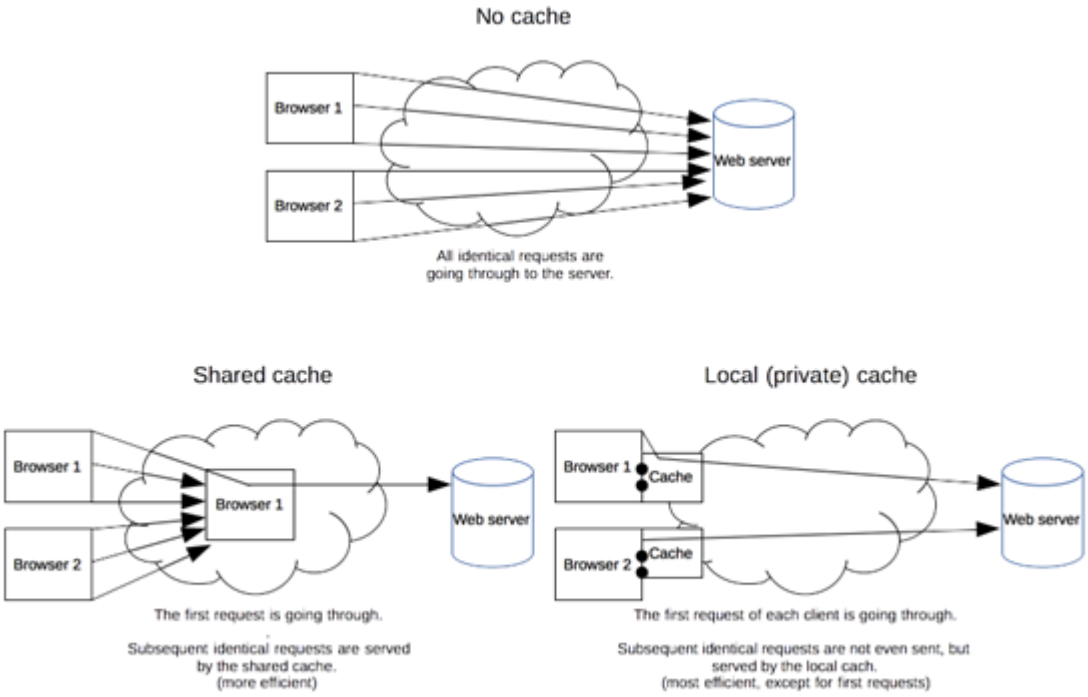


Рисунок 1.3 — Схема роботи кешу, налаштованого на сервері веб-сайту [21]

Приватний кеш призначений для окремого користувача. Ви, можливо, вже бачили параметри кешування в налаштуваннях свого браузера. Кеш браузера містить всі документи, завантажені користувачем по HTTP. Він використовується для доступу до раніше завантажених сторінок при навігації назад/вперед, дозволяє зберігати сторінки, або переглядати їх код, не звертаючись зайвий раз до сервера. Крім того, кеш корисний при відключенні від мережі [20].

За типом кешування можна виділити чотири рівні кешу: клієнтський, мережевий, серверний, додатка.

Кеш спільного використання — це кеш, який зберігає відповіді, щоб їх потім могли використовувати різні користувачі. Наприклад, в локальній мережі вашого провайдера або компанії, може бути встановлений проксі, обслуговуючий безліч користувачів, щоб можна було повторно

використовувати популярні ресурси, скорочуючи тим самим мережевий трафік і час очікування. [22]

1.3 Управління кешуванням та валідація кешу

Протокол HTTP надає можливість управляти кешуванням завдяки деяким спеціальним заголовкам запиту.

Поле Cache-Control загального заголовка HTTP / 1.1 використовується для завдання інструкцій по механізму кешування як в запитах, так і у відповідях, а також застосовується для завдання політик кешування [23].

Повна відсутність кешування, у кеші не повинно зберігатися нічого — ні за запитами клієнта, ні з відповідей сервера. Запит щоразу відправляється на сервер, відповідь завжди завантажується повністю. Можливі параметри для такої конфігурації: no-cache, no-store, must-revalidate

Кешувати, але перевіряти актуальність, перед тим, як видати копію, кеш запитує вихідний сервер на предмет актуальності ресурсу.

Приватний (private) і загальний (public) кеш, директива "public" вказує, що відповідь можна зберігати в будь-якому кеші. Це буває корисно, якщо виникає потреба зберегти сторінки з HTTP-аутентифікації, або такими кодами відповіді, які зазвичай не кешуються. Директива ж "private" вказує, що відповідь призначений окремому користувачеві і не повинен зберігатися в кеші спільного використання. У цьому випадку відповідь може зберігатися приватним кешем браузера [24, 25].

Термін дії (Expiration), найважливішою тут є директива "max-age = <seconds>" — максимальний час, протягом якого ресурс вважається "свіжим". На відміну від директиви Expires, вона прив'язана до моменту запиту. До незмінних файлів програми зазвичай можна застосовувати "агресивне" кешування. Прикладом таких статичних файлів можуть бути зображення, файли стилів (CSS) або скриптів (JavaScript).

Перевірка актуальності, при використанні директиви "must-revalidate" кеш зобов'язаний перевіряти статус ресурсів із закінченим терміном дії. Ті

копії, що втратили актуальність, використовуватися не повинні.

Pragma є заголовком HTTP / 1.0. Він не описаний для HTTP-відповідей і, таким чином, не може служити надійною заміною загальному заголовку Cache-Control протоколу HTTP / 1.1, хоча його поведінка аналогічно "Cache-Control: no-cache" коли поле заголовка Cache-Control опущено в запиті. Використовувати його слід тільки для сумісності з клієнтами HTTP / 1.0.

Одного разу потрапивши в кеш, ресурс, теоретично, може зберігатися там вічно. Однак, оскільки обсяг сховища кінцевий, записи періодично доводиться звідти видаляти. Цей процес називають витісненням даних з кеша (cache eviction). Крім того, ресурси можуть змінюватися на сервері, тому кеш потрібно оновлювати. Оскільки HTTP є клієнт-серверним протоколом, сервера не можуть самі звертатися до кешам і клієнтам при зміні ресурсу; їм необхідно домовитися про термін дії збереженої копії. До його закінчення ресурс вважається свіжим (fresh), після — застарілим (stale). Алгоритми витіснення віддають перевагу "свіжим" ресурсів. Проте, копія ресурсу не видаляється з кеша відразу ж після закінчення її терміну дії. При отриманні запиту на застарілий ресурс кеш передає його далі з заголовком If-None-Match (en-US) на випадок, якщо копія все ще актуальна. Якщо це так, сервер повертає заголовок 304 Not Modified («нічого не змінено»), а тіло ресурсу не посилає, економлячи тим самим трафік [26]. На рисунку 1.4 відображено приклад того, як протікає цей процес при використанні спільного кеша проксі.

Термін дії (freshnessLifetime) обчислюється на підставі декількох заголовків. Якщо заданий заголовок "Cache-control: max-age = N", то термін дії дорівнює N. Якщо його немає, а це буває дуже часто, перевіряється заголовок Expires, і, якщо він є, то термін дії береться рівним значенню заголовка Expires мінус значення заголовка Date. Нарешті, якщо немає ні того ні іншого, дивляться заголовок Last-Modified. Якщо він є, то термін дії дорівнює значенню заголовка Date мінус значення заголовка Last-modified розділити на 10.

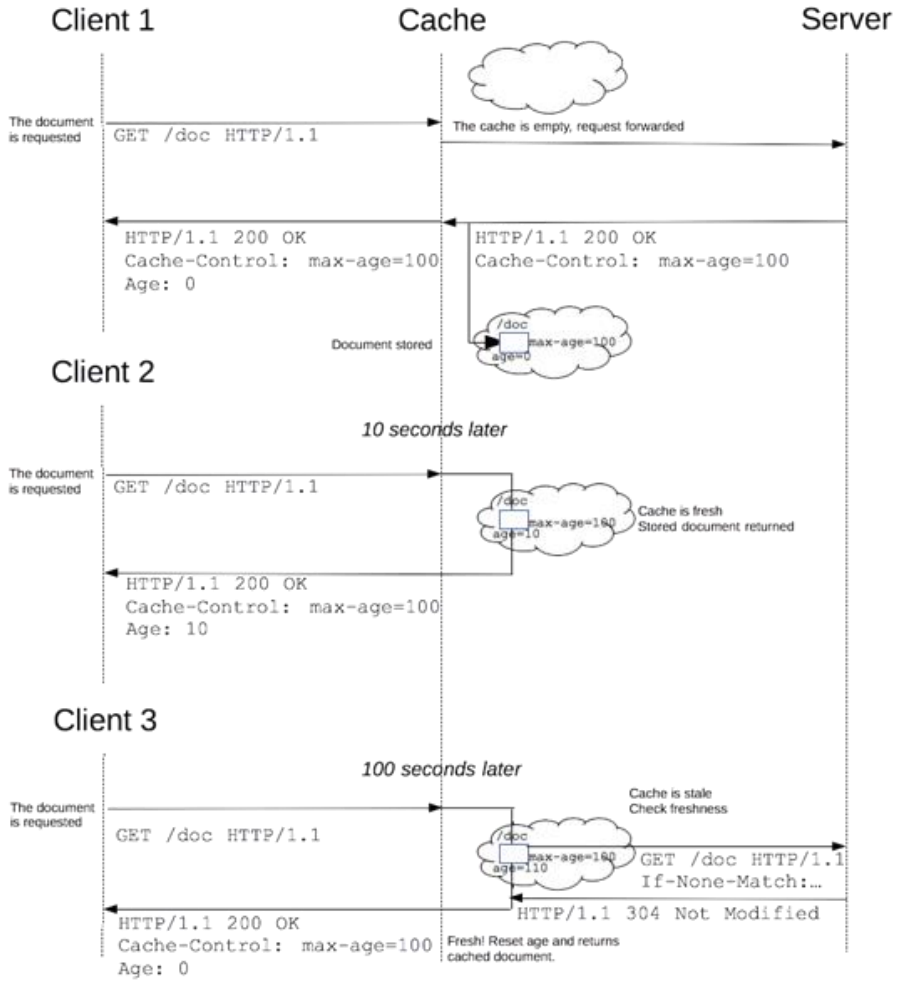


Рисунок 1.4 — Схема процесу використання спільного кешу проксі [27]

Час старіння (expirationTime) обчислюється таким чином.

expirationTime = responseTime + freshnessLifetime – currentAge,

де responseTime — це час отримання відповіді по годинах браузера;

currentAge — поточний вік кеша.

Валідація кеша запускається при натисканні користувачем кнопки перезавантаження. Крім того, вона може виконуватися в ході звичайного перегляду сторінок, якщо кеш відповідь включає заголовок "Cache-control: must-revalidate". Іншим фактором є налаштування кешування браузера — можна вимагати примусової валідації при кожному завантаженні документа.

При закінченні терміну придатності документа він або проходить

валідацію, або повторно доставляється з сервера. Валідація може виконуватися тільки якщо на сервері реалізований сильний валідатор або слабкий валідатор.

Заголовок відповіді ETag [28] є непрозорим для клієнтського додатка (агента) значенням, яке можна використовувати в якості сильного валідатора. Суть в тому, що клієнт, наприклад, браузер, не знає, що являє цей рядок і не може передбачити, яким буде її значення. Якщо у відповіді присутній заголовок ETag, клієнт може транслювати його значення через заголовок If-None-Match (en-US) майбутніх запитів для валідації кеш ресурсу.

Заголоваок відповіді Last-Modified можна використовувати в якості слабкого валідатора. Слабким він вважається через те, що має 1-секундне дозвіл. Якщо у відповіді присутній заголовок Last-Modified, то для валідації кеш документа клієнт може виводити в запитах заголовок If-Modified-Since.

При запиті на валідацію сервер може або проігнорувати валідацію і послати стандартну відповідь 200 OK, або повернути відповідь 304 Not Modified (з порожнім тілом), тим самим вказуючи браузеру взяти копію з кеша. В останньому випадку у відповідь можуть входити також заголовки для поновлення строку дії кеш ресурсу.

Заголовок HTTP-відповіді Vary визначає, як по заголовкам майбутніх запитів зрозуміти, чи може бути використана копія з кеша, або потрібно запитати нові дані у сервера [29].

Якщо кеш отримує запит, який можна задовольнити збереженим в кеші відповіддю з заголовком Vary, то використовувати цю відповідь можна тільки при збігу всіх зазначених в Vary полів заголовка вихідного (збереженого в кеші) запиту і нового запиту.

1.4 Кешування в мережах доставки контенту (CDN)

У зв'язку з поширенням мереж доставки контенту, CDN, важливо зрозуміти, як кешування працює на проміжних точках веб-мережі.

Першу мережа доставки контенту була запущена Akamai орієнтовно 20 років тому [30]. Він перевернув світ, розповсюдивши глобальні зворотні проксі

сервери у багатьох точках присутності (Points of Presence), які діяли б як кеші вмісту, і провели шар DNS зверху, щоб вибрати найближчий PoP до користувача.

На веб-сайти того часу, вплив був миттєвим. Зображення, які постійно передавалися з вихідних серверів почали передаватися кінцевому користувачу із зворотних проксі-серверів Akamai. Це означало меншу кількість запитів до вихідній серверу та збільшення швидкості доставки до клієнта через меншу затримку.

Вміст доставлявся з мережі зворотних проксі-серверів Akamai. І народилася назва CDN.

Розповсюджуючи контент ближче до відвідувачів веб-сайту за допомогою найближчого сервера CDN (серед інших оптимізацій), відвідувачі швидше завантажують сторінки. Оскільки відвідувачі більш схильні виходити з сайтів, що повільно завантажуються, CDN може зменшити показники відмов та збільшити кількість часу, який люди проводять на сайті. Іншими словами, чим більша швидкість обробки даних на веб-сайті — тим більше відвідувачів залишатимуться і продовжують ним користуватися.

Витрати на споживання пропускної спроможності для розміщення веб-сайтів є основними витратами для веб-сайтів.

Якщо 1000 відвідувачів відкриють одну й ту саму сторінку веб-сайту протягом 1 хвилини, документ потрібно буде відправити з вихідного сервера 1000 разів.

Через цей стрес на вихідних серверах менеджерам веб-сайтів необхідно планувати максимальний обсяг трафіку, який, на їхню думку, вони матимуть у будь-який час. Це означає купівлю серверів для пікових періодів трафіку, навіть якщо ці піки досягаються лише в 1% випадків [31 – 32].

Коли документ кешується, кешуючий сервер є єдиним, хто робить запит до вихідного сервера: Якщо даний документ налаштовано кешувати протягом 1 хвилини, то сервер кешування зробить один запит до вихідного серверу за хвилину, незалежно від того, чи відвідує веб-сайт 10 відвідувачів чи 1000

відвідувачів за цю хвилину (рис. 1.5).

Це означає, що сервери веб-сайту звільняються для критичних навантажень, збільшуючи кількість відвідувачів, яких можна обслуговувати одночасно. В той самий час це зменшує кількість серверів, які потрібно купувати у разі пікового трафіку. Це заощаджує витрати підприємства на розміщення та гарантує, що користувачі матимуть хороший досвід роботи.

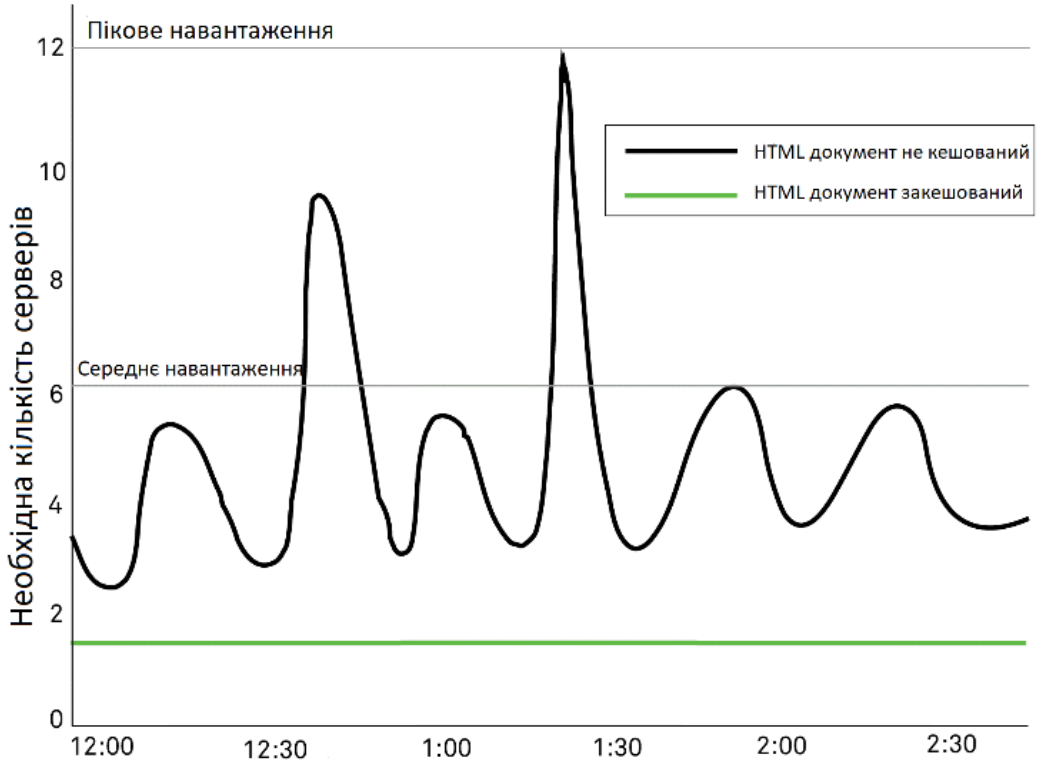


Рисунок 1.5 — Порівняння навантаження на вихідний сервер з та без використання кешування в CDN [33]

Місткі обсяги трафіку або збої обладнання можуть призводити до некоректної роботи веб-сайту. Завдяки чіткій та розподіленій структурі CDN, надається можливість обробляти об’ємніше кількість трафіку і витримувати апаратні збої краще, ніж багато серверів-початківців.

CDN може покращити безпеку, забезпечуючи пом'якшення DDoS, покращення сертифікатів безпеки та інші оптимізації.

При правильному налаштуванні, CDN передає ваш контент клієнтам через найшвидший і найближчий до нього сервер (рис. 1.6). Крім цього, CDN

працює буфером між сервером і користувачами. Важливим є відсоток кількості кешованих запитів – тих запитів, які CDN опрацювало, не перевантажуючи сервер. Залежно від трафіку і архітектури цей номер може досягати і 90% [34], хоча ефект буде помітно і при менших цифрах. Треба зауважити, що при невеликій кількості запитів велика їх частина буде відправлятися на сервер, тому цей відсоток має сенс тільки разом з часом кешування і загальним навантаженням сайту. Але якщо ви налаштуєте один лише кеш, а заголовки кешування працюватимуть неправильно, підсумкові показники можуть навіть стати гіршою, ніж були.

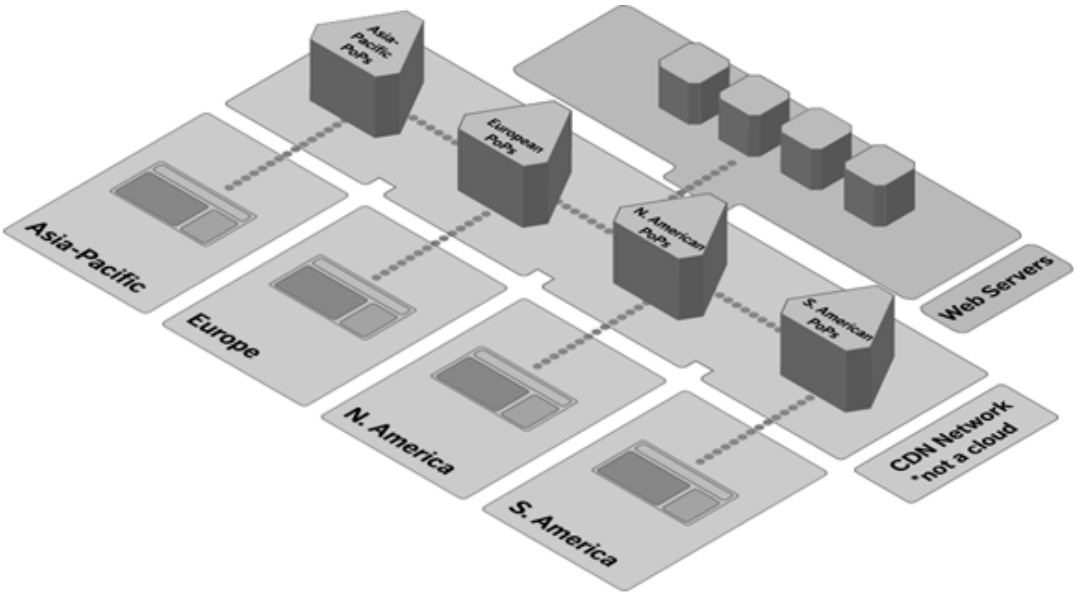


Рисунок 1.6 — Схема географічного розподілення мережі доставки контенту [35]

Здебільшого література рекомендує при запитах динамічного вмісту веб-сторінок ставити cache-control: no-cache, щоб CDN кешував його. Хоча, велика частина динамічного контенту не така вже й непостійна. Список активних користувачів, наприклад, має час життя від 10 до 20 секунд [36 – 37].

Сторінки з графіками напевно можуть пожити кілька хвилин. Новинна стрічка може деякий час і покешитися, особливо при наявності etag. Тому є сенс кешування динамічного контенту, при правильному вибірковому налаштуванні параметрів такого кешування.

1.5 Аналіз часу кешування

Час, який витрачається на кешування певних ресурсів залежить від кількості трафіку, розміру ресурсів, розміру кеша. Крім того, треба подумати про головний недолік кешування – зменшенні контролю над ресурсами.

Якщо існує необхідність оновити ресурс миттєво, то можуть виникнути проблеми, в тому випадку, якщо деякий час назад йому був заданий час життя в рік. Особливо, якщо це значення було задано не тільки для CDN (s-maxage), але і для користувачів (maxage).

Найтриваліший період для кеша — рік, або 31536000. Але це погана ідея. Якщо ваші сервера не витримують хоча б щоденних запитів від CDN з приводу того, чи змінився ресурс – найкращим рішенням буде підвищити продуктивніть та пропускні можливості сервера а не кешу [38, 39].

У таблиці 1.1 відображено втрати часу на кеш. Припускаючи, що ресурс отримує 500 запитів в хвилину, для різних варіантів часу зберігання ресурсу виходять наступний відсоток запитів до кешу.

Таблиця 1.1 — Використання часу для здійснення кешування

Час кешування (хв)	Відсоток запитів до кешу	Запити до оригіналу на год.
1	99.8%	60
5	99.96%	12
20	99.99%	3
60	99.97%	1
86400	99.9998%	<1

Зазвичай від 60 секунд до 60 хвилин - нормальний час життя для ресурсу. Для псевдо-динамічного контенту можна встановлювати час кешування в проміжку до 60 секунд.

1.6 Підсумки до розділу

Отже, в даному розділі здійснювалось дослідження та аналіз матеріалу обраної галузі, а саме вивчено особливості статичного та динамічного вмісту веб-сторінок, поняття кешування та класифікація кешів, управління

кешуванням та валідація кешу, кешування в мережах доставки контенту (CDN), аналіз часу кешування.

В результаті проведеного аналізу теоретичного матеріалу та виходячи з мети і актуальності теми, були поставлені наступні задачі подальшої роботи:

— розроблено метод кешування динамічного вмісту веб-сторінок із можливістю безпечного конфігурування;

— запропоновано алгоритм кешування динамічного вмісту веб-сторінок;

— обрано інструментарій для розроблення програмного продукту;

— перевірити ефективність створеного програмного додатку;

— дослідити економічну доцільність створеного програмного додатку.

В результаті виконання поставлених завдань, планується досягти основної мети роботи, а саме підвищення рівня відповідності динамічного вмісту веб-сторінок вимогам користувача засобами конфігурування кешування динамічного вмісту веб-сторінок.

2 РОЗРОБЛЕННЯ МЕТОДУ КЕШУВАННЯ ДИНАМІЧНОГО ВМІСТУ ВЕБ-СТОРІНОК

2.1 Принципи безпечного конфігурування кешування динамічного вмісту веб-сторінок

Оптимальний час очікування завантаження сторінки — 2 секунди з десктопу та від 3 до 4 секунд з мобільної версії. Це цифри, які більшість користувачів вважають найбільш прийнятними. Насправді, все виглядає дещо інакше. Середній час завантаження сайтів у різних сегментах eCommerce не відповідає ні очікуванням користувача, ні рекомендаціям пошукових систем. Таку статистику представив Google, проаналізувавши середні показники швидкості завантаження сайтів США (рис. 2.1).

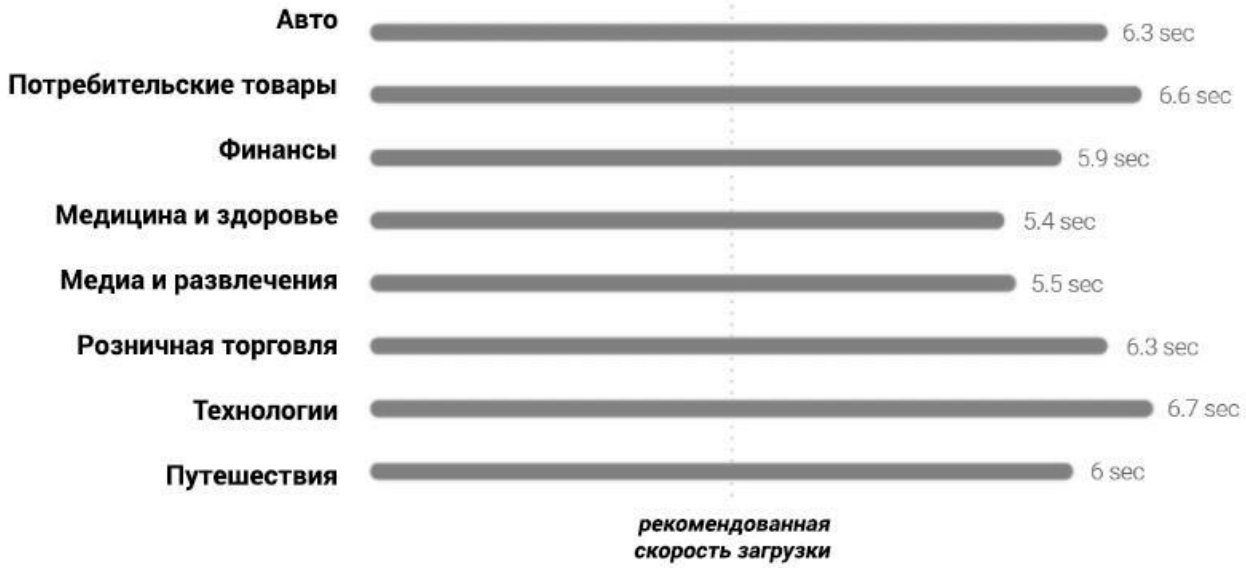


Рисунок 2.1 — Аналіз швидкості завантаження файлів [41]

Перевіривши сайт кількома інструментами, можна отримати точну картину швидкості завантаження сторінок. Для виправлення більшості проблем не потрібно глибоких знань та навичок. Але частину оптимізації, насамперед реалізовану на рівні коду, краще делегувати спеціалістам.

Далі розглянемо типовий перелік проблем, які погіршують час

завантаження сторінок, і з’ясуємо, як їх усунути.

Зменшення розміру HTML-коду. Розмір HTML-коду безпосередньо впливає на швидкість завантаження: чим більше файлів, тим більше ресурсів витрачає браузер для візуалізації сторінки. Оптимально, якщо розмір HTML не перевищує 100-200 кілобайт. Оптимізація у разі передбачає видалення всього непотрібного: повторюваних прогалин, переносів рядків, коментарів до коду, зайвих знаків та ін.

Зробити це можна двома способами: вручну та за допомогою спеціальних інструментів. Зі зрозумілих причин серед вебмайстрів більш популярний другий спосіб: він не вимагає навичок верстки і дозволяє значно прискорити роботу. Перед тим як оптимізувати документи, бажано зробити їх резервні копії. Незважаючи на те, що до сучасних сервісів і плагінів для стиснення HTML-коду немає особливих нарікань, запобіжні заходи все ж таки не зашкодять [42].

Знайти сторінки з великим розміром HTML коду допомагають сервіси Sitechecker, вже згаданий GTMetrix або Screaming Frog.

Для безпосередньої оптимізації можна використати сервіс Daruse. Це онлайн інструмент, який за кілька секунд компресує код, не порушуючи його функціональності. Для сайтів на WordPress HTML-код можна оптимізувати за допомогою популярних плагінів Better WordPress Minify та W3 Total Cache.

Оптимізація зображень. На більшості сайтів картинки є одними з найважчих файлів, на завантаження яких йде основна частина пропускної спроможності. У середньому 45% ваги сторінки відводиться під зображення. Вони стають найвужчим місцем, яке збільшує час відповіді. Тому оптимізація графічного контенту є пріоритетним завданням тих, хто хоче прискорити свій сайт. Найбільш ефективними є три шляхи вирішення цього питання.

Використання форматів, адаптованих до web. Йдеться про такі формати як WebP та JPEG-XR. Їхні ключові переваги — висока якість при мінімальній вазі. Але є й суттєві недоліки: досі їх підтримують далеко не всі браузери та

індексують не всі пошукові системи [43].

Використання механізму відкладеного завантаження зображень. Цей метод дозволяє відтермінувати завантаження картинки до того моменту, доки користувач не доскролить сторінку до неї. Застосування цього способу дає можливість суттєво прискорити сайт та підвищити його продуктивність. Для швидкого використання цієї опції на CMS-сайтах можна застосувати плагіни, такі як A3 Lazy Load для WordPress.

Метод «лінивого» підвантаження ефективно використовують і для інших елементів. Наприклад, якщо JavaScript-файл не потрібен користувачеві на початку сторінки, доцільно відкласти його завантаження до того моменту, поки до нього не докролять. Це дозволить досягти більш швидкого стартового завантаження. Аналогічно можна вчинити з відео або іншими файлами на сторінці [44].

Деактивація непотрібних плагінів. Однією з важливих переваг сайтів на CMS є доступність установки всіляких плагінів, що розширюють функціональність веб-ресурсу. З ними у багатьох вебмайстрів швидко втрачається відчуття міри. Перспектива в кілька кліків зробити свій сайт більш функціональним та ефективним здається дуже привабливою. Але це явище має і зворотний бік — для роботи кожного плагіна потрібні серверні ресурси. Додаткові програмні модулі збільшують кількість запитів до бази даних, що значно підвищує час завантаження сторінки. Якщо надлишок плагінів уповільнює сайт, слід відключити зайві з них. Другий варіант вирішення цієї проблеми — перехід на тариф хостингу, зі збільшеним обсягом ресурсів, що виділяються.

Налаштування клієнтського кешування. У разі клієнтського кешування браузер локально зберігає файли і не робить до них повторних звернень. Під час відвідування вже переглянутих сторінок браузер не буде робити щоразу запит на сервер, а завантажить контент локально — з кеша. Такий механізм значно прискорює завантаження сторінок, особливо коли йдеться про картинки та CSS/Javascript-файли. Клієнтське кешування можна налаштувати за

допомогою CMS-плагінів або за допомогою файлу .htaccess.

GZIP стиск. GZIP стиск — це ще один механізм кешування, але вже на рівні сервера. Технологія мінімізує загальний обсяг даних, які сервер надсилає кінцевому користувачеві. GZIP може зменшувати розмір веб-сторінок та CSS до 50-70%, за рахунок чого досягають відчутного зниження часу завантаження сторінки.

Перенесення відеоконтенту на зовнішні платформи. Відео, безумовно, додає цінності будь-якому сайту. Це ефективний маркетинговий інструмент, який мотивує відвідувачів на цільові дії — 64% споживачів із більшою ймовірністю купують товари в інтернеті, якщо можуть переглянути презентаційне відео про продукт. Крім маркетингу, відеоконтент — хороший засіб покращення поведінкових факторів на сторінці: він збільшує час, проведений користувачем на сайті, та знижує показники відмов.

Важливим питанням є місце збереження відео. Від завантаження таких файлів на власний сервер слід відмовитися, навіть якщо використовується прокачений хостинг. Це пояснюється кількома причинами. Важкий контент уповільнює швидкість завантаження сторінок, крім того, необхідно постійно думати про вільне місце на хостингу та масштабувати його ресурси, витрачаючи на це додаткові кошти (рис. 2.2).

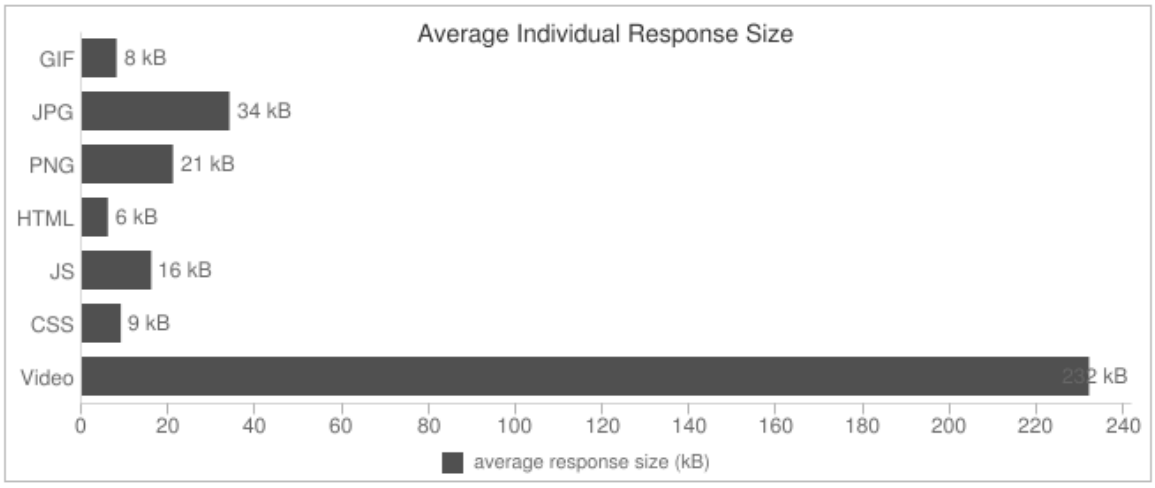


Рисунок 2.2 — Завантаження відеоконтенту [45]

Також зберігання файлів на власному сервері може погіршити взаємодію

користувачів із сайтом. При одночасному перегляді відео сервери з обмеженою пропускною здатністю будуть викликати затримки і гальмувати відео, що не найкраще позначиться на якості досвіду користувача.

Оптимальне рішення в цьому випадку — розміщення або перенесення файлів на сторонній відеохостинг. Необхідно завантажувати відео на сервіси YouTube, Vistia або Vimeo, а потім додавати посилання на сторінку. Це заощаджує місце на сервері, прискорює роботу сайту і уможливлює його постійну доступність для користувачів, оскільки зі стабільністю серверів того ж YouTube сьогодні може порівнятися далеко не кожен хостинг-провайдер.

Використання CDN. Мережі доставки контенту (CDN) здатні вирішити багато проблем, пов'язаних із продуктивністю сайту та швидкістю його завантаження.

CDN-інфраструктура є мережею серверів зі спеціальним ПЗ, розташованих у різних частинах світу (рис. 2.3).

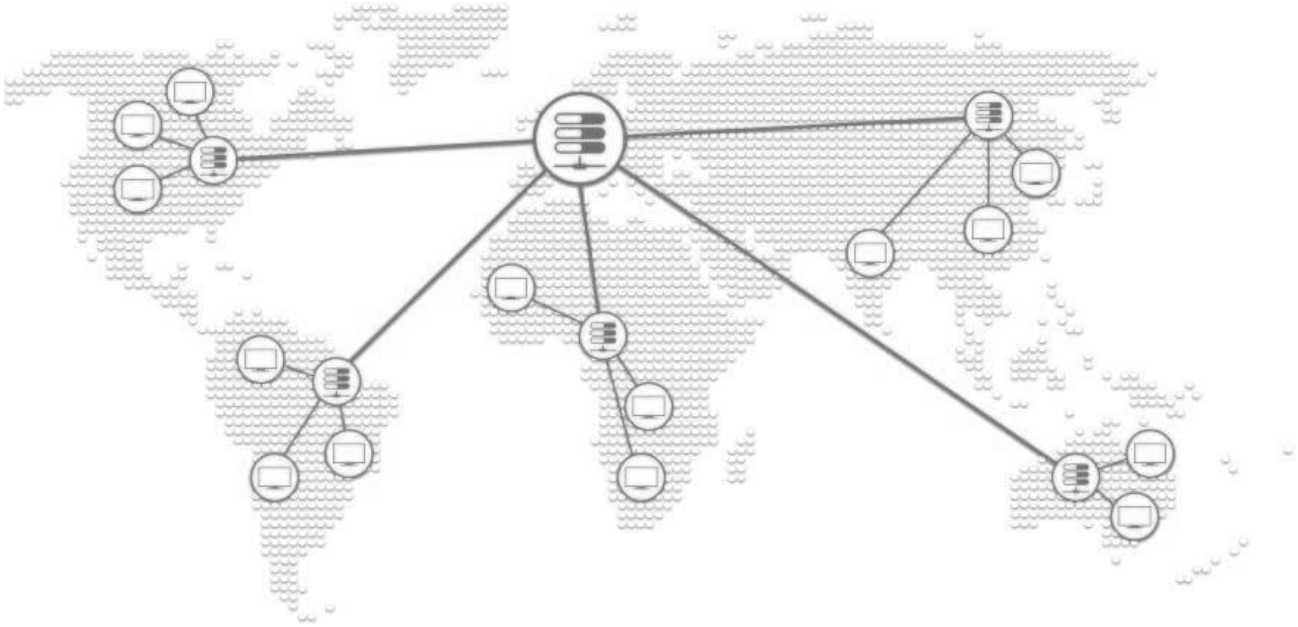


Рисунок 2.3 — Використання контенту CDN [46]

Система кешує вміст сайту та зберігає його на всіх фізичних машинах мережі. При запиті клієнт відправляється на найближчий до нього сервер, що істотно скорочує затримку між запитом і відповіддю.

Таким чином, перша проблема, яку ефективно вирішує CDN —

оптимізація часу завантаження користувачів, географічно віддалених від сервера.

На відвідуваних сайтах з великою кількістю звернень обробка запитів займає багато часу, що уповільнює швидкість відгуку сайту. Це друга проблема, яку вирішує CDN. Мережа доставки контенту розподіляє навантаження між різними серверами, завдяки чому навіть за критично великого обсягу трафіку сайт вантажиться максимально швидко.

Крім цього, використання CDN зменшує ймовірність втрати доступу до вмісту сайту через падіння сервера — при виникненні форс-мажорних ситуацій користувачі будуть перенаправлені на резервні сервери мережі.

Таким чином, CDN — незамінна річ для відвідуваних сайтів, орієнтованих на глобальну (в географічному плані) аудиторію [47, 48].

При великому обсязі трафіку, що надходить з різних континентів, це, мабуть, єдина можливість ефективно знизити затримку сайту та забезпечити швидку роздачу важкого контенту для всіх користувачів.

Отже, важливість швидкості завантаження сторінок не викликає сумнівів. Яким би якісним та оптимізованим із базового SEO не був сайт, його повільне завантаження може звести нанівець успіх всього проекту.

Стандартний пул робіт із прискорення сайту передбачає оптимізацію зображень, чищення html-коду від зайвого сміття, настроювання клієнтського та серверного кешування, використання AMP та Турбо-сторінок, а також інші практики, детально розглянуті вище.

Зосереджуючись на внутрішній оптимізації, важливо не забувати про використання якісного хостингу, який відповідає поточним навантаженням сайту. Крім того, швидкість завантаження має перевірятися на регулярній основі.

2.2 Особливості побудови програмного засобу

Програмне забезпечення веб-кешування — це проксі-сервер, який використовується для веб-прискорення або скорочення часу, необхідного для

доступу до веб-сайту. Його можна завантажити на певний пристрій або він може бути автономним [49, 50].

Веб-кешування прискорює використання вмісту сторінок без залучення додаткового серверного обладнання.

Програмне забезпечення для динамічного веб-кешування може функціонувати за допомогою різноманітних методів, таких як оптимізація HTTP, кешування та попередня вибірка, стиснення та обробка SSL/TLS. Ці рішення використовуються веб-сайтами, які мають бути функціональними для тисяч або десятків тисяч відвідувачів і допомагають підтримувати роботу служб у звичайному режимі для всіх відвідувачів.

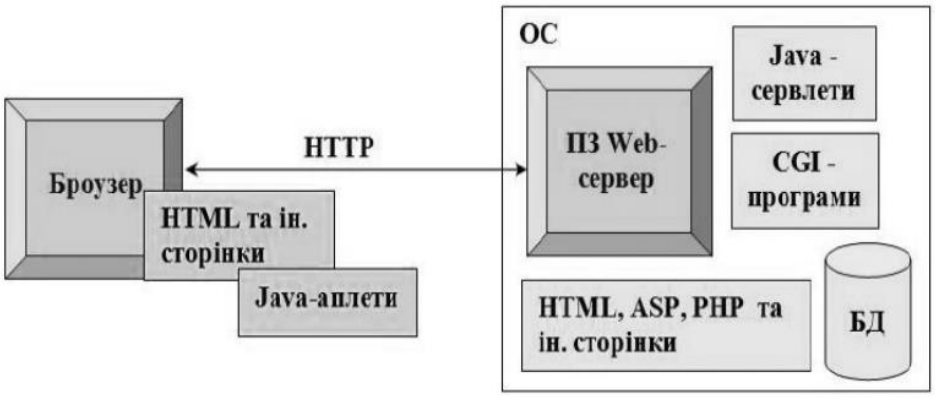


Рисунок 2.4 — Програмне забезпечення веб-сервера [51]

Для реалізації поставленої мети було розроблено інтелектуальний модуль аналізу параметрів та налаштувань для оцінювання можливості кешування та правильного способу кешування веб сторінки, що містить динамічний контент.

Складовими елементами такої розробки є:

— web-browser;

— server;

— база даних кешу;

— база даних налаштувань кешу;

— модуль кешування даних;

— модуль перевірки кешу;

— модуль конфігурації кешу;

— клієнт.

Загальна схема роботи такого програмного забезпечення зображена на рис. 2.5.

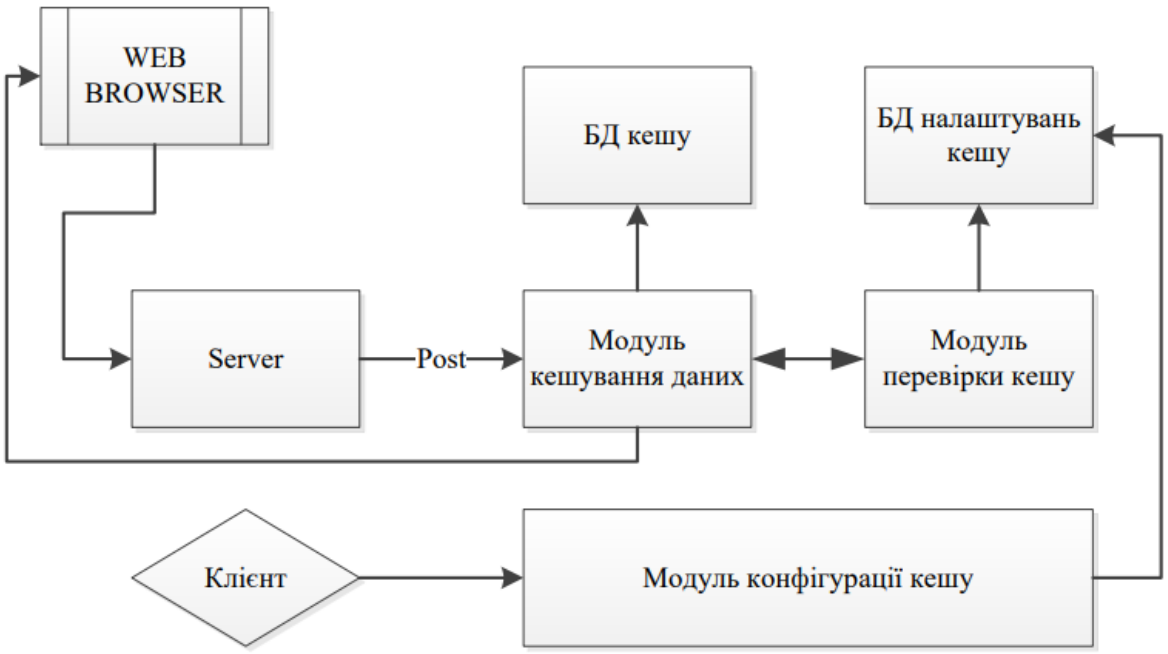


Рисунок 2.5 — Загальна схема програмного засобу для кешування динамічного вмісту веб-сторінок

Клієнт або власник веб-сайту за допомогою «Модулю конфігурації кешу» та зручного інтерфейсу налаштовує спосіб кешування кожної окремої сторінки сайт. Далі ці дані потрапляють до «БД налаштувань кешу», де міститься інформація щодо налаштувань для кожної сторінки.

Коли веб-браузер звертається до веб-серверу, той, в свою чергу, відправляє запит до «модулю кешування данних». Цей модуль відповідає за збереження та отримання сторінок з кешу. Крім того він звертається до «модуля перевірки», що необхідний для того, щоб визначити, чи можна кешувати сторінку за поданим запитом.

Якщо можна, тоді модуль кешування дістає сторінку з кешу. Якщо не можна, то сервер самостійно збирає сторінку. А, якщо можна, але такої

сторінки в кеші поки що немає, то сервер самостійно збирає сторінку, і перед відправленням її користувачу, передає її до «модулю кешування данних», який, власне, і кешує її.

Задачу ефективного розподілу навантаження даниими та відповідно процесами кешування, можна розділити на декілька простіших завдань;

— управління доступом до ресурсів з метою визначення чи достатньо ресурсів для додання нового навантаження;

— розміщення робочого навантаження на програмні додатки з метою отримання рекомендації щодо оптимального розміщення навантаження для зменшення кількості використовуваних серверів;

— прогнозування потреб навантаження.

Таким чином, слід враховувати фактори, які впливають на швидкість отриманні відповіді від сервера (рис. 2.6)

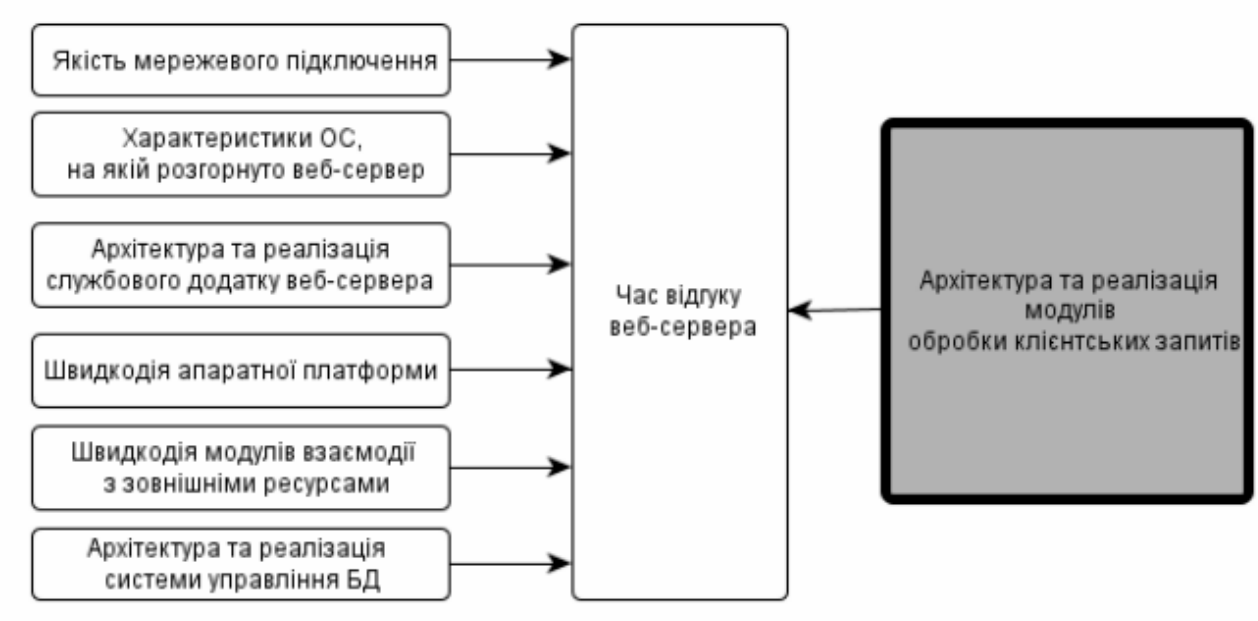


Рисунок 2.6 — Умови, що впливають на час відгуку веб-сервера [52]

Практично всі великі програмні системи є розподіленими. Розподіленою називається така система, в якій оброблення інформації зосереджене не на одній обчислювальній машині, а розподілене на кількох ЕОМ. Клауд сервіси, такі як Azure дозволяють вирішити питання, які дадуть нам можливість

створити декілька екземплярів кожного модуля, якими необхідно користуватись, фізично розмістити їх в будь-якій точці, якомога ближче до користувача, налаштувати автоматичне масштабу-вання та безпечну і стійку комунікацію між різними компонентами.

Microsoft Azure надає декілька різних способів розміщення та виконання коду або робочих процесів без використання віртуальних машин (ВМ), включаючи Azure функції. Функція

Azure — це простий спосіб запускати невеликі фрагменти коду в хмарі, не турбуючись про інфраструктуру, необхідну для розміщення цього коду. Функцію можна написати на C#, Java, JavaScript, PowerShell, Python чи багатьох інших популярних мовах програмування. Azure автоматично масштабує функцію у відповідь на запит користувачів.

Якщо програма складається з компонентів, запущених на різних комп’ютерах, серверах і мобільних пристроях, зв’язок між цими компонентами може бути складним і ненадійним. Azure надає кілька техно-логій, які можна використовувати для надійнішого зв’язку, зокрема черги Storage queues, Event Hubs, Event Grid, and Service Bus.

Саме тому було прийняте рішення будувати даний додаток на базі хмарних сервісів Azure, а для комунікації між компонентами використовувати Azure Event Grid та Azure Service Bus.

2.3 Розроблення алгоритмів роботи модулів з налаштувань та перевірки кеша

Для початку клієнт завантажує сайтмап свого сайту, після чого всі сторінки сайту будуть графічно відображені в деревовидній структурі. Далі клієнт проходиться по всіх сторінках, обирає які сторінки кешувати не можна, конфігурує параметри за якими можна кешувати сторінку та налаштовує валідацію кешу. Представимо граф-схему алгоритму роботи модуля з налаштувань на рис. 2.7.

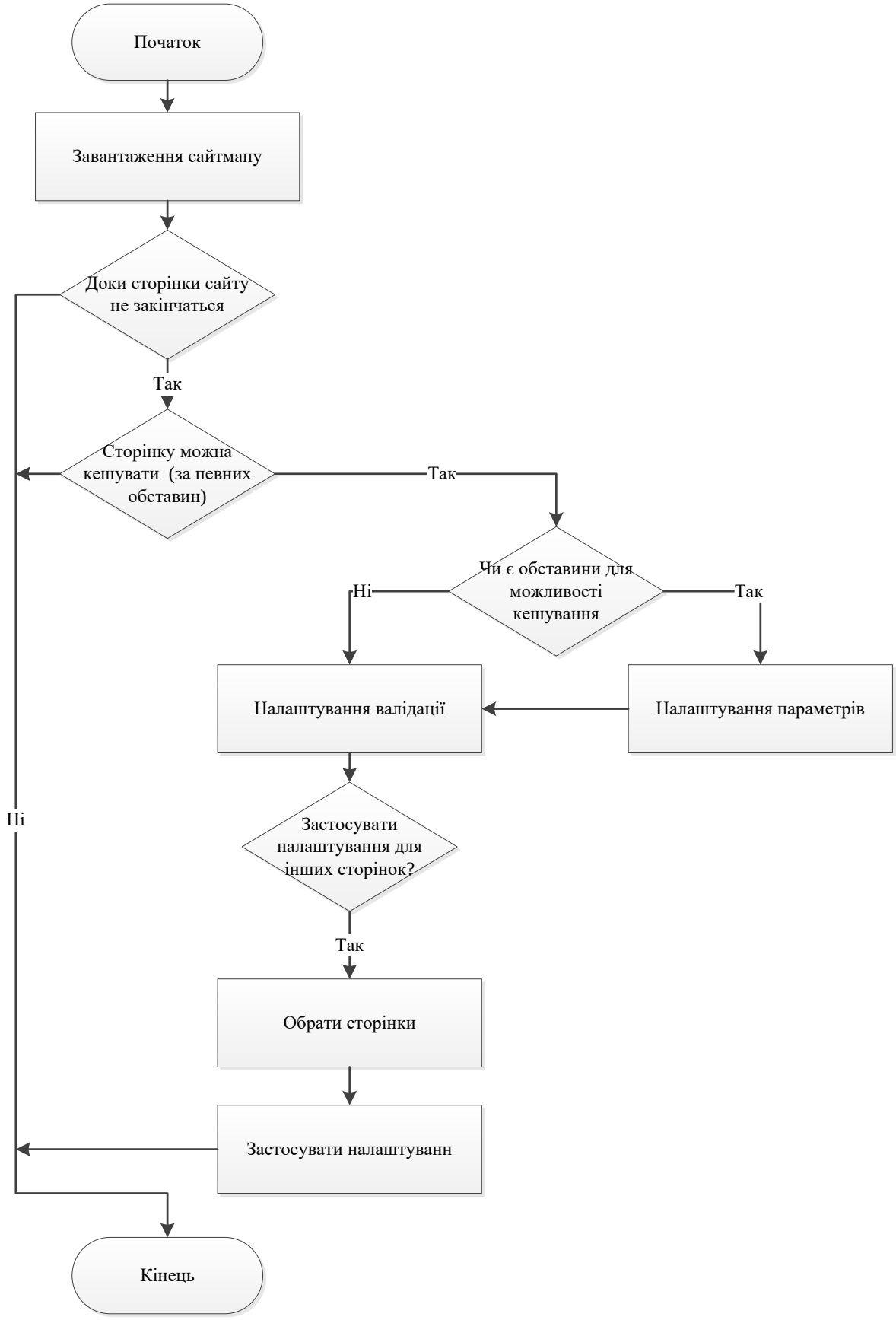


Рисунок 2.7 — Граф-схема алгоритму роботи модуля з налаштувань

Алгоритм роботи даного модулю можна описати наступним чином.

Крок 1 — початок роботи з розроблюваним програмним засобом.

Крок 2 — завантаження сайт мапу для подальшої роботи.

Крок 3 — визначення чи можлива перевірка усіх сторінок ресурсу.

Крок 3.1 — якщо перевірка можлива то відбувається перехід до кроку 4.

Крок 3.2 — якщо перевірка заборонена то відбувається перехід до кроку 11.

Крок 4 — перевірка дозволу кешування сторінки.

Крок 4.1 — якщо дозвіл на кешування можливий то відбувається перехід до кроку 5.

Крок 4.2 — якщо дозвіл на кешування заборонений то відбувається перехід до кроку 11.

Крок 5 — перевірка наявності можливості кешування.

Крок 5.1 — якщо така можливість існує то відбувається перехід до кроку 6.

Крок 5.2 —якщо така можливість відсутня то відбувається перехід до кроку 7.

Крок 6 — налаштування параметрів кешування динамічного вмісту веб-сторінок та перехід до кроку 7.

Крок 7 — налаштування валідації для кешування динамічного вмісту веб-сторінок, та перехід до кроку 8.

Крок 8 — перевірка можливості застосування реалізованих налаштувань для інших сторінок.

Крок 8.1 — якщо така перевірка успішна, відбувається перехід до кроку 9.

Крок 8.2 —я кщо така перевірка не успішна, відбувається перехід до кроку 11.

Крок 9 — процес обрання сторінок для застосування налаштувань.

Крок 10 — застосування реалізованих налаштувань до сторінок обраних під час виконання кроку 9.

Крок 11 — завершення роботи програмного засобу.

Таким чином, модуль конфігурації кешу буде генерувати бази даних та інші модулі в хмарі у вигляді Azure сервісів, з використанням всіх переваг та найсучасніших технологій побудови розподілених програмних систем, які пропонує Azure. Всі сервіси будуть створюватися в тій кількості, яку захоче клієнт, і до того ж розташовуватися фізично найближче до кінцевого користувача. Завдяки цьому буде зменшено час відповіді та підсилено пропускову здатність всього додатку.

Розглянемо граф-схему алгоритму роботи модуля перевірки кеша, що запропоновано на рис. 2.8.

Алгоритм роботи даного модулю можна описати наступним чином:

Крок 1 — початок роботи з модулем.

Крок 2 — у форматі ISON приходить відповідь на попередній запит даних.

Крок 3 — здійснення запиту HTML-сторінки.

Крок 3.1 — якщо запит коректний, відправляється запит на дозвіл кешування сторінки. Перехід до кроку 4.

Крок 3.2 — якщо запит сторінки відхилений, повертається негативна відповідь, перехід до кроку 7. Закінчення роботи з модулем.

Крок 4 — отримання відповіді на дозвіл для кешування.

Крок 4.1 —у випадку отримання дозволу на відправлений запит, відбувається перехід до кроку 5.

Крок 4.2 —у випадку відхиленого запиту повертається негативна відповідь, перехід до кроку 7. Закінчення роботи з модулем.

Крок 5 — перевірка валідності кешу.

Крок 5.1 — у випадку, якщо валідність кешу підтверджена, відбувається перехід до кроку 6.

Крок 5.2 —у випадку, якщо валідність кешу не підтверджена, повертається негативна відповідь, перехід до кроку 5. Закінчення роботи з модулем.

Крок 6 — надання даних про результати кешування. Перехід до кроку 7.

Крок 7 — Завершення роботи з модулем.

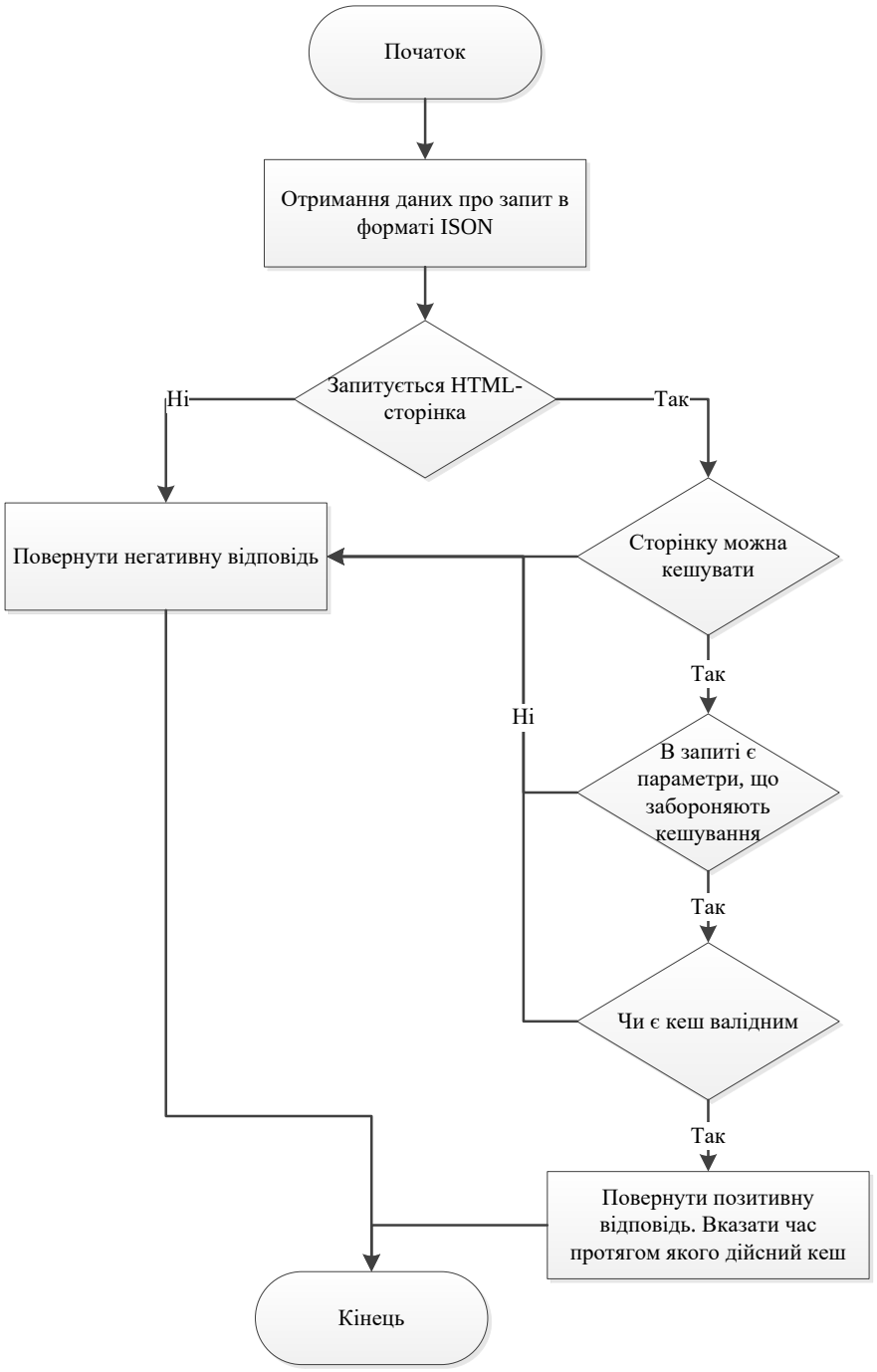


Рисунок 2.8 — Загальна схема ПЗ для кешування вмісту веб-сторінок

Отже, такий модуль перевірки кеша після отримання запиту перевіряє чи це запит саме HTML сторінки, а не статичного ресурсу, дозвіл на кешування наявність параметрів, що забороняють кешування, валідність кешу, та повертає відповідний результат.

2.4 Підсумки до розділу

Отже, в даному розділі було здійснено розроблення методу кешування динамічного вмісту веб-сторінок. В ході роботи були досліджені та проаналізовані принципи безпечного конфігурування кешування динамічного вмісту веб-сторінок, описані особливості побудови програмного засобу, практично здійснено розроблення алгоритмів роботи модулів з налаштувань та перевірки кеша, зокрема, модуль конфігурації кешу, який буде генерувати бази даних та інші модулі в хмарі у вигляді Azure сервісів, з використанням всіх переваг та найсучасніших технологій побудови розподілених програмних систем, які пропонує Azure та модуль перевірки кеша після отримання запиту перевіряє чи це запит саме HTML сторінки, а не статичного ресурсу, дозвіл на кешування наявність параметрів, що забороняють кешування, валідність кешу, та повертає відповідний результат.

3 РОЗРОБКА ПРОГРАМНОГО ЗАСОБУ БЕЗПЕЧНОГО КОНФІГУРУВАННЯ КЕШУВАННЯ ВЕБ-СТОРІНОК

3.1 Вибір інструментарію реалізації програмного засобу

Враховуючи поставлені мету та задачі магістерської дипломної роботи для практичної реалізації програмного засобу безпечного конфігурування кешування динамічного вмісту веб-сторінок було обрано сучасні та практичні засоби реалізації, а саме мову програмування C#, платформу Asp.net core, візуалізатор сторінок Razor Pages, сервіси Microsoft Azure (Azure functions, Azure Storage, Azure Cosmos DB), бібліотеки System.Threading, System.Threading.Tasks для асинхронних операцій.

C# — це мова програмування загального призначення, надзвичайно популярна, проста та гнучка у використанні. Це структурована мова програмування, яка не залежить від машини і широко використовується для написання різних програм, операційних систем, таких як Windows, та багатьох інших складних програм, таких як база даних Oracle, Git, інтерпретатор Python тощо [53].

Вважається, що «C#» — це мова надважлива програмування. Можна сказати, C є основою програмування. Якщо ви знаєте «C», ви можете легко осягнути знання інших мов програмування, які використовують концепцію «C».

Також, важливою умовою є наявність досвіду роботи з механізмами пам’яті комп’ютера, оскільки це важливий аспект при роботі з мовою програмування C.

C# — це скомпільована мова. Компілятор — це спеціальний інструмент, який компілює програму та перетворює її в об'єктний файл, який є доступний для апаратного зчитування. Після процесу компіляції компонувальник об’єднає різні об’єктні файли та створить єдиний ініціюючий файл для запуску програми. На рис. 3.1 показано виконання програми «C#».

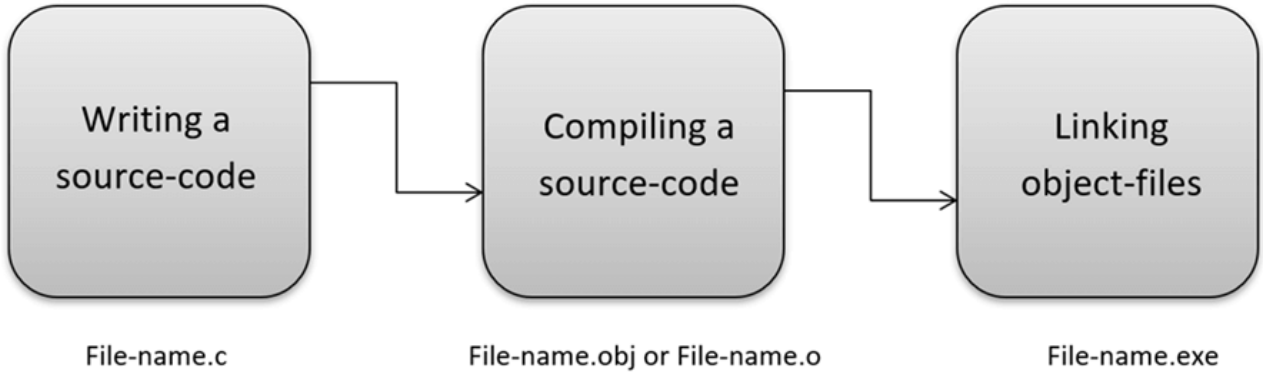


Рисунок 3.1 — Схема виконання програмного засобу на C# [54]

Нині в Інтернеті доступні різноманітні компілятори, якими можна послугуватися. Їх функціональність практично не відрізняється і більшість з них надають функції, необхідні для виконання програм «C» і «C++».

Нижче наведено список популярних компіляторів, доступних в Інтернеті: компілятор Clang; компілятор MinGW (мінімалістський GNU для Windows); Портативний компілятор «C»; Турбо C.

ASP.NET — це платформа для розробки в Інтернеті (сайти, додатки). Вона підтримує роботу з кількома мовами програмування, що входять до складання фреймворку: Basic NET, C#, J# та інших. На цій платформі можна створювати як найпростіші веб-ресурси, так і дуже складні сайти, здатні до оброблення багатотисячного потоку користувачів [55].

ASP.NET за швидкістю роботи значно перевершує інші скриптові мови. Причина швидкої обробки полягає в тому, що основа компілюється під час першого підключення користувача і додається до кеша комп'ютера. Всі наступні переходи по сайту використовують кешований код, який просто виймається з пам'яті, а не скачується з сервера повторно. Такий підхід суттєво заощаджує час на парсингу, завантаженні та обробці файлів.

В ASP.NET застосовується традиційна схема MVC — Модель-Вид-Контролер (ASP NET Core MVC). Ця схема часто зустрічається в C# (C# MVC). Усі елементи відповідають за конкретні дії. Наприклад, користувач запускає процес реєстрації та відправляє на сервер реєстраційні дані. Контролер

інтерпретує дії людини та передає моделі інформацію про внесені зміни до статусу користувача. Модель реагує на дії контролера і працює з даними, що поставляються. Візуалізація сторінки відповідає за відображення інформації з моделі.

Функціонування стандартної реалізації архітектурного принципу MVC наведено на рис. 3.2.

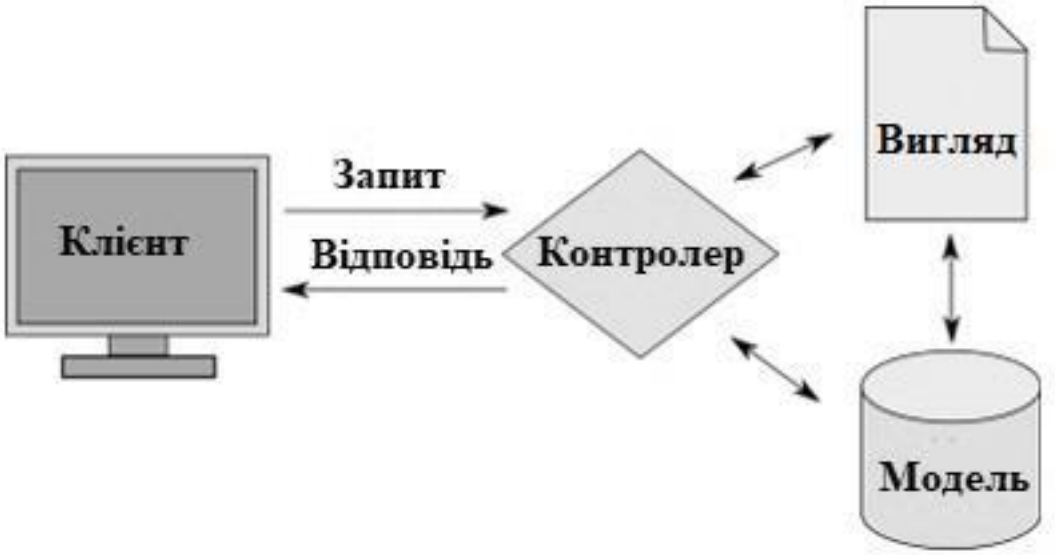


Рисунок 3.2 — Функціонування стандартної реалізації архітектурного принципу MVC [56]

Після надсилання запиту на сервер, його починає обробляти контролер, потім передає зміни до моделі, яка реагує на оновлення та видає все необхідне для відображення сайту. Вигляд виконує лише роль відображення зовнішнього вигляду сторінки — стандартний HTML-шаблон.

.NET Core — це модульна реалізація бібліотеки та середовища виконання, до складу якої входить піднабір .NET Framework. .NET Core працює на Windows, Mac та Linux.

Версія складається з колекції бібліотек CoreFX та невеликого оптимізованого середовища виконання CoreCLR. .NET Core — проект із відкритим вихідним кодом, тому можна спостерігати за його розвитком та підтримувати його на GitHub. Елементи архітектури .NET Core представлено на рис. 3.3.

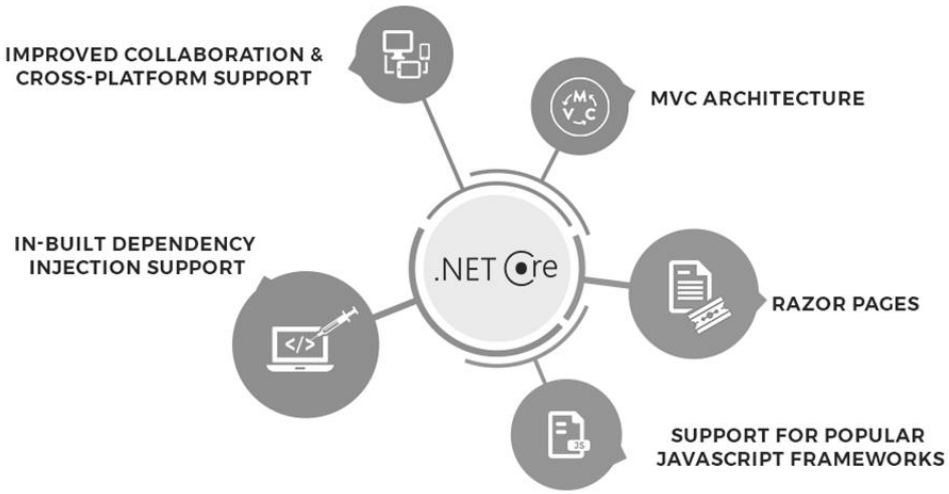


Рисунок 3.3 — Архітектура .NET Core [57]

Середовище виконання CoreCLR (Microsoft.CoreCLR) та бібліотеки CoreFX розповсюджуються через NuGet. Оскільки версія .NET Core є компонентизованим набором бібліотек, ви можете обмежити кількість API для своєї програми та використовувати тільки потрібні елементи. Крім того, можна виконувати програми на базі .NET Core у набагато обмеженіших середовищах (наприклад, в ASP.NET Core на Nano Server).

Razor Pages — новий елемент ASP.NET Core, який робить програмні сценарії, засновані на веб-сторінках більш продуктивними. З технічної точки зору Razor Pages — модель кодування, заснована на веб-сторінках, яка спрощує створення веб-інтерфейсу. За допомогою Razor Pages кожна веб-сторінка стає автономною з компонентом View, код також чітко налагоджений.

Перевага такого рішення очевидна — це відмова від непотрібного прошарку — моделі сторінки (модель даних у вигляді, наприклад, Entity).

Бекенд сторінки є контролером і моделлю — класика ООП — інкапсуляція даних та методів роботи з ними в одному об'єкті. Зрештою модель сторінки — це просто клас, немає жодних причин, чому цим класом не може бути контролер [58].

Microsoft Azure є однією з хмарних платформ, яка поєднує як рішення обчислювальної інфраструктури як сервіс (IaaS) (сервери, сховища даних,

мережі, операційні системи), так і набір інструментів і сервісів, що полегшують розробку і розгортання хмарних додатків (PaaS). Все це дає розробникам можливість швидко створювати, розгортати та керувати масштабними програмами для задоволення потреб конкретних організацій. Сервіси Microsoft Azure дозволяють замовникам уникнути складнощів та додаткових витрат, пов'язаних із заміною існуючого обладнання, управлінням базовою інфраструктурою та, відповідно, забезпечують більш ефективне управління IT-бюджетами: замовники платять лише за ресурси, які їм необхідні [59, 60].

Структура та компоненти Microsoft Azure відображені на рис. 3.4.

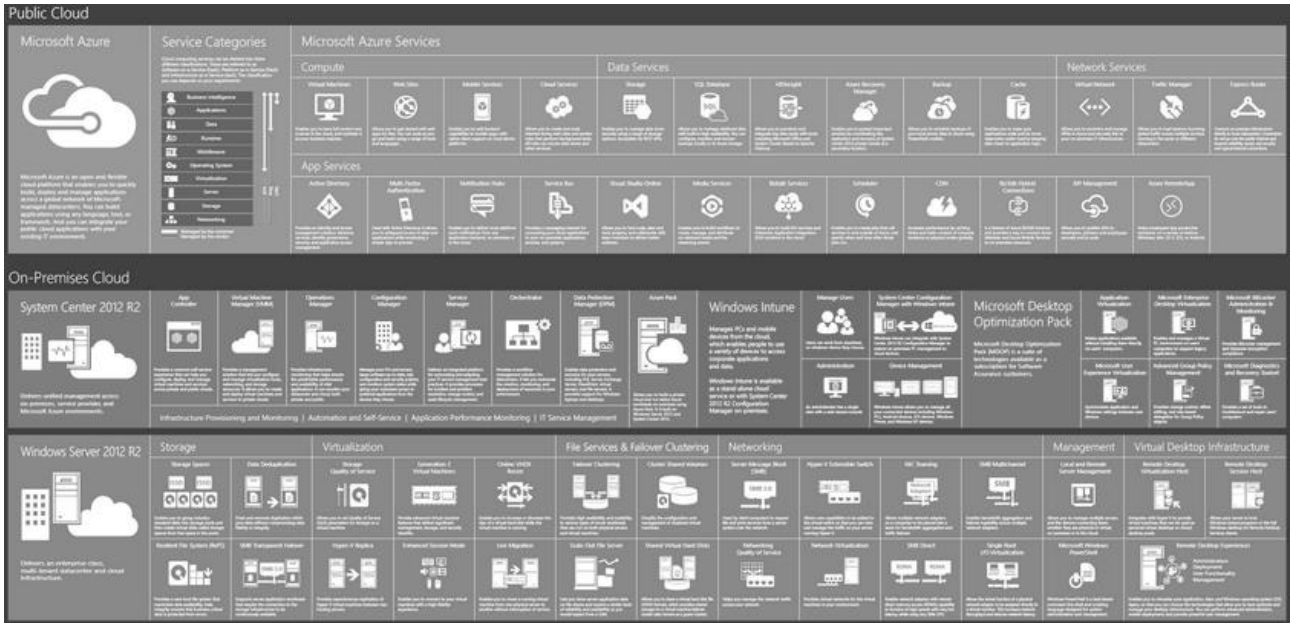


Рисунок 3.4 — Структура Microsoft Azure [61, 62]

Стандартні послуги Microsoft Azure Services. Категорії сервісів:

— надання обчислювальних потужностей у найм (Compute Services),

— сервіси зберігання (Data Services) та служби додатків (App Services).

Сервіси зі створення приватних інфраструктур віртуалізації на базі продуктів сімейства System Center 2012 R2, включаючи такі компоненти як: App Controller, Operations, Service і Configuration Manager, Orchestrator, Data Protection Manager та Azure Pack.

Сервіси платформи Windows Server 2012 R2, включаючи такі технології та складові: Storage Spaces, Data Deduplication, Resilient File System, SMB

Transparent Failover, Storage Quality of Service, Online VHDX Resize, Enhanced Session Mode, Live Migration, Failover Clustering, Cluster Shared Volumes, Scale-Out File Server, Shared Virtual Hard Disks, Hyper-V Extensible Switch, Remote Desktop Services, SMB Direct, SMB Multi-channel та NICTeaming (рис. 3.5) [63].

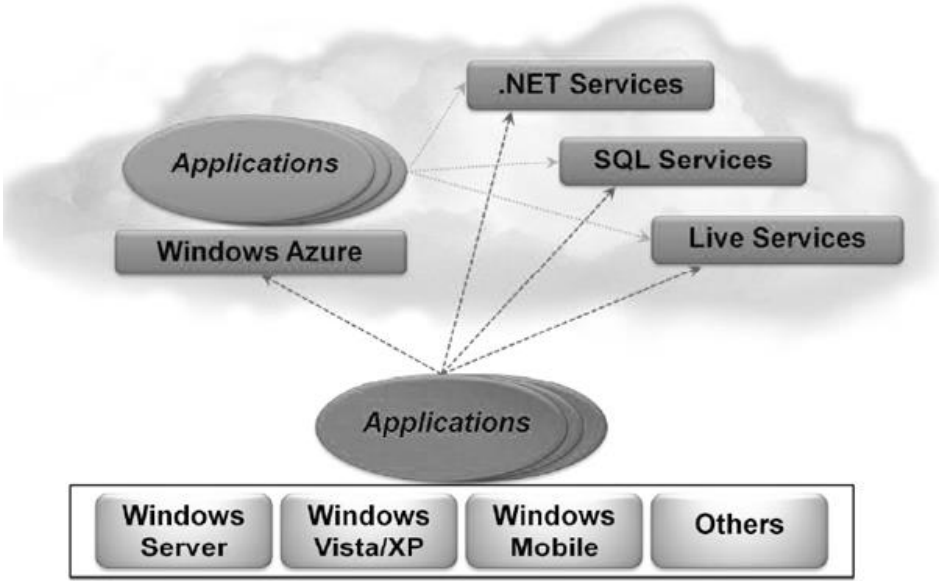


Рисунок 3.5 — Сервіси платформи Microsoft Azure [64]

PaaS-архітектура від Microsoft Azure складається з кількох компонентів:

— Windows Azure — програма, призначена для розгортання та управління сервісами, обробки та масштабованого зберігання даних, організації мережі, забезпечує можливість еластичних обчислень;

— інструмент роботи з базами даних та звітністю Microsoft SQL Services;

— Microsoft .NET Services — сервісну реалізацію компонентів .NET Framework;

— Microsoft Live Services — набір послуг для роботи з документами. Забезпечує зберігання, розповсюдження та синхронізацію документів, фотографій та інших файлів між комп'ютерами, телефонами, програмами та веб-сайтами;

— Microsoft SharePoint Services — набір сервісів для роботи над проектами;

— Microsoft Dynamics CRM Services — сервіси для управління бізнес-інформацією та взаємовідносинами з клієнтами.

Оскільки хмарні програми, як правило, вимагають створення систем, що об'єднують кілька віртуальних машин і різних сервісів, в архітектурі від Microsoft є компонент Azure AppFabric, який відповідає за планування, виділення ресурсів, управління пристроями та відмовостійкість, а також додаткові інструменти для моніторингу та контролю.

Основне призначення Microsoft Azure — можливість швидкої розробки веб-сервісів та веб-додатків. Для цього Windows Azure братиме на себе рішення таких завдань як зберігання даних, інформаційний пошук, трудомісткі обчислення, тощо, у той час як веб-розробникам залишиться лише виконувати запити до Windows Azure.

3.2 Розроблення програмного засобу безпечного конфігурування кешування динамічного вмісту веб-сторінок

Однією із важливих програмних частин розроблюваного програмного засобу безпечного конфігурування кешування динамічного вмісту веб-сторінок є Cosmos Db — нереляційна база даних (рис. 3.6). Її особливістю є відсутність зв’язків, кожен об'єкт зберігається як окремий json файл із певними властивостями (рис. 3.7).

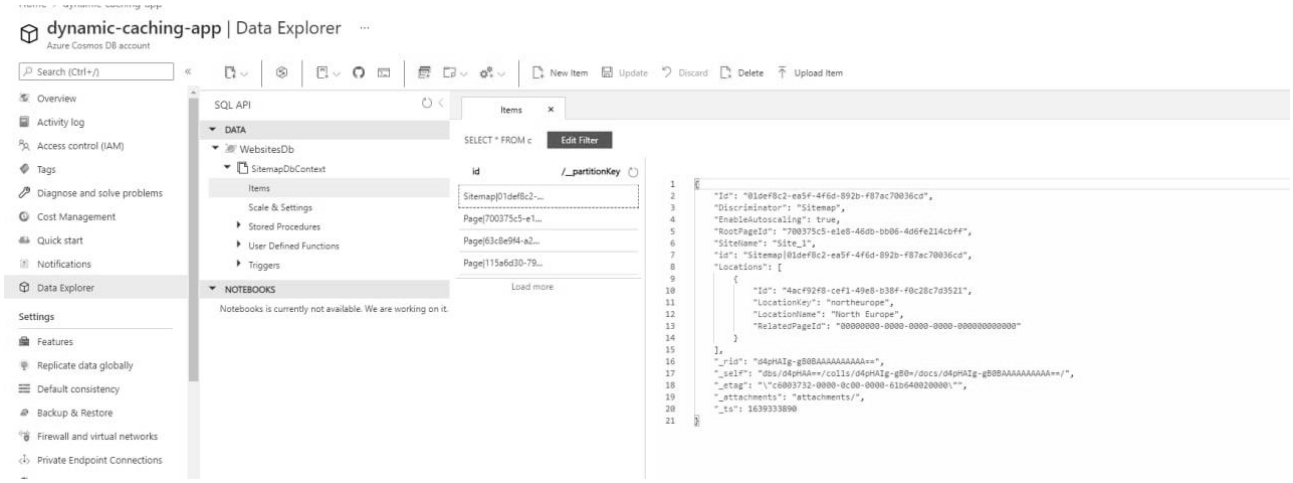


Рисунок 3.6 — Інтерфейс взаємодії з базою даних Cosmos DB

```
{
    "Id": "01def8c2-ea5f-4f6d-892b-f87ac70036cd",
    "Discriminator": "Sitemap",
    "EnableAutoscaling": true,
    "RootPageId": "700375c5-e1e8-46db-bb06-4d6fe214cbff",
    "SiteName": "Site_1",
    "id": "Sitemap|01def8c2-ea5f-4f6d-892b-f87ac70036cd",
    "Locations": [
        {
            "Id": "4acf92f8-cef1-49e8-b38f-f0c28c7d3521",
            "LocationKey": "northeurope",
            "LocationName": "North Europe",
            "RelatedPageId": "00000000-0000-0000-0000-000000000000"
        }
    ],
    "_rid": "d4pHAIg-gB0BAAAAAAAAAA==",
    "_self": "dbs/d4pHAA==/colls/d4pHAIg-gB0=/docs/d4pHAIg-gB0BAAAAAAAAAA==/",
    "_etag": "\"c6003732-0000-0c00-0000-61b640020000\"",
    "_attachments": "attachments/",
    "_ts": 1639333890
```


Рисунок 3.7 — Приклад json файлу властивостей певного екземпляру сутності в Cosmos Db

За необхідності перегляду властивостей сторінки, можна скористатись меню вибору відповідної сторінки зліва, здійснювати перегляд та редагування (рис 3.8). Варто зазначити, що у реляційній базі даних зв’язки набувають іншого вигляду. Їх структуру можна побачити на рис. 3.9.

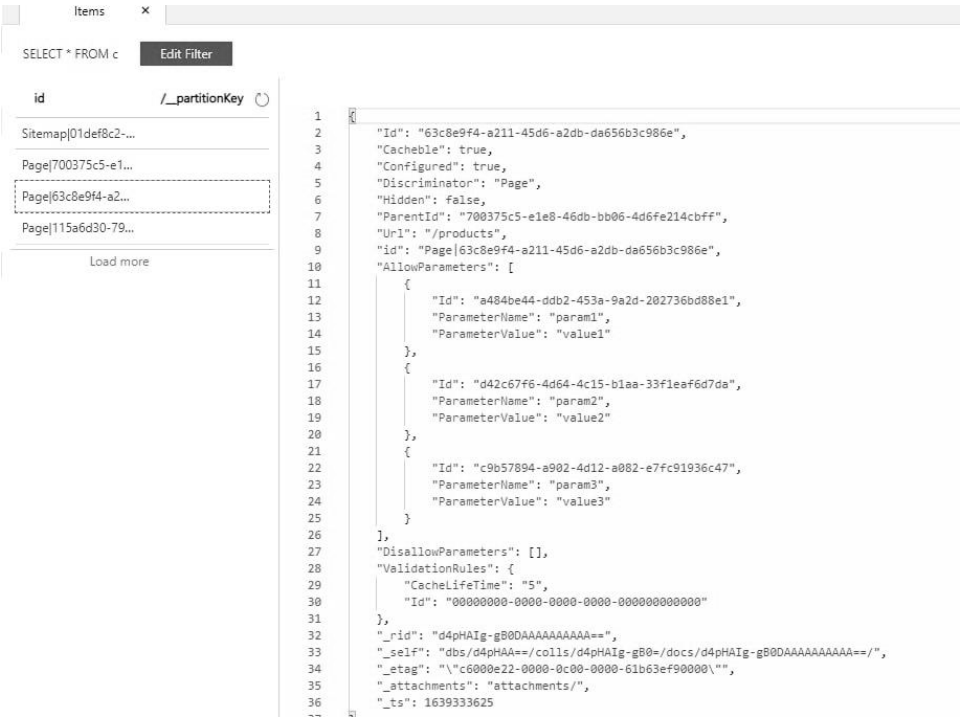


Рисунок 3.8 — Перегляд властивостей сторінки у нереляційній базі даних

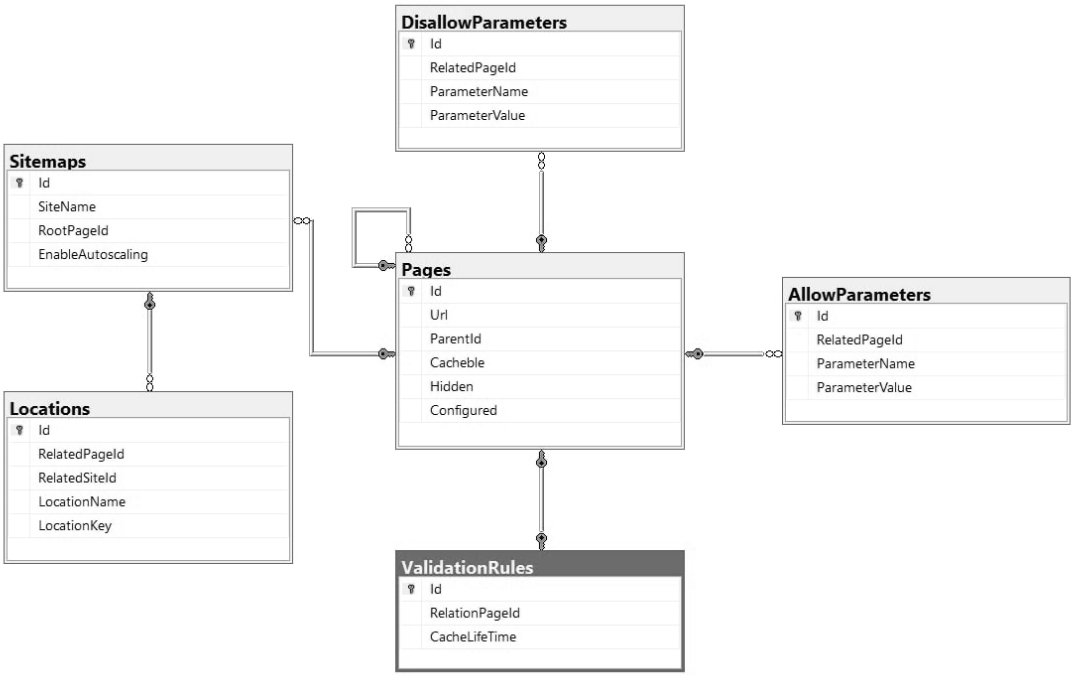


Рисунок 3.9 — Зв’язки у реляційній базі даних

Вибір бази даних NoSql зумовлений наявністю у розроблюваному програмному засобі сайту, який містить дерево сторінок. Тобто, такий веб-ресурс посилається на сторінку, що є коренем сайту. Таким чином, у кожної сторінки є низка дочірніх сторінок, а в кожної дочірньої – низка своїх дочірніх тошо.

У випадку необхідності отримати з бази даних дерево веб-ресурсу, необхідно здійснити певні операції Join для кожного ряду підсторінок, проте, з використанням даних NoSql, така робота значно спрощується (здійснюється запит коду кореневої сторінки та підтягується далі все дерево сторінок).

Відповідно, така робота із ресурсом відбувається значно простіше і дозволяє зекономити значну частину часу.

Далі, на рисунку 3.10 зображено реалізовану функцію для додавання аналогічних конфігурацій однаковим сторінкам, за допомогою чекбоксу Apply to similar pages. Така можливість є корисною, наприклад, у випадку необхідності задання однакових параметрів для усіх публікацій блогу, тощо.

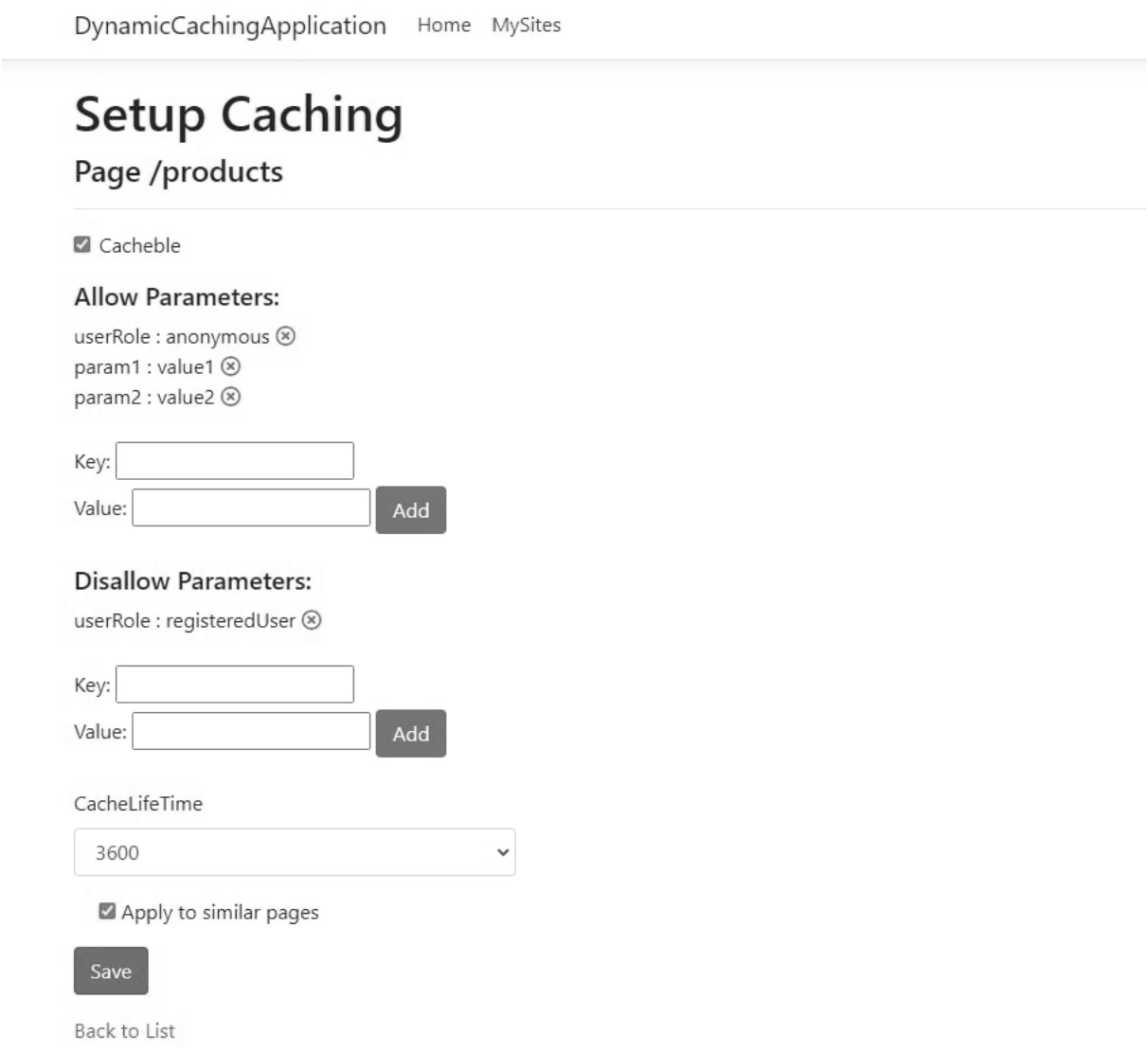


Рисунок 3.10 — Чекбокс Apply to similar pages

Далі наведено фрагмент кодової послідовності для реалізації згадуваної вище функції.

```
if (applyToSimilar)
        {
          foreach(var childpage in page.ParentPage.ChildPages.ToList())
          {
            if (childpage == page) continue;

            childpage.Cacheble = page.Cacheble;
            childpage.ValidationRules = new ValidationRules {
              CacheLifeTime = page.ValidationRules.CacheLifeTime
            };
            childpage.Configured = page.Configured;
            childpage.Parameters.Clear();
```

```
                    foreach (var param in page.Parameters)
                    {
                        childpage.Parameters.Add(new Parameter
                        {
                            ParameterName = param.ParameterName,
                            ParameterValue = param.ParameterValue
                        });
                    }
                }
            }
```

Програмна реалізація завантаження налаштувань кешування сторінок до бази даних наведена нижче:

```
        public async Task<ItemResponse<Sitemap>> UploadSiteSettingsAsync(Sitemap sitemap, string _endpointUri, string _primaryKey)
        {
            using (CosmosClient client = new CosmosClient(_endpointUri, _primaryKey))
            {
                DatabaseResponse databaseResponse = await client.CreateDatabaseIfNotExistsAsync("WebsitesDb");
                Database targetDatabase = databaseResponse.Database;

                IndexingPolicy indexingPolicy = new IndexingPolicy
                {
                    IndexingMode = IndexingMode.Consistent,
                    Automatic = true,
                    IncludedPaths =
                    {
                        new IncludedPath
                        {
                            Path = "/*"
                        }
                    }
                };
                var containerProperties = new ContainerProperties("sitemaps", "/id")
                {
                    IndexingPolicy = indexingPolicy
                };
                var containerResponse = await targetDatabase.CreateContainerIfNotExistsAsync(containerProperties, 10000);
                var customContainer = containerResponse.Container;

                ItemResponse<Sitemap> responce = await customContainer
                    .CreateItemAsync<Sitemap>(sitemap, new PartitionKey(sitemap.Id.ToString()));

                return responce;
            }
```

}

Процес створення бази даних налаштувань кешу реалізується наступним чином. Першим кроком створюється база даних, далі відбувається налаштування індексації та створюється контейнеру із необхідними властивостями. Після налаштувань кешування сторінок сайту, створюється інфраструктура хмарних сервісів Azure, індивідуальна для кожного клієнта (рис. 3.11).

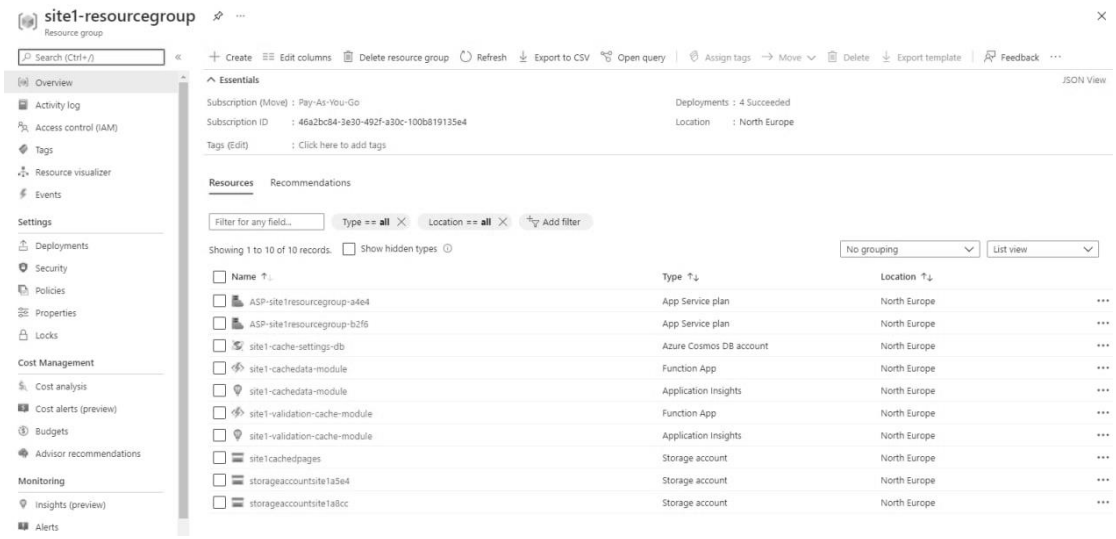


Рисунок 3.11 — Набір створених сервісів Microsoft Azure

Для програмної реалізації модулю кешування даних та перевірки кешу створено дві відповідних Azure функції. Здійснені налаштування кешу зберігаються в Azure Cosmos Db, а закешовані сторінки в Azure storage. Графічне відображення ресурсів, створених в хмарі, з яких складається архітектури додатку представлено на рис. 3.12.

Для того, щоб спростити та пришвидшити процес налаштування кешування кожної окремої сторінки, передбачена можливість застосування однакових конфігурацій для певного набору сторінок.

Зазвичай сторінки, які повинні мати однакові конфігурації, мають схожий контент, і розміщуються на одному рівні, та мають спільну батьківську сторінку. Тому було розроблено спеціальний чекбокс, розміщений на сторінці налаштувань кешування, який дозволяє примінити задані налаштування до

схожих сторінок (рис. 3.13).

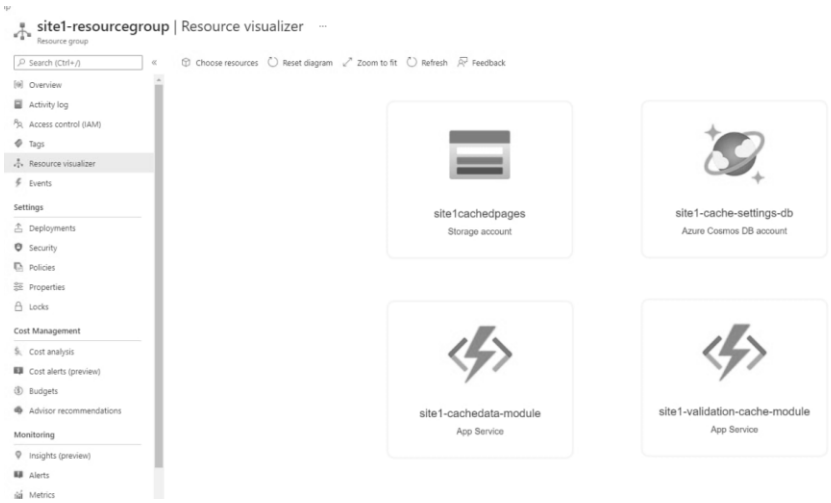


Рисунок 3.12 — Візуальне представлення Azure Cosmos Db

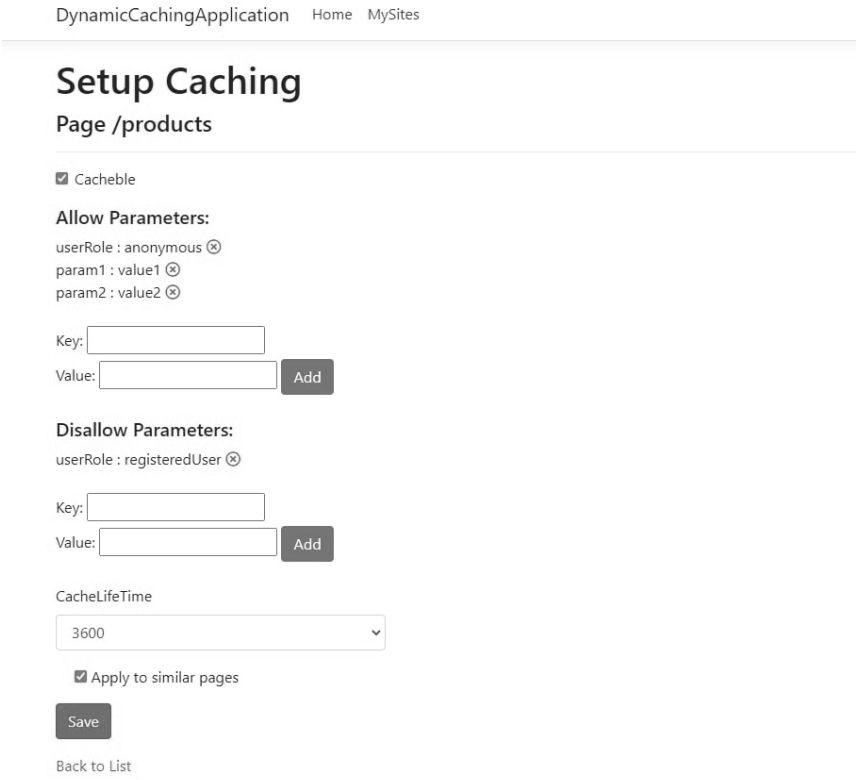


Рисунок 3.13 — Чекбокс Apply to similar pages

Далі наведено фрагмент кодової послідовності для реалізації згадуваної вище функції.

```
if (applyToSimilar)
        {
```

```
            foreach(var childpage in page.ParentPage.ChildPages.ToList())
            {
               if (childpage == page) continue;
               childpage.Cacheble = page.Cacheble;
               childpage.ValidationRules = new ValidationRules {
                  CacheLifeTime = page.ValidationRules.CacheLifeTime
               };
               childpage.Configured = page.Configured;
               childpage.Parameters.Clear();
               foreach (var param in page.Parameters)
               {
                  childpage.Parameters.Add(new Parameter
                  {
                     ParameterName = param.ParameterName,
                     ParameterValue = param.ParameterValue
                  });
               }
            }
```

Логіка модуля перевірки кешу наведена нижче. Даними функціями здійснюється аналіз даних сторінки та можливості їх подальшої обробки. Залежно від отриманих значень, отримується відповідь можливості чи недоступності кешування тієї чи іншої сторінки. Іншими словами, являє собою модуль валідації кешу, що представлений на рисунку 2.4.

```
        [FunctionName("ValidationCacheFunction")]
            public static async Task<IActionResult> Run(
               [HttpTrigger(AuthorizationLevel.Anonymous, "get", "post", Route = null)]
HttpRequest req,
               ILogger log)
            {
               log.LogInformation("C# HTTP trigger function processed a request.");
               string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
               dynamic body = JsonConvert.DeserializeObject(requestBody);
               string reqUrl = body?.requestedUrl;
               if (string.IsNullOrEmpty(reqUrl))
               {
                  return new BadRequestObjectResult(new Result { Success = false, Message =
"Need to specify requestedUrl in request body" });
               }
               var pageSettings = GetPageSettings(endpointUrl, primaryKey, dbName, reqUrl);
               if (pageSettings == null)
               {
                  return new OkObjectResult(new Result { Success = false, Message = "Can not find
settings for this page" });
               }
```

```
            if (!pageSettings.Cacheble)
            {
                return new OkObjectResult(new Result { Success = true, Message = "Page can not be cached", Cacheble = false });
            }
            if (pageSettings.DisallowParameters.Any() && RequestContainsDisallowedParameters(req, body, pageSettings.DisallowParameters))
            {
                return new OkObjectResult(new Result { Success = true, Message = "Request contains disallowed parameters", Cacheble = false });
            }
            if (pageSettings.AllowParameters.Any() && !RequestContainsAllowedParameters(req, body, pageSettings.AllowParameters))
            {
                return new OkObjectResult(new Result { Success = false, Message = "Request does not contain needed allow parameters", Cacheble = false });
            }
            return new OkObjectResult(new Result {
                Success = true,
                Message = "Page can be cached",
                Cacheble = true,
                ValidationRules = pageSettings.ValidationRules
            });
        }
```

Нижче наведено допоміжний метод Azure функції, який звертається до відповідної бази даних Azure CosmosDb, та отримує дані про налаштування кешування необхідної сторінки. Для встановлення з'єднання з базою даних, необхідно мати адресу точки доступу до бази даних, ключ доступу до бази даних, назву бази даних та контейнера в якому зберігається необхідна інформація.

```
    private static PageUploadModel GetPageSettings(string _endpointUri, string _primaryKey, string dbName, string url)
        {
            using (CosmosClient client = new CosmosClient(_endpointUri, _primaryKey))
            {
                var database = client.GetDatabase(dbName);
                var container = database.GetContainer($"{dbName}Context");
                var pageQueriable = container.GetItemLinqQueryable<PageUploadModel>(true);
                PageUploadModel page = pageQueriable
                    .Where(p => p.Url == url)
                    .AsEnumerable()
                    .FirstOrDefault();
                return page;
```

```
}
```

Зокрема адреса точки доступу до бази даних та ключ доступу до бази даних являються вразливими з точки зору інформаційної безпеки, оскільки маючи ці дані, зловмисники зможуть отримати дані з бази даних, видалити або змінити дані та структуру даних, що там зберігаються. Тому ці дані ні в якому разі не можна зберігати в відкритому доступі, коді програми, чи конфігураційних файлів.

Для безпечного отримання таких даних рекомендується використовувати Azure Key Vault — спеціалізованому централізованому хмарному сервісі для зберігання секретів програми. Далі наведено фрагмент програмної реалізації вище згаданої Azure функції, що відповідає за пошук параметрів, необхідних для прийняття рішення про можливість кешування сторінки, та значень цих параметрів в тілі, заголовках та атрибутах запиту.

```
        private static bool RequestContainsAllowedParameters(HttpRequest req, dynamic requestBody, List<AllowParameter> parameters)
            {
                foreach (var param in parameters)
                {
                    if (req.Headers.ContainsKey(param.ParameterName) && param.ParameterValue == req.Headers[param.ParameterName])
                        return true;
                    if (req.Query.ContainsKey(param.ParameterName) && param.ParameterValue == req.Headers[param.ParameterName])
                        return true;
                    if (requestBody[param.ParameterName] != null && param.ParameterValue == requestBody[param.ParameterName].ToString())
                        return true;
                }
                return false;
            }
```

Таким чином, на основі обраних засобів програмування та можливостей, що вони надають, було здійснено практичну реалізацію програмного засобу безпечного конфігурування кешування динамічного вмісту веб-сторінок. У наступному підрозділі більш детальніше розглянемо застосування розробленого додатку на практиці.

3.3 Перевірка якості роботи програмного засобу

Розглянемо практичну роботу розроблюваного додатку на прикладах.

На сторінці управління сайтами, відображається деревовидна структура сторінок сайту (рис. 3.14). На кожну з таких сторінок можна додати налаштування, позначками у вигляді «хрестика» та «галочки» відмічається можливість кешування. Відповідно, якщо іконки немає, то налаштування відсутні і їх можна збити за замовчуванням.

Рисунок 3.14 — Візуальне відображення деревовидної структури сторінок сайту

Далі наведено зразки візуального представлення сторінок при реалізації можливості додавання нового сайту (рис. 3.15).

Після натискання на кнопку Setup Caching відповідної сторінки, користувачу відображається сторінки налаштування кешування (рис. 3.16).

Якщо позначити сторінку, як ту, що може бути закешована (чекбокс Cachable), то з’явиться можливість налаштувати параметри кешування сторінки та поля для налаштувань валідації кешу (рис. 3.17).

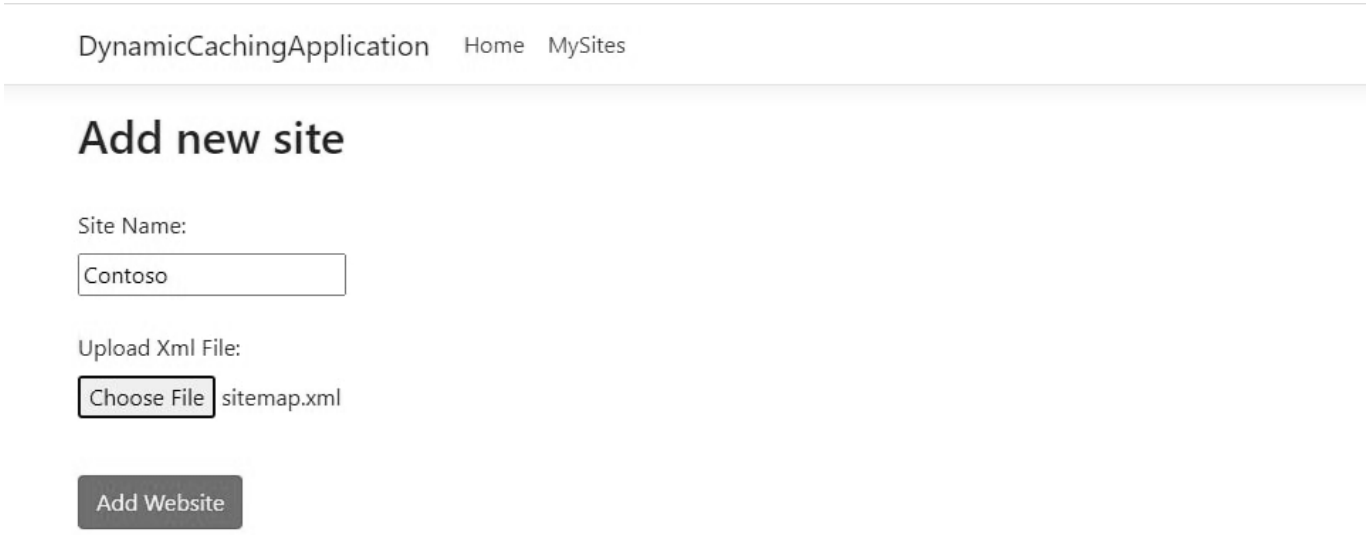


Рисунок 3.15 — Додавання нового сайту за допомогою sitemap.xml

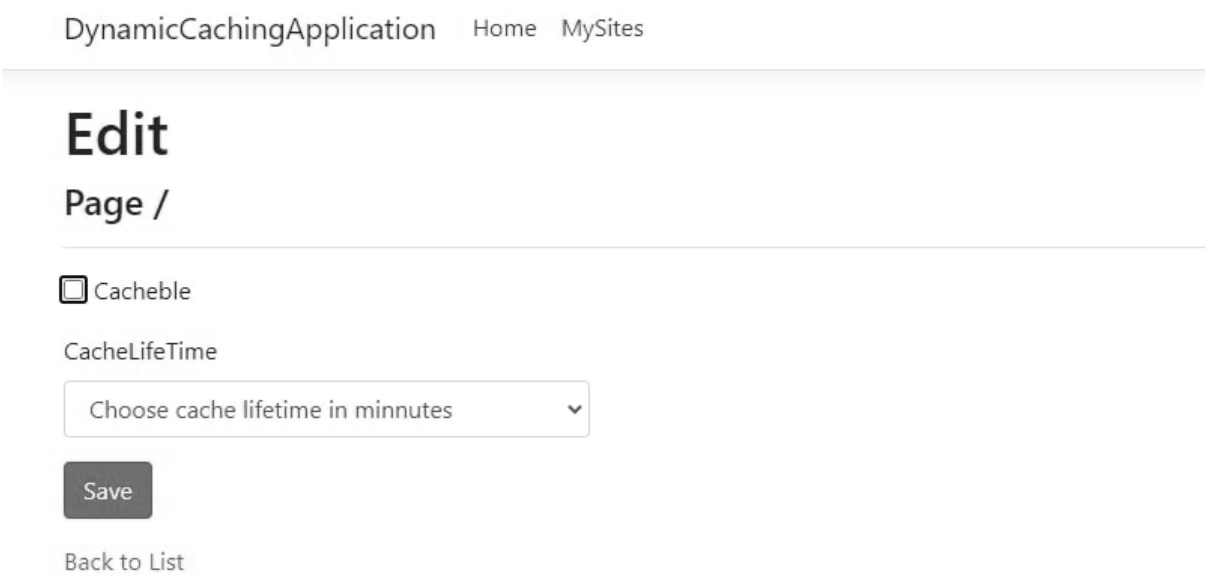


Рисунок 3.16 — Налаштування кешування для певної сторінки

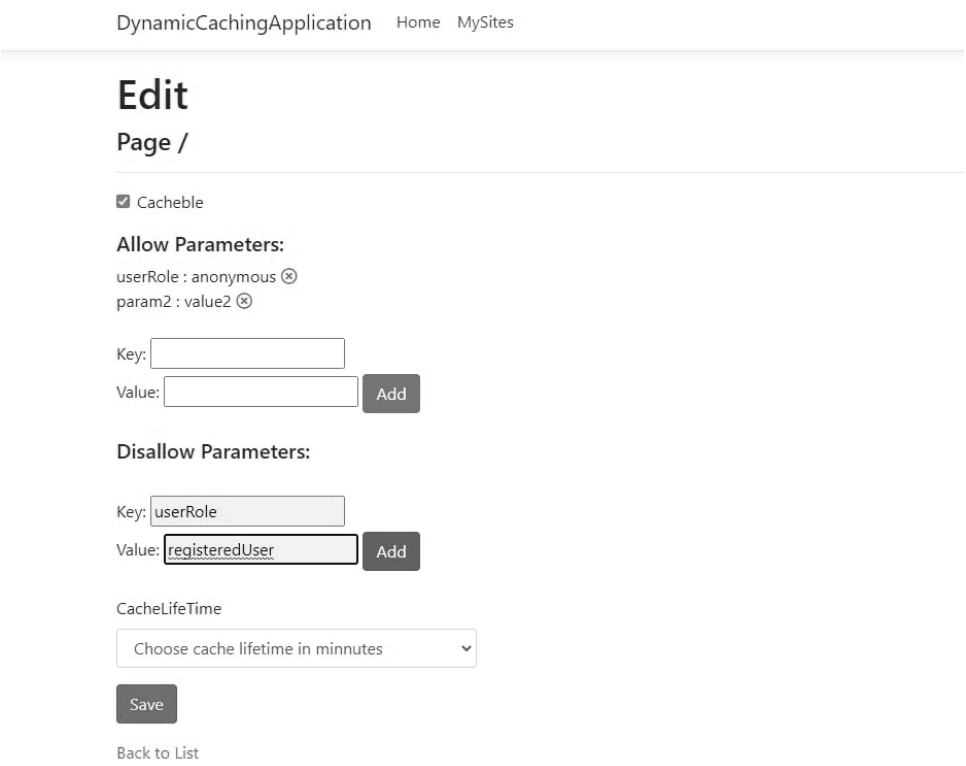


Рисунок 3.17 — Вигляд сторінки налаштувань кешування

Загальний вигляд сторінки налаштувань сервісів Microsoft Azure представлений на рисунках нижче. Зокрема існує можливість обрати список локацій, в яких створити екземпляри необхідних ресурсів та сервісів (рис. 3.18), а також налаштувати автоматичне розширення (рис. 3.19).

Рисунок 3.18 — Процес вибору локацій для програмної інфраструктури додатку

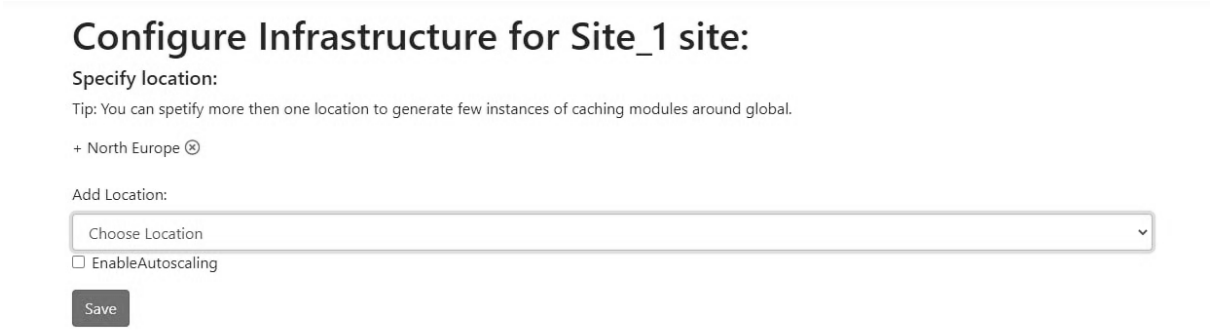


Рисунок 3.19 — Форма налаштувань інфраструктури додатку.

В такому функціоналі програмного засобу для кожного сайту налаштовується інфраструктура, існує можливість обрати декілька локацій або одну, за необхідності вмикати автоматичне розширення — функціонал, який Azure пропонує прямо із коробки для широкого спектру своїх сервісі. Дана функція полягає у тому, що в моменти пікової активності, коли пам’ять або CPU певного ресурсу перевищує нормоване значення, Azure автоматично створює дублікат або декілька дублікатів даного ресурсу та розподіляє навантаження між ними, для максимальної продуктивності. Після того як навантаження на ресурси спадає, Azure автоматично видаляє додаткові екземпляри ресурсів, тим самим зменшуючи витрати клієнта.

При натисненні кнопки «Create infrustructure» у хмарі створюються всі необхідні сервіси, в тій кількості і локаціях скільки їх вказано, а клієнт отримує дані для звернення до модулю кешування данних (рис. 3.20).

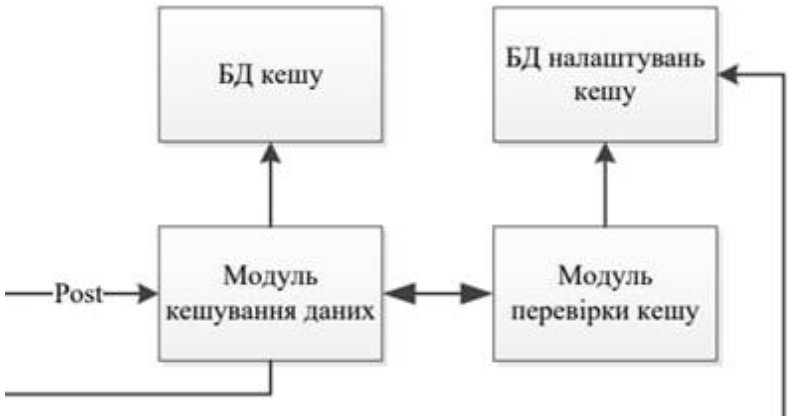


Рисунок 3.20 — Модулі бази даних сервісів для роботи з даними

Модуль кешування даних посилає запит, до модулю перевірки кешу і вказує в цьому запиті інформацію про сторінку, яку запитує клієнт веб-сайту, тіло, квері-параметри, та заголовки оригінального запиту, та необхідні додаткові параметри.

В свою чергу, модуль перевірки дістає налаштування кешування запрошуваної сторінки з бази даних, порівнює налаштовані для неї параметри з тими, що у запиті, і повертає у відповідь інформацію щодо того чи можливо кешувати таку сторінку, чи ні і як має бути провалідована закешована сторінка для даного запиту.

Нище представлені варіанти позитивної та негативної відповіді від модулю перевірки кешу (рис. 3.21, 3.22).

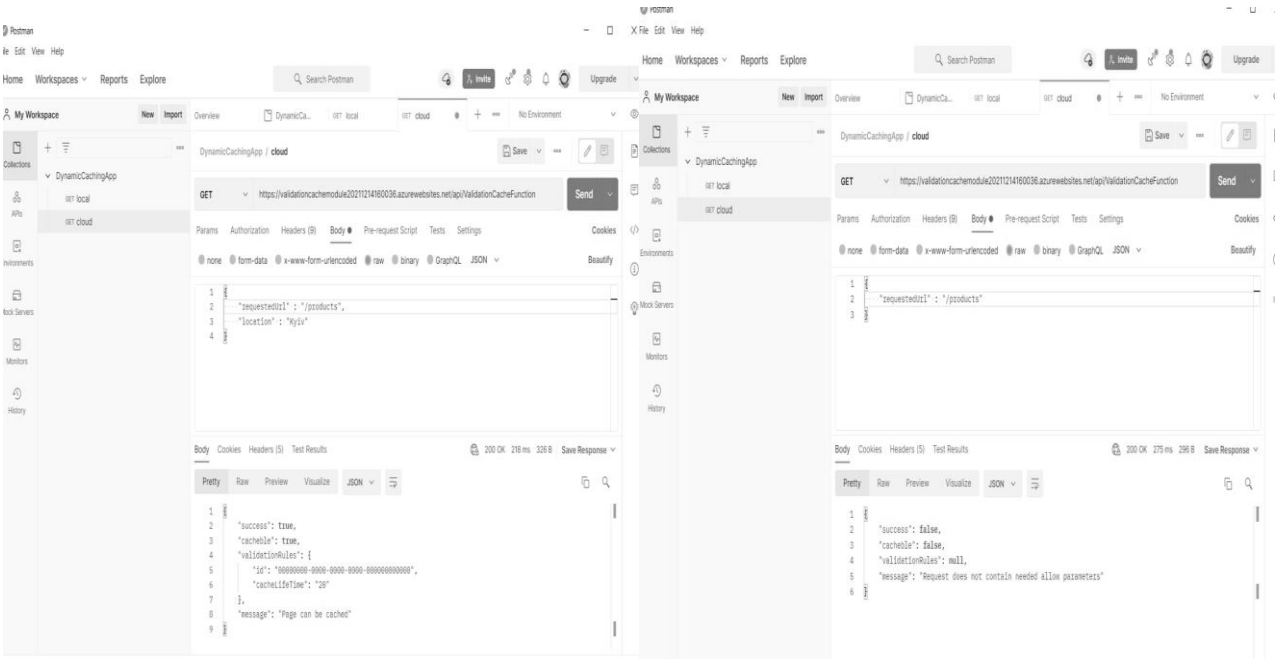


Рисунок 3.21 — Вигляд відповіді модулю перевірки кешу на запит, за яким можна кешувати сторінку

Рисунок 3.22 — Вигляд відповіді модулю перевірки кешу на запит, за яким не можна кешувати сторінку

Таким чином, на основі використання сервісів Microsoft Azure, налаштуванні відповідних функцій та можливостей сторінок було здійснено та представлено практичну реалізацію програмного засобу безпечного конфігурування кешування динамічного вмісту веб-сторінок.

3.4 Підсумки до розділу

Отже, в дамому розділі була здійснена розробка програмного засобу безпечного конфігурування кешування веб-сторінок. Описано вибір інструментарію реалізації програмного засобу, його практичну розроку та наведено результати тестування.

Враховуючи поставлені мету та задачі роботи для практичної реалізації програмного засобу б було обрано сучасні та практичні засоби реалізації, а саме мову програмування C#, платформу Asp.net core, візуалізатор сторінок Razor Pages, сервіси Microsoft Azure, бібліотеки System.Threading, System.Threading.Tasks для асинхронних операцій. Під час тестування додатку було на практиці продемонстровано доцільність та практичність розроблюваного програмного засобу.

4 РОЗРАХУНОК ЕКОНОМІЧНОЇ ДОЦІЛЬНОСТІ СТВОРЕННЯ ПРОГРАМНОГО ЗАСОБУ

4.1 Оцінювання комерційного потенціалу розробки ПЗ безпечного конфігурування кешування динамічного вмісту веб-сторінок

Метою проведення технологічного аудиту є оцінювання комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності [65].

Результатом магістерської кваліфікаційної роботи є розробка програмного засобу, який відрізняється збільшеною відповідністю до критеріїв користувача. Сферою застосування результатів роботи є комп'ютерні системи маркетингу, реклами, торгівлі, тощо.

Для проведення технологічного аудиту залучено трьох незалежних експертів. У нашому випадку такими експертами є викладачі кафедри ОТ: Семеренко В. П. (к.т.н., доц. кафедри ОТ), Крупельницький Л.В. (к.т.н., доц. кафедри ОТ), Черняк О.І. (к.т.н., доц. кафедри ОТ). Оцінювання комерційного потенціалу було здійснене за критеріями, що наведені в таблиці 4.1

Таблиця 4.1 — Критерії оцінювання комерційного потенціалу розробки

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
Кри-терій	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів

Продовження таблиці 4.1

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
Кри-тер.	0	1	2	3	4
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатацій ні витрати значно вищі, ніж в аналогів	Експлуатацій ні витрати дещо вищі, ніж в аналогів	Експлуатацій ні витрати на рівні експл. витрат аналогів	Експлуатацій ні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навч. наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовую ться у військово-промисловом у комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві

Закінчення таблиці 4.1

11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Процедура отримання документів для в-тва та реалізації вимагає незначних коштів та часу	Процедура отримання документів для в-тва та реалізації вимагає незначних коштів та часу	Процедура отримання документів для в-тва та реалізації вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання комерційного потенціалу експертами розробки зведено в таблицю 4.2.

Таблиця 4.2 — Результати оцінювання комерційного потенціалу розробки

Критерії	Прізвище, ініціали, посада експерта		
	Семеренко В. П,	Крупельницький Л.В.	Черняк О.І.
1	3	4	4
Ринкові переваги (недоліки):			
2	3	4	3
3	4	4	3
4	3	3	4
5	4	4	4
Ринкові перспективи			
6	4	4	3
7	4	3	4
Практична здійсненність			
8	4	3	4
9	3	4	3
10	4	4	3
11	4	3	3
12	3	3	4
Сума балів	$СБ_1 = 43$	$СБ_1 = 43$	$СБ_1 = 42$
Середньоарифметична сума балів $\overline{СБ}$	$\overline{СБ} = 42{,}7$		

За даними таблиці 4.2 можна зробити висновок, щодо рівня комерційного потенціалу розробки. Зважимо на результат й порівняємо його з рівнями комерційного потенціалу розробки, що представлено в таблиці 4.3.

Таблиця 4.3 — Рівні комерційного потенціалу розробки

Середньоарифметична сума балів $\overline{СБ}$, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0 – 10	Низький
11 – 20	Нижче середнього
21 – 30	Середній
31 – 40	Вище середнього
41 – 48	Високий

Рівень комерційного потенціалу розробки, становить 42,7 балів, що відповідає рівню «високий».

Проаналізуємо суть технічної проблеми та розглянемо аналоги. Зі збільшенням кількості користувачів глобальної мережі інтернет, проблема доставки об'ємного контенту в інтернеті стає все більш актуальною. Виникає необхідність пошуку методів оптимізації процесів обробки запитів та доставки вмісту веб сторінок. Особливо, це актуально для контенту, який потрібно одночасно роздати велику кількість користувачів. Виникають нові і нові технології, такі як балансування трафіку, розподілені мережі доставки контенту, кешування та ін. Але проблема досі не вирішена. І одним з підходів для оптимізації процесу обробки та доставки контенту є пошук способу кешувати динамічно змінювані дані. Актуальність дослідження пов’язана з необхідністю вирішення задачі надання динамічного вмісту веб-сторінок максимально відповідного до критеріїв користувача. Зокрема, щоб забезпечити максимальний комфорт користувача при користуванні веб-сайтом, надати йому рекомендації(товар, послуга сайту, тощо) відповідно до його критеріїв.

Наукова новизна розробки полягає в наступному: вперше запропоновано та реалізовано, в якості алгоритма кешування динамічного вмісту веб-сторінок, метод з безпечним конфігуруванням, що дозволяє підвищити відповідність даних до критеріїв користувача.

Практична цінність роботи полягає в створенні алгоритму та програмного продукту, який відрізняється збільшенною відповідністю до критеріїв користувача. Сферою застосування результатів роботи є комп'ютерні системи маркетингу, реклами, торгівлі, тощо.

Враховуючи такі переваги розробленого методу, можемо порівняти його з основними аналогами.

У таблиці 4.4 наведені основні технічні показники аналога і нового програмного продукту.

Таблиця 4.4 — Порівняння основних технічних показників

Показники, %	Аналог	Нова розробка	Відношення параметрів нової розробки до параметрів аналога
Функціональність	80	100	1,25
Надійність	75	100	1,33
Сумісність	95	100	1,05
Супровід	70	100	1,43
Економія ресурсів і часу	90	100	1,11
Простота використання	80	100	1,25

На сьогоднішній день одним з найпопулярніших підходів до управління кешуванням вмісту сторінок є заголовки запиту Http протоколу. Він містить поле Cache-Control що використовується для завдання інструкцій по механізму кешування як в запитах, так і у відповідях. Застосовується для завдання політик кешування, серед яких: повна відсутність кешування (параметри no-cache, no-store, must-revalidate); кешувати, але перевіряти актуальність; (араметри: no-cache (приватний (private) і загальний (public) кеш, термін дії (Expiration)).

Крім того, оскільки обсяг сховища кешу кінцевий, а ресурси на сервері можуть змінюватись то кеш потрібно чистити та оновлювати. Для цього доступні заголовки ETag та заголовок HTTP відповіді Vary.

Даний метод управління кешем є досить складним, для його грамотного використання потрібно мати неаби які технічні знання, і він не є достатньо

гнучким для використання його в управлінні кешуванням динамічного контенту. Для цього необхідний інший підхід, який буде більш інтуїтивно зрозумілий та гнучкий.

На підставі вищевикладеного можна стверджувати, що нове технічне рішення, що пропонується для розробки, буде мати кращі показники, ніж у аналога та більшою мірою задовольнить потреби споживачів. Тому його розробка та впровадження є актуальним та доцільним.

Програмний засіб на сьогодні має перспективу та користь як для пересічних користувачів так і для спецслужб. Продукт, який пропонується є реалізованим засобом, що дозволяє проводити автентифікацію користувачів в системі. Готовий програмний продукт буде реалізовуватись на ринку програмних засобів шляхом щомісячної підписки за певну плату.

Під час встановлення ціни та попиту на новий програмний продукт основна увага повинна акцентуватися на унікальності об’єкта купівлі-продажу, цінах продуктів конкурентів, перевагах порівняно з аналогами, витратах, які зазнає покупець у разі заміни старого продукту новим, ступені терміновості та гостроті потреби.

Програмний засіб готовий для використання. Фахівці відповідної кваліфікації наявні, трудові та фінансові ресурси теж, обслуговування програми може відбуватись в режимі он-лайн, з будь-якої точки світу, оскільки немає проблем з передачею на нього прав. Комерціалізація розробки знаходиться на початковому етапі. Ведуться пошуки інвесторів та партнерів. Наявні зацікавлені особи, що готові першими випробувати програмний засіб в обмін на акт впровадження та подальшу рекламу від їх імені. Просування на ринок планується шляхом реалізації та продажу через спеціалізовані магазини програмного забезпечення.

4.2 Прогнозування витрат на виконання наукової роботи та впровадження її результатів

Прогнозування витрат на виконання науково-дослідної, дослідно-

конструкторської та конструкторсько-технологічної роботи складається з таких етапів:

— 1-й етап — розрахунок витрат, які безпосередньо стосуються виконавців даного розділу роботи;

— 2-й етап — розрахунок загальних витрат на виконання даної роботи;

— 3-й етап — прогнозування загальних витрат на виконання та впровадження результатів даної роботи.

Виконаємо розрахунок витрат, які безпосередньо стосуються виконавців даного розділу роботи, за такими статтями та формулами, приймаючи до уваги те, що для розробки інформаційної технології було залучено одного розробника програмного забезпечення.

Основна заробітна $З_о$.

$$З_о = \frac{М}{Т_р} \cdot t, \text{грн.} \quad (4.1)$$

де М — місячний посадовий оклад – 20 000 грн.;

$Т_р$ — число робочих днів в місяці; приблизно $Т_р = 20$ днів;

t — число робочих днів роботи – 30 днів.

Таким чином:

$$З_о = \frac{20000}{20} \cdot 30 = 30\,000 \text{ (грн.)}$$

Таблиця 4.5 — Витрати по заробітній платі

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату
Розробник	15 000	750	30	22500
Керівник розробки	10 000	500	30	15000
Всього				37 500

Додаткова заробітна плата $З_д$ працівників розраховується як 12% від основної заробітної плати:

$$З_д = 0{,}12 \cdot 22\,500 \ = 2\,700(\text{грн.}) - \text{для розробника}$$

$$З_д = 0{,}12 \cdot 15\,000 \ = 1\,800(\text{грн.}) - \text{для керівника розробки}$$

Нарахування на заробітну плату $Н_{зп}$ розробника становить:

$$Н_{зп} = (З_о + З_д) \cdot \frac{\beta}{100} \qquad (4.2)$$

де $З_о$ — основна заробітна плата розробника;

$З_д$ — додаткова заробітна плата розробника;

β — ставка єдиного внеску на загальнообов'язкове державне соціальне страхування – 22%.

$$Н_{зп} = (22\,500 \ + 2\,700) \cdot 0{,}22 = 5\,544 \text{ (грн.) розробнику}$$

$$Н_{зп} = (15\,000 \ + 1\,800) \cdot 0{,}22 = 7\,392 \text{ (грн.) керівнику}$$

Амортизація обладнання, комп'ютерів та приміщень, які використовувались під час виконання даного етапу роботи розраховуємо за формулою:

$$А = \frac{Ц \cdot Т}{12 \cdot Т_В} \qquad (4.3)$$

де Ц — загальна балансова вартість обладнання, приміщення тощо, грн.;

Т — фактична тривалість використання, міс;

$Т_В$ — термін використання обладнання, приміщень тощо, роки.

Розробка програмного забезпечення ведеться 1,5 місяці.

Розрахунки зведено до таблиці 4.6.

Таблиця 4.6 — Амортизаційні відрахування

Найменування	Балансова вартість (грн.)	Термін використання (років)	Фактична тривалість використання, (міс.)	Величина амортизаційних відрахувань, (грн..)
Офісне приміщення	50 000	20	1,5	625
Комп'ютер	10 000	2	1,5	300
Монітор	4 000	2	1,5	100
			Всього	1 025

Витрати на комплектуючі К, що були використані під час виконання даного етапу роботи, розраховуються за формулою:

$$К = \sum_{1}^{n} Н_i \cdot Ц_i \cdot К_i \text{ (грн.)} \quad (4.4)$$

де $Н_i$ — кількість комплектуючих *i*-го виду, шт.;

$Ц_i$ — ціна комплектуючих *i*-го виду, грн.;

$К_i$ — коефіцієнт транспортних витрат, $К_i = (1{,}1 \ldots 1{,}15)$;

n — кількість видів комплектуючих.

Таблиця 4.7 — Витрати на комплектуючі

Найменування комплектувальних	Кількість	Ціна за штуку, грн.	Сума, грн.	Примітка
Клавіатура (тип 1)	1	200 грн.	200 грн.	
Клавіатура (тип 2)	1	350 грн.	350 грн.	
Всього:			$К_i = 1{,}2$	550 грн.

Витрати на силову електроенергію $В_e$ розраховуються за формулою:

$$В_е = В \cdot П \cdot Ф \cdot К_п \text{ (грн.)} \quad (4.5)$$

де В — вартість 1 кВт-год.;

П — установлена потужність обладнання – 0,8 кВт;

Ф — фактична кількість годин роботи обладнання – 480 годин,

$К_п$ — коефіцієнт використання потужності.

$$В_е = 5 \cdot 0{,}8 \cdot 480 \cdot 0{,}14 = 269 \text{ (грн).}$$

Інші витрати $В_{ін}$ охоплюють:

— витрати на управління організацією;

— оплату службових відряджень;

— витрати на утримання, ремонт та експлуатацію основних засобів;

— витрати на опалення, освітлення, водопостачання, охорону праці тощо.

Інші витрати $В_{ін}$ можна прийняти як 100% від суми основної заробітної плати розробника та керівника:

$$В_{ін} = (22\,500 + 15\,000) \cdot 1 = 37\,500 \text{ (грн)}$$

Послуги Інтернету – 250 грн., канцтовари – 200 грн. Загальна вартість становить:

$$250 + 200 = 450 \text{ (грн.)}$$

Сума всіх попередніх статей витрат дає витрати на виконання даної частини роботи – В.

$$В = 37\,500 + 5\,400 + 9\,240 + 1025 + 550 + 269 + 37\,500 + 450 = 91\,934 \text{ (грн.)}$$

Проведемо прогнозування загальних витрат ЗВ на виконання та впровадження результатів виконаної наукової роботи. Прогнозування здійснюється за формулою:

$$ЗВ = \frac{В_{заг}}{\beta}, грн. \quad (4.6)$$

де β — коефіцієнт, який характеризує етап (стадію) виконання даної роботи;

$В_{заг}$ — загальна вартість всієї наукової роботи.

Так, якщо розробка знаходиться:

— на стадії науково-дослідних робіт, то $\beta \approx 0{,}1$;

— на стадії технічного проектування, то $\beta \approx 0{,}2$;

— на стадії розробки конструкторської документації, то $\beta \approx 0{,}3$;

— на стадії розробки технологій, то $\beta \approx 0{,}4$;

— на стадії розробки дослідного зразка, то $\beta \approx 0{,}5$;

— на стадії розробки промислового зразка, $\beta \approx 0{,}7$;

— на стадії впровадження, то $\beta \approx 0{,}9$.

$$В = 121\ 334\ (грн.)$$

$$ЗВ = \frac{121\ 334\ 505}{0{,}7} = 115\ 122 (грн.)$$

Отже, прогноз загальних витрат ЗВ на виконання та впровадження результатів виконаної наукової роботи складає $115\ 122$ (грн.)

Витрати на службові відрядження. Під час розробки програмного забезпечення відрядження штатних працівників, працівників організацій, які працюють за договорами цивільно-правового характеру, магістрів, зайнятих розробленням досліджень, відрядження, пов’язані з проведенням випробувань

машин та приладів, а також витрати на відрядження на наукові з'їзди, конференції, наради, пов'язані з виконанням конкретних досліджень, не плануються.

Витрати на роботи, які виконують сторонні підприємства, установи і організації. Витрати за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації» не плануються, так як у цьому не має потреби.

4.3 Прогнозування комерційних ефектів від реалізації результатів розробки

У даному підрозділі проведемо кількісне прогнозування, яку вигоду, зиск можна отримати у майбутньому від впровадження результатів виконаної наукової роботи.

В умовах ринку узагальнюючим позитивним результатом, що його отримує підприємство від впровадження результатів тієї чи іншої розробки, є збільшення чистого прибутку підприємства. Зростання чистого прибутку можна оцінити у теперішній вартості грошей.

Зростання чистого прибутку забезпечить інвестору надходження додаткових коштів, які дозволять покращити фінансові результати діяльності.

Виконання даної наукової роботи та впровадження її результатів складає приблизно 1 рік. Позитивні результати від впровадження розробки очікуються вже в перші місяці після впровадження.

Проведемо детальне прогнозування позитивних результатів та кількісне їх оцінювання по роках.

Обчислимо збільшення чистого прибутку підприємства $\Delta П_i$ для кожного із років, протягом яких очікується отримання позитивних результатів від впровадження розробки, розраховується за формулою:

$$\Delta П_i = \sum_{1}^{n} (\Delta П_я \cdot N + П_я \cdot \Delta N)_i \qquad (4.7)$$

де $\Delta П_я$ — покращення основного якісного показника від впровадження результатів розробки у даному році;

N — основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження результатів наукової розробки;

ΔN — покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки;

$П_я$ — основний якісний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки;

n — кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки.

Припустимо, що внаслідок впровадження результатів наукової розробки чистий прибуток підприємства збільшиться на 70 грн., а кількість одиниць реалізованої послуги збільшиться:

— протягом першого року — на 400 од.,

— протягом другого року — ще на 750 од.,

— протягом третього року — ще на 900 од.

Орієнтовно реалізація продукції до впровадження результатів наукової розробки складала 1 шт., а прибуток, що його отримувало підприємство на одиницю продукції до впровадження результатів наукової розробки — 60 грн.

Потрібно спрогнозувати збільшення чистого прибутку підприємства від впровадження результатів наукової розробки у кожному році відносно базового.

Збільшення чистого прибутку підприємства $\Delta П_1$ протягом першого року складе:

$$\Delta П_1 = 60 \cdot 1 + (60 + 70) \cdot 400 = 76\,000 \text{ (грн.)}$$

Обчислимо збільшення чистого прибутку підприємства $\Delta П_2$ протягом другого року:

$$\Delta П_2 = 60 \cdot 1 + (60 + 70) \cdot (400 + 750) = 218\,500 \text{ (грн.)}$$

Збільшення чистого прибутку підприємства $\Delta П_3$ протягом третього року становитиме:

$$\Delta П_3 = 60 \cdot 1 + (60 + 70) \cdot (400 + 750 + 900) = 389\,500 \text{ (грн.)}$$

Отже, розрахунки показують, що відповідно прогнозуванню комерційний ефект від впровадження розробки виражається у значному збільшенні чистого прибутку підприємства.

4.4 Розрахунок ефективності вкладених інвестицій та періоду їх окупності

Основними показниками, які визначають доцільність фінансування наукової розробки певним інвестором, є абсолютна і відносна ефективність вкладених інвестицій та термін їх окупності.

Розрахунок ефективності вкладених інвестицій передбачає розрахунок теперішньої вартості інвестицій PV, що вкладаються в наукову розробку.

Такою вартістю ми можемо вважати прогнозовану величину загальних витрат ЗВ на виконання та впровадження результатів НДДКР, тобто $ЗВ = PV = 83\,439 \text{ (грн.)}$

Розрахуємо очікуване збільшення прибутку$\Delta П_i$, що його отримає підприємство (організація) від впровадження результатів наукової розробки, для кожного із років, починаючи з першого року впровадження. Таке збільшення прибутку також було розраховане нами раніше та становить:

$$\Delta П_1 = 76\,000(\text{грн}), \Delta П_2 = 218\,500(\text{грн}), \Delta П_3 = 389\,500(\text{грн}).$$

Будуємо вісь часу, на якій відображаємо всі платежі (інвестиції та прибутки), що мають місце під час виконання науково-дослідної роботи та впровадження її результатів. Рисунок 4.1 характеризує рух платежів (інвестицій та додаткових прибутків).

Рисунок 4.1 — Вісь часу з фіксацією платежів, що мають місце під час розробки та впровадження результатів НДДКР

Розрахуємо абсолютну ефективність вкладених інвестицій $Е_{абс}$ за формулою:

$$Е_{абс} = (ПП - PV), (грн.) \quad (4.8)$$

де ПП — приведена вартість всіх чистих прибутків, що їх отримає підприємство (організація) від реалізації результатів наукової розробки, грн.;

PV — теперішня вартість інвестицій $PV = ЗВ, грн.$

Приведена вартість всіх чистих прибутків ПП розраховується за формулою:

$$ПП = \sum_{1}^{Т} \frac{\Delta П_i}{(1+\tau)^t}, (грн) \quad (4.9)$$

де $\Delta П_i$ — збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДДКР, грн.;

Т — період часу, протягом якого виявляються результати впровадженої НДДКР, роки;

τ — ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні – 0,1;

t — період часу (в роках) від моменту отримання чистого прибутку до точки «0»;

$$ПП = \frac{76\,000}{(1+0{,}1)^1} + \frac{218\,500}{(1+0{,}1)^2} + \frac{389\,500}{(1+0{,}1)^3} = 542\,306{,}5\ (\text{грн.})$$

$$Е_{абс} = 542\,306{,}5 - 115\,122 = 427\,184{,}5\ (\text{грн.})$$

Оскільки $Е_{абс} > 0$, результат від проведення наукових досліджень щодо розробки програмного продукту та їх впровадження принесе прибуток, тобто є доцільним, проте це ще не свідчить про те, що інвестор буде зацікавлений у фінансуванні даної програми.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $Е_в$ за формулою:

$$Е_в = \sqrt[Т_ж]{1 + \frac{Е_{абс}}{PV} - 1} \qquad (4.10)$$

де $Е_{абс}$ — абсолютна ефективність вкладених інвестицій, грн.;

PV — теперішня вартість інвестицій $PV = 3В, грн.$

$Т_ж$ — життєвий цикл наукової розробки, роки.

$$E_в = \sqrt[3]{1 + \frac{427\,184{,}5}{115\,122}} - 1 = \sqrt[3]{5{,}1} - 1 = 0{,}705 \text{ або } 70{,}5\%$$

Порівняємо $E_в$ з мінімальною (бар'єрною) ставкою дисконтування τ_{min}, яка визначає ту мінімальну дохідність, нижче за яку інвестиції вкладатися не

будуть.

Спрогнозуємо величину τ_{min}. У загальному вигляді мінімальна (бар'єрна) ставка дисконтування τ_{min}визначається за формулою:

$$\tau_{min} = d + f \tag{4.11}$$

де d — середньозважена ставка за депозитними операціями в комерційних банках; $d = 0{,}2$;

f — показник, що характеризує ризикованість вкладень, величина $f = 0{,}5$.

$$\tau_{min} = 0{,}2 + 0{,}5 = 0{,}7$$

Оскільки

$$E_{\text{в}} = 70{,}5\% > \tau_{min} = 70\%,$$

то у інвестора є потенційна зацікавленість у фінансуванні даної наукової розробки.

Розрахуємо термін окупності вкладених у реалізацію наукового проекту інвестицій $\text{Т}_{\text{ок}}$ за формулою:

$$\text{Т}_{\text{ок}} = \frac{1}{\text{Е}_{\text{в}}}, \text{рік} \tag{4.12}$$

$$\text{Т}_{\text{ок}} = \frac{1}{0{,}71} = 1{,}4 \text{ (року)}$$

Оскільки термін окупності вкладених у реалізацію наукового проекту інвестицій менше трьох років ($\text{Т}_{\text{ок}} < 3$ років), то фінансування нової розробки є доцільним.

4.5 Підсумки до розділу

В даному розділі було виконано оцінювання комерційного потенціалу

розробки програмного засобу, який відрізняється збільшеною відповідністю до критеріїв користувача. Сферою застосування результатів роботи є комп’ютерні системи маркетингу, реклами, торгівлі, тощо.

Проведено технологічний аудит з залученням трьох незалежних експертів. Визначено, що рівень комерційного потенціалу розробки вище середнього. Проведено порівняння з аналогом. Згідно з проведеним оцінюванням нова розробка є якісною та конкурентоспроможною.

Рівень комерційного потенціалу розробки, становить 42,7 балів, що відповідає рівню «високий».

Згідно із розрахунками всіх статей витрат на виконання науково-дослідної, дослідно-конструкторської та конструкторсько-технологічної роботи загальні витрати на розробку складають 115 122 (грн.). Розрахована абсолютна ефективність вкладених інвестицій в сумі 427 184,5 (грн.) свідчить про отримання прибутку інвестором від комерціалізації програмного продукту.

Щорічна ефективність вкладених в наукову розробку інвестицій складає 71%, що вище за мінімальну бар'єрну ставку дисконтування, яка складає 70%. Це означає потенційну зацікавленість інвесторів у фінансуванні розробки.

Термін окупності вкладених у реалізацію проекту інвестицій становить 1,4 (року), що також свідчить про доцільність фінансування нової розробки.

Отже, проаналізувавши отримані економічні показники, можна вважати, що запропонована розробка програмного засобу має високий комерційний потенціал, а тому є доцільною для подальшого впровадження.

ВИСНОВКИ

В магістерській кваліфікаційній роботі розроблено програмний засіб для безпечного конфігурування кешування динамічного вмісту веб-сторінок, метою якого є спрощення роботи користувача в процесі конфігурування динамічного контенту, який може бути закешований, зменшення ризику кешування персональних даних, збільшення відсотку закешованих сторінок, які містять динамічний контент, зниження навантаження на основний сервер, та пришвидшення завантаження сторінок для кінцевого користувача.

В першому розділі роботи здійснено аналіз теоретичного матеріалу обраної галузі. А саме, здійснено аналіз методів та засобів кешування вмісту веб-сторінок, досліджено особливості статичного та динамічного вмісту веб-сторінок, кешування та класифікацію кешів, поняття «кешу» та важливість процесу кешування, проаналізовано принципи роботи кешу, управління кешуванням та валідація кешу, кешування в мережах доставки контенту (CDN), часу кешування.

В другому розділі роботи здійснена розробка методу та алгоритму роботи програмного засобу. А саме, обґрунтовані принципи безпечного конфігурування кешування динамічного вмісту веб-сторінок, особливості побудови програмного засобу, розроблені алгоритм модуля налаштування кешування та алгоритм роботи модулю перевірки кешу.

В третьому розділі роботи здійснена розробка програмного засобу безпечного конфігурування кешування веб-сторінок , обґрунтовано вибір інструментарію реалізації програмного засобу (мова програмування C#, ASP.NET Core Фреймворк, візуалізатор Razor Pages , сервіс Microsoft Azure), здійснено програмну розробка спроектованого засобу та перевірку якості роботи програмного засобу.

В четвертому розділі роботи було виконано оцінювання комерційного потенціалу розробки програмного засобу, який відрізняється збільшеною відповідністю до критеріїв користувача. Сферою застосування результатів

роботи є комп’ютерні системи маркетингу, реклами, торгівлі, тощо. Проаналізувавши отримані економічні показники, можна вважати, що запропонована розробка програмного засобу має високий комерційний потенціал, а тому є доцільною для подальшого впровадження.

Таким чином, в магістерській кваліфікаційній роботі розроблений підхід щодо прискорення відображення динамічного контенту відбувається за рахунок створення власної розподіленої системи для налаштування кешування динамічного вмісту веб сторінок, на основі хмарних технологій. Дана система дозволяє безпечно кешувати можливості використання HTML сторінки, що значно знижує навантаження на вихідний сервер, в наслідок чого зменшується вар-тість хостингу веб-сайтів. За рахунок використання хмарних технологій, розроблений додаток є більш гнучким, легко масштабованим та надійним, внаслідок чого витрати на обслуговування додатку є також значно зниженими.

Даний програмний засіб рекомендовано використовувати лише для великих систем, з великою кількістю даних, ресурсів та сторінок. Якщо ж мова йде про невеликий сайт, то витрати на впровадження такого програмного рішення можуть бути більшими, ніж його користь.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl, “Globally Distributed Content Delivery,” IEEE Internet Computing, pp. 50-58, September/October 2002

2. Akamai Technologies, Inc., www.akamai.com, 2007

3. J. Ni, and D. H. K. Tsang, “Large Scale Cooperative Caching and Applicationlevel Multicast in Multimedia Content Delivery Networks,” IEEE Communications, Vol. 43, Issue. 5, pp. 98-105, May 2005.

4. Internet Content Distribution: Developments and Challenges. Adrian Popescu, David Erman, Dragos Ilie, Doru Constantinescu

5. Akamai Technologies, Inc. A Distributed Infrastructure for e-Business — Real Benefits, Measurable Returns.23

6. Yair Bartal. Probabilistic approximation of metric space and its algorithmic applications. In 37th Annual IEEE Symposium on Foundations of Computer Science, October 1996

7. M. Baentsch, L. Baum, G. Molter, S. Rothkugel, and P. Sturm. Enhancing the web infrastructure - from caching to replication. IEEE Internet Computing, pages 18– 27,Mar-Apr 1997

8. Rajkumar Buyya, Al-Mukaddim Khan Pathan, James Broberg and Zahir Tari. A Case for Peering of Content Delivery Networks

9. G. Peng, CDN: Content Distribution Network, Technical Report TR-125, Experimental Computer Systems Lab, Department of Computer Science, State University of New York, Stony Brook, NY 2003. http://citeseer.ist.psu.edu/peng03cdn.html

10. A. Vakali and G. Pallis, Content Delivery Networks: Status and Trends, IEEE Internet Computing, IEEE Computer Society, pp. 68-74, November-December 2003

11. Architecture and Performance Models for QoS-Driven Effective Peering of Content Delivery Networks. Mukaddim Pathan, Rajkumar Buyya

12. Amazon CloudFront. http://aws.amazon.com/cloudfront/

13. oStream: Asynchronous Streaming Multicast in Application-Layer Overlay Networks. Cui Y., Li B. and Nahrstedt K., , IEEE Journal on Selected Areas in Communications, Vol 22, No 1, January 2004

14. Martin Casado, Tal Garfinkel, Aditya Akella, Michael J. Freedman Dan Boneh, Nick McKeown, Scott Shenker. SANE: A Protection Architecture for Enterprise Networks, 15-th Usenix SS, Vancouver, Canada, August 2006

15. N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner. Openflow: Enabling innovation in campus networks. SIGCOMM Computer Communication Review, vol. 38, no. 2, pp. 69–74, 2008

16. Arnaud Legout, Guillaume Urvoy-Keller: Understanding BitTorrent: An Experimental Perspective, PLANETE(INRIA Sophia Antipolis), 2005.

17. Openflow project site, http://www.openflow.org/.

18. Ньюкомер Э. Веб-сервисы. Для профессионалов: Пер. с англ. - С.-Петербург: Питер, 2003. - 256 с.

19. Хабибулин И.Ш. Самоучитель XML: Пер. с англ. - С.-Петербург: BHVCПб, 2003. - 336 с.

20. Бин Д. XML для проектировщиков: Пер. с англ. - М.:Символ-Плюс, 2004. -256 с.

21. Мартин Д., Бирбек М., Лозген Б., Пиннок Д., Ливингстон С. XML для профессионалов: Пер. с англ. - С.-Петербург: Лори, 2001. - 900 с.

22. Боуэн Р., Лиска А. Apache. Настольная книга администратора: Пер. С англ. - М.:ДиаСофт ЮП, 2002. - 384 с.

23. Айлебрехт Л. Web-сервер Apache: Пер. с англ. - М.: Новое знание, 2002. –592 с.

24. Веллинг Л. Разработка Web-приложений с помощью PHP и MySQL 3-е издание: Пер. с англ. - М.: Вильямс, 2005. - 910 с.

25. Дюбуа П. MySQL. Сборник рецептов: Пер. с англ. - М.:Символ-Плюс, 2004. - 1056 с.

26. Аргерих К., Чой В., Коггсхол Д., Эгервари К., Сколло К.

Профессиональное PHP программирование. 2-е издание: Пер. с англ. - М.:СимволПлюс, 2003. -1048 с.

27. PostgreSQL: Руководство разработчика и администратора: Пер. с англ. - М.:ДиаСофт ЮП, 2002. - 608 с.

28. Уорсли Д., Дрейк Д. PostgreSQL. Для профессионалов: Пер. с англ. - С.-Петербург: Питер, 2002. - 496 с.

29. Стоунз Р., Мэттью Н. PostgreSQL. Основы: Пер. с англ. - М.:СимволПлюс, 2002. - 640 с.

30. Цвики Э., Купер С., Чапмен Б. Создание защиты в Интернете (2 издание): Пер. с англ. - М.:Символ-Плюс, 2002. - 928 с.

31. PHP и MySQL. Создание интернет-магазина: Кристиан Дари, Эмилиан Баланеску — Санкт-Петербург, Вильямс, 2010 г.- 640 с.

32. Козье Д. Электронная комерция: Пер. с англ. - Москва: Издательськоторговий дом «Руская редакция». 1999. - 288 с.: ил.

33. Кобелев О.А. Электронная коммерция: учеб. пособие / С.В. Пирогов (ред.). — 3-е изд., перераб. и доп. — М. : Дашков и Ко, 2008. — 683с

34. Гаврилов Л. П. Электронная коммерция. Учебное пособие по выполнению практических работ Издательство: Солон-Пресс, 2006 г. 112 с.

35. Скотт Хокинс. Администрирование веб-сервера Apache и руководство по электронной коммерции = Apache Web Server Administration and e-Commerce Handbook. — М.: Вильямс, 2001. — 336 с.

36. Sravanthi Kalepu, Shonali Krishnaswamy, Seng Wai Loke, Verity: A QoS Metric for Selecting Web Services and Providers [Електронний ресурс] Режим доступу до файлу: http://www.acs.org.au/vic/socsig/IEEE-VeritySOA

37. Терейковська Л.О. Архітектура марківської моделі зміни навантаження Web-сервера / Л.О. Терейковська // Вісник ДУІКТ. – 2012. – Т.10, №1. – С. 95-100.

38. Tom Vercauteren, Pradeep Aggarwal, Xiaodong Wang Hierarchical Forecasting of Web Server Workload Using Sequential Monte Carlo Training // IEEE transactions on signal processing. – 2007. – VOL. 55. – № 4. – PP. 635-644.

39. Memcached [Електронний ресурс]. – Режим доступу до файлу: http://en.wikipedia.org/wiki/ Memcached

40. Membase Server is Now Couchbase Server [Електронний ресурс]. – Режим доступу до файлу: http://www.couchbase.com/membase.

6. Disk Storage [Електронний ресурс]. – Режим доступу до файлу: http://www.couchbase.com/docs/

41. Couchbase features [Електронний ресурс]. – Режим доступу до файлу: http://www.couchbase.com/couchbase-server/features.

42. Квєтний Р.Н. Імовірнісні нейронні мережі в задачах ідентифікації часових рядів / В. В. Кабачій, О. О. Чумаченко [Електронний ресурс]. – Режим доступу до файлу.: http://www.nbuv.gov.ua/ejournals/vntu/2010_3/2010-3.files/

43. Agustín C. Caminero, Salvador Ros, Roberto Hernández, Antonio Robles-Gómez: A Use Case [Електронний ресурс]. – Режим доступу до журн.: http://fie-conference.org/fie2011/papers/1085.pdf.

44. Джон Дакетт - HTML и CSS. Разработка и дизайн веб-сайтов – 2013 – «Эксмо» - 120 с.

45. Эрик Мейер "CSS - Каскадные таблицы стилей. Подробное руководство" – 2008 - ООО Издательство «Питер» - 341 с.

46. Реализация MVC паттерна на примере создания сайта-визитки [Електронний ресурс]. – Режим доступу: http://habrahabr.ru/post/150267/

47. Обзор способов и протоколов аутентификации в веб-приложениях [Електронний ресурс]. – Режим доступу: https://habrahabr.ru/company/dataart/

48. Methods For Tracing and Debugging Redis Lua Scripts [Virtual Resource] / Access Mode: URL: https://redislabs.com/blog/5-6-7-methods

49. Lua: A Guide for Redis Users [Virtual Resource] / Access Mode: URL: https://www.redisgreen.net/blog/intro-to-lua-for-redis-programmers/

50. Embedding JavaScript in HTML [Virtual Resource] / Access Mode: URL: https://docstore.mik.ua/orelly/webprog/jscript/ch12_02.htm Title from Screen

51. Сэм Руби, Дейв Томас, Дэвид Хэннсон - Rails 4. Гибкая разработка вебприложений – 2014 - ООО Издательство «Питер» - 231 с.

52. Веб-сервер [Електронний ресурс]. – Режим доступу: https://ru.wikipedia.org/wiki/Веб-сервер

53. Шилдт, Г.C# 3.0 : руководство для начинающих [Текст] : учебное пособие. М. : ООО "И.Д. Вильямс", 2009., 688 с.

54. Гуннерсон Э. Введение в C#. Питер, 2005. – 304с.

55. ASP.NET Core [Електронний ресурс]. – Режим доступу: https://ru.wikipedia.org/wiki/ASP.NET_Core

56. Model-View-Controller [Електронний ресурс]. – Режим доступу: https://ru.wikipedia.org/wiki/Model-View-Controller

57. .Net Core [Електронний ресурс]. – Режим доступу: https://ru.wikipedia.org/wiki/.NET

58. Razor Pages. Введение в Razor Pages [Електронний ресурс]. – Режим доступу: https://metanit.com/sharp/aspnet5/29.1.php

59. Преимущества службы приложений azure [Електронний ресурс]. – Режим доступу: https://docs.microsoft.com/ru-ru/azure/app-service/overview

60. Azure geographies [Електронний ресурс]. – Режим доступу: https://azure.Microsoft.com/en-us/globalinfrastructure/geographies/

61. Azure Resource Manager. 2021 [Електронний ресурс]. – Режим доступу: https://docs.Microsoft.com/ruru/azure/azure

62. Общие сведения о службе Azure Monitor. 2019 [Електронний ресурс]. – Режим доступу: https://docs.Microsoft.com/ru-ru/azure/azure-Monitor/overview

63. Обзор оповещений в Microsoft Azure. 2021 [Електронний ресурс]. – Режим доступу: https://docs.Microsoft.com/ru-ru/azure/azure-Monitor/alerts/alerts-overview

64. База данных Azure для PostgreSQL [Електронний ресурс]. – Режим доступу: https://azure.microsoft.com/ru-ru/pricing/details/postgresql/

65. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад.: В. О. Козловський, О. Й. Лесько, В. В. Кавецький. — Вінниця : ВНТУ, 2021. — 42 с.

ДОДАТОК А

Технічне завдання

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
професор, д.т.н.
_____ О. Д. Азаров

«__» _____ 2021 р.

ТЕХНІЧНЕ ЗАВДАННЯ

до магістерської кваліфікаційної роботи на тему:

Програмний засіб безпечного конфігурування кешування динамічного вмісту веб-сторінок
08-23.МКР.023.00.00 ТЗ

Науковий керівник
магістерської кваліфікаційної роботи
проф. Азарова А.О.
_____

Вінниця 2021

1 Підстава для виконання магістерської кваліфікаційної роботи (МКР)

1.1 Актуальність дослідження пов’язана з необхідністю вирішення задачі надання динамічного вмісту веб-сторінок максимально відповідного до критеріїв користувача. Зокрема, щоб забезпечити максимальний комфорт користувача при користуванні веб-сайтом, надати йому рекомендації(товар, послуга сайту, тощо) відповідно до його критеріїв.

1.2 Наказ про затвердження теми магістерської кваліфікаційної роботи.

2 Мета і призначення МКР

2.1 Мета магістерської роботи полягає у підвищенні рівня відповідності динамічного вмісту веб-сторінок вимогам користувача засобами конфігурування кешування динамічного вмісту веб-сторінок.

2.2 Призначення розробки – виконання магістерської кваліфікаційної роботи.

3 Вихідні дані для виконання МКР

Реалізувати процес конфігурування кешування динамічного вмісту веб-сторінок. Дослідити методи і алгоритми кешування динамічного вмісту веб-сторінок.

4 Вимоги до виконання МКР

МКР повинна задовольняти такі вимоги:

— проаналізувати методи і засоби кешування динамічного вмісту веб-сторінок;

— виявити та обрати найбільш ефективні технології конфігурування кешування веб-сторінок;

— розробити метод кешування динамічного вмісту веб-сторінок з можливістю безпечного конфігурування;

— розробити алгоритм кешування динамічного вмісту веб-сторінок;

— обрати інструментарій для розробки програмного продукту;

— розробити та дослідити ефективність створеного прикладного програмного продукту.

5 Етапи МКР

Етапи МКР та очікувані результати приведені в таблиці А.1

Таблиця А.1 — Етапи виконання роботи

№	Назва етапу	Термін виконання		Очікувані результати
		початок	кінець	
1	Аналіз завдання. Вступ	07.09.2021	10.09.2021	Вступ
2	Аналіз літературних джерел для розпізнавання особи	13.09.2021	22.09.2021	розділ 1
3	Розробка технічного завдання	23.09.2021	24.09.2021	Технічне завдання
3	Розробка структури системи розпізнавання особи за зображенням обличчя	25.09.2021	08.10.2021	Розділ 2, розробка структури
4	Розробка програми, проектування програмного продукту	11.10.2021	29.10.2021	Розділ 3, розробка програми
5	Практична реалізація, результати.	01.11.2021	14.11.2021	Розділ 3
6	Розробка економічної частини	15.11.2021	30.11.2021	Розділ 4
7	Оформлення пояснювальної записки	02.12.2021	15.12.2021	ПЗ, презентація

6 Матеріали, що подаються до захисту МКР

До захисту МКР подаються: пояснювальна записка , ілюстративні та графічні матеріали, протокол попереднього захисту на кафедрі, відзив наукового керівника, відзив рецензента, протоколи складання державних екзаменів, анотації українською та іноземною мовами.

7 Порядок контролю виконання та захисту МКР

Виконання етапів розрахункової та графічної документації МКР контролюється науковим керівником згідно зі встановленими термінами. Захист МКР відбувається на засіданні Державної екзаменаційної комісії, затвердженою наказом ректора.

8 Вимоги до оформлення МКР

Вимоги викладені в ДСТУ 3008-2015 та «Положенні про кваліфікаційні роботи на другому (магістерському) рівні вищої освіти ВНТУ 2021».

Технічне завдання до виконання отримав _____ Курко В.С.

ДОДАТОК Б

Блок-схема програмного засобу для кешування динамічного вмісту веб-сторінок

Рисунок Б.1 — Блок-схема програмного засобу для кешування динамічного вмісту веб-сторінок

ДОДАТОК В

Блок-схема роботи модулю перевірки кешування

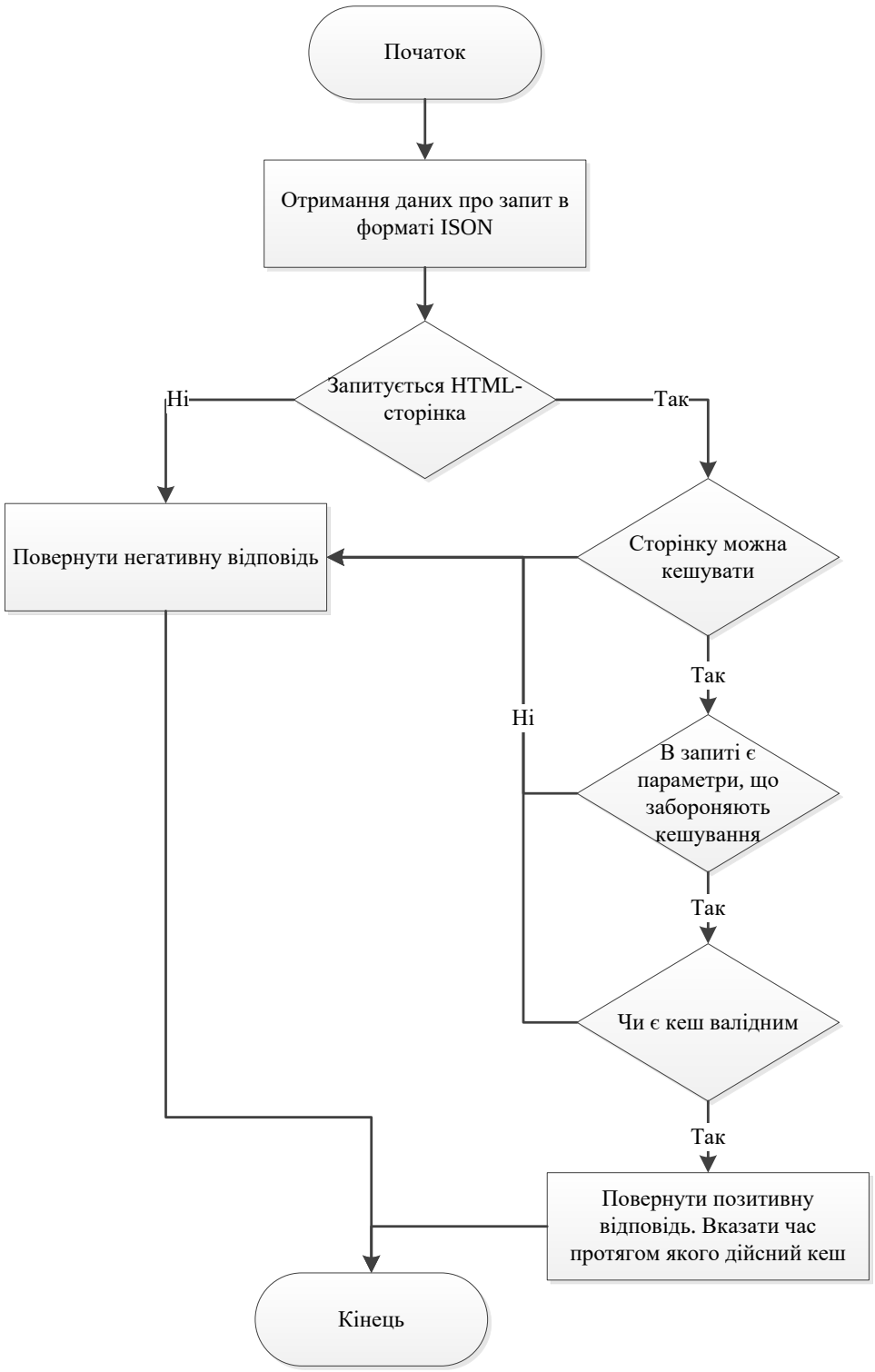


Рисунок В.1 — Блоксхема роботи модулю перевірки кешування

ДОДАТОК Г

Текст розмітки списку налаштувань сайтів та сторінок

```
@{
   ViewData["Title"] = "Index";
}
<h1>Manage your sites</h1>
<p>
   <a class="btn btn-primary" asp-action="Create">Add New Site</a>
</p>
@foreach (var sitemap in Model)
{
   <div class="site">
      <div class="container">
         <div class="row">
            <div class="col-md-6 siteName"><h3>@Html.DisplayFor(modelItem => sitemap.SiteName)</h3></div>
            <div class="col-md-6 page-inner-right">
               <a class="btn btn-primary" asp-action="ConfigureInfrastructure" asp-route-id="@sitemap.Id">Configure Infrastructure</a>
            </div>
         </div>
      </div>
      <div class="container page level-1">
         <div class="row">
            <div class="col-md-6 page-inner-left">@sitemap.RootPage.Url</div>
            <div class="col-md-6 page-inner-right">
               @if (sitemap.RootPage.Configured)
               {
                  if (sitemap.RootPage.Cacheble)
                  {
                     <i class="far fa-check-square"></i>
                  }
                  else
                  {
                     <i class="fas fa-times"></i>
                  }
               }
               <a href="~/Sitemap/Edit/@sitemap.RootPage.Id">Setup Caching</a> |
               <a asp-action="Details" asp-route-
```

```
id="@sitemap.RootPage.Id">Details</a> |
                    <a asp-action="Clear" asp-route-
id="@sitemap.RootPage.Id">Clear</a>
                    @if (sitemap.RootPage.ChildPages.Any())
                    {
                        <a asp-action="Collapse" asp-route-
id="@sitemap.RootPage.Id"> <i class="fas fa-chevron-down"></i></a>
                    }
                </div>
            </div>
        </div>
        @foreach (var childpage in @sitemap.RootPage.ChildPages)
        {
            <div class="container page level-2 @(childpage.Hidden ? "hidden" :
"")">
                <div class="row">
                    <div class="col-md-6 page-inner-left">@childpage.Url</div>
                    <div class="col-md-6 page-inner-right">
                        @if (childpage.Configured)
                        {
                            if (childpage.Cacheble)
                            {
                                <i class="far fa-check-square"></i>
                            }
                            else
                            {
                                <i class="fas fa-times"></i>
                            }
                        }
                        <a href="~/Sitemap/Edit/@childpage.Id">Setup Caching</a> |
                        <a asp-action="Details" asp-route-
id="@childpage.Id">Details</a> |
                        <a asp-action="Clear" asp-route-id="@childpage.Id">Clear</a>
                        @if (childpage.ChildPages.Any())
                        {
                            <a asp-action="Collapse" asp-route-id="@childpage.Id"> <i
class="fas fa-chevron-down"></i></a>
                        }
                    </div>
                </div>
            </div>
        }
    </div>
}
```

ДОДАТОК Д

Текст контролеру модулю налаштування кешу

```
using DynamicCachingApplication.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Xml;
using DynamicCachingApplication.Services;

namespace DynamicCachingApplication.Controllers
{
   public class CasheSettingsController : Controller
   {
      private readonly ILogger<SitemapController> _logger;
      private readonly SitemapDbContext _context;
      private List<Sitemap> sitemaps;
      private CosmosDbService dbService = new CosmosDbService();

      public SitemapController(ILogger<SitemapController> logger,
SitemapDbContext context)
      {
         _logger = logger;
         _context = context;

         GenerateSitemap();

         //Cosmos
         this.sitemaps = _context.Sitemaps.ToList();
      }

      public IActionResult Index()
      {
         return View(sitemaps);
      }

      public IActionResult Create()
      {
         return View(new SitemapXml());
```

```
}

[HttpPost]
public IActionResult Create(SitemapXml model)
{

   if (model.Xml != null)
   {
      XmlDocument doc = new XmlDocument();
      doc.Load(model.Xml.OpenReadStream());

      string xmlstring = doc.InnerXml;
   }
   return RedirectToAction("Index", "Sitemap");
}

public IActionResult Collapse(Guid? id)
{
   var pages = _context.Pages.Where(p => p.ParentId == id).ToList();
   foreach (var page in pages)
   {
      if (page.Hidden) page.Hidden = false;
      else {
         page.Hidden = true;
         foreach (var subpage in page.ChildPages)
         {
            subpage.Hidden = true;
         }
      }

   }

   _context.SaveChanges();

   return RedirectToAction("Index");
}

public IActionResult Edit(Guid? id)
{
   if (id == null) return RedirectToAction("Index");

   var page = _context.Pages.Find(id);
   ViewBag.PageId = id;
   return View(page);
}
```

```
        [HttpPost]
        public IActionResult Edit(Guid? id,
                                  Page input,
                                  string allowParamName,
                                  string allowParamValue,
                                  string disallowParamName,
                                  string disallowParamValue,
                                  bool applyToSimilar)
        {
            var page = _context.Pages.Single(p => p.Id == id);

            if (page != null && ModelState.IsValid)
            {
                page.Cacheble = input.Cacheble;

                if (input.ValidationRules.CacheLifeTime != null)
                {
                    if (page.ValidationRules == null)
                    {
                        page.ValidationRules = new ValidationRules();
                    }
                    page.ValidationRules.CacheLifeTime                                    =
input.ValidationRules.CacheLifeTime;
                }
                page.Configured = true;
                _context.SaveChanges();

                if (applyToSimilar)
                {
                    foreach(var childpage in page.ParentPage.ChildPages.ToList())
                    {
                        if (childpage == page) continue;

                        childpage.Cacheble = page.Cacheble;
                        childpage.ValidationRules = new ValidationRules {
                            CacheLifeTime = page.ValidationRules.CacheLifeTime
                        };
                        childpage.Configured = page.Configured;
                        childpage.AllowParameters.Clear();
                        foreach (var param in page.AllowParameters)
                        {
                            childpage.AllowParameters.Add(new AllowParameter
                            {
                                ParameterName = param.ParameterName,
```

```
                        ParameterValue = param.ParameterValue
                    });
                }
                childpage.DisallowParameters.Clear();
                foreach (var param in page.DisallowParameters)
                {
                    childpage.DisallowParameters.Add(new DisallowParameter
                    {
                        ParameterName = param.ParameterName,
                        ParameterValue = param.ParameterValue
                    });
                }
            }
        }

        _context.SaveChanges();
        if (!string.IsNullOrEmpty(allowParamName) &&
!string.IsNullOrEmpty(allowParamValue))
        {
            page.AllowParameters.Add(new AllowParameter {
ParameterName = allowParamName, ParameterValue = allowParamValue });
            _context.SaveChanges();
            return View(page);
        }
        if (!string.IsNullOrEmpty(disallowParamName) &&
!string.IsNullOrEmpty(disallowParamValue))
        {
            page.DisallowParameters.Add(new DisallowParameter {
ParameterName = disallowParamName, ParameterValue = disallowParamValue });
            _context.SaveChanges();
            return View(page);
        }
        return RedirectToAction("Index");
    };
    return View(page);
}

public IActionResult Clear(Guid? id)
{
    var page = _context.Pages.Single(p => p.Id == id);

    if (page != null)
    {
        page.Cacheble = false;
        page.ValidationRules.CacheLifeTime = "";
```

```
                page.Configured = false;
                page.AllowParameters.Clear();
                page.DisallowParameters.Clear();
                _context.SaveChanges();
            };
            return RedirectToAction("Index");
        }

        public IActionResult RemoveParameter(Guid? pageId, Guid? paramId,
bool allowed)
        {
            var page = _context.Pages.Single(p => p.Id == pageId);
            if (allowed)
            {
                page.AllowParameters.Remove(page.AllowParameters.Single(par =>
par.Id == paramId));
            }
            else
            {

page.DisallowParameters.Remove(page.DisallowParameters.Single(par => par.Id ==
paramId));
            }

            _context.SaveChanges();

            return RedirectToAction("Edit", new { id = pageId});
        }

        public IActionResult RemoveLocation(Guid? id, Guid? locationId)
        {
            var sitemap = _context.Sitemaps.Single(s => s.Id == id);

            sitemap.Locations.Remove(sitemap.Locations.Single(l => l.Id ==
locationId));

            _context.SaveChanges();
            var regions = Constants.SiteConstants.AzureLocations;
            ViewBag.Regions = regions;
            return RedirectToAction("ConfigureInfrastructure", new { id =
sitemap.Id});
        }
        public IActionResult ConfigureInfrastructure(Guid? id)
        {
            var sitemap = _context.Sitemaps.Single(s => s.Id == id);
```

```
            var regions = Constants.SiteConstants.AzureLocations;
            ViewBag.Regions = regions;
            return View(sitemap);
        }
        [HttpPost]
        public async Task<IActionResult> ConfigureInfrastructureAsync(Guid? id,
                                Sitemap input,
                                string locationKey)
        {
            var sitemap = _context.Sitemaps.Single(s => s.Id == id);
            var regions = Constants.SiteConstants.AzureLocations;
            ViewBag.Regions = regions;
            if (sitemap != null && ModelState.IsValid)
            {
                sitemap.EnableAutoscaling = input.EnableAutoscaling;
                _context.SaveChanges();

                if (!string.IsNullOrEmpty(locationKey))
                {
                    sitemap.Locations.Add(new Location
                    {
                        LocationKey = locationKey,
                        LocationName = Constants.SiteConstants.AzureLocations[locationKey]
                    });
                    _context.SaveChanges();
                    return View(sitemap);
                }
                var responces = await dbService.UploadSiteSettingsAsync(
                    new PageMapperService().MapPages(_context.Pages.ToList()),
                    endpointUrl,
                    primaryKey
                );
                return RedirectToAction("Index");
            };
            return View(sitemap);
        }
        private void GenerateSitemap()
        {
            if (!_context.Sitemaps.ToList().Any())
            {
                var sitemap1 = new Sitemap
                {
                    SiteName = "Site_1",
```

```
                RootPage = new Page
                {
                    Url = "/",
                    ValidationRules = new ValidationRules
                    {
                        CacheLifeTime = string.Empty
                    },
                    ChildPages = new List<Page>
                {
                    new Page
                    {
                        Url = "/products",
                        ValidationRules = new ValidationRules
                        {
                            CacheLifeTime = string.Empty
                        }
                    },
                    new Page
                    {
                        Url = "/blog",
                        ValidationRules = new ValidationRules
                        {
                            CacheLifeTime = string.Empty
                        }
                    }
                }
                }
            };
            _context.Sitemaps.Add(sitemap1);
            _context.SaveChanges();
        }
    }
  }
}
```

ДОДАТОК Е

Ілюстративний матеріал

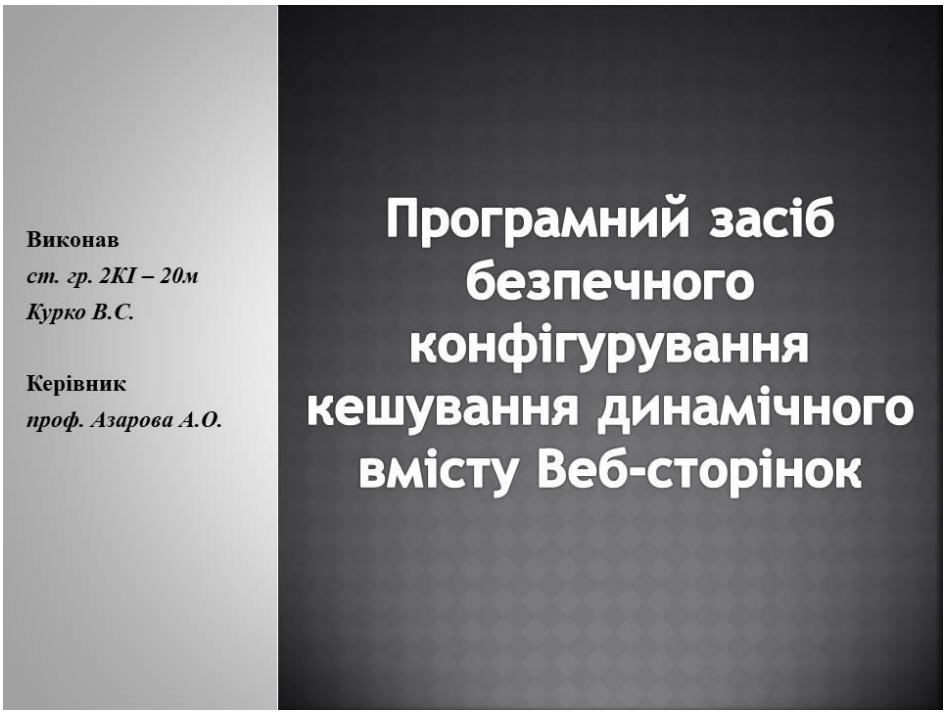


Рисунок Е.1 — Слайд 1

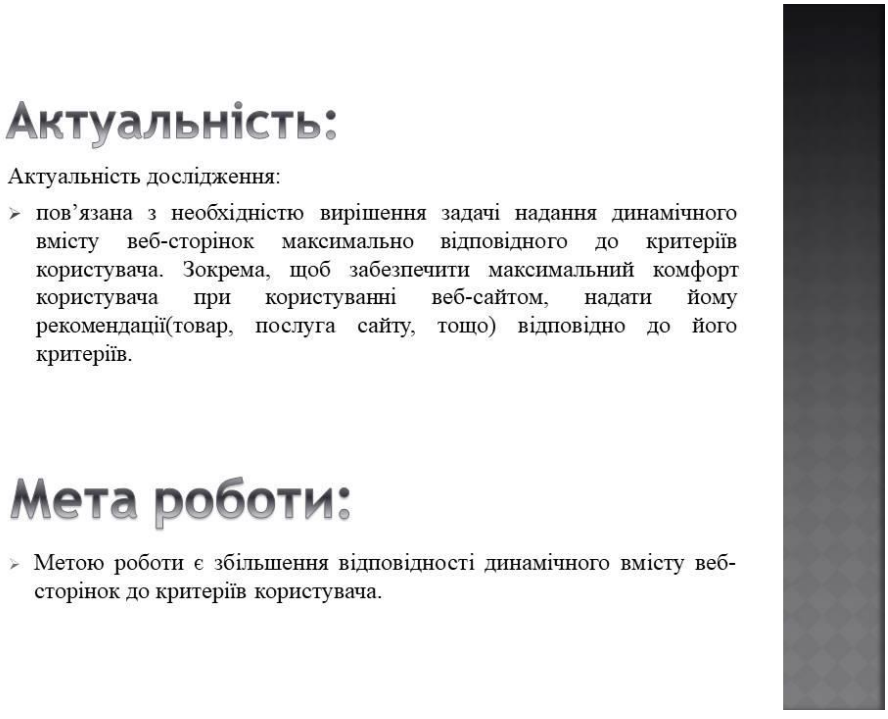
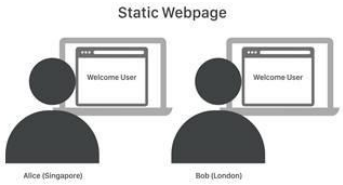


Рисунок Е.2 — Слайд 2

Статичний та динамічний вміст веб-сторінок

- Статичний контент — це будь-який файл, який зберігається на сервері, і той самий щоразу, коли він доставляється користувачам. HTML-файли та зображення є прикладами такого змісту.



- Динамічний вміст — це вміст, який змінюється залежно від факторів, характерних для користувача, таких як час відвідування, місце-знаходження та пристрій.

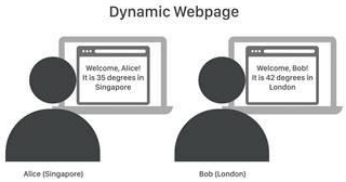


Рисунок Е.3 — Слайд 3

Поняття та принципи кешування

- «Вебкеш» (або «кеш HTTP») – інформаційна технологія для тимчасового зберігання (кешування) вебдокументів і медіа контенту задля зменшення серверних затримок.
- Основною перевагою кешування є пришвидшення роботи веб-сторінок. А більш швидкі веб-сторінки покращують якість користування, а це означає, що відвідувачі веб-сайту будуть щасливішими. Численні дослідження показали, що користувачі відвідують більше сторінок веб-сайту, коли він завантажується швидше

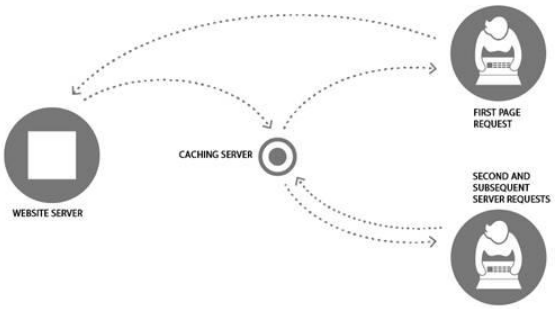


Схема роботи кешу, налаштованого на сервері веб-сайту

Рисунок Е.4 — Слайд 4

Кешування динамічного контенту

- Динамічні веб-сторінки не зберігаються у вигляді статичних файлів HTML. Натомість сценарії на стороні сервера генерують HTML-файл у відповідь на деякі події, або характеристики певного користувача. Оскільки динамічний вміст створюється на стороні сервера, він зазвичай подається з вихідних серверів, а не з кешу.
- Довгий час динамічний вміст вважався некешованим. Але нові технології, такі як мережі доставки контенту, Edge Side Includes (ESI) та ін. дозволяють веб-сайтам подавати динамічний вміст із кешу, значно скорочуючи затримку, зберігаючи при цьому взаємодію з користувачами.
- Проте, велика частина динамічного контенту не така вже й непостійна. Список активних користувачів, наприклад, має час життя 10-20 секунд. Сторінки з графіками або новинна стрічка також може деякий час бути закешованою. А персоналізовані сторінки за такими параметрами, як вік, стать, чи локація користувача можуть мати декілька закешованих версій для різних кластерів користувачів. Тому є сенс кешування динамічного контенту, при правильному вибірковому налаштуванні параметрів такого кешування.

Рисунок Е.5 — Слайд 5

Загальна схема програмного засобу для кешування динамічного вмісту веб-сторінок

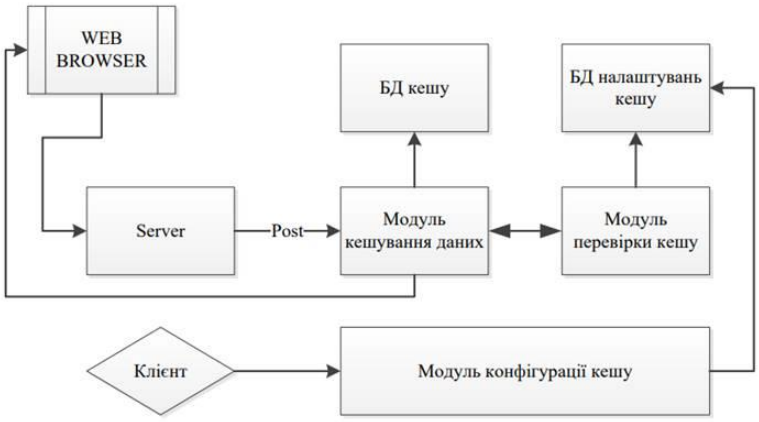


Рисунок Е.6 — Слайд 6

Рисунок Е.7 — Слайд 7

Рисунок Е.8 — Слайд 8

Рисунок Е.9 — Слайд 9

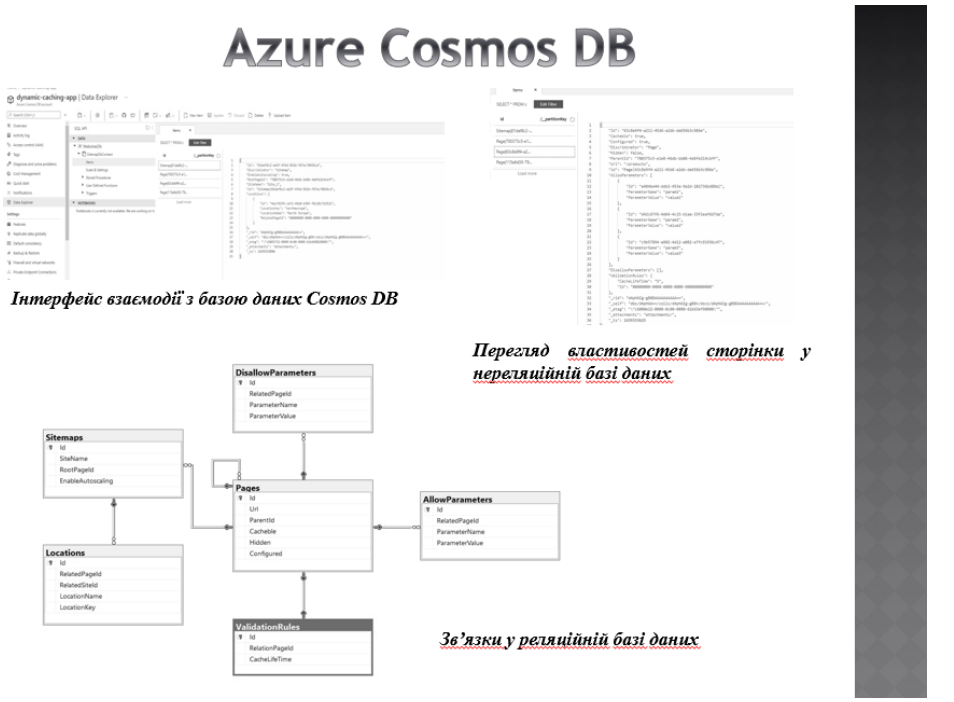


Рисунок Е.10 — Слайд 10

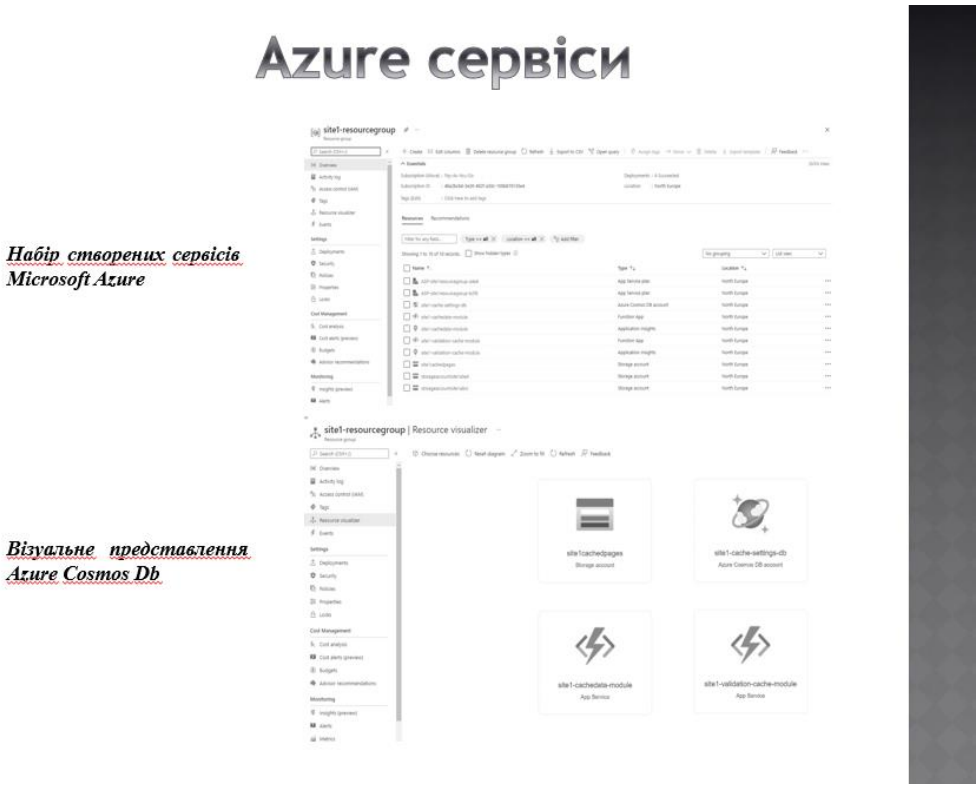


Рисунок Е.11 — Слайд 11

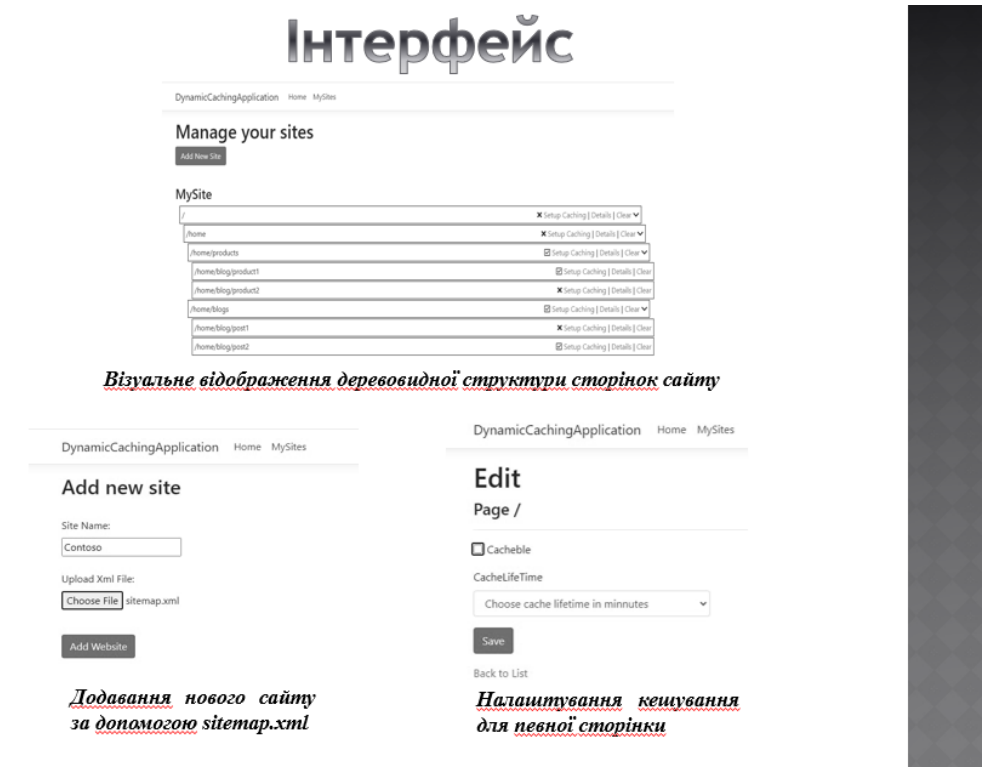
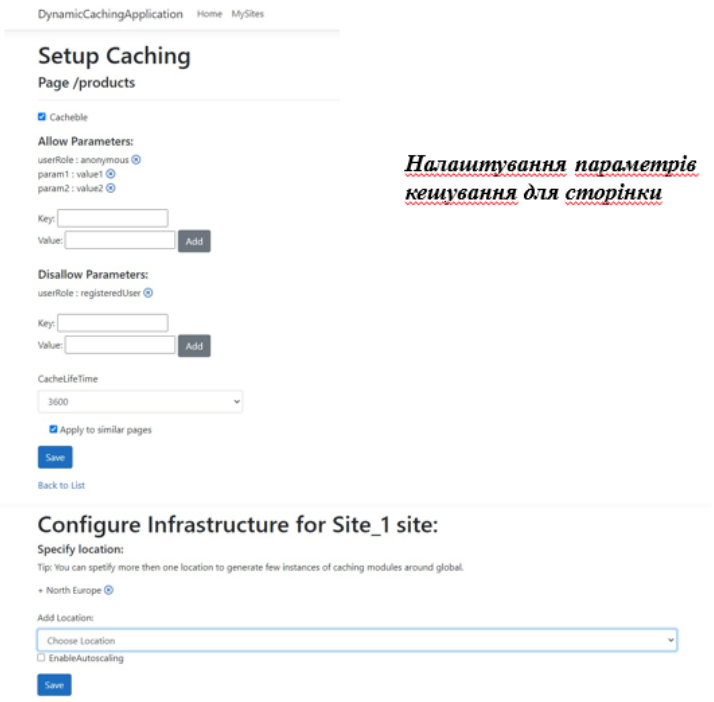


Рисунок Е.12 — Слайд 12

Інтерфейс



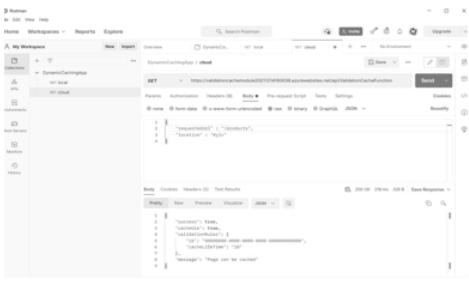
Налаштування параметрів кешування для сторінки

Налаштування хмарної інфраструктури Azure

Рисунок Е.13 — Слайд 13

Інтерфейс

Вигляд відповіді модулю перевірки кешу на запит, за яким можна кешувати сторінку



Вигляд відповіді модулю перевірки кешу на запит, за яким не можна кешувати сторінку

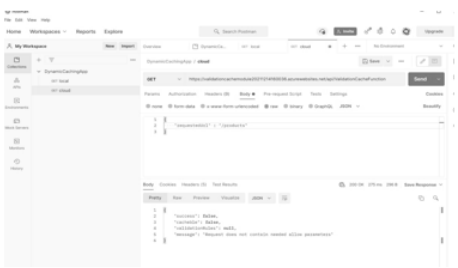


Рисунок Е.14 — Слайд 14

Висновки

Отже, в магістерській кваліфікаційній роботі здійснювалась розробка програмного засобу безпечного конфігурування кешування динамічного вмісту Веб-сторінок.

Розроблений підхід щодо прискорення відображення динамічного контенту відбувається за рахунок створення власної розподіленої системи для налаштування кешування динамічного вмісту веб-сторінок, на основі хмарних технологій.

Дана система дозволяє безпечно кешувати HTML сторінки, що містять динамічний контетн, що значно знижує навантаження на вихідний сервер, в наслідок чого зменшується вартість хостингу веб-сайтів.

За рахунок використання хмарних технологій, розроблений додаток є більш гнучким, легко масштабованим та надійним, внаслідок чого витрати на обслуговування додатку є також значно зниженими.

Дякую за увагу!

Рисунок Е.15 — Слайд 15

ДОДАТОК Ж

Протокол перевірки навчальної (кваліфікаційної) роботи

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи Програмний засіб безпечного конфігурування кешування динамічного вмісту веб-сторінок

Тип роботи: магістерська кваліфікаційна робота

(кваліфікаційна роботи, курсовий проект (робота), реферат, аналітичний огляд, інше (зазначити))

Підрозділ кафедра обчислювальної техніки

(кафедра, факультет (інститут), навчальна група)

Науковий керівник Азарова А. О., к.т.н., доц. каф. ОТ

(прізвище, ініціали, посада)

Показники звіту подібності

Plagiat.pl (StrikePlagiarism)		Unicheck	
КП1			
КП2		Оригінальність	97,4
Тривога/Білі знаки	/	Схожість	2,6

Аналіз звіту подібності (відмінити подібне)

- ✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.
- □ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності і відсутності самостійності її автора. Робот направити на доопрацювання.
- □ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Заявляю, що ознайомлений(-на) з повним звітом подібності, який був згенерований Системою щодо роботи (додається)

Автор _____ Курко В.С.

(підпис) (прізвище, ініціали)

Опис прийнятого рішення

Ступінь оригінальності роботи відповідає вимогам, що висуваються до МКР

Особа, відповідальна за перевірку_____ Захарченко С.М.

(підпис) (прізвище, ініціали)

Експерт _____

(за потреби) (підпис) (прізвище, ініціали)