Вінницький національний технічний університет
(повне найменування вищого навчального закладу)
Факультет інформаційних технологій та комп'ютерної інженерії
(повне найменування інституту)
Кафедра обчислювальної техніки
(повна назва кафедри)

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Методи та програмний засіб комплексного кешування даних у високонавантажених інформаційних системах»

Виконав: студент 2-го курсу, групи 1КІ-20м
спеціальності 123 – Комп'ютерна інженерія
(шифр і назва напряму підготовки, спеціальності)
Горбачов Г. О.
(прізвище та ініціали)
Керівник: к.т.н., доцент каф. ОТ
Ткаченко О. М.
(прізвище та ініціали)
«____» _____ 2021 р.
Опонент: к.т.н., професор каф. ЗІ
Кондратенко Н. Р.
(прізвище та ініціали)
«____» _____ 2021 р.

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

Вінниця ВНТУ 2021

АНОТАЦІЯ

УДК 004.4

Горбачов Г. О. Методи та програмний засіб комплексного кешування даних у високонавантажених інформаційних системах. Магістерська кваліфікаційна робота зі спеціальності 123 — комп'ютерна інженерія, освітня програма — комп'ютерна інженерія. Вінниця: ВНТУ, 2021. 118 с.

На укр. мові. Бібліогр.: 28 назв; рис.: 21; 13 табл.

В даній магістерській дипломній роботі було досліджено та проаналізовано існуючі методи кешування даних у високонавантажених інформаційних системах. У теоретичній частині розглянуто загальну концепцію кешування даних , проблематику яку дана концепція має вирішувати, розглянуті основні методи кешування даних, їх переваги, недоліки та специфіка використання. Розроблено удосконалений метод кешування даних у високонавантажених інформаційних системах. Розроблено інформаційну систему та модуль кешування даних. Було проведено тестування інформаційної системи та проаналізовано вплив модуля кешування на продуктивність системи.

Інформаційна система та модуль кешування даних розроблені для використання в операційних системах Windows, Linux, Mac OS на платформі .NET Core за допомогою середовища розробки Visual Studio 2019 та мови програмування C#.

Ключові слова: інформаційна система, мультипоточність, кешування, кеш, http запит, асимптотична складність, шар доступу до бази даних, ASP.NET Core, C#, об’єктно-реляційне перетворення, тест, об'єкт передачі даних, хешування, фреймворк.

ABSTRACT

UDC 004.4

Hennadii H. O. Methods and software for complex data caching in highly loaded information systems. Master's thesis in the specialty 123 - computer engineering, educational program - computer engineering. Vinnytsia: VNTU, 2021. 118 p.

In Ukrainian language. Bibliographer.: 28 titles; fig .: 21; 13 tab.

In the master's thesis the existing methods of data caching in highly loaded information systems were investigated and analyzed. The theoretical part considers the general concept of data caching, the problems that this concept should solve, the main methods of data caching, their advantages, disadvantages and specifics of use. An improved method of data caching in highly loaded information systems has been developed. An information system and a data caching module have been developed. The information system was tested and the impact of the caching module on system performance was analyzed.

The information system and data caching module are designed for use in Windows, Linux, Mac OS on the .NET Core platform using the Visual Studio 2019 development environment and C # programming language.

Keywords: information system, multithreading, caching, cache, http request, asymptotic complexity, database access layer, ASP.NET Core, C#, object relational mapping, test, data transfer object, hashing, framework.

Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інформаційних технологій та комп’ютерної інженерії

Кафедра обчислювальної техніки

Рівень вищої освіти ІІ-й (магістерський)

Спеціальність — 123 — Комп’ютерна інженерія

Освітньо-професійна програма — Комп'ютерна інженерія

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

«___»_____2021 р.

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ КВАЛІФАКАЦІЙНУ РОБОТУ СТУДЕНТУ

Горбачову Геннадію Олександровичу

1 Тема роботи: «Методи та програмний засіб комплексного кешування даних у високонавантажених інформаційних системах», керівник роботи: Ткаченко Олександр Миколайович, к.т.н., доцент, затверджені наказом вищого навчального закладу від “06” березня 2021 року № 75.

2 Строк подання студентом роботи 21.12.21.

3 Вихідні дані до роботи: проаналізувати існуючі методи та технологічні засоби кешування даних у високонавантажених інформаційних системах, розробити інформаційну систему з модулем кешування даних.

4 Зміст текстової частини: аналіз методів та засобів кешування даних та їх застосування, дослідження можливостей удосконалення методів кешування даних, математичне обґрунтування ефективності системи кешування з різними структурами даних, розробка інформаційної системи та модулю кешування, тестування додатку та дослідження впливу модулю кешування на продуктивність системи, обґрунтування економічної доцільності розробки.

5 Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень): узагальнена схема методу кешування Write-Through, асимптотичні функції складності алгоритмів, інтерактивна документація REST-сервісу Swagger, графіки витрат комп'ютерних ресурсів інформаційною системою, графік розподілення трафіку інформаційної системи.

6 Консультанти розділів роботи приведені в таблиці 1

Таблиця 1 — Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1,2,3,4	Ткаченко О. М., к.т.н., доцент каф. ОТ		
5	Кавецький В. В., к.т.н., доцент каф. ЕПВМ		

7 Дата видачі завдання 12.09.21

Таблиця 2 — Календарний план

№ з/п	Назва етапів виконання бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Постановка задачі роботи	15.09.21	
2	Аналіз методів кешування даних у інформаційних системах	18.10.21	
3	Огляд існуючих програмних засобів кешування даних у інформаційних системах	03.11.21	
4	Математичне обґрунтування ефективності системи кешування з різними структурами даних	28.11.21	
5	Розробка інформаційної системи та модулю кешування	06.12.21	
6	Дослідження впливу модулю кешування на продуктивність системи	08.12.21	
7	Обґрунтування економічної доцільності розробки	12.12.21	
8	Оформлення пояснювальної записки	21.12.21	

Студент _____ Горбачов Г. О.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Ткаченко О. М.
(підпис) (прізвище та ініціали)

Консультант з економічної частини _____ Кавецький В. В.
(підпис) (прізвище та ініціали)

ЗМІСТ

ВСТУП

Актуальність обраної тематики зумовлена швидким зростанням обсягів цифрових даних у інформаційних системах та необхідністю швидкого опрацювання запитів з ефективним використанням ресурсів серверів. Необхідно проаналізувати використання ресурсів інформаційною системою та на основі отриманих результатів сформувати висновки про використання методів кешування даних та їх вплив на швидкодію та надійність роботи інформаційних систем. У наш час кешування широко використовується для підвищення ефективності доступу до даних та підвищення швидкодії запитів у різноманітних інформаційних системах. Однак існує багато методів кешування, кожен з яких має свої недоліки та переваги у порівнянні з іншими, а також може комбінуватися з іншими методами для оптимізації роботи системи.

Метою дослідження магістерської роботи є підвищення швидкості оброблення інформації у високонавантажених системах шляхом розробки та програмної реалізації методу комплексного кешування даних.

Задачі дослідження магістерської роботи:

— здійснити аналіз існуючих методів та засобів кешування даних у високонавантажених інформаційних системах;

— запропонувати кращий підхід для оптимізації використання ресурсів та підвищення швидкодії інформаційної системи;

— створити алгоритм та програму комплексного кешування даних у високонавантажених інформаційних системах.

Об’єкт дослідження магістерської роботи — процес оброблення даних у високонавантажених інформаційних системах.

Предмет дослідження магістерської роботи — методи кешування даних у високонавантажених інформаційних системах.

Методи дослідження магістерської роботи: використовувались методи кешування даних, методи організації баз даних, архітектурні методи до побудови інформаційної системи, методи математичної статистики для виконання аналізу

отриманих результатів. У роботі використано принципи об’єктно-орієнтованого програмування для реалізації запропонованого підходу.

Наукова новизна отриманих результатів магістерської роботи полягає у тому, що вдосконалено метод комплексного кешування даних у високонавантажених інформаційних системах, який відрізняється від відомих комплексним підходом до інвалідації даних, що дозволяє більш ефективно використовувати ресурси інформаційної системи.

Практичне значення одержаних результатів магістерської роботи: розроблено програмний модуль комплексного кешування даних у високонавантажених інформаційних системах.

Апробація. Основні результати роботи повідомлено та затверджено на Всеукраїнській науково-практичній онлайн-конференції «Молодь у науці: дослідження, проблеми, перспективи» (МН-2021) (Вінниця, 05.01.2021 — 14.05.2021) та Всеукраїнській науково-практичній онлайн-конференції «Молодь у науці: дослідження, проблеми, перспективи» (МН-2022) (Вінниця, 11.05.2022 — 13.05.2022).

Публікації. За результатами дослідження опубліковано тези доповіді:

Аналіз методів кешування даних у високонавантажених інформаційних системах [Текст] Г. О. Горбачов «Молодь в науці: дослідження, проблеми, перспективи (МН-2021)» Тез. доп. — Вінниця, 2021. — Режим доступу https://conferences.vntu.edu.ua/index.php/mn/mn2021/paper/view/13144/11053 [1].

Дослідження можливостей удосконалення методів кешування у інформаційних системах [Текст] Г. О. Горбачов «Молодь в науці: дослідження, проблеми, перспективи (МН-2022)» Тез. доп. — Вінниця, 2021. — Режим доступу https://conferences.vntu.edu.ua/index.php/mn/mn2022/paper/view/13144/11053 [2].

1 СУЧАСНІ МЕТОДИ ТА ЗАСОБИ КЕШУВАННЯ ДАНИХ У ІНФОРМАЦІЙНИХ СИСТЕМАХ

1.1 Кешування даних у інформаційних системах

Кешування даних — це механізм високошвидкісного зберігання даних, який зберігає підмножину даних, як правило, тимчасових за своєю природою, тому майбутні запити на ці дані обслуговуються швидше, ніж це можливо, якщо отримати доступ до основного місця зберігання даних. Кешування дозволяє ефективно повторно використовувати раніше отримані або обчислені дані. Дані в кеші зазвичай зберігаються в апаратному забезпеченні швидкого доступу, такому як оперативна пам’ять (пам’ять із довільним доступом), і також можуть використовуватися у кореляції з програмним компонентом. Основна мета кешу — підвищити продуктивність пошуку даних за рахунок зменшення необхідності доступу до базового, повільнішого рівня зберігання.[3] Замінюючи потужність за швидкість, кеш зазвичай зберігає підмножину даних тимчасово, на відміну від баз даних, дані яких зазвичай повні та довговічні. За рахунок цього ефективність використання ресурсів системою підвищується. Основними перевагами використання кешування у інформаційних системах є наступні наслідки.

Покращення продуктивності програми — оскільки оперативна пам’ять працює на порядок швидше, ніж постійна пам’ять (магнітна або SSD), читання даних із кешу в пам’яті відбувається надзвичайно швидко. Значно швидший доступ до даних покращує загальну продуктивність програми.

Зменшення витрат на підтримку баз даних — один екземпляр кешу може забезпечувати сотні тисяч IOPS (операцій введення/виводу в секунду), потенційно замінюючи декілька екземплярів бази даних, знижуючи таким чином загальну вартість. Це особливо важливо, якщо є багато запитів на зчитування даних із бази даних, у таких випадках витрати можуть знизитися у декілька разів.

Зменшення навантаження на бекенд — перенаправляючи значну частину навантаження на читання з серверної бази даних на рівень в пам’яті, кешування може зменшити навантаження на базу даних і захистити її від зниження

продуктивності під навантаженням або навіть від збою під час пікових навантажень.

Передбачувана продуктивність — поширеною проблемою в інформаційних системах є боротьба зі сплесками використання системи. Прикладом можуть бути системи що освітлюють спортивні заходи у день фіналу чемпіонату або веб-сайти електронної комерції в день «чорної п’ятниці» тощо. Збільшення навантаження на базу даних призводить до збільшення затримок на отримання даних, що робить загальну продуктивність системи непередбачуваною. Використовуючи кеш-пам’ять високої пропускної здатності, цю проблему можна пом’якшити.

Вирішення проблеми надмірного блокування ресурсів бази даних — у багатьох системах імовірно, що невелика підмножина даних, наприклад, загальна інформація, профіль знаменитості або популярний продукт, буде фігурувати у запитах до бази даних частіше, ніж інші. Це може призвести до появи гарячих точок у базі даних і може вимагати надмірного надання ресурсів бази даних на основі вимог пропускної здатності для найбільш часто використовуваних даних. Зберігання такої інформації у кеші в пам’яті пом’якшує потребу в надмірному забезпеченні, забезпечуючи швидку та передбачувану продуктивність роботи з даними, до яких найчастіше звертаються.[4]

Збільшення пропускної здатності читання (IOPS) — окрім зменшення затримки на запити зчитування інформації, системи з використанням кешування даних також пропонують набагато вищі показники кількості запитів в секунду (IOPS).

1.2 Аналіз сучасних методів кешування даних

Для різних цілей система кешування може розміщуватися на різних рівнях інформаційної системи, але основною задачею кешування на будь якому рівні є запобігання надлишкових викликів до нижчих рівнів системи. Найнижчим рівнем системи зазвичай є база даних, і у випадку з реляційними базами даних популярні системи управління базами даних такі як MySQL, MsSQL, PostgreSQL вже містять

в собі реалізовані методи кешування запитів на рівні таблиці. У вищих шарах архітектури додатку також є багато реалізацій що пропонуються фреймворками, операційними системами та сторонніми бібліотеками, однак дані засоби зазвичай є лише базою для побудови системи кешування що відповідає вимогам інформаційної системи. Окрім того налаштування та використання систем кешування вимагають глибоких знань теми, адже при неправильному використанні кешування може навпаки знизити ефективність роботи системи у декілька разів або призвести до різноманітних програмних помилок.[5] Основними методами кешування у інформаційних системах є наступні:

Cache-Aside — метод кешування що користується найбільшою популярністю, основною ідеєю якого є розміщення системи кешування окремо від сервера, подібно базі даних. Система кешування такого типу постійно оновлює кеш через додаток асинхронно. При необхідності доступу до даних додаток перевіряє, чи існують дані в кеші, і у випадку якщо вони записані зчитує їх, у іншому випадку зчитування відбувається напряму з бази даних. Системи кешування даного типу добре підходять для інформаційних систем що працюють з великою кількістю даних та високими навантаженнями. Така архітектура досить стійка до збоїв кешу, адже у випадку неполадки інформаційна система може звертатися до бази даних. Ще однією перевагою даного підходу є те що модель даних системи кешу може відрізнятись від моделі даних у базі даних [6]. Наприклад, відповідь, сформована в результаті декількох запитів, може зберігатись по спеціальному ідентифікатору запиту. Однак у даного підходу є свої недоліки, через те що система кешування працює незалежно від бази даних інформація у них з часом може відрізнятися. Такий метод кешування дуже простий у реалізації, але складний в використанні та управлінні. Для оновлення даних у кеші доводиться проводити інвалідацію даних щоразу при оновленні даних у базі даних або використовувати токени валідності даних (TTL) і продовжувати обслуговувати застарілі дані до закінчення терміну дії TTL. Схема методу кешування Cache-Aside зображена на рисунку 1.1.

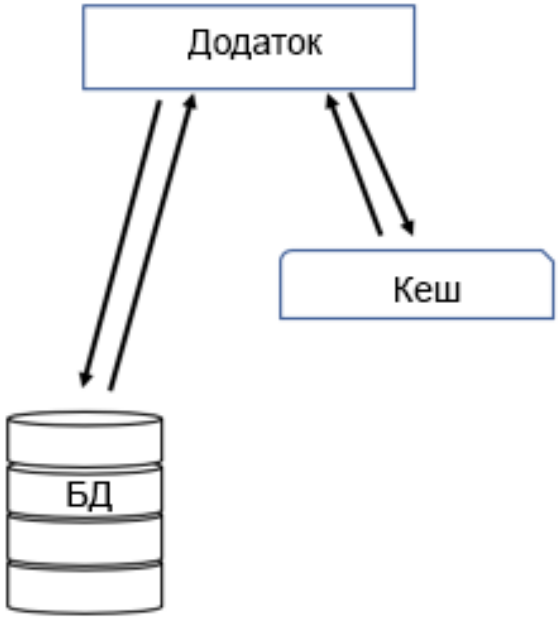

Рисунок 1.1 — Схема методу кешування Cache-Aside

Read-Through — метод кешування де система кешування розміщується між сервером та базою даних, або клієнтським додатком та сервером. При використанні кешу такого типу вся інформація що завантажується з бази даних або сервера зберігається в кеші. Таким чином інформація що повертається у такій системі завжди надсилається з кешу, а сам кеш там база даних тісно пов’язані. Такий тип кешування добре підходить у випадках коли дані не змінюються при цьому зчитуються багато раз, як це відбувається наприклад з архівами або новинами. На відміну від методу Cache-Aside у даній реалізації кешу модель даних що зберігається у кеші не може відрізнятися від моделі даних бази даних. Недоліком такого методу є те що при першому зчитуванні інформації швидкість запиту тільки впаде. Окрім того у даній реалізації також присутня проблема свіжості даних, яка може вирішуватися за допомогою використання токенів валідності даних або доповненням реалізації іншими методами кешування даних.

Write-Through — метод кешування що часто використовується для доповнення методу Read-Through та гарантує актуальність даних у кеші. Сам по собі він лише збільшує затримку перед записом інформації у базу даних, однак при

використанні разом з методом Read-Through ми отримуємо всі переваги при зчитуванні інформації а також узгоджуємо дані між базою даних та кешем. Такий метод кешування є досить ресурсоємні і його неправильне використання є особливо небезпечним. Write-Through метод варто застосовувати лише у системах де даних небагато та вони частіше зчитуються ніж оновлюються. Схема методу кешування Write-Through зображена на рисунку 1.2.

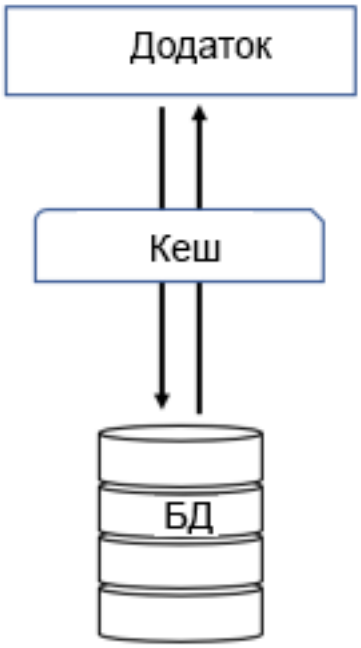

Рисунок 1.2 — Схема методу кешування Write-Through

Write-Back — ще один метод кешування що розміщується між базою даних та сервером. При використанні даного методу кешування інформація завжди спочатку записується в кеш і лише тоді система кешування записує дані у базу даних. Write-Back стійкий до збоїв у роботі бази даних і може сам якийсь час слугувати у якості бази даних. Окрім того за допомогою даного методу можливо буферизувати дані перед записом у базу даних, таким чином захищаючи базу даних від перенавантаження під час пікових годин навантаження на систему. Також значною перевагою даної реалізації є те що дані у кеші завжди актуальні. Однак при використанні даного типу кешування система дуже залежна від кешу, і при

виході кешу із ладу дані можуть не потрапити у базу даних або може постраждати цілісність даних. [7]

Також у багатопоточних та розподілених інформаційних системах потрібно враховувати фактор конкурентного доступу до даних у кеші. Існує декілька стратегій вирішення даної проблеми, вибір яких залежить від методу кешування та вимог щодо актуальності даних.

Найпростіша стратегія доступу до кешованих даних у багатопоточних та розподілених інформаційних системах називається Read-Only. При використанні даної стратегії кешуються лише запити що не змінюють стан записів у базі даних. Дана стратегія може використовується у методах кешування Read-Through та Cache-Aside. Дана стратегія легко реалізовується та має кращу продуктивність в порівнянні з іншими стратегіями, однак її варто використовувати лише тоді коли робота ведеться з даними які часто зчитуються та рідко змінюються. Якщо дані змінюються необхідно час від часу очищати кеш щоб актуалізувати дані у ньому.

Коли дані у кеші змінюються необхідно використовувати стратегію Read-Write, що набагато складніше, адже у багатопоточних системах важливо підтримувати цілісність даних при одночасному доступі до об’єкта. Дана стратегія може використовуватися у методах кешування Write-Through та Write-Back. Синхронізація доступу до об’єктів реалізується блокуванням потоків що одночасно взаємодіють з об’єктом, тому дана стратегія має нижчу продуктивність ніж Read-Only.

При проектуванні системи кешування потрібно обирати метод та стратегію конкурентного доступу базуючись на потребах інформаційної системи. В залежності від типів даних та навантажень на систему найкращим вибором можуть бути різні методи кешування або їх комбінації.

1.3 Аналіз сучасних засобів кешування даних

Існує багато фреймворків що можуть бути інтегровані у інформаційну систему і використовуватися як готові засоби кешування даних або слугувати

базою для побудови власних засобів кешування даних. Також для багатьох мов програмування реалізовані спеціальні бібліотеки що надають базовий функціонал кешування даних. Розглянемо найпопулярніші із них.

Одним із найпопулярніших засобів кешування є Memcached. Memcached це безкоштовний програмний засіб з відкритим кодом. Для роботи з Memcached використовує протокол TCP, тому при необхідності система може розміщуватися на окремому сервері, або може бути розподілена між кількома серверами, підсумовуючи велику хеш-таблицю для зберігання даних. [8] Коли місце закінчується, старі дані перезаписуються, тому Memcached може використовуватись лише як тимчасовий (незберігаючий) кеш, що означає, що необхідних даних там може все ще не бути. Memcached завоював свою популярність ще дуже давно, коли у нього фактично не було конкурентів. Основним недоліком даного кешу є те що він працює лише з рядковими змінними, і у випадку коли нам необхідно зберегти у кеш великий об’єкт багато ресурсів витрачається на серіалізацію та десеріалізацію об’єкта. Тому Memcached погано підходить для кешування складних об’єктів і краще його використовувати для зберігання згенерованих http сторінок. Даний програмний засіб дуже легко конфігурується і може використовуватися для швидкого підвищення продуктивності інформаційної системи.

Альтернативою є програмний засіб Redis, за своєю ідеєю подібний до Memcached але розроблений враховуючи недоліки свого предка. Redis дуже потужний інструмент що класифікується як сховище структурованих даних, і може використовуватися не тільки у якості системи кешування, а і як незалежна NoSQL база даних або посередник повідомлень у системах що використовують технологію Message Queue. Хоча Redis і може використовуватися як автономна база даних, дані зазвичай зберігаються в оперативній пам’яті, що дозволяє виконувати операції над ними надзвичайно швидко. Redis був створений Сальваторе Санфіліппо в 2009 році, і сьогодні Санфіліппо залишається провідним розробником проекту. Redis іноді описують як покращений Memcached, що не дивно, враховуючи, що частини Redis були створені базуючись на досвіді Memcached [9]. Redis має більше

можливостей, ніж Memcached, і, таким чином, є більш потужним і гнучким. Як Memcached, так і Redis, які використовуються багатьма компаніями і в незліченних критично важливих виробничих середовищах, підтримуються клієнтськими бібліотеками на всіх можливих мовах програмування, і вони включені в безліч пакетів для розробників.

Також Redis додатково пропонує можливість збереження даних на постійну пам’ять, призначену для завантаження кешу після запланованого вимкнення або збою. Хоча зазвичай дані в кеші розглядаються як нестабільні та тимчасові, дані, що зберігаються на диску, можуть бути дуже цінними в сценаріях кешування. Наявність даних кешу, доступних для завантаження відразу після перезапуску, дозволяє значно швидше «розігрівати» кеш і знімає навантаження, пов’язане з повторним заповненням і перерахунком вмісту кешу з основного сховища даних.

При використанні ORM бібліотек для доступу до бази даних часто виникає проблема низької продуктивності доступу до бази даних при високих навантаженнях інформаційної системи. Фреймворк Hibernate якраз орієнтований для вирішення даної проблеми, і часто використовується у серверних додатках написаних мовою Java. Даний фреймворк функціонує як окремий шар між додатком та базою даних і реалізовує метод кешування Read-Through. Також функціонал фреймворку дозволяє сконфігурувати кешування методом Write-Through. Використовуючи Hibernate, дані завантажені з БД одразу записуються у кеш. Це допомагає зменшити трафік між додатком та БД, оскільки більшість випадків отримання даних потрапляє в кеш. Щоразу коли потрібні нові дані, буде зроблено запит до БД. Оскільки час, необхідний для доступу до БД, більший у порівнянні з часом доступу до кешу, трафік буде зменшений між програмою та БД. Важливо пам’ятати що в кеші зберігаються лише дані, пов'язані з поточною запущеною програмою. Отже, кеш повинен бути очищений щоразу, коли додаток змінюється. У Hibernate використовується дворівнева модель кешування. Схема дворівневої моделі зображена на рисунку 1.3.

Кеш першого рівня пов’язаний з об’єктом сесії і є обов’язковим, тобто через нього проходять всі запити що надходять на сервер. Hibernate використовує даний

кеш за замовчуванням. При необхідності оновити об’єкт що лежить у базі даних декілька разів Hibernate затримає виконання даних операцій наскільки це можливо і оновить всі зміни однією транзакцією, що суттєво знизить кількість запитів оновлення даних, і як наслідок знизить навантаження на базу даних.

Кеш другого рівня пов’язаний з об’єктом Session Factory, і є опціональним. Він доступний для використання між усіма сесіями що створюються конкретним об’єктом Session Factory. Кеш другого рівня може бути окремо сконфігурований для кожного класу та колекції, і зазвичай відповідає за кешування між сеансами. Це означає що коли екземпляр Session Factory закривається всі кешовані дані пов’язані з ним очищуються. [10]

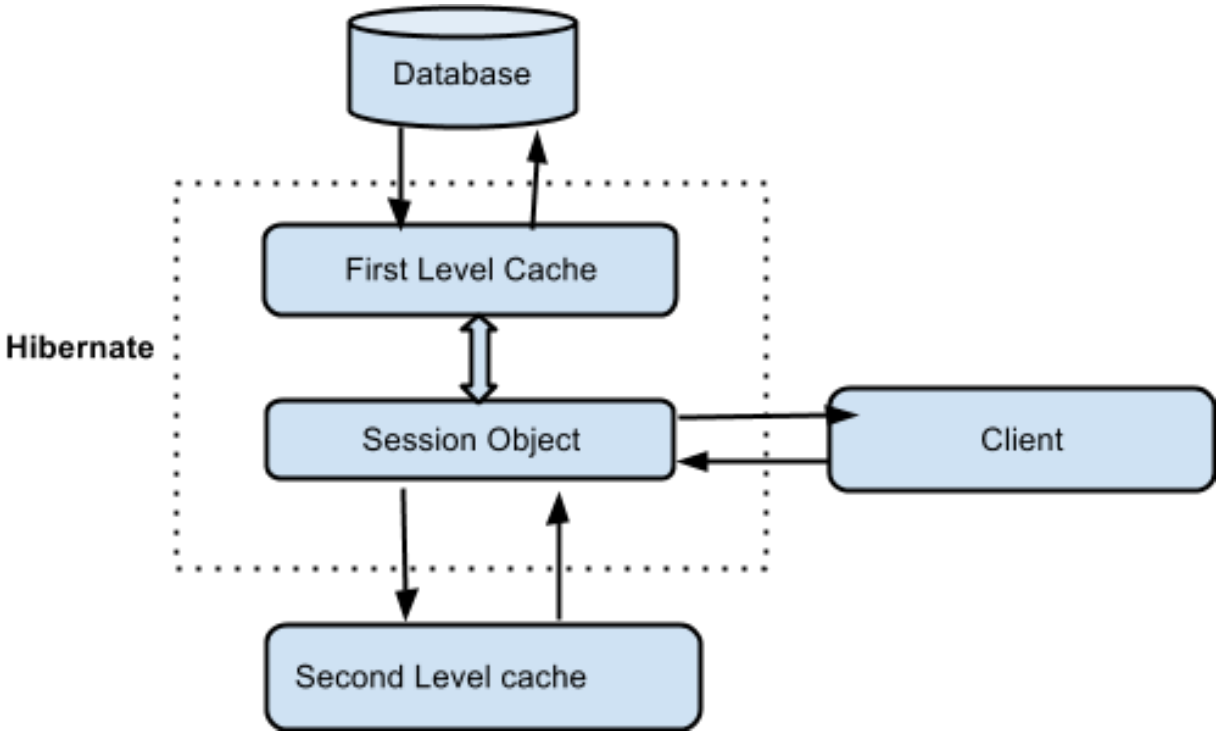

Рисунок 1.3 — Дворівнева модель кешування Hibernate

Окрім цього Hibernate може розширюватися іншими бібліотеками кешування або власною імплементацією системи кешування. Однак даний засіб кешування може бути застосований лише до інформаційних систем написаних мовою Java, що обмежує його використання.

В сучасних мовах програмування що використовуються для написання серверної частини інформаційних систем наявні стандартні бібліотеки, що містять

базові класи з фунціоналом кешування даних, що можуть використовуватися як простий засіб кешування або база для власної імплементації кешу. Такі бібліотеки ще називають in memory cache, вони імплементується у різних мовах програмування по різному, однак основна ідея залишається однією. При використанні даного засобу фактично розробляється окремий модуль додатку, що використовується у якості шару між додатком та базою даних для швидкої доставки відповідей шляхом зберігання даних із попередніх запитів або копіювання безпосередньо з баз даних. Кеш у пам’яті усуває затримки продуктивності, коли програма, побудована на базі даних на диску, повинна отримати дані з диска перед обробкою. У даного засобу є певні обмеження у розширені системи коли ресурсів однієї машини не вистачає, і доводиться використовувати кластери серверів. Однак для невеликих інформаційних систем та учбових цілей даний засіб кешування підходить найкраще, адже не вимагає встановлення додаткових додатків та платних сертифікатів продукції, що часто використовуються у складніших системах кешування.

1.4 Дослідження можливостей застосування та ефективності використання методів кешування даних в залежності від інформаційної системи

Підхід до проектування інформаційної системи обирається базуючись на вимогах до сервісу. В залежності від типу архітектури обирається метод кешування та його тип, адже не існує універсального рішення яке було б для всіх інформаційних систем. Ключовим пунктом у виборі архітектури є наявність графічного інтерфейсу, та технічні вимоги до нього. Також потрібно враховувати очікуване навантаження на сервіс, кількість даних, над якими він оперує, ймовірність розширення сервісу у майбутньому, використання кластерів серверів для розподілених систем та необхідність виконання програми у різних операційних системах[11]. Архітектурні стилі визначають сукупність взаємопов'язаних систем, які мають спільні структурні та семантичні властивості. Таким чином метою використання архітектурних стилів є використання спільної структури для всіх

компонентів системи.

Використовуючи один з класичних архітектурних стилів, інформаційна система стає зрозумілішою для інших спеціалістів, що допомагає при розширенні команди розробників та дозволяє полегшити інтеграції з іншими сервісами. Ще однією перевагою використання архітектурних стилів є можливість повторного використання коду та дизайну, а також підвищення сумісності програмного забезпечення. Отже розглянемо основні архітектурні стилі, та їх сумісність з методами кешування.

Архітектура потоків даних — архітектурний стиль що використовується в основному в інформаційних системах, що отримують дані, проводять над ними певні операції та обчислення після чого повертають опрацьовані дані. У даній архітектурі кожен модуль, відомий також як фільтр, трансформує дані та пересилає їх наступному фільтру для подальшої обробки. Кожен модуль працює як незалежна сутність, і має мінімум зв’язків з іншими фільтрами, які обробляють дані. Дана архітектура використовується системах обробки сигналів, компіляторах, функціональному програмуванні, паралельному програмуванні та розподілених обчислювальних системах, однак вона незручна для використання кінцевим користувачем. При використанні даної архітектури у системі часто відсутній графічний інтерфейс, а керується забезпечується за допомогою текстових команд[12]. При використанні даної архітектури доцільно використовувати методи кешування Read-Through та Write-Through, адже їх концепція корелює з загальною ідеєю архітектурного стилю, і у таких системах модуль кешування буде представлено як проміжний фільтр в ланцюгу фільтрів системи.

Мікро-сервісна архітектура — стиль що базується на декількох ключових концепціях: сервіси працюють незалежно один від одного, кожен сервіс оперує над власною базою даних. При використанні даного архітектурного стилю легко додаватися нові компоненти, вони можуть використовувати інші технології та зазвичай розміщуються на різних фізичних машинах. Однак недоліками даного стилю є неможливість повторного використання коду, адже кожен сервіс повинен бути незалежним та працювати з власною базою даних. Системи що базуються на

даному архітектурному стилі легко розширяються та можуть витримувати гігантські навантаження, однак вони надзвичайно складні і вимагають велику кількість окремих команд розробників для незалежної роботи над модулями програми [13]. При використанні даної архітектури можливі багато сценаріїв використання кешування, як наприклад використання внутрішнього кешу окремого сервісу базуючись на методах Write-Through або Write-Back, так і використання глобального кешу що зберігає результати сумісної роботи декількох сервісів базуючись на методиці Cache-Aside. Зазвичай саме з цим методом кешування і асоціюються інформаційні системи побудовані на мікро-сервісній архітектурі.

Клієнт-серверна архітектура — стиль що базується на чіткому розподілі сервісу на частину відповідальну за представлення та частину відповідальну за логіку системи. Взаємодіють компоненти за допомогою спеціальних протоколів, які визначають сукупність правил, які виконуються під час взаємодії між компонентами. Даний архітектурний стиль надає можливість легко розширювати функціонал інформаційної системи, а також повторно використовувати код та модулі програми. Окрім того компоненти можуть замінятися на аналогічні, тобто немає жорсткої залежності від графічного інтерфейсу що використовується, або бази даних, вони можуть бути легко замінені коли це потрібно. Використовуючи клієнт-серверну архітектуру сервіс поділяється на серверну та клієнтську частини, де клієнтська частина імплементує представлення для зручного використання сервісу користувачами, а серверна частина оперує над даними та містить бізнес логіку. Такий підхід дає можливість чітко розмежувати відповідальності між компонентами, що робить їх незалежними один від одного[14]. Як правило веб-сервіси використовуючи інтернет протокол http або протоколи вищих рівнів. При використанні клієнт-серверної архітектури немає обмежень стосовно використання методів кешування. Часто в таких системах у серверній частині інформаційної системи використовуються декілька рівнів кешу, наприклад для кешування готових відповідей клієнту та кешування частини даних із бази даних.

Також в контексті архітектурних стилів інформаційних систем існують дві

концепції: stateful та stateless. Stateful — це підхід до побудови системи зі збереженням стану. Його суть полягає в тому, щоб враховувати при обчисленнях стан об’єкта у часі і в залежності від нього та вхідних даних змінювати алгоритм виконання програми. Прикладом даної концепції може слугувати традиційний FTP-сервер. Змінюючи стан користувача, наприклад запис про останній переглянутий товар, зберігається на сервері як стан клієнта. Таким чином кожна зміна, що приходить на сервер, реєструється як зміна стану. Якщо користувач відключається, його стан змінюється на від’єднаний. При використанні даного підходу можуть виникати деякі труднощі. У випадку збоїв системи з’являється велика кількість незакінчених транзакцій та сесій, які не можуть автоматично виправлятися, адже стан об’єкту у ненормальному стані. Також може бути незрозумілим як довго інформаційна система має зберігати сеанс відкритим, або як дізнатися чи клієнт від’єднаний. Існують обхідні шляхи для вирішення даних проблем, однак зазвичай збереження стану корисне тільки тоді коли самі функції залежать від стану[15]. Окрім того користувачі зазвичай можуть взаємодіяти з інформаційною системою різними способами, і тому збереження стану серверної частини не залежить від програми-клієнта, тоді і в збереженні стану немає необхідності.

Stateless підхід — є альтернативним підходом який протиставляють stateful. Він базується на відсутності будь-якого станів сервера, що дозволяє уникати проблеми характерні для stateful підходу. Це дає можливість обробляти запити базуючись лише на вхідних даних та наборі інструкцій по їх обробці, що також є частиною запиту. Концепція stateless є фундаментальною при розробці сучасних інформаційних систем. У сервісах що використовують дану концепцію зазвичай застосовуються методи кешування тільки для зчитування інформації, адже у іншому випадку порушується сама ідея підходу.

Базуючись на даних концепціях було створено дві реалізації: SOAP та REST. SOAP це стандартизований протокол, що передає повідомлення з використанням протоколів HTTP, SMTP, TCP, UDP та інших. Специфікації SOAP є офіційними веб-стандартами що розробляються та підтримуються Консорціумом World Wide

Web. Перевагами протоколу SOAP є чіткі правила використання, розширений функціонал безпеки, включаючи базову авторизацію та ACID принципи. З іншої сторони використовуючи SOAP сильно ускладняється структура інформаційної системи, що в свою чергу робить важчим розширення сервісу. Концепція REST була створена для вирішення проблем протоколу SOAP, відмовляючись від строгих специфікацій інформаційна система стає більш гнучкою, адже розробники можуть створювати власну реалізацію низькорівневих деталей системи, або використовувати стороні бібліотеки [16]. Вся логіка системи будується на ідеї, що всі необхідні дані та конфігурації знаходяться безпосередньо у запиті.

Отже у першому розділі було розглянуто загальну концепцію кешування даних проблематику яку дана концепція має вирішувати, були розглянуті основні методи кешування даних, їх переваги, недоліки та специфіку використання. Також були розглянуті та проаналізовані популярні фреймворки та платформи для побудови систем кешування даних, освітлені основні архітектурні підходи до побудови інформаційної системи, а також досліджені можливості застосування та ефективність використання методів кешування даних в залежності від архітектури інформаційної системи. За результатами досліджень я прийшов до висновку що у різних інформаційних системах можуть застосовуватися різні методи кешування, а в залежності від архітектури та вимог системи обирається оптимальний метод та засіб кешування.

2 УДОСКОНАЛЕННЯ МЕТОДУ КЕШУВАННЯ ДАНИХ У ВИСОКОНАВАНТАЖЕНІЙ ІНФОРМАЦІЙНІЙ СИСТЕМІ

2.1 Дослідження можливостей удосконалення методу кешування

У сучасних інформаційних системах для підвищення продуктивності та швидкості виконання запитів часто використовується метод Read-Through. Визначимо критерії ефективності кешування даних, та розглянемо можливі удосконалення методу.

Одним з основних критеріїв ефективності кешування є відношення кількісті оброблених запитів за одиницю часу з використанням системи кешування та кількості оброблених запитів за одиницю часу без використання системи кешування:

$$k_e = \frac{n_1 \times t_2}{n_2 \times t_1} \quad (2.1)$$

де n_1, n_2 — кількість запитів до інформаційної системи за сесію;

t_1, t_2 — тривалість сесії.

Коефіцієнт ефективності системи кешування може відрізнятися в межах однієї системи в залежності від загального навантаження системи, вільної оперативної пам’яті, часу затримки мережі, вибірки даних. Для отримання репрезентативних результатів ефективності системи кешування необхідно провести дослідження ефективності декілька разів в умовах максимально наближених до реальних, включаючи особливості нерівномірного розподілення трафіку.

$$k_\text{c} = \frac{\sum_{i=1}^{n} k_n}{n} \quad (2.1)$$

В реальних системах даний коефіцієнт змінюється в залежності від дня тижня та часу, адже коефіцієнт ефективності кешування напряму залежить від навантаження на інформаційну систему. Для порівняння коефіцієнтів ефективності декількох методів кешування потрібно використовувати стабільну систему, та

виконувати заміри декілька разів змінюючи кількість запитів та вибірку даних. Для отримання результатів наближених до реальних вибірка даних повинна бути рандомізованою. В кінцевому результаті ефективнішим буде метод кешування при використанні якого було здійснено найменша кількість викликів до бази даних, а пошук результатів запиту потребуватиме мінімум ресурсів.

Іншим важливим критерієм ефективності кешування даних є затрати оперативної пам'яті. У багатьох інформаційних системах розмір бази даних може сягати десятків терабайт. Очевидно що неможливо зберігати у оперативній пам'яті таку кількість даних, тому потрібно фільтрувати які дані повинні затримуватися в кеші, а які ні. Зазвичай дана задача вирішується простим методом зберігання даних у кеші на певний час T, після якого дані видаляються з кешу звільнюючи місце для нових даних. Таким чином дані які запитувалися одноразово будуть видалені і не займатимусь місце в оперативній пам'яті, проте у випадку якщо дані запитуються часто виникне необхідність доступатися до них через базу даних. Це призведе до додаткових витрат, адже замість того щоб один раз звернутися до бази даних доведеться періодично завантажувати ці дані з бази даних, де період дорівнюватиме T.

Для вирішення даної проблеми можна замінити константний час, після якого дані у кеші видаляються, на період, який буде оновлюватися кожен раз коли ці дані використовуються. Таким чином у кеші не будуть зберігатися дані які запитуються одноразово, а для збереження даних що використовуються часто буде виконуватися лише один виклик бази даних [17]. Однак при використанні такого підходу можливі ситуації коли дані знаходяться у кеші дуже довго залишаючись неактуальними. Наприклад за час поки дані знаходяться у кеші аналогічні дані у базі даних могли бути змінені, або навіть видалені. Метод кешування Read-Through не передбачає механізму який просто вирішував дану проблему. Зазвичай для пом'якшення даної проблеми використовують комбінацію двох підходів, де встановлюють період T_1, після якого дані видаляються з кешу і тим самим при наступному виклику актуалізуються, та період T_2, який оновлюється кожен раз коли дані фігурують у запиті, тим самим не створюючи надмірне навантаження на

базу даних. При цьому важливо виконувати умову $T_1 > T_2$, інакше сенс використання періоду T_2 втрачається.

Також для вирішення проблеми актуальності даних має сенс удосконалити метод кешування шляхом інвалідації кешованих даних при виконанні операцій над ними. Такими операціями є змінення вмісту даних, або їх видалення. Дану операцію потрібно виконати одразу після внесення змін, тоді при наступному запиті дані будуть отримані з бази даних та знову запишуться у кеш. Таким чином можна гарантувати що у кеші завжди будуть знаходитися актуальні дані.

Однак у високонавантажених інформаційних системах, що оперують великими вибірками даних, перелічених засобів оптимізації вмісту кешу може бути недостатньо. Кожен окремий запит характеризується часом виконання запиту у базі даних та кількістю виконання аналогічних запитів. Метод періодичної очистки кешу забезпечує можливість видаляти з кешу запити одноразові запити, та запити які виконуються досить рідко. Однак ефективно зберігати дані не тільки тих запитів що використовуються часто, а й дані для витягнення яких витрачається багато ресурсів бази даних. Індикаторами таких запитів перш за все є час їх виконання базою даних, також це можуть бути кількість блокувань всередині реляційних таблиць, навантаження на процесор серверу бази даних тощо. Час виконання запиту, на відміну від метрик навантаження процесору, досить легко виміряти, і водночас дана інформація дає релевантну інформацію щодо ресурсоємність запиту. Отже має сенс зберігати в системі кешування дані, запити до яких виконуються довго, навіть якщо такі запити використовуються рідко. Для цього потрібно додати додаткову метрику до об’єктів кешованих даних, що міститиме час витрачений на виконання відповідного запиту базою даних. Тоді під час періодичного процесу інвалідації кешу, такі дані не будуть видалятися, що додатково підвищить ефективність роботи системи.

При застосуванні вдосконаленого методу кешування даних , для оптимальної роботи інформаційній системи необхідно визначити періоди інвалідації кешу та вагові коефіцієнти запитів. Це можливо виконати під час тестування системи емпіричним шляхом.

2.2 Аналіз структур даних при використанні у вдосконаленому методі кешування даних

Дані що зберігаються у кеші становлять собою колекцію об’єктів. Необхідно проаналізувати структури даних та використати оптимальну для поставленої задачі структуру колекції та об’єктів що в ній зберігаються.

При роботі з кешем часто доводиться виконувати операції над даними що там зберігаються, наприклад пошук, додавання нового об’єкта в кеш або його видалення . Складність виконання даних операцій при використанні різних структур даних відрізняється. В програмуванні для оцінки ефективності алгоритмів та структур даних прийнято використовувати поняття асимптотичної складності алгоритму, яке позначається як $\theta(f)$.

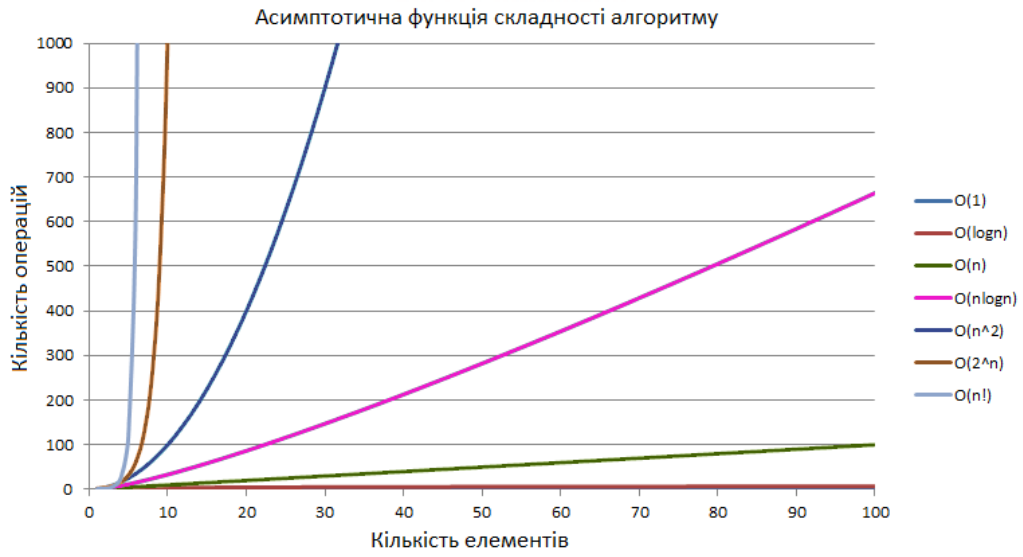

Рисунок 2.1 — Типові асимптотичні функції складності алгоритмів

Коли ми оцінюємо складність, ми говоримо про порядок підрахунку операцій, а не про їх точну кількість. Дані значення корелюють з затратами комп’ютерних ресурсів, таких як CPU, час виконання операцій, об’єм оперативної пам’яті тощо. Для вираження залежності затрат комп’ютерних ресурсів від кількості елементів з якими працює алгоритм використовують асимптотичні функції. Типові асимптотичні функції складності алгоритмів зображені на рисунку 2.1.

Припустимо у кеші знаходиться колекції об’єктів кількістю n. В такому випадку для того щоб знайти серед них об’єкт з певним ім’ям в масиві доведеться пройтися колекцією поки не буде знайдено потрібний об’єкт. Іншими словами виконання алгоритму пошуку потрібного об’єкту у кеші буде коштувати від 1 до n операцій. У випадку з пошуком об’єкту у масиві даних в середньому знадобиться $\frac{n}{2}$ операцій, при цьому константою при оцінці алгоритму можна знехтувати, тому середня складність алгоритму оцінюється $\theta(n)$ [18]. Також під час використанні кеша часто виникатиме необхідність додавати нові об’єкти та видаляти старі.

Таблиця 2.1 — Асимптотичні функції середньої ефективності виконання операцій

Структура даних	**Доступу**	**Пошук**	**Додавання**	**Видалення**
Масив	Θ(1)	Θ(N)	Θ(N)	Θ(N)
Стек	Θ(N)	Θ(N)	Θ(1)	Θ(1)
Черга	Θ(N)	Θ(N)	Θ(1)	Θ(1)
Однозв’язний список	Θ(N)	Θ(N)	Θ(1)	Θ(1)
Двозв’язний список	Θ(N)	Θ(N)	Θ(1)	Θ(1)
Словник	Θ(1)	Θ(1)	Θ(1)	Θ(1)
Бінарне дерево	Θ(log N)	Θ(log N)	Θ(log N)	Θ(log N)

Розглянемо асимптотичні функції складності алгоритмів для необхідних для системи кешування операцій у різних структурах даних. Оскільки в залежності від вибірки даних ефективність виконання алгоритмів змінюється варто оцінювати середню ефективність та ефективність у найгіршому варіанті. В таблиці 2.1. наведено основні структури даних та асимптотичні функції середньої ефективності виконання необхідних операцій при оптимальній вибірці даних. В таблиці 2.2 наведено основні структури даних та асимптотичні функції ефективності виконання необхідних операцій у найгіршому випадку.

Таблиця 2.2 — Асимптотичні функції мінімальної ефективності виконання операцій

Структура даних	**Доступу**	**Пошук**	**Додавання**	**Видалення**
Масив	O(1)	O(N)	O(N)	O(N)
Стек	O(N)	O(N)	O(1)	O(1)
Черга	O(N)	O(N)	O(1)	O(1)
Однозв’язний список	O(N)	O(N)	O(1)	O(1)
Двозв’язний список	O(N)	O(N)	O(1)	O(1)
Словник	O(N)	O(N)	O(N)	O(N)
Бінарне дерево	O(N)	O(N)	O(N)	O(N)

Проаналізувавши таблицю асимптотичних функцій середньої ефективності виконання операцій можна зробити висновок що більшість структур даних мають приблизно однакову ефективність при виконанні операцій. Однак варто також пам’ятати що наприклад масив погано пристосований для роботи з нефіксованою кількістю даних, тобто не може бути динамічно розширеним і тому погано підходить для задачі кешування даних. Такі структури даних як стек та черга можуть легко бути розширені, однак з них важко видаляти неактуальні дані з середини колекції. Однозв’язний список, двозв’язний список, словник та деревовидний список найкраще підходять для поставленої задачі, адже дозволяють швидко та легко додавати нові дані та видаляти неактуальні. Серед них найкращі показники ефективності виконання операцій мають такі структури даних як бінарне дерево та словник. При цьому дані структури даних мають одні з найгірших показників ефективності при роботі з неоптимальним набором даних. Розглянемо детальніше чому так може відбуватись та як можливо підвищити ефективність виконання операцій над даними у кеші.

Для того щоб зрозуміти чому так відбувається, необхідно оглянути внутрішню реалізацію даної структури даних. Кожен елемент даної колекції

складається з ключа та безпосередньо об’єкта який в ній зберігається. При виконанні операцій над колекцією для пошуку відповідних елементів використовується не сам ключ, а його хеш. Після визначення хешу ключа елемента в колекції з N елементів, хеш ключа ділиться з остачею на кількість елементів в колекції.

$$I = \frac{F_{hash}(K)}{N} \tag{2.1}$$

де I — індекс елементу колекції;

F_{hash} — функція хешування об’єкту;

K — ключ елементу колекції;

N — кількість елементів в колекції.

Таким чином для кожного окремого елемента ми отримуємо відповідний індекс значення якого лежать в межах $0 \leq I < N$. В залежності від вибірки даних можуть виникати колізії індексів, тобто коли залишок від ділення з остачею хеш-функції ключа повертає однакові значення для різних об’єктів. В такому випадку новий елемент додається до колекції, його індекс зберігається у кошику, а індекс старого елемента — у його полі next [19]. Фактично створюється однозв’язний список елементів всередині словника. Складність доступу до елементів однозв’язного списку виходячи з таблиці 2.1 дорівнює $\Theta(N)$. Саме тому у випадку коли виникає колізія між всіма елементами колекції складність виконання операцій над словником зростає до $\Theta(N)$, як це вказано у таблиці асимптотичних функцій мінімальної ефективності виконання операцій 2.2. Для того щоб запобігти вирішенню даної проблеми необхідно забезпечити унікальність індексів, для цього необхідно щоб ключі елементів словника були унікальними. Також у деяких випадках використовується метод накладання бітових масок на результат хеш-функції, однак даний метод не може замінити необхідність використання унікальних ключів елементів.

Таким чином словник, за умови використання унікальних ключів, є найкращою структурою даних для збереження кешованих даних. Оскільки

філософія реляційних баз даних вимагає щоб у відношеннях зберігались лише унікальні об’єкти, це не є проблемою, адже у якості ключа може бути використаний ідентифікатор або унікальне поле об’єкта.

Отже у другому розділі було проведене дослідження можливостей удосконалення методу кешування, розроблено та математично обґрунтовано доповнений алгоритм інвалідації даних у кеші для оптимізації витрат ресурсів інформаційної системи. Також були проаналізовані структури даних, що можуть використовуватися для реалізації методу кешування, розглянуті їх переваги, недоліки, асимптотичні функції ефективності виконання алгоритмів та досліджено можливості оптимізації колекції даних у системі кешування. За результатами досліджень для реалізації методу кешування обрано структуру даних словник з оптимізацією вибірки даних для пришвидшення виконання операцій над даними.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДУ КОМЛЕКСНОГО КЕШУВАННЯ ДАНИХ ТА ТЕСТОВОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Вибір технологій

3.1.1 Вибір мови програмування

Для тестування методу кешування даних і високонавантажених інформаційних системах перш за все необхідна сама інформаційна система. Для даної задачі доцільно створити спеціальну інформаційну систему з типовою структурою, та створити штучне навантаження у відповідністю з математичною моделлю описаною в другому розділи. Перш ніж приступити до реалізації інформаційної системи та модуля кешування даних потрібно визначитися з мовою програмування та фреймворком, якщо такий буде використовуватися. Основними мовами програмування що орієнтовані на написання серверної частини інформаційних систем є такі мови програмування як Node.js, Java, C#, PHP та Python. Розглянемо переваги та недоліки даних мов програмування.

Node.js — популярний фреймворк, що базується на java script та часто сприймається як окремий інструмент. Java script здебільшого застосовується у розробці клієнтської веб частини інформаційних систем, однак може використовуватися і для реалізації сервера. Із переваг можна виділити те що писати код java script відносно просто. Серед недоліків можна виділити притаманні для java script проблеми динамічної типізації, проблеми з багатопоточністю, нижчу в порівнянні з аналогами швидкодію та недостатню кількості інформації по використанню технології в інтернеті[20].

Java — є однією з найпопулярніших мов програмування у світі, на ній можуть розроблятися як сервери, так і мобільні додатки. Завдячує Java своєю варіативністю застосування віртуальній машині JVM, яка виконує роль посередника між кодом Java та машинними кодами для керування пристроєм. Це дозволяє запускати код Java незалежно від операційної системи та платформи де він був написаний. Хоч дана мова є популярною серед розробників програмного забезпечення, вона менш зручна для новачків в порівнянні з сучаснішими мовами програмування, оскільки

синтаксис мови є громіздким та вимагає більше коду для створення функціоналу ніж в аналогах. Як наслідок на Java важче створювати програми та підтримувати вже створені бібліотеки[21]. Java знаходиться на ринку дуже давно, і її спільнота є дуже великою, що забезпечує велику кількість корисних бібліотек та полегшує пошуки необхідної інформації під час написання додатку.

PHP — майже 60% серверів що функціонують нині розроблені за допомогою цієї мови програмування. Важливо одразу звернути увагу, що дана мова програмування використовує динамічну типізацію типів, що водночас дозволяє витрачати менше зусиль написання коду та його дизайн, але й може призвести до неочікуваних помилок з якими непросто боротися. Окрім цього, сервери написані за допомогою мови програмування PHP погано масштабуються, тому при написанні сучасних інформаційних систем PHP використовують рідко.

Python — відносно нова мова програмування, що нині стрімко розвивається і вже здобула високу популярність у сфері алгоритмів та нейронних мереж. Як і у випадку з Java, пошук інформації не є проблемою, що робить її використання зручним для початківців [22]. Окрім цього, синтаксис що використовує Python дуже компактний, і еквівалентний код написаний на Python займатиме набагато менший об’єм ніж у С-подібних мовах програмування. Проте варто звернути увагу що Python як і PHP використовує динамічну типізацію об’єктів, що може призводити до тих же проблем, що і у випадку з PHP.

C# — мова програмування створена компанією Microsoft, по своїй концепції схожа на Java. На даний момент є популярною для написання програмного забезпечення під операційну систему Windows та може використовуватися для розробки кросплатформних серверів. На відміну від Java C# продовжує активно розвиватися та постійно покращується, не так давно з’явилася нова платформа що використовує код C# — .NET Core, яка пропонує багато сучасних функцій та можливість виконання коду під різними операційними системами. C# є строго типізованою мовою та має великий вибір фреймворків для створення серверів.

При розробці сучасної інформації системи орієнтованої на високі навантаження варто обирати між Java та C#, як є об’єктно-орієнтованими мовами

зі строгою типізацією, зі всіма випливаючими перевагами, та можуть виконуватися під різними платформами. Між цими двома мовами я надаю перевагу C#, адже ця мова продовжує активно розвиватися, і в останніх версіях отримала значні оптимізації саме для написання веб серверів. Окрім того використовуючи платформу .NET ми зможемо використовувати популярний ORM для роботи з базами даних — Entity Framework Core. За його допомогою можна швидко імплементувати шар доступу до бази даних, який суттєво спростить взаємодію з базою даних, адже робота з записами буде вестися через їх представлення у вигляді колекцій та об’єктів, що в свою чергу дозволить використовувати такі потужні бібліотеки для роботи з колекціями як LINQ.

3.1.2 Вибір фреймворку

Використовуючи мову програмування C# є декілька доступних фреймворків для побудови серверу інформаційної системи. Серед них можна виділити такі фреймворки як WebAPI, WCF, ASP.NET Core та ASP.NET MVC. Розглянемо детальніше кожен фреймворк, його переваги та недоліки.

Web API — фреймворк що пропонує простий функціонал REST сервісу, а саме операції створення, видалення, перегляду та зміни об’єктів. Структура сервісу організована як ланцюг методів що перевіряють дані надіслані в запиті, виконують бізнес логіку та повертають відповідь. Даний фреймворк можна легко комбінувати з будь яким клієнтським додатком, адже він базується на уніфікованих http запитах, отже може викликатись на різних пристроях та платформах.

WCF — фреймворк орієнтований для побудови SOAP сервісів зі складною бізнес логікою та підтримкою різноманітних специфікацій та транспортних протоколів. На базі даного фреймворку можна реалізовувати і REST сервіс, але в такому випадку значна кількість можливостей не буде використовуватись. Загальна концепція фреймворку описується трьома термінами що описують взаємодію між компонентами, а саме контракт, прив’язка та адреса, які в сукупності узагальнюються в поняття endpoint. Контрактом є інтерфейс сервісу, що буде викликатися з інших додатків, тому важливо щоб контракт містив всі вимоги

до даних що використовує сервіс та інформацію про всі методи які він містить[21]. Прив’язка описує специфікацію що використовується, протокол транспортного рівня, по якому будуть передаватися дані, а також визначає рівень безпеки та час що виділяється на запит. Однак в сервісах що базуються на фреймворку WCF немає можливості вибирати формат даних що передаються, тому дані передаються у форматі xml, що може бути неефективно та незручно. Також варто відмітити що використовувати даний фреймворк доволі складно. Нині WCF вважається застарілою та неоптимальною технологією і використовується лише в великих корпораціях, де можуть бути корисними можливості які він надає.

ASP.NET MVC — фреймворк що надає можливість реалізації сумістити реалізацію серверної частини разом із клієнтською частиною, адже за допомогою технології Razor View можливо створювати та керувати html сторінками через класи та методи C#. Окрім цього залишається можливість передачі даних на інші реалізації клієнтів. Структура обробки запиту схожа на обробку запитів Web API, але додатково може містити механізм формування html сторінки.

ASP.NET Core — фреймворк що дозволив створювати серверні додатки C# під різні платформи. Сам фреймворк з’явився разом з платформою .NET Core і вмістись в собі функціональні можливості як Web API так і MVC. Зараз фреймворк активно розвивається і вже встиг завоювати популярність, адже у даному фреймворку враховані недоліки попередніх двох. Також варто згадати що ASP.NET Core за замовчуванням підтримує специфікації структури OWIN, контейнеризацію, в тому числі Docker, Web-сокети та інші сучасні веб технології.

Серед перечислених фреймворків для створення нових інформаційних систем рекомендується використовувати ASP.NET Core, адже він активно удосконалюється компанією Microsoft, містить функціональні можливості Web API та MVC, дозволяє додатку запускатися незалежно від операційної системи та пристрою сервісу під різними операційними системами.

3.1.3 Вибір середовища розробки

Для того щоб писати програми мовою C# достатньо мати на комп’ютері пакет

SDK, що містить платформу .net та компілятор, однак набагато зручніше та ефективніше використовувати середовища розробки з великою кількістю додаткових функцій для моніторингу ресурсів, використання шаблонів , відлагодження програми, та швидкої навігації по коду. Розглянемо декілька популярних IDE, що підтримують програмування мовою C#.

Rider — середовище розробки від компанії IntelliJ IDEA, має сучасний інтерфейс та всі необхідні засоби для створення програм мовою C#. Підтримує як платформу .NET так і .NET Core, може працювати на машинах під різними операційними системами, містить засоби для роботи з базою даних та контролю версій без встановлення додаткового програмного забезпечення. Також за замовчуванням містить в собі плагін IntelliJ IDEA Resharper, який вважається найкращим на ринку інструментом для рефакторингу та аналізу C# коду. Особливо зручно використовувати Rider людям які раніше програмували мовою Java та вже використовували програмне забезпечення від компанії IntelliJ IDEA.

Eclipse — середовище розробки яке не може похвастатись таким сучасним функціоналом як Rider, однак Eclipse існує доволі давно, та містить плагіни для роботи з багатьма мовами програмування, і багато програмістів вже звикли до нього. Eclipse надає базовий функціонал та неперевантажений інтерфейс, що буде хорошим вибором для людей що цінують мінімалізм або звикли працювати з Eclipse.

Visual Studio — середовище розробки від компанії Microsoft, яке значною мірою створювалося якраз під потреби платформи .NET та мови програмування C#. Містить величезну кількість додаткових функцій для рефакторингу коду, його аналізу, створення макетів програм та побудови діаграм. Підтримує встановлення багатьох плагінів та розширень, що інтегрують інші корисні інструменти в IDE. Варто також згадати що Visual Studio оновлюється синхронно з платформою .NET, що гарантує комфортну роботу та підтримку нового функціоналу починаючи з дня релізу версії платформи.

Visual Studio Code — нове середовище розробки від компанії Microsoft, що дуже швидко завоювало свою популярність. Основними перевагами є те що сама

IDE займає мало дискового простору, і підтримує багато мов програмування. Функціонал IDE може бути легко розширений за допомогою плагінів. Загалом IDE має лише основний необхідний функціонал для комфортного програмування, і що важливо цей функціонал схожий незалежно від мови програмування, що спрощує вивчення та використання нових мов.

Так як для написання інформаційної системи ми використовуватимо платформу .NET Core, включаючи останні оновлення платформи, оптимальним вибором буде Visual Studio, адже у ній наявна підтримка нових функцій, а також в ході розробки системи нам знадобляться розширені можливості IDE для діагностики та відлагодження системи.

3.2 Встановлення необхідного програмного забезпечення

Перед тим як почати безпосередньо розробку потрібно завантажити та встановите все необхідне ПЗ. Під час розробки я буду використовувати операційну систему Windows, тому необхідно завантажувати сумісні з нею компоненти. Першою необхідно завантажити та встановити інтегроване середовище розробки Visual Studio, знайти файл інсталяції можна на офіційному сайті Microsoft, вказавши останню версію та виконуючи інструкцію зі встановлення. Після першого запуску Visual Studio запропонує завантажити та встановити додаткові інструменти для розробки, де нам потрібно обрати пункти ASP.NET web development та .NET desktop development, як на рисунку 3.1. Після закінчення загрузки на комп’ютері будуть встановлені необхідні SDK та фреймворки для початку розробки. Після цього необхідно встановити базу даних та систему управління базою даних.

Після встановлення SQL Server з офіційного сайту програма встановлення запропонує також інсталювати SQL Server Management Studio, програмне забезпечення для керування базами даних, аналізу та профайлингу запитів. Після завершення інсталяції ми зможемо авторизуватися на сервері використовуючи акаунт Microsoft, як показано на рисунку 3.2.

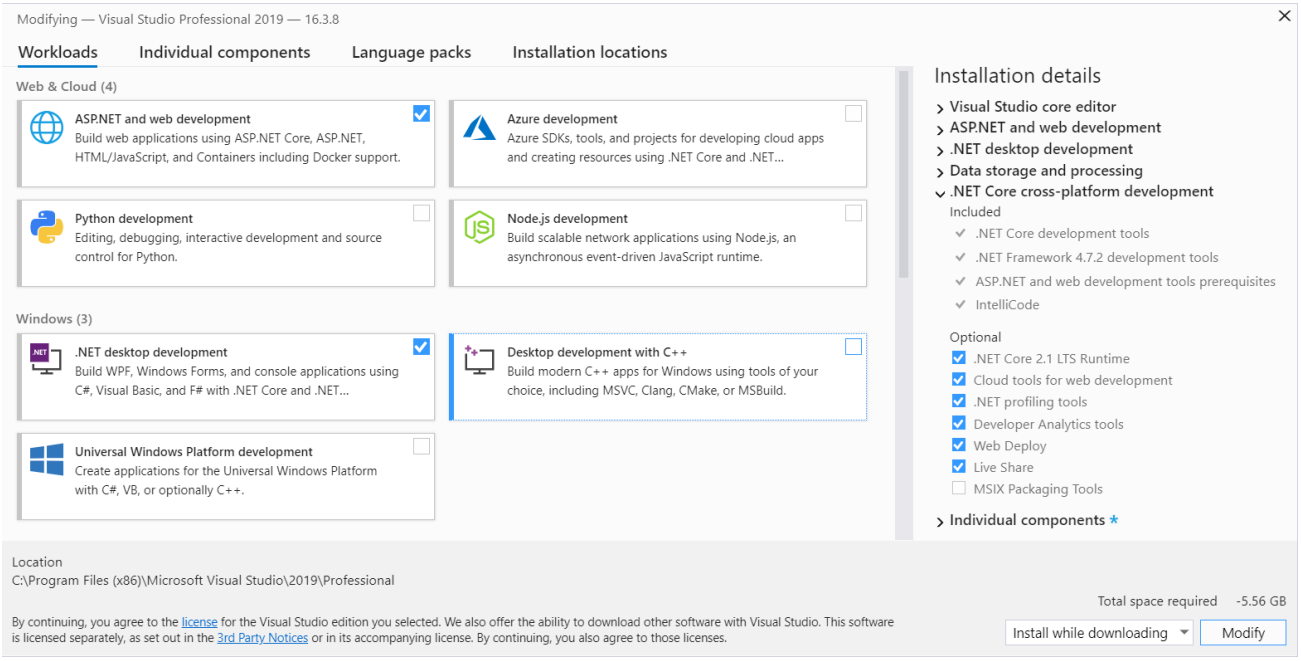

Рисунок 3.1 —Вікно вибору компонентів Visual Studio

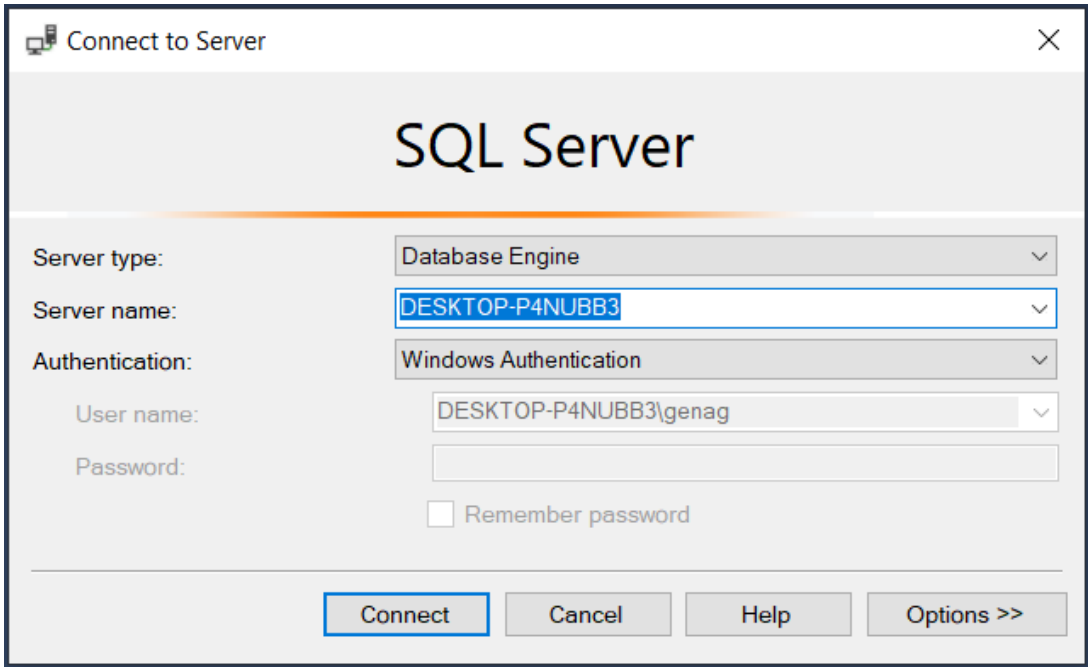

Рисунок 3.2 — Вікно авторизації до SQL Server

3.3 Створення проекту інформаційної системи та базові налаштування

Створимо каркас проекту використавши шаблон ASP.NET Core у Visual Studio. Після цього потрібно видалити класи-приклади і замінити їх на власні класи. Розглянемо детально ключові файли та папки в проекті, та їх призначення. Базова структура проекту зображена на рисунку 3.3.

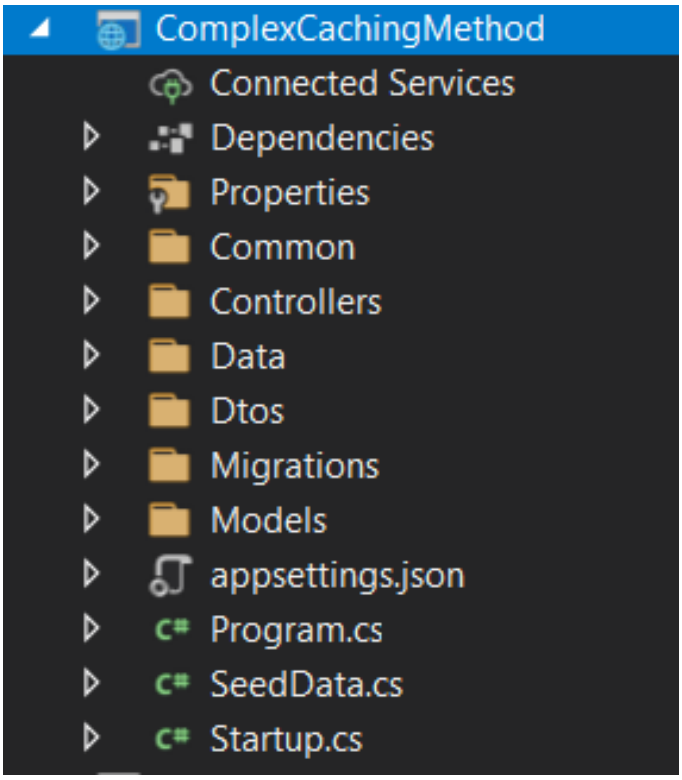

Рисунок 3.3 — Базова структура проекту інформаційної системи

У класі StartUp знаходиться код конфігурування сервісу, у ньому підключаються сторонні бібліотеки та класи, реєструються в inversion of control контейнері класи та інтерфейси, налаштовується підключення до бази даних, механізми авторизації користувачів, логування та інші ключові конфігурації. З даним класом нам доведеться працювати впродовж всього проекту, адже у ньому реєструватимуться нові класи та додаватимуться новий функціонал та бібліотеки.

У класі Program безпосередньо починається виконання програма, запускається білдер сервісу та клас конфігурації.

У папці Controllers знаходяться класи-контролери, так у фреймворку ASP.NET Core називаються класи що є кінцевими точками для доступу до функціоналу за допомогою веб запитів. Запит надходить через протокол HTTP, опрацьовуються у класі контролера за допомогою викликів методів інших класів, після чого контролер надсилає результат або помилку у форматі HTTP response.

У папці Data знаходяться класи призначенні для роботи з базою даних. Це можуть бути контексти бази даних, її структурні класи, репозиторії до відношень бази даних тощо. Також у даній папці буде знаходитись реалізація методу

кешування.

У папці Migrations знаходяться файли що описують міграції бази даних, а саме код який було виконано під час змінення схеми бази даних та зворотній код, який при виконанні анулює всі зміни внесені міграцією. Дані файли генеруються автоматично фреймворком Entity Framework Core при оновленні структури бази даних, а також можуть змінюватися вручну.

У папці Models зберігаються класи моделей бази даних — об’єктів що відповідають структурі таблиць у базі даних. Дані класи можуть містити публічні властивості та коментарі з описом полів, і зазвичай не містять функціональні методи або методи помічники. Ці класи використовуються фреймворком Entity Framework Core для конфігурування та організації роботи з базою даних.

У папці Common зберігаються класи-помічники та класи розширення, функціонал яких може використовуватись або розширювати функціонал інших класів.

У папці Dtos зберігаються класи передачі об’єктів (data transfer object). Дані класи використовуються для передачі інформації про моделі у тілі http запитів.

У файлі appsettings.json зберігаються динамічні налаштування, такі як адреса підключення до бази даних, ключі та секрети доступу до сторонніх API що використовуються, значення рівню логування тощо (лістинг 3.1). Кожне окреме оточення може використовувати окремий файл конфігурацій з актуальними для даного оточення параметрами.

Лістинг 3.1 — Динамічні конфігурації сервісу

```
 "Logging": {
  "LogLevel": {
   "Default": "Information",
   "Microsoft": "Warning",
   "Microsoft.Hosting.Lifetime": "Information"
  }
 },
```

```
  "AllowedHosts": "*",

  "ConnectionStrings": {
"DefaultConnection": "Data Source=.;Initial Catalog=
ComplexCachingMethod;Integrated Security=True"
  }
```

Інші папки та файли будуть додаватись по необхідності і їх призначення буде пояснюватись в ході реалізації проекту, однак більшість класів розміщуватимуться у описаних папках. Таким чином проект є чітко структурованим, що дозволяє легко розширювати функціонал та швидко знаходити потрібний код та конфігурації.

3.4 Підключення та налаштування бази даних

Тепер необхідно підключити базу даних та ORM бібліотеки для роботи з нею. Щоб зробити це потрібно завантажити пакети NuGet для Entity Framework у менеджері сторонніх бібліотек Visual Studio, так як на рисунку 3.4. Після встановлення пакетів потрібно створити клас що відповідатиме за структуру бази даних. За конвенціями назвемо даний клас DataContext та розмістимо у папку Data. Конструктор класу приймає такі параметри як DbContextOptions, в яких можуть вказуватися специфічні для бази даних властивості та налаштування. У даному класі будуть створюватися властивості, які відповідають таблицям бази даних, та використовуються для роботи з базою даних як з об’єктами рівня додатку.

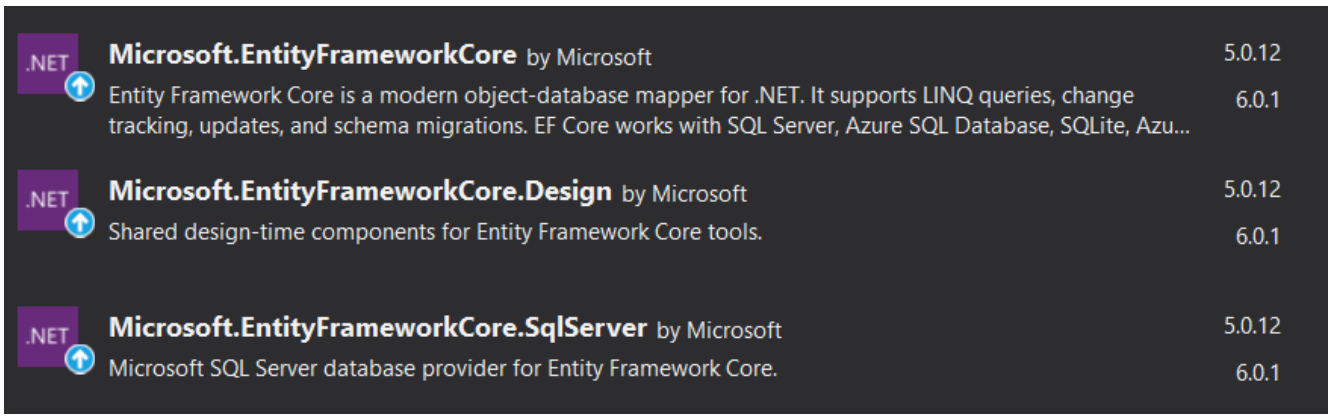

Рисунок 3.4 — NuGet packages для роботи з базою даних MsSQL

Після створення класу контексту бази даних необхідно зареєструвати його у класі конфігурацій StartUp. Там ми також передаємо інформацію шляху підключення до бази даних, що розміщується у файлі appsettings.json.

Лістинг 3.2 — Реєстрація контексту бази даних

```
public void ConfigureServices(IServiceCollection services) {
    services.AddDbContext<DataContext>(options => options.UseSqlServer(
        Configuration.GetConnectionString("DefaultConnection")));
```

Тепер коли Entity Framework налаштовано, можна виконати першу міграцію яка створить початкову версію бази даних. Для цього потрібно відкрити Power Shell від імені адміністратора та виконати команду: ef migrations add InitialCreate, після чого застосувати дану міграцію, тим самим створивши базу даних, виконавши команди ef database update. Тепер ми можемо використовувати контекст бази даних у коді і еквівалентні зміни будуть застосовані до бази даних.

3.5 Розробка контролерів інформаційної системи

У інформаційній системі у якості головної моделі використаємо клас User. Діаграма класу користувача зображена на рисунку 3.5 збереження в базі даних про користувача.

Рисунок 3.5 — Діаграма класу User

Створимо для роботи з класом користувача два контролера, щоб розділити функціональність яка відповідає за реєстрацію нового користувача та авторизацію, та звичайну CRUD функціональність. Назвемо контролер для роботи з авторизацією AuthController, та створимо у ньому такі методи як Register та Login. Також за допомогою Dependency injection включимо до класу такі інтерфейси як IAuthRepository, IConfiguration, IMapper. IAuthRepository використовується для роботи з базою даних, і буде детально розглянутий у наступному підрозділі, IConfiguration потрібен для роботи з конфігураціями проекту, а IMapper виконує перетворення об'єкту передачі даних у об'єкт класу користувача, та навпаки.

Варто зауважити, що методи Login та Register звертаються до бази даних щоб отримати або зберегти дані користувача, і тому варто зробити їх виконання асинхронним. Справа у тому що звернення до бази даних займає деякий час, тоді як потік що виконує програму зупиняється та очікує. Спершу здається що це не велика проблема, адже в сучасних операційних системах потоків багато, однак якщо на сервер надходить велика кількість паралельних запитів кількість вільних потоків грає ключову роль, адже від цього залежить швидкість роботи сервісу. Для створення асинхронних викликів у C# є багато інструментів, однак оптимальним варіантом є застосування бібліотеки TPL та ключових слів async await. В сигнатурі методу, який буде виконуватися асинхронно, пишеться ключове слово async, що дозволяє робити асинхронні виклики в тілі методу використовуючи ключове слово await. Також потрібно замінити тип T, який повертав метод, на об'єкт узагальненого класу Task<T>.

Лістинг 3.3 — Методи та властивості класу AuthController

```
[Route("api/[controller]")]
[ApiController]
public class AuthController : ControllerBase
{
    private readonly IAuthRepository _repository;
    private readonly IConfiguration _config;
```

```
        private readonly IMapper _mapper;
        public AuthController(IAuthRepository repository, IConfiguration config,
IMapper mapper)
        [HttpPost("register")]
        public async Task<IActionResult> Register(UserForRegisterDto userDto)
        [HttpPost("login")]
        public async Task<IActionResult> Login(UserForLoginDto
userForLoginDto)
    }
```

Метод Register приймає дані які конвертується в об’єкт типу User, та повертає відповідь після збереження користувача в базу даних. Оскільки вирішено виконувати даний метод асинхронно повертати він буде Task<IActionResult>. Об’єкт що передається в тілі запиту як параметр метода не повинен бути об’єктом класу User, адже даний клас містить властивості які не повинні бути доступними для користувачів API. Для вирішення даної проблеми використовуються додаткові класи, які іменують DTO, що розшифровується як data transfer object.

DTO — це об'єкт, який використовують для інкапсуляції даних та передачі їх між сервісами за допомогою http запитів. Використання DTO є прийнятою практикою у API аплікаціях. Додатковою перевагою використання DTO є те, що це зменшує кількість даних, які надсилаються по мережі, що економить мережеві ресурси та підвищує швидкість комунікації між сервісами. Також DTO широко використовуються у додатках побудованих за шаблоном MVC. Іншим прикладом застосування DTO може бути інкапсуляція параметрів методів. Це може бути корисно, якщо метод має велику кількість параметрів, як наприклад оновлення користувача або реєстрації. Однак всередині сервера робота ведеться над доменними моделями, тому необхідно конвертувати DTO в об’єкти моделей. Це можна робити вручну, або скористатися сторонніми бібліотеками що надають можливість автоматично конвертувати об’єкти за допомогою рефлексії. У даному проекті використовується бібліотека AutoMapper.

AutoMapper завантажується як nuget package за допомогою менеджера сторонніх бібліотек Visual Studio, після чого конфігурується у окремому класі для конвертації потрібних нам об’єктів. Для цього в класі Startup необхідно зареєструвати конфігурацію AutoMapper та передати об’єкт assembly у якості параметра (лістинг 3.4).

Лістинг 3.4 — Реєстрація класу бібліотеки AutoMapper

```
services.AddControllers();
services.AddAutoMapper(typeof(Startup));
```

Аналогічно реалізовуються і другий контролер, який відповідає за виконання операцій над користувачами, а саме зчитування, видалення та оновлення інформації користувача. У даному випадку в обов’язки контролера входить конвертація об’єктів DTO у об’єкти моделі, формування HTTP відповіді на запит та коректна обробка виключень. За допомогою атрибутів класу позначається шлях url [Route("api/users")], та вимога авторизації для доступу до контролера за допомогою атрибуту Authorize (лістинг 3.5).

Лістинг 3.5 — Методи та властивості класу UsersController

```
[Route("api/users")]
[ApiController]
public class UsersController : ControllerBase
{
    private readonly IRepository<User> _repository;
    private readonly IMapper _mapper;
    public UsersController(IRepository<User> repository, IMapper mapper)
    [HttpGet("{id}", Name = "GetUser")]
    public async Task<IActionResult> GetUser(int id)
    [HttpPut("{id}/update")]
    public async Task<IActionResult> UpdateUser(int id, UserForUpdateDto
userForUpdateDto)
```

```
        [HttpPut("{id}/delete")]
        public async Task<IActionResult> DeleteUser(int id)
    }
```

Аналогічно контролеру авторизації ми передаємо об’єкти необхідних інтерфейсів через контролер за допомогою механізму dependency injection. Метод Наступним методом є пошук конкретного користувача по його ідентифікаційному номеру. Метод GetUser використовується для запитів інформації про конкретного користувача. Метод приймає id користувача у якості вхідного параметра, та повертає DTO об’єкту користувача у відповідь. Метод UpdateUser використовується для запитів на оновлення інформації користувача. Даний метод приймає у якості параметра DTO користувача та повертає результат чи були зміни успішно застосовані. Усередині методу запит перевіряється на коректність та чи існує такий користувач, у випадку невідповідностей надсилається відповідь з кодом помилки. Третім методом даного контролеру є метод видалення користувача з бази даних. Сигнатура методу містить ідентифікатор користувача по якому знаходиться відповідний користувач у базі даних якого потрібно видалити.

3.6 Розробка шару доступу до бази даних

Контролери містять високорівневі інструкції та використовують функціонал інших класів, в той час коли робота з базою даною виконується у класах рівня доступу до бази даних. Такі класи називаються ще репозиторіями, працюють вони безпосередньо з контекстом бази даних. Часто кількість репозиторіїв у системі відповідає кількості контролерів. У даному випадку ми створимо два репозиторії, AuthRepository для інкапсуляції роботи з базою даних у AuthController та UserRepository для UsersController.

Лістинг 3.6 — Методи та властивості класу AuthRepository

```
public class AuthRepository : IAuthRepository
    public async Task<User> Login(string username, string password)
```

private bool VerifyPasswordHash(string password, byte[] passwordHash, byte[] passwordSalt)

public async Task<User> Register(User user, string password)

private void CreatePasswordHash(string password, out byte[] passwordHash, out byte[] passwordSalt)

public async Task<bool> UserExists(string username)

У класі репозиторію міститься об’єкт контексту бази даних та два публічних методи, Login та Register. Метод Register приймає об’єкт користувача та пароль у рядковому вигляді, після чого шифрує пароль та зберігає об’єкт користувача у базі даних належним чином. Пароль шифрується за допомогою алгоритму SHA512 (Secure Hash Algorithm). Даних алгоритм широко використовується у кріптографії для захисту конфіденційних даних. Він заснований на хеш-таблиці, має довжину 512 біт і його злам займає дуже багато часу, що робить його одним із найнадійніших алгоритмів шифрування.

Після шифрування пароля та успішного збереження користувача в базі даних користувач отримає унікальний ідентифікатор для швидкого доступу до інформації в майбутньому. Потім метод реєстрації перетворює об’єкт User у спеціальний DTO, повертає відповідь з http кодом 200 та об’єктом користувача у повідомленні. Код стану 200 вказує на те, що операція виконалась успішно.

Наступний метод — це метод Login, який дозволяє користувачам входити до свого облікового запису. Як і метод Register, цей метод працює асинхронно і приймає спеціальний DTO як параметр, а також ім’я користувача та пароль. Завдання цього методу — перевірити дані, введені користувачем, і якщо дані правильні, створити JWT токен, що дозволяє системі ідентифікувати користувача.

Токен безпеки JWT — це відкритий міжнародний стандарт, який представляє собою компактний та автономний спосіб безпечної передачі інформації між сторонами за допомогою об’єктів JSON. JWT токен підписаний закритим ключем, тому перевірити інформацію, зашифровану маркером зможе тільки той хто має парний ключ. Інформація записується у вигляді об’єкта JSON, перетворюється у

формат Base64 і кодується алгоритмом шифрування SHA512.

Клас UsersController простіший за своєю функціональністю і в початковому варіанті слугує просто обгорткою стандартним методам Entity Framework, таким як пошук по ідентифікатору, видалення та збереження даних у базу даних. Методи та властивості класу UsersController знаходяться у лістингу 3.7.

Лістинг 3.7 — Методи та властивості класу UsersController

```
public class UserRepository : IRepository<User>
{
    private readonly DataContext _context;
    public UserRepository(DataContext context)
    public void Add(User user)
    public void Delete(User user)
    public async Task<User> GetAsync(int id)
    public async Task<bool> SaveAsync()
    public async Task<User> UpdateAsync(User userForUpdate)
}
```

Усі методи що працюють з базою даних у даному класі асинхронні. Методи Add та Delete не працюють з базою даних напряму, при їх використанні змінюється колекція моделей EntityFramework, і всі зроблені зміни будуть застосовані до бази даних лиш тоді коли виконається метод SaveAsync.

3.7 Реалізація комплексного методу кешування даних

Основне навантаження розробленої інформаційної системи припадає на клас UsersController, адже інформація про користувачів зчитується набагато частіше ніж реєструється новий користувач, окрім того клас UsersController також виконує операції зміни та видалення даних. В такому випадку доцільно застосувати метод кешування даних до даного контролеру, що повинно суттєво знизити навантаження на базу даних та пришвидшити інформаційну систему загалом. UsersController у

якості шару доступу до бази даних використовує клас UserRepository, тому саме у ньому буде використовуватись логіка кешування даних.

У якості структури даних було вирішено використовувати словник, однак у інформаційну систему може надходити багато одночасних запитів. Щоб уникнути властиві для багатопоточних додатків проблеми варто використовувати спеціально створену для таких потреб потоко-безпечну версію словника ConcurrentDictionary. Фреймворк ASP.NET Core додатково до ConcurrentDictionary пропонує клас обгортку для даної структури даних IMemoryCache, з набором стандартних методів для роботи з даними та реалізацією механізму інвалідації даних. Візьмемо дану структуру даних за базу нашого класу кешування даних.

Створимо у папці Data створимо клас який міститиме логіку методу кешування, так як це було описано та обґрунтовано у другому розділі, та назвемо його UserCache. Структура класу, його властивості та методи можна знайти у лістингу 3.8.

Лістинг 3.7 — Методи та властивості класу UserCache

```
public class UserCache : ICache<User>
{
    private readonly IMemoryCache _cache;
    public UserCache(IMemoryCache cache)
    public User Get(int id)
    public void Set(User user)
    public void Remove(int id)
    private string GetCacheKey(int id)
}
```

Даний клас наслідується від спеціального створеного інтерфейсу ICache<T>, для того щоб його можна було зареєструвати у контейнері dependency injection, та легко передавати у якості параметра конструкторів інших класів. Також даний клас містить властивість з об’єктом структури даних IMemoryCache, яка є потокобезпечним словником для збереження інформації та ініціалізується у

конструкторі класу. Окрім цього клас реалізовує декілька методів для роботи з об’єктами кешування, розглянемо детально кожен із них.

Приватний метод GetCacheKey використовується для генерації унікального ключа, за допомогою якого буде збережено об’єкт користувача. У якості параметра даний метод приймає ціле число, після чого конвертує його у рядок, додаючи метаінформацію про тип об’єкта який буде збережено за даним ключем, та повертає його.

Метод Set використовується для збереження об’єкту користувача у кеш. Даний метод приймає у якості параметра об’єкт користувача, після чого базуючись на ідентифікаторі користувача, за допомогою методу GetCacheKey, створює унікальний ключ для збереження даного об’єкту. Також у даному методі конфігурується абсолютний час інвалідації об’єкту кешу, період інвалідації об’єкту та пріоритет збереження об’єкту. Використовуючи дані параметри даний метод зберігає об’єкт користувача в кеш.

Метод Get використовується для зчитування об’єкту користувача з кешу. Метод приймає у якості параметра ціле число Id, після чого звертається до методу GetCacheKey для генерації унікального ключа. Після цього програма намагається знайти у кеші за допомогою даного ключа користувача з відповідним ідентифікатором, якщо такого користувача було збережено у кеші. Якщо користувача знайдено метод повертає його, у іншому випадку метод повертає значення null.

Метод Remove використовується для позачергового видалення об’єкту користувача з кешу. Метод подібно методу Get приймає ідентифікатор користувача, отримує унікальний ключ та видаляє потрібний об’єкт по даному ключу.

Разом ці методи надають необхідний функціонал для оперування над об’єктами в кеші та управління циклами життя даних об’єктів. Щоб система запрацювала використовуючи кеш необхідно модифікувати відповідний клас репозиторію. Перш за все потрібно додати у конструктор класу інтерфейс кешу, для того щоб dependency injection контейнер при необхідності створить об’єкт

кешу.

Змінимо метод GetAsync в класі UserRepository наступним чином, додами в сигнатуру методу новий параметр типу bool useCache, на основі якого визначатиметься чи буде програма працювати напряму з базою даних, чи спробує знайти потрібні дані у кеші. У випадку якщо параметр useCache дорівнює true створимо пустий об’єкт користувача та спробуємо зчитати інформацію за допомогою методу Get класу UserCache. Якщо такий користувач є у кеші, його буде повернуто із методу репозиторію. Якщо такого користувача немає або useCache дорівнює false, виконається звичайний потік програми, де репозиторій за допомогою контексту бази даних зчитає інформацію з бази даних, після чого збереже її у кеш за допомогою методу Set класу UserCache.

Також змінимо метод оновлення даних користувача UpdateAsync. Після того як нова інформація вступила в силу необхідно також оновити дані у кеші. Але, як згадувалось раніше, метод UpdateAsync не зберігає змінені дані до бази даних одразу, а робить це лише після виконання асинхронного методу SaveAsync. Тому і дані в кеші не можна оновлювати одразу, адже вони можуть не бути змінені у базі даних і тоді виникне розбіжність між даними що може призвести до багатьох помилок. Для вирішення даної проблеми я використаю механізм відкладеного виконання операцій по оновленню кешу. Для цього потрібно спочатку створити список який зберігатиме об’єкти типу Task. У один із таких об’єктів при виконанні методу UpdateAsync необхідно записати лямбда функцію, у якій буде міститися логіка по оновленню об’єкту в кеші. Далі також потрібно модифікувати метод SaveAsync, щоб він перевіряв даний список задач, та у разі успішного виконання оновлення даних у базі даних також виконував всі задачі що містяться у списку задач, після чого очищав би його.

Аналогічно методу UpdateAsync, метод Delete також потребує оновлення інформації в кеші після виконання логіки методу. Однак на відміну від UpdateAsync, після видалення об’єкту з бази даних потрібно також стерти його з кешу. Це досягається використанням описаного вище механізму відкладеного виконання лямбда функції, у тілі якої повинен викликатися метод Remove класу

UserCache. Після даних змін репозиторій буде працювати коректно використовуючи систему кешування для зниження навантаження на базу даних.

Отже у третьому розділі були обрані оптимальні технології для створення тестової інформаційної системи та реалізації комплексного методу кешування даних. Також у ході роботи були встановлені всі необхідні для проекту інструменти та бібліотеки. Була підключена та налаштована база даних MsSQL, сконфігурований для використання та роботи над даними фреймворк Entity Framework Core. Також були спроектовані та імплементовані шар доступу до бази даних, класи старту інформаційної системи, класи контролери, класи репозиторії, класи моделей та класи передачі даних. У розробленій інформаційній системи був імплементований та застосований метод комплексного кешування даних, спроектований та описаний у другому розділі даної роботи.

4 ТЕСТУВАННЯ ТА АНАЛІЗ ПРОДУКТИВНОСТІ РОБОТИ ЗАСОБУ КОМПЛЕКСНОГО КЕШУВАННЯ ДАНИХ

4.1 Тестування інформаційної системи

Перед тим як вимірювати та аналізувати вплив використання системи кешування на інформаційну систему потрібно пересвідчитись що інформаційна система працює коректно. Виконуватись тестування системи буде за допомогою безкоштовного програмного засобу Postman, який містить функціонал для створення http викликів та їх аналізу. Також за допомогою Postman можна легко змінювати параметри запиту та форматувати результат виконання запиту у зручному для сприйняття форматі.

Спочатку протестуємо контролер створення користувача. Для цього потрібно створити HTTP POST запит з наступним url http://localhost:5000/api/auth/register. (Лістинг 4.1)

Лістинг 4.1 — Тіло http запиту реєстрації користувача

Body:

```
{
    "username": " Hennadii",
    "password": "Horbachov",
    "gender": "male",
    "knownas": "Gena ",
    "dateofbirth": "1999-08-01",
    "city": "Vinnytsya",
    "country": "Ukraine",
    "interests": "Programming"
}
```

Результат виконання запиту зображено на рисунку 4.1.

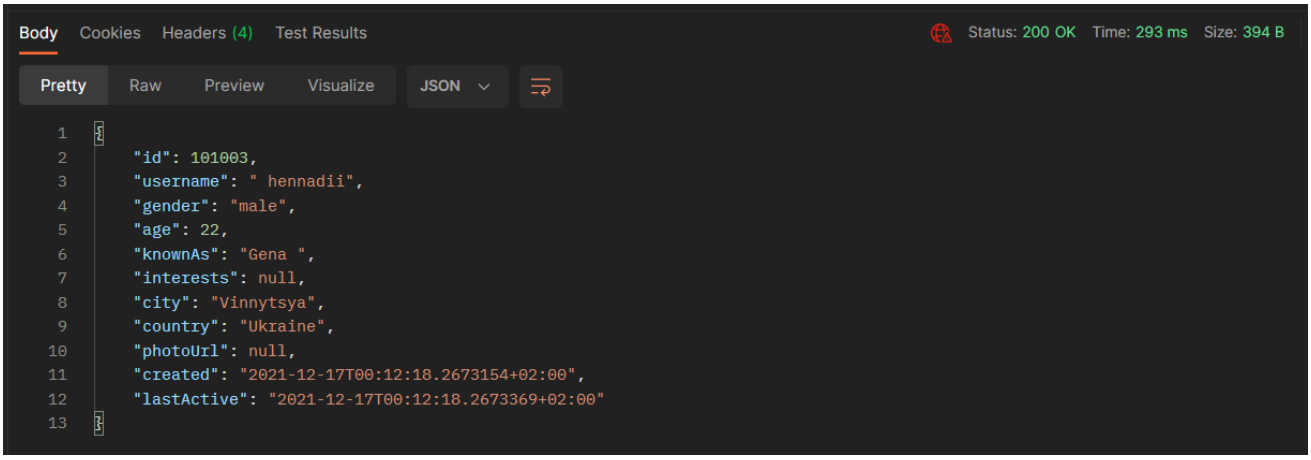

Рисунок 4.1 — Результат виконання запиту реєстрації користувача

Тепер перевіримо функціонал аутентифікації користувача. Для цього потрібно створити HTTP POST запит на наступний url: http://localhost:5000/api/auth/login. (Лістинг 4.2)

Лістинг 4.2 — Тіло http запиту аутентифікації користувача

```
Body:
{
    "username": "Hennadii",
    "password": "Horbachov"
}
```

Результат виконання http запиту аутентифікації користувача на рисунку 4.2.

Рисунок 4.2 — Результат виконання http запиту аутентифікації користувача

У тілі відповіді видно що сервер знайшов користувача у базі даних та згенерував токен для подальшої авторизації. Використовуючи даний токен ми можемо звертатися до захищених методів контролерів, адже у ньому міститься інформація про того хто надсилає запит.

Протестуємо також метод Get. Для цього створимо http Get запит з наступним url: http://localhost:5000/api/users/101003.

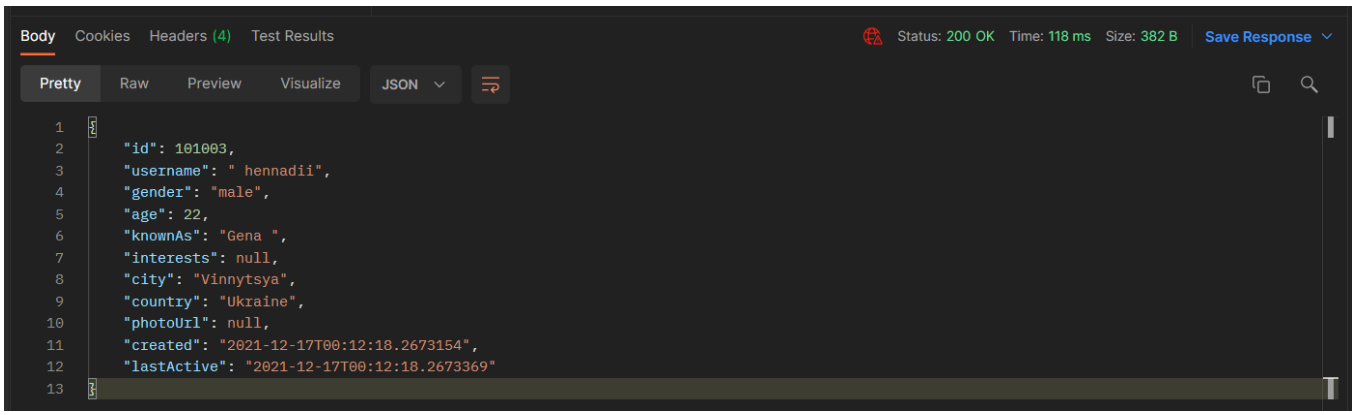

Рисунок 4.3 — Результат виконання http запиту пошуку користувача

На рисунку 4.3 видно що сервером була зчитана інформація користувача, після чого він сформував HTTP відповідь. Час виконання запиту становив 118 мілісекунд. Таким чином сервер було протестовано та доведено коректність його роботи.

4.2 Тестування роботи інформаційної системи з використанням кешування

Для того щоб увімкнути модуль кешування необхідно при виклику методу пошуку користувача встановити параметр useCache в значення true. Тепер знову виконаємо http запит пошуку користувача. При першому виконанні запиту нічого не змінилося, однак при наступному інформація вже буде зчитана з кешу. Як видно на рисунку 4.4 після ввімкнення модулю кешування ми отримуємо від сервера аналогічну відповідь, однак на виконання запиту було витрачено не 118 мілісекунд, а 4 мілісекунди.

Якщо приєднати до інформаційної системи дебагер можна побачити що тепер при виконанні запиту код звертається до кешу, де знаходиться запис

збереженого користувача (рисунок 4.5). На даному рисунку у полі AbsoluteExpiration також видно дату коли дані перестануть бути актуальними та будуть видалені з кешу.

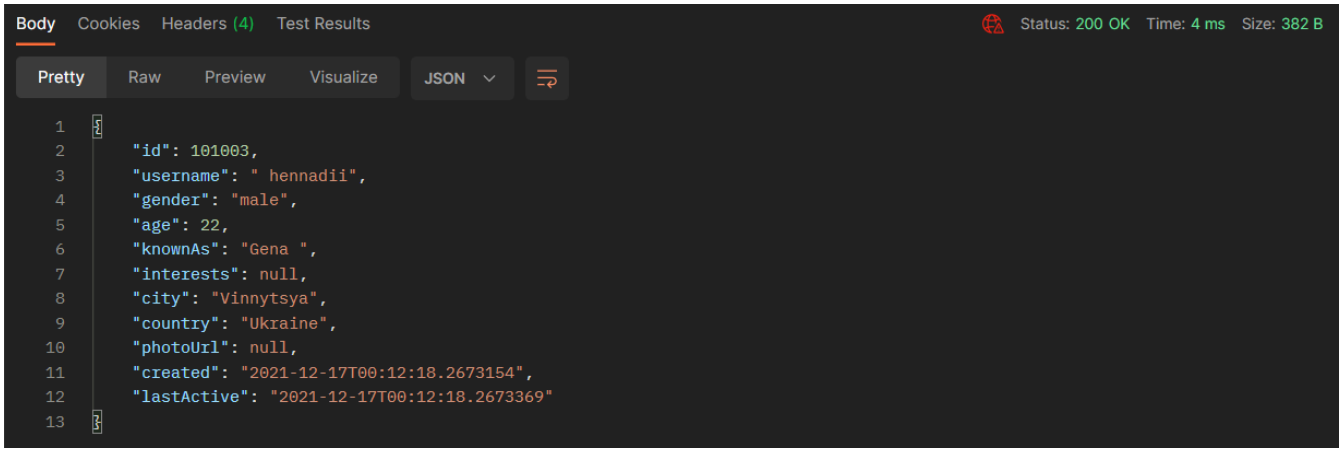

Рисунок 4.4 — Результат виконання http запиту пошуку користувача з використанням модулю кешування

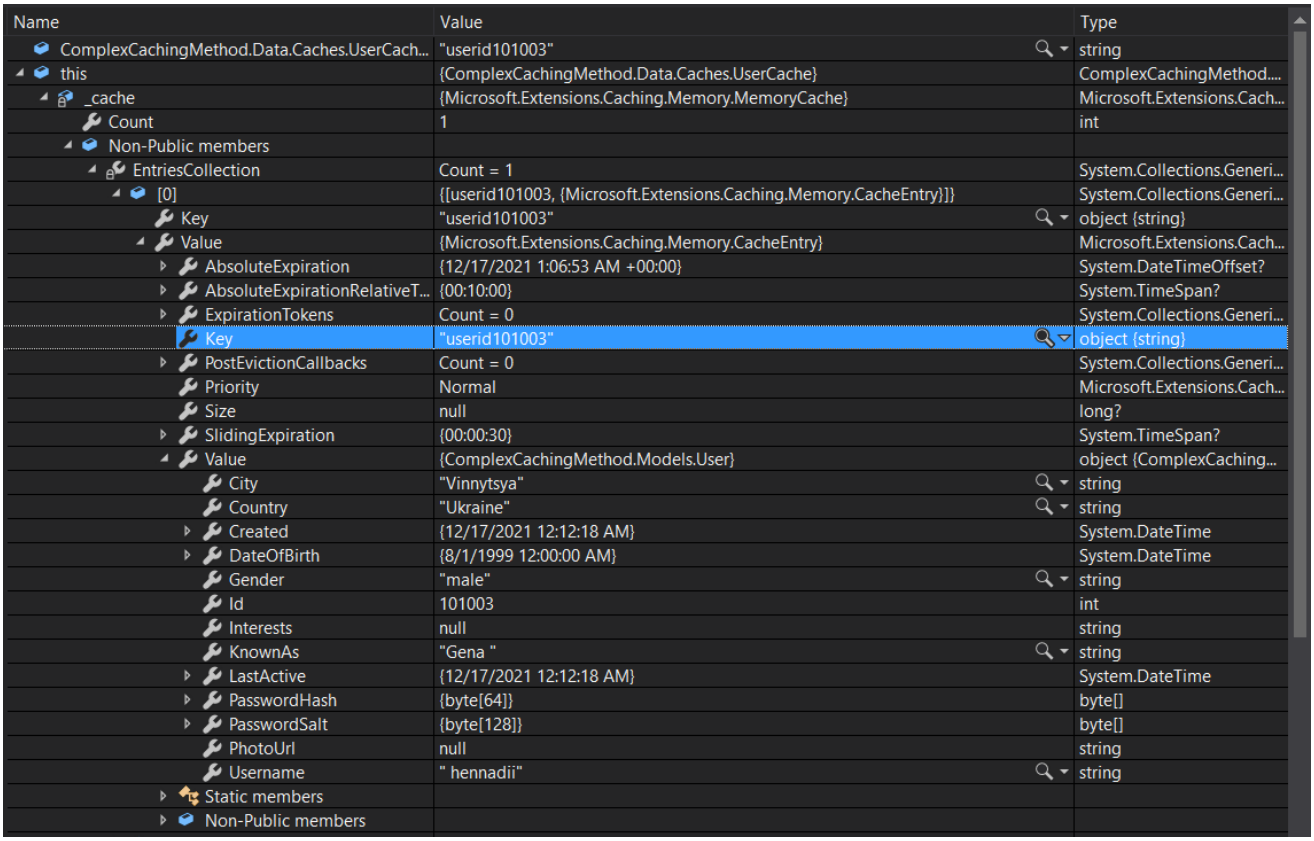

Рисунок 4.5 — Властивості об’єкту користувача збереженого в кеші

Тепер спробуємо змінити даного користувача використавши метод контролеру Users Update. Для цього створимо Http post запит з наступним вмістом:

Лістинг 4.3 — Тіло http запиту оновлення даних користувача

Body:

```
{
    "knownas": "Genadious",
    "city": "New York",
    "country": "USA"
    "interests": "Programming"
}
```

Після виконання запиту за допомогою інструменту Debug перевіримо вміст кешу.

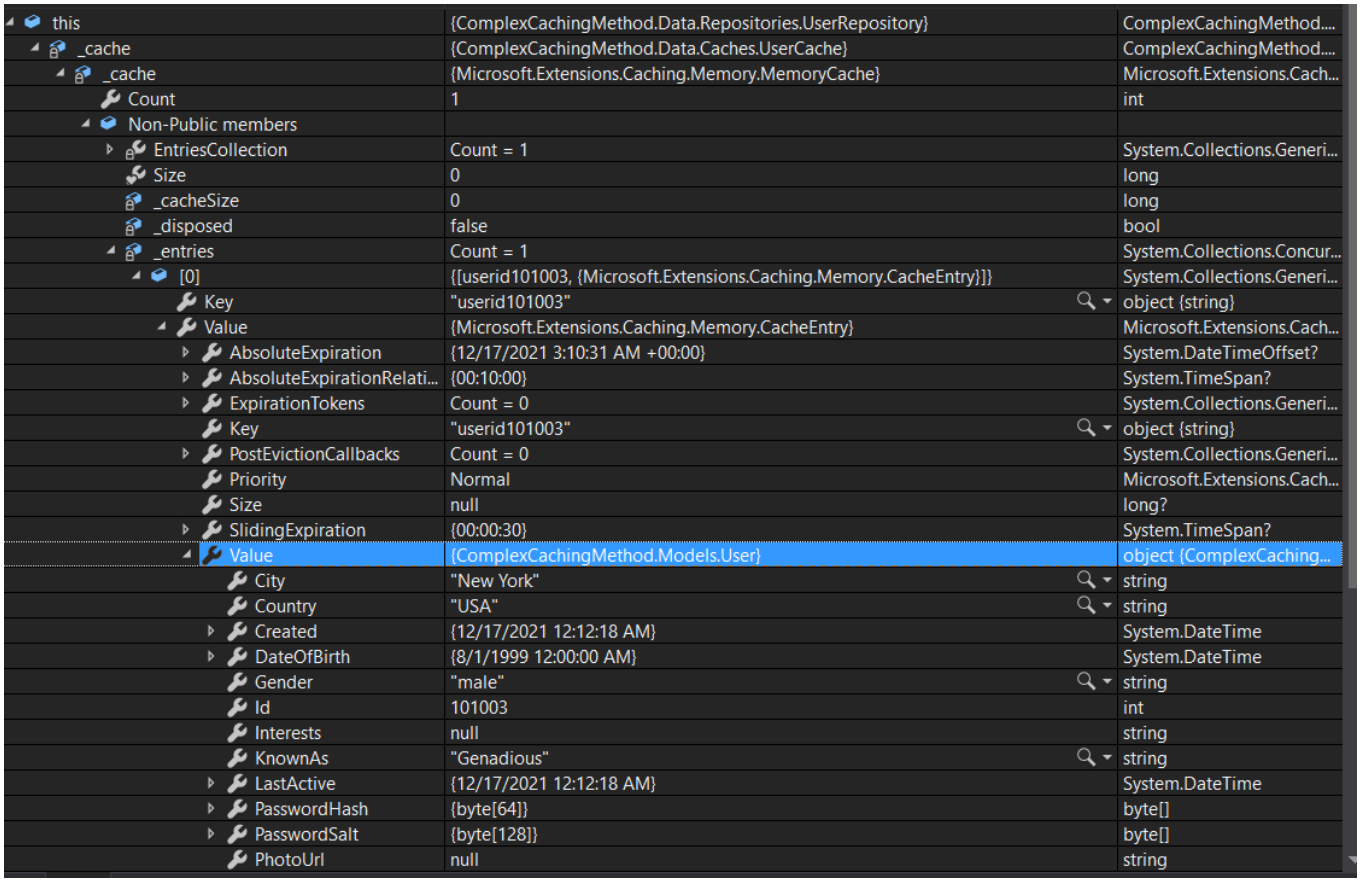

Рисунок 4.6 — Властивості об’єкту користувача в кеші після оновлення даних

Як видно на рисунку 4.6 дані були також змінені у кеші, і залишились цілісними. Таким чином, методом тестування доведено що модуль кешування працює коректно.

4.3 Тестування роботи інформаційної системи в умовах високого навантаження

Для створення високого трафіку на інформаційну систему скористаємося фреймворком тестування NUnit. Згідно досліджень розподілення трафіку реальних інформаційних системах використаємо наступну гіперболічну функцію для симуляції розподілення трафіку наближеного до реального. Для ста тисяч запитів схема розподілення трафіку буде наступною (рисунок 4.7).

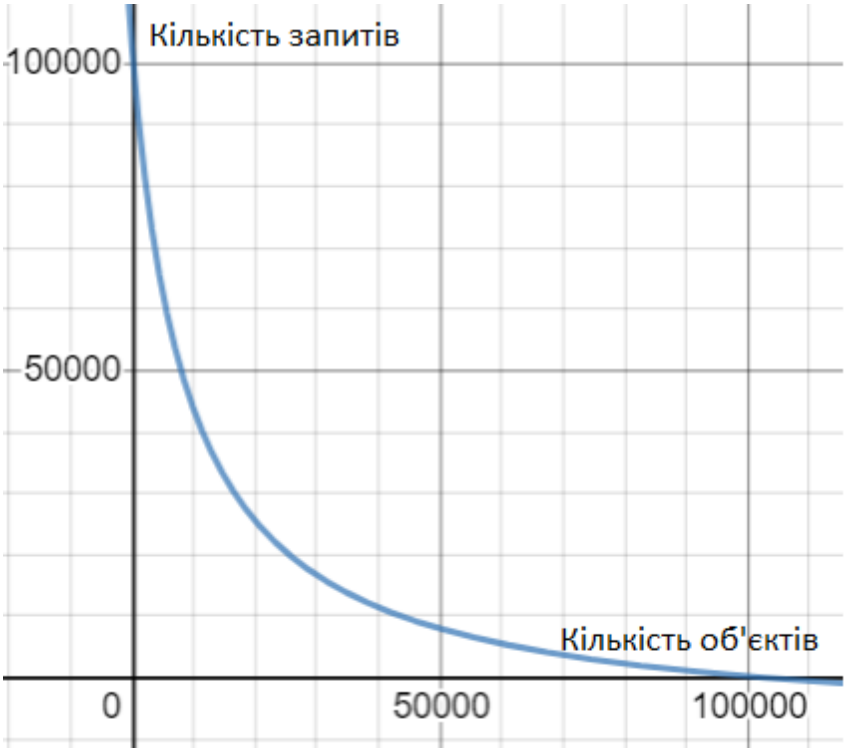

Рисунок 4.7 — Графік розподілення трафіку інформаційної системи

Створимо проект StressTests, та метод який буде створювати велику кількість запитів за короткий проміжок часу (лістинг 4.4).

Лістинг 4.4 — Код методу StressTest

```
public async Task StressTestAsync()
{
   var httpClient = new HttpClient();
   httpClient.DefaultRequestHeaders.CacheControl = new
CacheControlHeaderValue { NoCache = true};
```

```
            var random = new Random();
            var sw = new Stopwatch();
            int n = 100;
            sw.Start();
            for (int i = 0; i < 100; i++)
            {
                var tasks = new List<Task>(n);
                for (int j = 0; j < n; j++)
                {
                    var task = Task.Run(async () => {
                        int x = random.Next(N);
                        double y = (N * N / 10) / (x + K) - K;
                        var request = new HttpRequestMessage(HttpMethod.Get,
$"https://localhost:5001/api/users/{Math.Round(y)}");
                        try
                        {
                            var result = await httpClient.SendAsync(request);
                        }
                        catch (Exception ex)
                        {
                            Console.WriteLine(ex.Message);
                        }
                    });
                    tasks.Add(task);
                }
                await Task.WhenAll(tasks);
                if (i%10 == 0)
                    Console.WriteLine($"{i*n+1}-{(i+10)*n} request sent. Time:
{sw.ElapsedMilliseconds} ms.");
            }
```

Як видно з коду ми відправляємо по сто паралельних запитів на інформаційну систему сто раз. Таким чином створюється навантаження в десять тисяч запитів, після чого ми можемо зафіксувати результати роботи системи та проаналізувати вплив модуля кешування на її швидкодію. Спочатку запустимо на виконання тесту на інформаційній системі з вимкненим модулем кешування (рисунок 4.8).

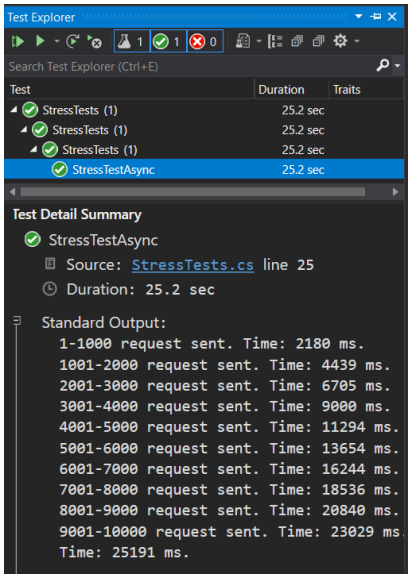

Рисунок 4.8 — Виконання тесту на інформаційній системі без модулю кешування

Як видно з рисунку 4.8, час виконання тесту становить 25 секунд, а час виконання наступної порції запитів приблизно рівномірний. Графік витрат комп’ютерних ресурсів зображено на рисунку 4.9.

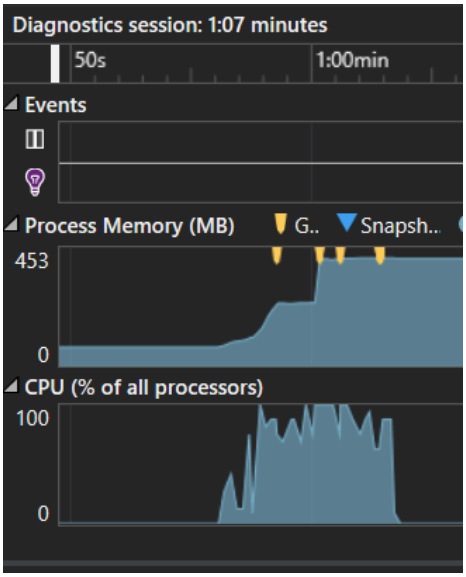

Рисунок 4.9 — Графік витрат комп’ютерних ресурсів без модулю кешування

Тепер спробуємо виконати даний тест на інформаційній системі з увімкненим модулем кешування даних.

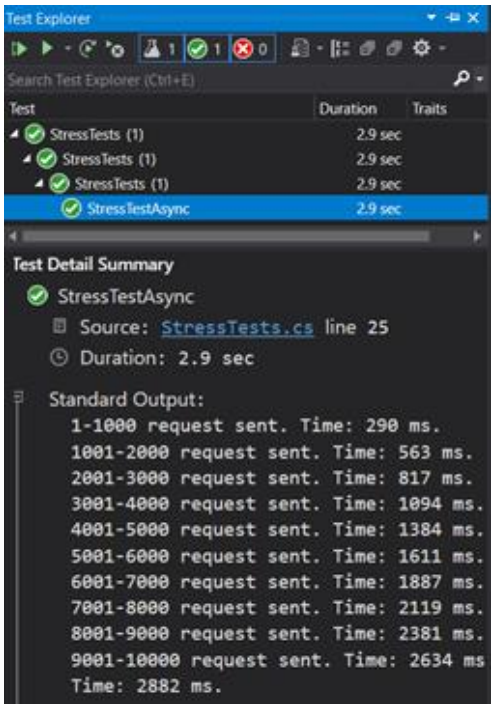

Рисунок 4.10 — Виконання тесту на інформаційній системі з модулем кешування

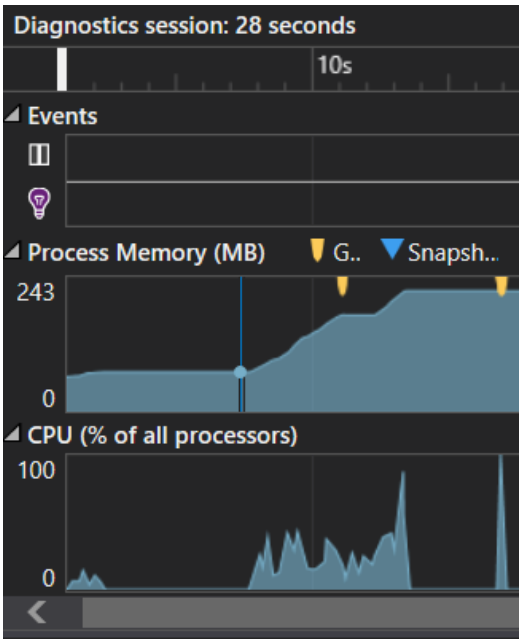

Рисунок 4.11 — Графік витрат комп’ютерних ресурсів з модулем кешування

Як можна побачити на рисунку 4.10 на виконання тесту знадобилося 2.8 секунди, що значно швидше ніж при запуску без модулю кешування. Також було витрачено суттєво менше оперативної пам’яті та процесорних ресурсів.

З проведених дослідів можна зробити висновок що модуль кешування сильно впливає на роботу інформаційної системи, та підвищує ефективність її роботи на 890 відсотків. Такий приріст продуктивності можливий у оптимальних умовах, адже як згадувалося раніше, продуктивність інформаційної системи залежить від об’ємів даних, дисперсії запитів, затримки мережі та інших факторів. В порівнянні зі стандартним методом кешування ASP.NET Core ми отримали продуктивність вищу на 30%, а головне, дані збережені в кеші залишаються актуальними на всьому циклі роботи інформаційної системи.

Отже у четвертому розділі було продемонстровано роботу методів інформаційної системи та перевірено коректність їх виконання. Також було створено проект стрес тестування системи за допомогою фреймворку NUnit, та протестовано інформаційну систему під високим навантаженням з увімкненим модулем кешування даних та без. Були проаналізовані результати дослідження та доведена ефективність роботи модулю кешування даних.

5 ЕКОНОМІЧНА ЧАСТИНА

Науково-технічна розробка має право на існування та впровадження, якщо вона відповідає вимогам часу, як в напрямку науково-технічного прогресу та і в плані економіки. Тому для науково-дослідної роботи необхідно оцінювати економічну ефективність результатів виконаної роботи.

Магістерська кваліфікаційна робота з розробки та дослідження «Методи та програмний засіб комплексного кешування даних у високонавантажених інформаційних системах» відноситься до науково-технічних робіт, які орієнтовані на виведення на ринок (або рішення про виведення науково-технічної розробки на ринок може бути прийнято у процесі проведення самої роботи), тобто коли відбувається так звана комерціалізація науково-технічної розробки. Цей напрямок є пріоритетним, оскільки результатами розробки можуть користуватися інші споживачі, отримуючи при цьому певний економічний ефект. Але для цього потрібно знайти потенційного інвестора, який би взявся за реалізацію цього проекту і переконати його в економічній доцільності такого кроку.

Для наведеного випадку нами мають бути виконані такі етапи робіт:

— проведено комерційний аудит науково-технічної розробки, тобто встановлення її науково-технічного рівня та комерційного потенціалу;

— розраховано витрати на здійснення науково-технічної розробки;

— розрахована економічна ефективність науково-технічної розробки у випадку її впровадження і комерціалізації потенційним інвестором і проведено обґрунтування економічної доцільності комерціалізації потенційним інвестором.

5.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Метою проведення комерційного і технологічного аудиту дослідження за темою «Методи та програмний засіб комплексного кешування даних у високонавантажених інформаційних системах» є оцінювання науково-технічного рівня та рівня комерційного потенціалу розробки, створеної в результаті науково-

технічної діяльності.

Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням 5-ти бальної системи оцінювання за 12-ма критеріями оцінювання науково-технічного рівня і комерційного потенціалу розробки наведеними в таблиці 5.1. [23]

Таблиця 5.1 — Рекомендовані критерії оцінювання науково-технічного рівня і комерційного потенціалу розробки та бальна оцінка

Бали (за 5-ти бальною шкалою)					
	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 працездатність 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
4	Технічні та споживчі властивості продукту значно гірші, ніж в	Технічні та споживчі влас-тивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в	Технічні та споживчі властивості продукту значно кращі, ніж в
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні 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

Закінчення таблиці 5.1

Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання 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 використовуються у військово промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуютьс я у виробництві
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	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту 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 науково-технічного рівня та комерційного потенціалу науково-технічної розробки необхідно звести до таблиці.

Таблиця 5.2 — Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

Критерії	Експерт (ПІБ, посада)		
	1	2	3
	Бали:		
1. Технічна здійсненність концепції	5	5	5
2. Ринкові переваги (наявність аналогів)	3	2	3
3. Ринкові переваги (ціна продукту)	3	4	3
4. Ринкові переваги (технічні властивості)	3	3	3
5. Ринкові переваги (експлуатаційні витрати)	1	0	2
6. Ринкові перспективи (розмір ринку)	3	3	3
7. Ринкові перспективи (конкуренція)	4	4	4
8. Практична здійсненність (наявність фахівців)	4	5	4
9. Практична здійсненність (наявність фінансів)	3	4	4
10. Практична здійсненність (необхідність нових матеріалів)	4	4	4
11. Практична здійсненність (термін реалізації)	4	3	4
12. Практична здійсненність (розробка документів)	4	3	3
Сума балів	41	40	42
Середньоарифметична сума балів $СБ_c$	41,0		

За результатами розрахунків, наведених в таблиці 5.2, зробимо висновок щодо науково-технічного рівня і рівня комерційного потенціалу розробки. При цьому використаємо рекомендації, наведені в табл. 5.3 [23].

Таблиця 5.3 — Науково-технічні рівні та комерційні потенціали розробки

Середньоарифметична сума балів СБ, розрахована на основі висновків експертів	Науково-технічний рівень та комерційний потенціал розробки
41…48	Високий
31…40	Вище середнього
21…30	Середній
11…20	Нижче середнього
0…10	Низький

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Методи та програмний засіб комплексного кешування даних у високонавантажених інформаційних системах» становить 41,0 бала, що, відповідно до таблиці 5.3, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки високий).

5.2 Визначення рівня конкурентоспроможності розробки

В процесі визначення економічної ефективності науково-технічної розробки також доцільно провести прогноз рівня її конкурентоспроможності за сукупністю параметрів, що підлягають оцінюванню.

Одиничний параметричний індекс розраховуємо за формулою [23]:

$$q_i = \frac{P_i}{P_{базі}}. \quad (5.1)$$

де q_i — одиничний параметричний індекс, розрахований за i-м параметром;

P_i — значення i-го параметра виробу;

$P_{базі}$ — аналогічний параметр базового виробу-аналога, з яким проводиться порівняння.

Загальні технічні та економічні характеристики розробки представлено в таблиці 5.4.

Таблиця 5.4 — Основні техніко-економічні показники аналога та розробки, що проектується

Показники (параметри)	Одиниця вимірю-вання	Аналог	Проектований пристрій	Відношення параметрів нової розробки до аналога	Питома вага показника
Базове підвищення ефективності роботи інформаційної системи	%	48	73	1,52	0,3
Щільність заповнення сформованої бази даних	%	85	95	1,12	0,12
Доступність інтерфейсу	бал	7	9	1,28	0,15

Закінчення таблиці 5.4

Кешовані мультимедійні ресурси	шт.	12	18	1,5	0,25
Захищеність БД від зовнішнього впливу (до 10 балів)	бал	7	7	1	0,18
Експлуатаційні витрати	грн	210	150	0,71	0,45
Ціна	грн	2800	1700	0,61	0,55

Нормативні параметри оцінюємо показником, який отримує одне з двох значень: 1 — пристрій відповідає нормам і стандартам; 0 — не відповідає.

Груповий показник конкурентоспроможності за нормативними параметрами розраховуємо як добуток частинних показників за кожним параметром за формулою [23]:

$$I_{НП} = \prod_{i=1}^{n} q_i \ , \tag{5.2}$$

де $I_{нп}$ — показник конкурентоспроможності за нормативними параметрами;

q_i — одиничний (частинний) показник за i-м нормативним параметром;

n — кількість нормативних параметрів, які підлягають оцінюванню.

За нормативними параметрами розроблюваний пристрій відповідає вимогам ДСТУ, тому $I_{нп} = 1$.

Значення групового параметричного індексу за технічними параметрами визначаємо з урахуванням вагомості (частки) кожного параметра [23]:

$$I_{ТП} = \sum_{i=1}^{n} q_i \cdot \alpha_i \ , \tag{5.3}$$

де $I_{тп}$ — груповий параметричний індекс за технічними показниками;

q_i — одиничний параметричний показник i-го параметра;

α_i — вагомість *i*-го параметричного показника, $\sum_{i=1}^{n}\alpha_i = 1$;

n — кількість технічних параметрів, за якими оцінюється конкурентоспроможність.

Проведемо аналіз параметрів згідно даних таблиці 5.4.

$$I_{mn} = 1{,}52 \cdot 0{,}3 + 1{,}12 \cdot 0{,}12 + 1{,}28 \cdot 0{,}15 + 1{,}5 \cdot 0{,}25 + 1 \cdot 0{,}18 = 1{,}34.$$

Груповий параметричний індекс за економічними параметрами розраховуємо за формулою [23]:

$$I_{ЕП} = \sum_{i=1}^{m} q_i \cdot \beta_i \quad , \tag{5.4}$$

де $I_{ЕП}$ — груповий параметричний індекс за економічними показниками;

q_i — економічний параметр *i*-го виду;

β_i — частка *i*-го економічного параметра, $\sum_{i=1}^{m}\beta_i = 1$;

m — кількість економічних параметрів, за якими здійснюється оцінювання.

Проведемо аналіз параметрів згідно даних таблиці.

$$I_{ЕП} = 0{,}71 \cdot 0{,}45 + 0{,}61 \cdot 0{,}55 = 0{,}66.$$

На основі групових параметричних індексів за нормативними, технічними та економічними показниками розрахуємо інтегральний показник конкурентоспроможності за формулою [23]:

$$K_{ІНТ} = I_{НП} \cdot \frac{I_{ТП}}{I_{ЕП}} \quad , \tag{5.5}$$

$$К_{ІНТ} = 1 \cdot 1{,}34 / 0{,}66 = 2{,}04.$$

Інтегральний показник конкурентоспроможності $К_{ІНТ} > 1$, отже розробка переважає відомі аналоги за своїми техніко-економічними показниками.

5.3 Розрахунок витрат на проведення науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи на тему «Методи та програмний засіб комплексного кешування даних у високонавантажених інформаційних системах», під час планування, обліку і калькулювання собівартості науково-дослідної роботи групуємо за відповідними статтями.

5.3.1 Витрати на оплату праці

До статті «Витрати на оплату праці» належать витрати на виплату основної та додаткової заробітної плати керівникам відділів, лабораторій, секторів і груп, науковим, інженерно-технічним працівникам, конструкторам, технологам, креслярам, копіювальникам, лаборантам, робітникам, студентам, аспірантам, інженерно-технічним працівникам, конструкторам, та іншим працівникам, безпосередньо зайнятим виконанням конкретної теми, обчисленої за посадовими окладами, відрядними розцінками, тарифними ставками згідно з чинними в організаціях системами оплати праці.

Витрати на основну заробітну плату дослідників ($З_о$) розраховуємо у відповідності до посадових окладів працівників, за формулою [23]:

$$З_о = \sum_{i=1}^{k} \frac{M_{ni} \cdot t_i}{T_p}, \tag{5.6}$$

де k — кількість посад дослідників залучених до процесу досліджень;

M_{ni} — місячний посадовий оклад конкретного дослідника, грн;

t_i — число днів роботи конкретного дослідника, дн.;

T_p — середнє число робочих днів в місяці, T_p=24 дні.

$$З_о = 13800{,}00 \cdot 24 / 24 = 13800{,}00 \; грн.$$

Проведені розрахунки зведемо до таблиці витрат на заробітну плату дослідників (таблиця 5.5).

Таблиця 5.5 — Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на заробітну плату, грн
Керівник проекту	13800,00	575,00	24	13800,00
Інженер-розробник програмного забезпечення	12100,00	504,17	16	8066,67
Інженер-розробник інформаційних систем	12100,00	504,17	15	7562,50
Лаборант	6540,00	272,50	7	1907,50
Всього				31336,67

Основна заробітна плата робітників

Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт НДР на тему «Методи та програмний засіб комплексного кешування даних у високонавантажених інформаційних системах» розраховуємо за формулою:

$$З_р = \sum_{i=1}^{n} C_i \cdot t_i \quad , \tag{5.7}$$

де C_i — погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

t_i — час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду C_i можна визначити за формулою:

$$C_i = \frac{M_M \cdot K_i \cdot K_c}{T_p \cdot t_{зм}} \quad , \tag{5.8}$$

де M_M — розмір прожиткового мінімуму працездатної особи, або мінімальної місячної заробітної плати (в залежності від діючого законодавства), приймемо $Мм$=2379,00 грн;

K_i — коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду;

$К_с$ — мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об'єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати.

$Т_р$ — середнє число робочих днів в місяці, приблизно $Т_р$ = 24 дн;

$t_{зм}$ — тривалість зміни, год.

$С_1$ = 2379,00 · 1,10 · 1,65 / *(24 · 8)* = 22,49 *грн.*

$З_{р1}$ = 22,49 · 6,00 = 134,93 *грн.*

Таблиця 5.6 — Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Тарифний коефіцієнт	Погодинна тарифна ставка, грн	Величина оплати на робітника грн
Установка обладнання для розробки програмного забезпечення	6,00	2	1,10	22,49	134,93
Підготовка робочого місця розробника програмного забезпечення	5,50	2	1,10	22,49	123,69
Інсталяція програмного забезпечення для моделювання та розробки	4,31	5	1,70	34,76	149,80
Компіляція програмних блоків	6,50	4	1,50	30,67	199,33
Налагодження програмних блоків	2,75	6	2,00	40,89	112,44
Тестування програмного засобу	8,00	2	1,10	22,49	179,91
Всього					900,11

Додаткова заробітна плата дослідників та робітників

Додаткову заробітну плату розраховуємо як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою:

$$З_{дод} = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%}, \quad (5.9)$$

де $Н_{дод}$ — норма нарахування додаткової заробітної плати. Приймемо 11%.

$З_{дод}$ = (31336,67 + 900,11) · 11 / 100% = 3546,05 грн.

5.3.2 Відрахування на соціальні заходи

Нарахування на заробітну плату дослідників та робітників розраховуємо як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%} \quad (5.10)$$

де $Н_{зп}$ — норма нарахування на заробітну плату. Приймаємо 22%.

Зн = (31336,67 + 900,11 + 3546,05) · 22 / 100% = 7872,22 грн.

5.3.3 Сировина та матеріали

До статті «Сировина та матеріали» належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби і предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень за темою «Методи та програмний засіб комплексного кешування даних у високонавантажених інформаційних системах».

Витрати на матеріали ($М$), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою:

$$М = \sum_{j=1}^{n} Н_j \cdot Ц_j \cdot К_j - \sum_{j=1}^{n} В_j \cdot Ц_{в\,j}, \quad (5.11)$$

де $Н_j$ — норма витрат матеріалу j-го найменування, кг;

n — кількість видів матеріалів;

$Ц_j$ — вартість матеріалу j-го найменування, грн/кг;

$К_j$ — коефіцієнт транспортних витрат, ($К_j = 1,1 \ ... \ 1,15$);

$В_j$ — маса відходів j-го найменування, кг;

$Ц_{вj}$ — вартість відходів j-го найменування, грн/кг.

$$М_1 = 4,00 \cdot 128,00 \cdot 1,1 - 0,000 \cdot 0,00 = 563,20 \ грн.$$

Проведені розрахунки зведемо до таблиці.

Таблиця 5.7 — Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за 1 кг, грн	Норма витрат, кг	Величина відходів, кг	Ціна відходів, грн/кг	Вартість витраченого матеріалу, грн
Офісний папір SOCRAT Ultra Plus	128,00	4,00	0,000	0,00	563,20
Папір для записів Parers Light A5	55,00	3,00	0,000	0,00	181,50
Органайзер офісний OFFICE Kozak	188,00	2,00	0,000	0,00	413,60
Канцелярське приладдя (набір офісного працівника)	203,00	4,00	0,000	0,00	893,20
Картридж для принтера Erixon EZ2500	506,00	2,00	0,000	0,00	1113,20
Диск оптичний NewOPtice CD-RW	11,80	4,00	0,000	0,00	51,92
Flesh-пам'ять Kingston 32 GB	230,00	1,00	0,000	0,00	253,00
Тека для паперів BOX-ZX	112,00	3,00	0,000	0,00	369,60
Всього					3839,22

5.3.4 Розрахунок витрат на комплектуючі

Витрати на комплектуючі *($К_в$)*, які використовують при проведенні НДР на

тему «Методи та програмний засіб комплексного кешування даних у високонавантажених інформаційних системах» відсутні.

5.3.5 Спецустаткування для наукових (експериментальних) робіт

До статті «Спецустаткування для наукових (експериментальних) робіт» належать витрати на виготовлення та придбання спецустаткування необхідного для проведення досліджень, також витрати на їх проектування, виготовлення, транспортування, монтаж та встановлення.

Балансову вартість спецустаткування розраховуємо за формулою:

$$B_{спец} = \sum_{i=1}^{k} Ц_i \cdot C_{пр.i} \cdot K_i \quad , \qquad (5.12)$$

де $Ц_i$ — ціна придбання одиниці спецустаткування даного виду, марки, грн;

$C_{пр.i}$ — кількість одиниць устаткування відповідного найменування, які придбані для проведення досліджень, шт.;

K_i — коефіцієнт, що враховує доставку, монтаж, налагодження устаткування тощо, ($K_i = 1,10…1,12$);

k — кількість найменувань устаткування.

$B_{спец} = 26298,00 \cdot 1 \cdot 1,09 = 28664,82$ *грн.*

Отримані результати зведемо до таблиці:

Таблиця 5.8 — Витрати на придбання спецустаткування по кожному виду

Найменування устаткування	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Ноутбук Dell Inspiron G5 15 5587 Black	1	26298,00	28664,82
Всього			28664,82

5.3.6 Програмне забезпечення для наукових робіт

До статті «Програмне забезпечення для наукових (експериментальних) робіт» належать витрати на розробку та придбання спеціальних програмних засобів

і програмного забезпечення, (програм, алгоритмів, баз даних) необхідних для проведення досліджень, також витрати на їх проектування, формування та встановлення.

Балансову вартість програмного забезпечення розраховуємо за формулою:

$$B_{прг} = \sum_{i=1}^{k} Ц_{iпрг} \cdot C_{прг.i} \cdot K_i \quad , \tag{5.13}$$

де $Ц_{iпрг}$ — ціна придбання одиниці програмного засобу даного виду, грн;

$C_{прг.i}$ — кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

K_i — коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, ($K_i = 1,10…1,12$);

k — кількість найменувань програмних засобів.

$B_{прг} = 8520,00 \cdot 1 \cdot 1,1 = 9372,00$ *грн.*

Отримані результати зведемо до таблиці:

Таблиця 5.9 — Витрати на придбання програмних засобів по кожному виду

Найменування програмного засобу	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Середовище розробки Visual Studio 2019 Pro	1	8520,00	9372,00
Всього			9372,00

5.3.7 Амортизація обладнання, програмних засобів та приміщень

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, розраховуємо з використанням прямолінійного методу амортизації за формулою:

$$A_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12}, \tag{5.14}$$

де $Ц_б$ — балансова вартість обладнання, програмних засобів, приміщень тощо, які

використовувались для проведення досліджень, грн;

$t_{вик}$ — термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$T_в$ — строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

$$А_{обл} = (25640{,}00 \cdot 1) / (2 \cdot 12) = 1068{,}33 \text{ грн.}$$

Проведені розрахунки зведемо до таблиці.

Таблиця 5.10 — Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Персональний комп'ютер розробника ПЗ	25640,00	2	1	1068,33
Робоче місце інженера-програміста	8250,00	5	1	137,50
Пристрої передачі даних	6540,00	4	1	136,25
Оргтехніка	7360,00	4	1	153,33
Приміщення лабораторії розробки інформаційних систем	195600,00	25	1	652,00
ОС Windows 11	7910,00	2	1	329,58
Прикладний пакет Microsoft Office 2019	6450,00	2	1	268,75
Всього				2745,75

5.3.8 Паливо та енергія для науково-виробничих цілей

Витрати на силову електроенергію ($B_е$) розраховуємо за формулою:

$$B_e = \sum_{i=1}^{n} \frac{W_{yi} \cdot t_i \cdot Ц_e \cdot K_{вni}}{\eta_i}, \quad (5.15)$$

де W_{yi} — встановлена потужність обладнання на визначеному етапі розробки, кВт;

t_i — тривалість роботи обладнання на етапі дослідження, год;

$Ц_e$ — вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії), приймемо $Ц_e = 4,50$ грн;

$K_{вni}$ — коефіцієнт, що враховує використання потужності, $K_{вni} < 1$;

η_i — коефіцієнт корисної дії обладнання, $\eta_i < 1$.

$B_e = 0,45 \cdot 160,0 \cdot 4,50 \cdot 0,95 / 0,97 = 324,00$ *грн.*

Проведені розрахунки зведемо до таблиці.

Таблиця 5.11 — Витрати на електроенергію

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
Персональний комп'ютер розробника ПЗ	0,45	160,0	324,00
Робоче місце інженера-програміста	0,25	160,0	180,00
Пристрої передачі даних	0,03	20,0	2,70
Оргтехніка	0,65	10,0	29,25
Ноутбук Dell Inspiron G5 15 5587 Black	0,09	80,0	32,40
Всього			568,35

5.3.9 Службові відрядження

До статті «Службові відрядження» дослідної роботи на тему «Методи та програмний засіб комплексного кешування даних у високонавантажених інформаційних системах» належать витрати на відрядження штатних працівників, працівників організацій, які працюють за договорами цивільно-правового характеру, аспірантів, 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.

Витрати за статтею «Службові відрядження» розраховуємо як 20…25% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{св} = (З_о + З_р) \cdot \frac{H_{св}}{100\%}, \quad (5.16)$$

де $H_{св}$ — норма нарахування за статтею «Службові відрядження», приймемо $Н_{св}$ = 20%.

$$В_{св} = (31336,67 + 900,11) \cdot 20 / 100\% = 6447,36 \text{ грн.}$$

Витрати на роботи, які виконують сторонні підприємства, установи і організації

Витрати за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації» відсутні.

5.3.10 Інші витрати

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками.

Витрати за статтею «Інші витрати» розраховуємо як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$I_в = (З_о + З_р) \cdot \frac{H_{ів}}{100\%}, \quad (5.17)$$

де $H_{ів}$ — норма нарахування за статтею «Інші витрати», приймемо $Н_{ів}$ = 50%.

$$І_в = (31336,67 + 900,11) \cdot 50 / 100\% = 16118,39 \text{ грн.}$$

5.3.11 Накладні (загальновиробничі) витрати

До статті «Накладні (загальновиробничі) витрати» належать: витрати,

пов’язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов’язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов’язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін.

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуємо як 100…150% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (5.18)$$

де $Н_{нзв}$ — норма нарахування за статтею «Накладні (загальновиробничі) витрати», приймемо $Н_{нзв}$ = 100%.

$B_{нзв} = (31336,67 + 900,11) \cdot 100 / 100\% = 32236,78$ *грн.*

Витрати на проведення науково-дослідної роботи на тему «Методи та програмний засіб комплексного кешування даних у високонавантажених інформаційних системах» розраховуємо як суму всіх попередніх статей витрат за формулою:

$$B_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + B_{спец} + B_{прг} + A_{обл} + B_е + B_{св} + B_{сп} + І_в + B_{нзв}. \quad (5.19)$$

$B_{заг}$*= 31336,67 +900,11 +3546,05 +7872,221198 +3839,22 +0,00 +28664,82 + 9372,00 + 2745,75 +568,35 +6447,36 +0,00 +16118,39 +32236,78 = 143647,71 грн.*

Загальні витрати *ЗВ* на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховується за формулою:

$$ЗВ = \frac{B_{заг}}{\eta}, \quad (5.20)$$

де η - коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної

роботи, приймемо η =0,9.

$$ЗВ = 143647,71 / 0,9 = 159608,56 \text{ грн.}$$

5.4 Розрахунок економічної ефективності науково-технічної розробки при її можливій комерціалізації потенційним інвестором

В ринкових умовах узагальнюючим позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку.

Результати дослідження проведені за темою «Методи та програмний засіб комплексного кешування даних у високонавантажених інформаційних системах» передбачають комерціалізацію протягом 4-х років реалізації на ринку.

В цьому випадку майбутній економічний ефект буде формуватися на основі таких даних:

ΔN — збільшення кількості споживачів продукту, у періоди часу, що аналізуються, від покращення його певних характеристик;

N — кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки, приймемо 9600 осіб;

$Ц_б$ — вартість програмного продукту у році до впровадження результатів розробки, приймемо 1500,00 грн;

$\pm\Delta Ц_о$ — зміна вартості програмного продукту від впровадження результатів науково-технічної розробки, приймемо 200,00 грн.

Можливе збільшення чистого прибутку у потенційного інвестора $\Delta П_i$ для кожного із 4-х років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховуємо за формулою [24]:

$$\Delta П_i = (\pm \Delta Ц_o \cdot N + Ц_o \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \quad (5.21)$$

де λ — коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2021 році ставка податку на додану вартість складає 20%, а коефіцієнт $\lambda = 0,8333$;

ρ — коефіцієнт, який враховує рентабельність інноваційного продукту). Приймемо ρ =40%;

ϑ — ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2021 році $\vartheta = 18\%$;

Збільшення чистого прибутку 1-го року:

$\Delta П_1$ =(200,00·9600,00+1700,00·1200)·0,83·0,4·(1-0,18/100%)=1078070,40 грн.

Збільшення чистого прибутку 2-го року:

$\Delta П_2$ =(200,00·9600,00+1700,00·4000)·0,83·0,4·(1-0,18/100%)=2373932,80 грн.

Збільшення чистого прибутку 3-го року:

$\Delta П_3$ =(200,00·9600,00+1700,00·7400)·0,83·0,4·(1-0,18/100%)=3947480,00 грн.

Збільшення чистого прибутку 4-го року:

$\Delta П_4$ =(200,00·9600,00+1700,00·11400)·0,83·0,4·(1-0,18/100%)=5798712,00 грн.

Приведена вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \quad (5.22)$$

де $\Delta П_i$ — збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

T — період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки;

τ — ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ =0,12;

t — період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$ПП = 1078070,4/(1+0,12)^1+2373932/(1+0,12)^2+394748/(1+0,12)^3+5798712/(1+0,12)^4=$
$=962562,86+1892484,69+2809738,29+3685186,31=9349972,15$ грн.

Величина початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки:

$$PV = k_{інв} \cdot ЗВ, \tag{5.23}$$

де $k_{інв}$ — коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, приймаємо $k_{інв}$ =1,5;

$ЗВ$ — загальні витрати на проведення науково-технічної розробки та оформлення її результатів, приймаємо 159608,56 грн.

$$PV = k_{інв} \cdot ЗВ = 1,5 \cdot 159608,56 = 239412,85 \text{ грн.}$$

Абсолютний економічний ефект $E_{абс}$ для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV \tag{5.24}$$

де $ПП$ — приведена вартість зростання всіх чистих прибутків від можливого

впровадження та комерціалізації науково-технічної розробки, 9349972,15 грн;

PV — теперішня вартість початкових інвестицій, 239412,85 грн.

$$E_{абс} = ПП - PV = 9349972,15 - 239412,85 = 9110559,31 \text{ грн.}$$

Внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1, \quad (5.25)$$

де $E_{абс}$ — абсолютний економічний ефект вкладених інвестицій, 9110559,31 грн;

PV — теперішня вартість початкових інвестицій, 239412,85 грн;

$T_ж$ — життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, 4 роки.

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1 = (1+9110559,31/239412,85)^{1/4} = 1,50.$$

Мінімальна внутрішня економічна дохідність вкладених інвестицій $\tau_{мін}$:

$$\tau_{мін} = d + f, \quad (5.26)$$

де d — середньозважена ставка за депозитними операціями в комерційних банках; в 2021 році в Україні d =0,09;

f — показник, що характеризує ризикованість вкладення інвестицій, приймемо 0,3.

$\tau_{мін}$ = 0,09+0,3 = 0,39 < 1,50 свідчить про те, що внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки вища мінімальної внутрішньої

дохідності. Тобто інвестувати в науково-дослідну роботу за темою «Методи та програмний засіб комплексного кешування даних у високонавантажених інформаційних системах» доцільно.

Період окупності інвестицій $T_{ок}$ які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$T_{ок} = \frac{1}{E_в}, \tag{5.27}$$

де $E_в$ — внутрішня економічна дохідність вкладених інвестицій.

$T_{ок}$ = 1 / 1,50 = 0,67 р.

$T_{ок}$ < 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Отже згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Методи та програмний засіб комплексного кешування даних у високонавантажених інформаційних системах» становить 41,0 бала, що, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки високий).

При оцінюванні рівня конкурентоспроможності, згідно узагальненого коефіцієнту конкурентоспроможності розробки, науково-технічна розробка переважає існуючі аналоги приблизно в 2,04 рази.

Також термін окупності становить 0,67 р., що менше 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Отже можна зробити висновок про доцільність проведення науково-дослідної роботи за темою «Методи та програмний засіб комплексного кешування даних у високонавантажених інформаційних системах».

ВИСНОВКИ

У даній дипломній роботі було проведено дослідження методів кешування у високонавантажених інформаційних системах.

У першому розділі було розглянуто концепцію кешування даних та основні методи кешування даних. Також були розглянуті та проаналізовані сучасні технології що використовується для побудови систем кешування даних, проаналізовані архітектурні підходи до побудови інформаційної системи.

У другому розділі було проведене дослідження можливостей удосконалення методу кешування, розроблено та математично обґрунтовано доповнений алгоритм інвалідації даних у кеші для оптимізації витрат ресурсів інформаційної системи. Також було досліджено використання різних структури даних для кешування даних та проаналізовані асимптотичні функції складності алгоритмів операцій необхідних для кешування.

У третьому розділі було розглянуто та обрано оптимальні технології для програмної реалізації інформаційної системи та модулю кешування даних. Також був спроектована та реалізована інформаційна система. У розробленій інформаційній системи був імплементований та застосований метод комплексного кешування даних, спроектований та описаний у другому розділі даної роботи.

У четвертому розділі було протестовано роботу інформаційної системи та перевірено коректність їх виконання. Також було протестовано інформаційну систему під високим навантаженням з увімкненим модулем кешування даних та без. Були проаналізовані результати дослідження та доведена ефективність роботи модулю кешування даних.

У п’ятому розділі було проведено та проаналізовано дослідження рівню комерційного потенціалу розробки, а також економічно обґрунтовано доцільність проведення науково-дослідної роботи.

Таким чином, мету роботи, сформульовану у вступі, досягнуто, а кешування даних інформаційних систем є перспективним напрямком досліджень, і може суттєво підвищити продуктивність роботи інформаційної системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Аналіз методів кешування даних у високонавантажених інформаційних системах [Текст] Г. О. Горбачов // Молодь в науці: дослідження, проблеми, перспективи (МН-2021) Тез. доп. — Вінниця, 2021. — Режим доступу https://conferences.vntu.edu.ua/index.php/mn/mn2021/ paper/view/13144/11053.

2. Дослідження можливостей удосконалення методів кешування у інформаційних системах [Текст] Г. О. Горбачов // Молодь в науці: дослідження, проблеми, перспективи (МН-2022) Тез. доп. — Вінниця, 2021. — Режим доступу https://conferences.vntu.edu.ua/index.php/mn/mn2022/ paper/view/13144/11053.

3. Caching Overview Caching helps applications perform dramatically faster and cost significantly less at scale [Електронний ресурс] — Режим доступу до ресурсу: https://aws.amazon.com/caching/

4. What is a cache hit ratio [Електронний ресурс] — Режим доступу до ресурсу: https://www.cloudflare.com/learning/cdn/what-is-a-cache-hit-ratio/

5. Advanced topics on caching [Електронний ресурс] — Режим доступу до ресурсу: https://www.apollographql.com/docs/react/caching/advanced-topics/

6. Caching challenges and strategies [Електронний ресурс] — Режим доступу до ресурсу: https://aws.amazon.com/builders-library/caching-challenges-and-strategies/

7. Deploy ASP.NET Core apps to Azure [Електронний ресурс] — Режим доступу до ресурсу: https://medium.com/the-cloud-architect/patterns-for-resilient-architecture-part-4-85afa66d6341

8. What is Memcached? [Електронний ресурс] — Режим доступу до ресурсу: https://memcached.org/

9. Redis vs. Memcached: In-Memory Data Storage Systems [Електронний ресурс] — Режим доступу до ресурсу: https://alibaba-cloud.medium.com/redis-vs-memcached-in-memory-data-storage-systems-3395279b0941

10. Как работает кэш в Hibernate? [Електронний ресурс] — Режим доступу до ресурсу: http://akorsa.ru/2016/11/kak-rabotaet-kesh-v-hibernate/

11. Thomas E., Service-Oriented Architecture (SOA): Concepts, Technology, and Design —1-е вид.,. — Prentice Hall, 2005. — 792 с.

12. Ajay D. Kshemkalyani, Distributed Computing: Principles, Algorithms, and Systems — перевидання, — Cambridge University Press, 2011. — 756 с.

13. What is Microservices [Електронний ресурс]. Режим доступу: https://smartbear.com/solutions/microservices/

14. Richardson L., Ruby S. RESTful Web Services. — O'Reilly Media, 2007. — 454 с.

15. Miller E. RESTful Web Service Architecture. — indle Edition, First Edition 2011— 19 с.

16. Defining Stateful vs Stateless Web Services. [Електронний ресурс]. —Режим доступу: https://nordicapis.com/defining-stateful-vs-stateless-web-services/

17. An Introduction to Cache Invalidation [Електронний ресурс] — Режим доступу до ресурсу: https://foshttpcache.readthedocs.io/en/latest/invalidation-introduction.html

18. Analysis of Algorithms | Big-O analysis [Електронний ресурс] — Режим доступу до ресурсу: https://www.geeksforgeeks.org/analysis-algorithms-big-o-analysis/

19. Concurrency структуры в .net. ConcurrentDictionary изнутри [Електронний ресурс] — Режим доступу до ресурсу: https://habr.com/ru/post/245727/

20. All you need to know about using node.js [Електронний ресурс]. Режим доступу: https://mobidev.biz/blog/node-js-for-backend-development.

21. Плюсы и минусы программирования на Java [Електронний ресурс]. Режим доступу: https://medium.com/nuances-of-programming/.

22. Advantages and Disadvantages of Python Programming Language [Електронний ресурс]. Режим доступу: https://medium.com/@mindfiresolutions.usa/advantages-and-disadvantages-of-python-programming-language-fd0b394f2121.

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

24. Кавецький В. В. Економічне обґрунтування інноваційних рішень: практикум / В. В. Кавецький, В. О. Козловський, І. В. Причепа — Вінниця : ВНТУ, 2016. — 113 с.

ДОДАТОК А

Міністерство освіти та науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ ВНТУ
д.т.н., проф. О. Д. Азаров
"__" _____ 2021 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської кваліфікаційної роботи
«Методи та програмний засіб комплексного кешування даних у високонавантажених інформаційних системах»
08-23.МКР.004.00.000 ПЗ

Науковий керівник к.т.н., доц. каф. ОТ
_____ Ткаченко О. М.
Студент групи 1КІ-20м
_____ Горбачов Г. О.

Вінниця 2021

1 Підстави для використання магістерської кваліфікаційної роботи (МКР)

— актуальність розробки полягає у вдосконаленні методу кешування даних у високонавантажених інформаційних системах, в якій, на відміну від існуючих, реалізовано комплексну інвалідацію даних, що дозволило зменшити затрати комп'ютерних ресурсів та підвищити продуктивність роботи інформаційної системи;

— наказ про затвердження теми дипломної роботи від "06" березня 2021 року №75.

2 Мета і призначення МКР

Мета роботи — підвищення швидкості оброблення інформації у високонавантажених системах шляхом розробки та програмної реалізації методу комплексного кешування даних.

Призначення розробки — здійснити аналіз методів кешування даних та запропонувати кращий підхід до організації роботи з даними в високонавантажених інформаційних системах.

3 Джерела розробки ДР

Інтегроване середовище розробки Visual Studio 2019 Professional для платформи .NET Core та мови програмування C# з використанням Entity Framework та NUnit.

4 Технічні вимоги до виконання МКР

— наявність встановленого пакету .NET 5.0 SDK;

— наявність інструментів моніторингу виконання додатку .NET Core;

— наявність встановленого пакету Entity Framework;

— наявність встановленої бази даних SQL Server 2016;

— наявність програмного забезпечення для роботи з HTTP запитами.

5 Етапи МКР та очікувані результати

Робота виконується за вісім етапів, таблиця А.1.

Таблиця А.1 — Етапи виконання роботи.

№ етапу	Назва етапу	Термін виконання		Очікувані результати
		початок	кінець	
1	Постановка задачі роботи	10.09.22	15.09.22	Вступ
2	Аналіз методів кешування даних у інформаційних системах	26.09.21	18.10.21	Розділ 1, Підрозділ 2
3	Огляд існуючих програмних засобів кешування даних у інформаційних системах	19.10.21	03.11.21	Розділ 1
4	Математичне обґрунтування ефективності системи кешування з різними структурами даних	04.11.21	28.11.21	Розділ 2
5	Розробка інформаційної системи та модулю кешування	29.11.21	06.12.21	Розділ 3
6	Дослідження впливу модулю кешування на продуктивність системи	07.12.21	08.12.21	Розділ 4, додатки
7	Обґрунтування економічної доцільності розробки	09.12.21	12.12.21	Розділ 5
8	Оформлення пояснювальної записки	13.12.21	21.12.21	ПЗ, презентація

6. Матеріали, що подаються до захисту МКР

Пояснювальна записка МКР, графічні і ілюстративні матеріали, протокол попереднього захисту роботи на кафедрі, відзив наукового керівника, відзив опонента, протоколи проходження перевірки на плагіат, анотації українською та іноземною мовами, нормоконтроль про відповідність оформлення діючим вимогам.

7. Порядок контролю виконання та захисту МКР

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

8. Вимоги до оформлення МКР

Вимоги викладені в Положеннях про МКР ВНТУ 2021, ДСТУ 3974-2000 «Правила виконання дослідно-конструкторських робіт. Загальні положення» та діючого ГОСТ 2.114-95 ЕСКД.

Технічне завдання до виконання отримав _____ Горбачов Г. О.

ДОДАТОК Б

Схема методу кешування

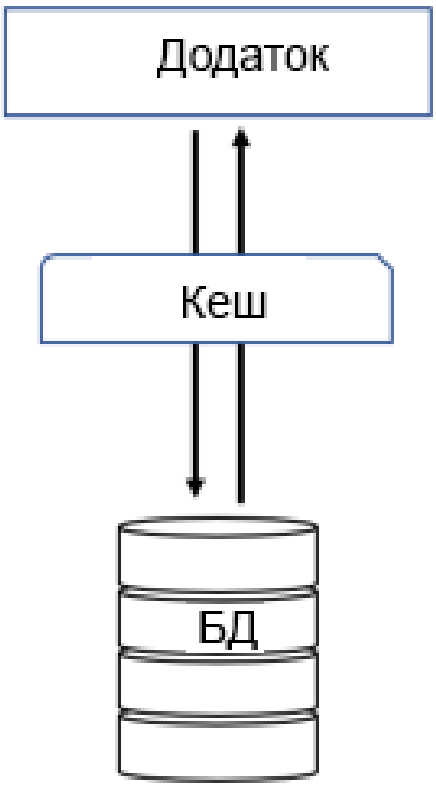

Рисунок Б.1 — Схема методу кешування

ДОДАТОК В

Асимптотичні функції складності алгоритмів

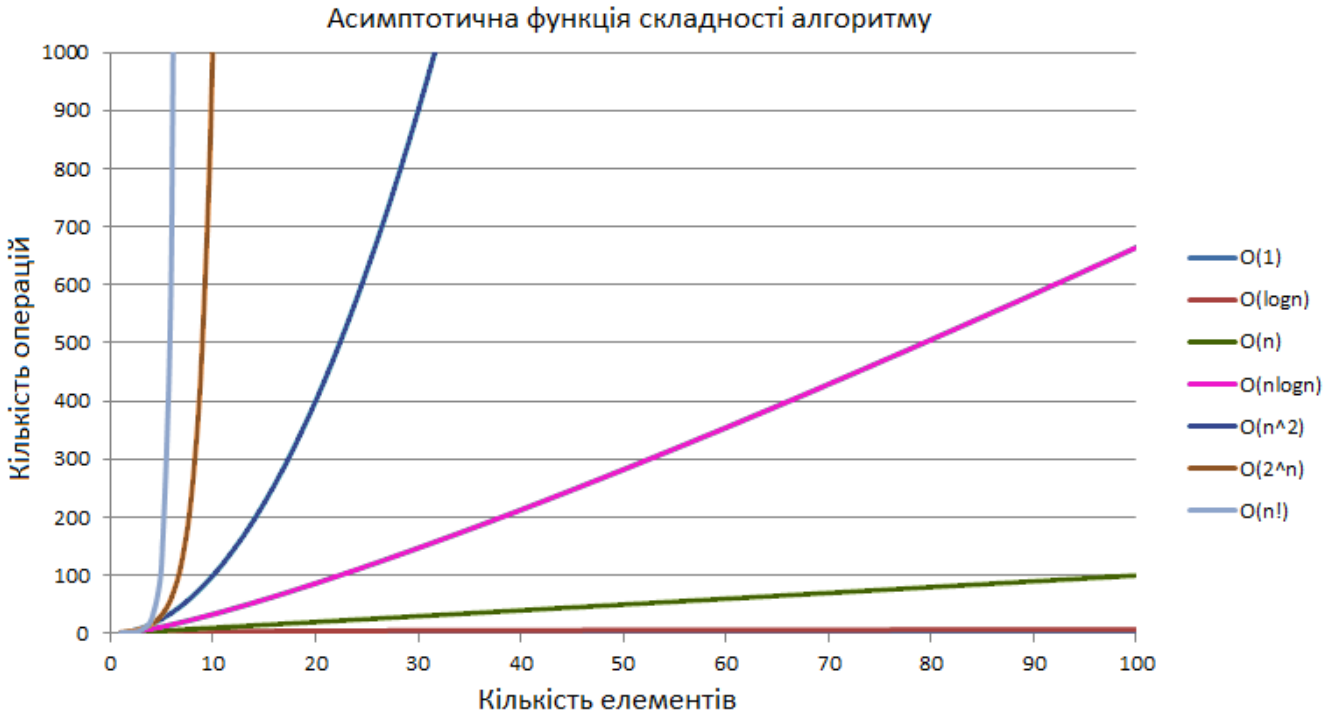

Рисунок В.1 — Асимптотичні функції складності алгоритмів

ДОДАТОК Г

Інтерактивна документація REST-сервісу Swagger

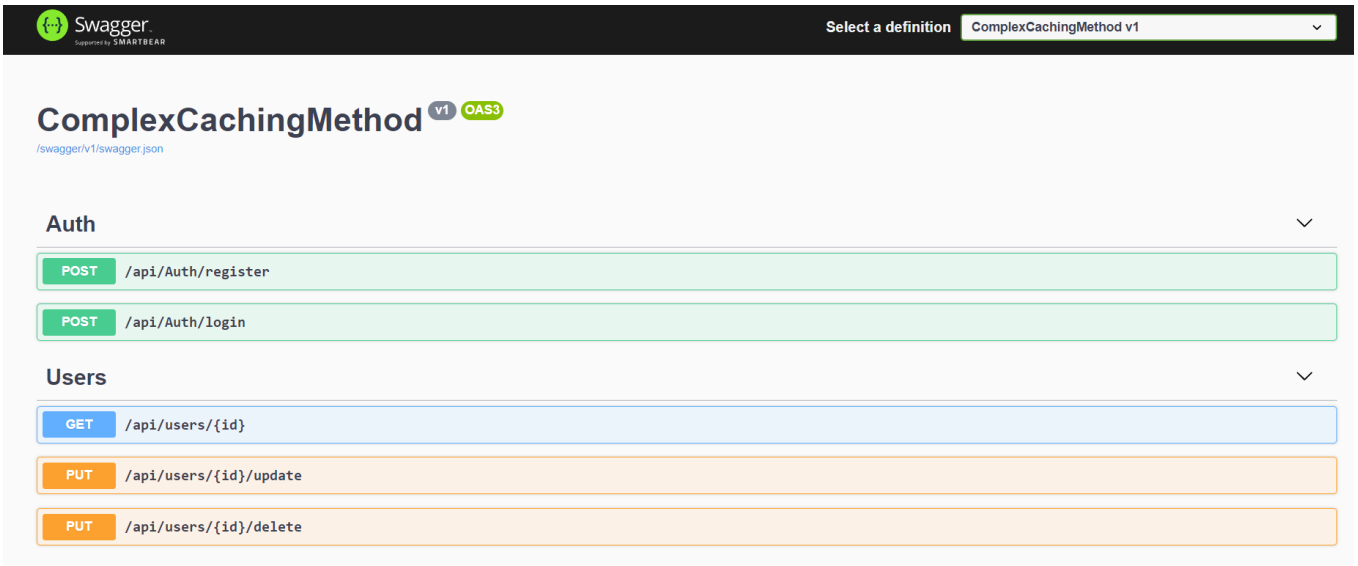

Рисунок Г.1 — Інтерактивна документація REST-сервісу Swagger

ДОДАТОК Д

Графіки витрат комп’ютерних ресурсів

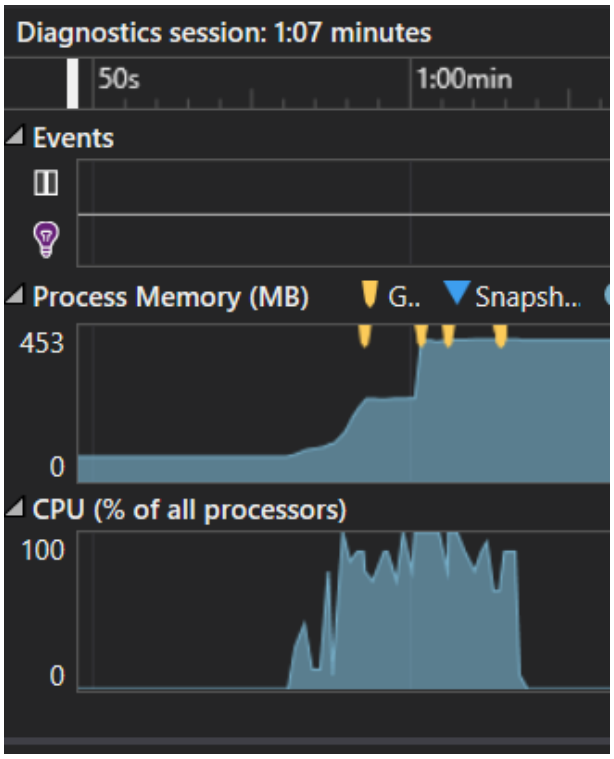

Рисунок Д.1 — Графік витрат комп’ютерних ресурсів без модулю кешування

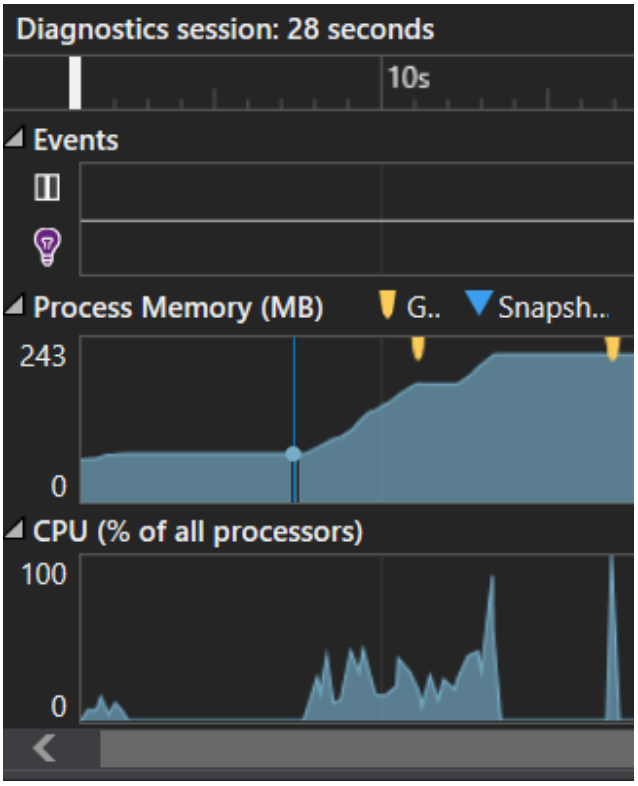

Рисунок Д.2 — Графік витрат комп’ютерних ресурсів з модулем кешування

ДОДАТОК Е

Графік розподілення трафіку інформаційної системи

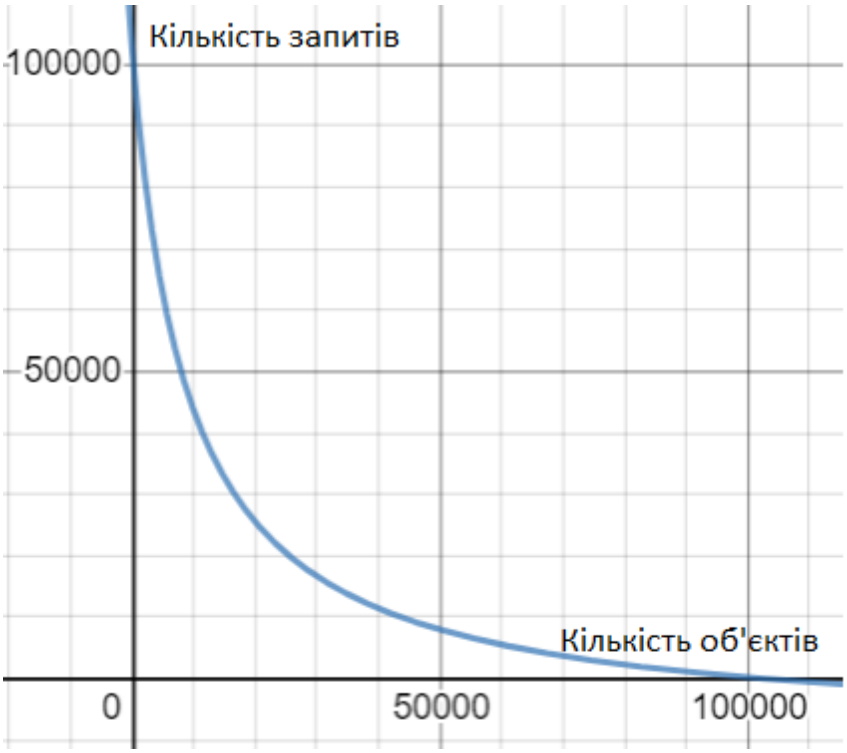

Рисунок Е.1 — Графік розподілення трафіку інформаційної системи

ДОДАТОК Ж

Лістинг коду програми

StartUp.cs

```
using ComplexCachingMethod.Data;
using ComplexCachingMethod.Data.Caches;
using ComplexCachingMethod.Data.Repositories;
using ComplexCachingMethod.Models;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.OpenApi.Models;

namespace ComplexCachingMethod
{
   public class Startup
   {
      public Startup(IConfiguration configuration)
      {
         Configuration = configuration;
      }

      public IConfiguration Configuration { get; }

      // This method gets called by the runtime. Use this method to add services to the
container.
      public void ConfigureServices(IServiceCollection services)
      {
         services.AddDbContext<DataContext>(options =>

options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));
         services.AddMemoryCache();
         services.AddControllers();
         services.AddAutoMapper(typeof(Startup));

         services.AddSingleton<ICache<User>, UserCache>();
         services.AddScoped<IAuthRepository, AuthRepository>();
         services.AddScoped<IRepository<User>, UserRepository>();
         services.AddSwaggerGen(c =>
```

```
        {
            c.SwaggerDoc("v1", new OpenApiInfo { Title = "ComplexCachingMethod",
Version = "v1" });
        });
    }

    // This method gets called by the runtime. Use this method to configure the HTTP
request pipeline.
    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)//,
DataContext dataContext)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
            app.UseSwagger();
            app.UseSwaggerUI(c => c.SwaggerEndpoint("/swagger/v1/swagger.json",
"ComplexCachingMethod v1"));
        }

        app.UseHttpsRedirection();

        app.UseRouting();

        app.UseAuthorization();

        app.UseEndpoints(endpoints =>
        {
            endpoints.MapControllers();
        });
        //var seedProvider = new SeedData(dataContext);
        //seedProvider.Seed(100000);
    }
  }
}
```

Program.cs

```
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Hosting;

namespace ComplexCachingMethod
{
  public class Program
  {
    public static void Main(string[] args)
```

```
        {
            CreateHostBuilder(args).Build().Run();
        }

        public static IHostBuilder CreateHostBuilder(string[] args) =>
            Host.CreateDefaultBuilder(args)
                .ConfigureWebHostDefaults(webBuilder =>
                {
                    webBuilder.UseStartup<Startup>();
                });
    }
}
```

AutoMapperProfiles.cs

```
using AutoMapper;
using System;
using ComplexCachingMethod.Dtos;
using ComplexCachingMethod.Models;

namespace ComplexCachingMethod.Common
{
    public class AutoMapperProfiles : Profile
    {
        public AutoMapperProfiles()
        {
            CreateMap<UserForLoginDto, User>();

            CreateMap<UserForRegisterDto, User>();

            CreateMap<UserForUpdateDto, User>();

            CreateMap<User, UserForReturnDto>()
                .ForMember(dest => dest.Age, src => src.MapFrom(u =>
CalculateAge(u.DateOfBirth)));
        }

        public int CalculateAge(DateTime theDateTime)
        {
            var age = DateTime.Today.Year - theDateTime.Year;
            if (theDateTime.AddYears(age) > DateTime.Today)
                age--;

            return age;
        }
```

```
    }
}
```

AuthController.cs

```
using AutoMapper;
using ComplexCachingMethod.Data.Repositories;
using ComplexCachingMethod.Dtos;
using ComplexCachingMethod.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Configuration;
using Microsoft.IdentityModel.Tokens;
using System;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;
using System.Threading.Tasks;

namespace DnDManager.API.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class AuthController : ControllerBase
    {
        private readonly IAuthRepository _repository;
        private readonly IConfiguration _config;
        private readonly IMapper _mapper;
        public AuthController(IAuthRepository repository, IConfiguration config, IMapper
mapper)
        {
            _mapper = mapper;
            _config = config;
            _repository = repository;
        }

        [HttpPost("register")]
        public async Task<IActionResult> Register(UserForRegisterDto
userForRegisterDto)
        {
            userForRegisterDto.Username = userForRegisterDto.Username.ToLower();

            var userToCreate = _mapper.Map<User>(userForRegisterDto);

            var createdUser = await _repository.Register(userToCreate,
userForRegisterDto.Password);
```

```
            var userToReturn = _mapper.Map<UserForReturnDto>(createdUser);

            //return CreatedAtRoute("GetUser", new { controller = "Users", id =
createdUser.Id }, userToReturn);
            return Ok(userToReturn);
        }

        [HttpPost("login")]
        public async Task<IActionResult> Login(UserForLoginDto userForLoginDto)
        {
            var userFromRepo = await
_repository.Login(userForLoginDto.Username.ToLower(),
userForLoginDto.Password);

            if (userFromRepo == null)
                return Unauthorized();

            var claims = new[]
            {
                new Claim(ClaimTypes.NameIdentifier, userFromRepo.Id.ToString()),
                new Claim(ClaimTypes.Name, userFromRepo.Username)
            };

            var key = new SymmetricSecurityKey(Encoding.UTF8
                .GetBytes(_config.GetSection("AppSettings:Token").Value));

            var creds = new SigningCredentials(key,
SecurityAlgorithms.HmacSha512Signature);

            var tokenDescriptor = new SecurityTokenDescriptor
            {
                Subject = new ClaimsIdentity(claims),
                Expires = DateTime.Now.AddDays(1),
                SigningCredentials = creds
            };

            var tokenHandler = new JwtSecurityTokenHandler();

            var token = tokenHandler.CreateToken(tokenDescriptor);

            var user = _mapper.Map<UserForReturnDto>(userFromRepo);

            return Ok(new
```

```
            {
                token = tokenHandler.WriteToken(token),
                user
            });
        }
    }
}
```

UsersController.cs

```
using AutoMapper;
using ComplexCachingMethod.Data.Repositories;
using ComplexCachingMethod.Dtos;
using ComplexCachingMethod.Models;
using Microsoft.AspNetCore.Mvc;
using System.Threading.Tasks;

namespace ComplexCachingMethod.Controllers
{
    [Route("api/users")]
    [ApiController]
    public class UsersController : ControllerBase
    {
        private readonly IRepository<User> _repository;
        private readonly IMapper _mapper;

        public UsersController(IRepository<User> repository, IMapper mapper)
        {
            _repository = repository;
            _mapper = mapper;
        }

        [HttpGet("{id}", Name = "GetUser")]
        [ResponseCache(NoStore = true, Location = ResponseCacheLocation.None)]
        public async Task<IActionResult> GetUser(int id)
        {
            var userFromRepo = await _repository.GetAsync(id, false);

            var user = _mapper.Map<UserForReturnDto>(userFromRepo);

            return Ok(user);
        }

        [HttpPut("{id}/update")]
```

```
        public async Task<IActionResult> UpdateUser(int id, UserForUpdateDto
userForUpdateDto)
        {
            var user = _mapper.Map<User>(userForUpdateDto);

            user = await _repository.UpdateAsync(id, user);

            if (user == null)
                return NotFound($"User {id} is not found");

            if (await _repository.SaveAsync())
                return Ok();

            return BadRequest($"User {id} update failed, you can't update it");
        }

        [HttpPut("{id}/delete")]
        public async Task<IActionResult> DeleteUser(int id)
        {
            var user = await _repository.GetAsync(id);
            if (user == null)
                return NotFound($"User {id} is not found, you can't delete it");

            _repository.Delete(user);

            if (await _repository.SaveAsync())
                return Ok();

            return BadRequest($"User update failed");
        }
    }
}
```

UserCache.cs

```
using ComplexCachingMethod.Models;
using Microsoft.Extensions.Caching.Memory;
using System;

namespace ComplexCachingMethod.Data.Caches
{
    public class UserCache : ICache<User>
    {
        private readonly IMemoryCache _cache;
```

```
        public UserCache(IMemoryCache cache)
        {
            _cache = cache;
        }

        public User Get(int id)
        {
            var key = GetCacheKey(id);
            var user = _cache.Get<User>(key);
            return user;
        }

        public void Set(User user)
        {
            if (user == null)
                return;

            var key = GetCacheKey(user.Id);
            var cacheExpiryOptions = new MemoryCacheEntryOptions
            {
                AbsoluteExpirationRelativeToNow = TimeSpan.FromMinutes(10),
                SlidingExpiration = TimeSpan.FromMinutes(2),
                Priority = CacheItemPriority.Normal
            };

            _cache.Set(key, user, cacheExpiryOptions);
        }

        public void Remove(int id)
        {
            var key = GetCacheKey(id);
            _cache.Remove(key);
        }

        private string GetCacheKey(int id)
        {
            return $"userid{id}";
        }
    }
}
```

AuthRepository.cs

```
u using ComplexCachingMethod.Models;
using Microsoft.EntityFrameworkCore;
```

```
using System.Threading.Tasks;

namespace ComplexCachingMethod.Data.Repositories
{
   public class AuthRepository : IAuthRepository
   {
      private readonly DataContext _context;
      public AuthRepository(DataContext context)
      {
         _context = context;
      }
      public async Task<User> Login(string username, string password)
      {
         var user = await _context.Users.FirstOrDefaultAsync(x => x.Username ==
username);

         if (user == null)
            return null;

         if (!VerifyPasswordHash(password, user.PasswordHash, user.PasswordSalt))
            return null;

         return user;
      }

      private bool VerifyPasswordHash(string password, byte[] passwordHash, byte[]
passwordSalt)
      {
         using (var hmac = new
System.Security.Cryptography.HMACSHA512(passwordSalt))
         {
            var computedHash =
hmac.ComputeHash(System.Text.Encoding.UTF8.GetBytes(password));
            for (int i = 0; i < computedHash.Length; i++)
            {
               if (computedHash[i] != passwordHash[i]) return false;
            }
            return true;
         }
      }

      public async Task<User> Register(User user, string password)
      {
         byte[] passwordHash, passwordSalt;
```

```
            CreatePasswordHash(password, out passwordHash, out passwordSalt);

            user.PasswordHash = passwordHash;
            user.PasswordSalt = passwordSalt;

            await _context.Users.AddAsync(user);

            await _context.SaveChangesAsync();

            return user;
        }

        private void CreatePasswordHash(string password, out byte[] passwordHash, out
byte[] passwordSalt)
        {
            using (var hmac = new System.Security.Cryptography.HMACSHA512())
            {
                passwordSalt = hmac.Key;
                passwordHash =
hmac.ComputeHash(System.Text.Encoding.UTF8.GetBytes(password));
            }
        }

        public async Task<bool> UserExists(string username)
        {
            if (await _context.Users.AnyAsync(x => x.Username == username))
                return true;

            return false;
        }
    }
}
```

UserRepository.cs

```
using ComplexCachingMethod.Data.Caches;
using ComplexCachingMethod.Models;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace ComplexCachingMethod.Data.Repositories
{
    public class UserRepository : IRepository<User>
    {
        private readonly DataContext _context;
```

```
private readonly ICache<User> _cache;
private List<Task> _tasks;
public UserRepository(DataContext context, ICache<User> cache)
{
    _context = context;
    _cache = cache;
    _tasks = new List<Task>();
}

public void Add(User user)
{
    _context.Users.Add(user);
}

public void Delete(User user)
{
    _context.Users.Remove(user);
    var invalidateCacheTask = new Task(() => _cache.Remove(user.Id));
    _tasks.Add(invalidateCacheTask);
}

public async Task<User> GetAsync(int id, bool useCache)
{
    User user;
    if (useCache)
    {
        user = _cache.Get(id);

        if (user != null)
            return user;
    }

    await Task.Delay(100);
    user = await _context.Users.FindAsync(id);
    _cache.Set(user);

    return user;
}


public async Task<bool> SaveAsync()
{
    bool success = await _context.SaveChangesAsync() > 0;
    if (success)
```

```
            {
                foreach(var task in _tasks)
                {
                    task.Start();
                }
                await Task.WhenAll(_tasks);
                _tasks.Clear();
            }

            return success;
        }

        public async Task<User> UpdateAsync(int id, User userForUpdate)
        {
            var user = await GetAsync(id, true);
            if (user == null)
                return null;

            user.KnownAs = userForUpdate.KnownAs;
            user.Interests = userForUpdate.Interests;
            user.City = userForUpdate.City;
            user.Country = userForUpdate.Country;

            var updateCacheTask = new Task(() => _cache.Set(user));
            _tasks.Add(updateCacheTask);

            return user;
        }
    }
}
```

UserForReturnDto.cs

```
using System;

namespace ComplexCachingMethod.Dtos
{
    public class UserForReturnDto
    {
        public int Id { get; set; }
        public string Username { get; set; }
        public string Gender { get; set; }
        public int Age { get; set; }
        public string KnownAs { get; set; }
        public string Interests { get; set; }
```

```
        public string City { get; set; }
        public string Country { get; set; }
        public string PhotoUrl { get; set; }
        public DateTime Created { get; set; }
        public DateTime LastActive { get; set; }
    }
}
```

UserForLoginDto.cs

```
namespace ComplexCachingMethod.Dtos
{
    public class UserForLoginDto
    {
        public string Username { get; set; }
        public string Password { get; set; }
    }
}
```

UserForRegisterDto.cs

```
using System;
using System.ComponentModel.DataAnnotations;

namespace ComplexCachingMethod.Dtos
{
    public class UserForRegisterDto
    {
        [Required]
        public string Username { get; set; }

        [Required]
        [StringLength(20, MinimumLength = 6, ErrorMessage = "You must specify
password between 4 and 8 characters")]
        public string Password { get; set; }

        [Required]
        public string Gender { get; set; }

        [Required]
        public string KnownAs { get; set; }

        [Required]
        public DateTime DateOfBirth { get; set; }
```

```
        [Required]
        public string City { get; set; }

        [Required]
        public string Country { get; set; }
        public DateTime Created { get; set; }
        public DateTime LastActive { get; set; }

        public UserForRegisterDto()
        {
            Created = DateTime.Now;
            LastActive = DateTime.Now;
        }
    }
}
```

UserForUpdateDto.cs

```
namespace ComplexCachingMethod.Dtos
{
    public class UserForUpdateDto
    {
        public string KnownAs { get; set; }
        public string Interests { get; set; }
        public string City { get; set; }
        public string Country { get; set; }
    }
}
```

User.cs

```
using System;

namespace ComplexCachingMethod.Models
{
    public class User
    {
        public int Id { get; set; }

        public string Username { get; set; }
        public byte[] PasswordHash { get; set; }
        public byte[] PasswordSalt { get; set; }
        public string Gender { get; set; }
        public string KnownAs { get; set; }
        public string Interests { get; set; }
```

```
        public string City { get; set; }
        public string Country { get; set; }
        public string PhotoUrl { get; set; }
        public DateTime DateOfBirth { get; set; }
        public DateTime Created { get; set; }
        public DateTime LastActive { get; set; }
    }
}
```

DataContextModelSnapshot.cs

```
using System;
using ComplexCachingMethod.Data;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Infrastructure;
using Microsoft.EntityFrameworkCore.Metadata;
using Microsoft.EntityFrameworkCore.Storage.ValueConversion;

namespace ComplexCachingMethod.Migrations
{
    [DbContext(typeof(DataContext))]
    partial class DataContextModelSnapshot : ModelSnapshot
    {
        protected override void BuildModel(ModelBuilder modelBuilder)
        {
#pragma warning disable 612, 618
            modelBuilder
                .HasAnnotation("Relational:MaxIdentifierLength", 128)
                .HasAnnotation("ProductVersion", "5.0.12")
                .HasAnnotation("SqlServer:ValueGenerationStrategy",
SqlServerValueGenerationStrategy.IdentityColumn);

            modelBuilder.Entity("ComplexCachingMethod.Models.User", b =>
                {
                    b.Property<int>("Id")
                        .ValueGeneratedOnAdd()
                        .HasColumnType("int")
                        .HasAnnotation("SqlServer:ValueGenerationStrategy",
SqlServerValueGenerationStrategy.IdentityColumn);

                    b.Property<string>("City")
                        .HasColumnType("nvarchar(max)");

                    b.Property<string>("Country")
                        .HasColumnType("nvarchar(max)");
```

```
                b.Property<DateTime>("Created")
                    .HasColumnType("datetime2");

                b.Property<DateTime>("DateOfBirth")
                    .HasColumnType("datetime2");

                b.Property<string>("Gender")
                    .HasColumnType("nvarchar(max)");

                b.Property<string>("Interests")
                    .HasColumnType("nvarchar(max)");

                b.Property<string>("KnownAs")
                    .HasColumnType("nvarchar(max)");

                b.Property<DateTime>("LastActive")
                    .HasColumnType("datetime2");

                b.Property<byte[]>("PasswordHash")
                    .HasColumnType("varbinary(max)");

                b.Property<byte[]>("PasswordSalt")
                    .HasColumnType("varbinary(max)");

                b.Property<string>("PhotoUrl")
                    .HasColumnType("nvarchar(max)");

                b.Property<string>("Username")
                    .HasColumnType("nvarchar(max)");

                b.HasKey("Id");

                b.ToTable("Users");
            });
#pragma warning restore 612, 618
        }
    }
}
```

SeedData.cs

```
using ComplexCachingMethod.Data;
using ComplexCachingMethod.Models;
using System;
```

```
namespace ComplexCachingMethod
{
   public class SeedData
   {
      private readonly DataContext _context;
      private string _chars =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
      public SeedData(DataContext context)
      {
         _context = context;
      }

      public void Seed(int count)
      {

         for (int i = 0; i < count; i++)
         {
            var user = GenerateUser();
            _context.Users.Add(user);

         }
         _context.SaveChanges();
      }

      private User GenerateUser()
      {
         var random = new Random();
         var hmac = new System.Security.Cryptography.HMACSHA512();

         var randomInt = random.Next(10) + 5;
         var randomString = GetRandomString(randomInt);

         var user = new User
         {
            Username = randomString,
            PasswordSalt = hmac.Key,
            PasswordHash =
hmac.ComputeHash(System.Text.Encoding.UTF8.GetBytes(randomString)),
            Gender = randomInt % 2 == 1 ? "Male" : "Female",
            KnownAs = randomString + randomInt,
            Interests = randomString,
            City = GetRandomString(randomInt-3),
            Country = "Ukraine",
```

```
            DateOfBirth = new DateTime(1980, 1, 1).AddDays(random.Next(10000)),
            Created = DateTime.Today,
            LastActive = DateTime.Now
        };

        return user;
    }

    private string GetRandomString(int lenght)
    {
        var stringChars = new char[lenght];
        var random = new Random();
        for (int i = 0; i < lenght; i++)
        {
            stringChars[i] = _chars[random.Next(_chars.Length)];
        }
        return new string(stringChars);
    }
  }
}
```

StressTests.cs

```
using NUnit.Framework;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Threading;
using System.Threading.Tasks;

namespace StressTests
{
  public class StressTests
  {
    [SetUp]
    public void Setup()
    {
    }
    private static int N = 1000;
    private static int K = Convert.ToInt32(0.09*N);

    [Test]
    public async Task StressTestAsync()
```

```
        {
            var httpClient = new HttpClient();
            httpClient.DefaultRequestHeaders.CacheControl = new
CacheControlHeaderValue
            {
                NoCache = true
            };
            var random = new Random();
            var sw = new Stopwatch();
            int n = 100;
            sw.Start();
            for (int i = 0; i < 100; i++)
            {
                var tasks = new List<Task>(n);
                for (int j = 0; j < n; j++)
                {
                    var task = Task.Run(async () => {
                        int x = random.Next(N);
                        double y = (N * N / 10) / (x + K) - K;
                        var request = new HttpRequestMessage(HttpMethod.Get,
$"https://localhost:5001/api/users/{Math.Round(y)}");
                        try
                        {
                            var result = await httpClient.SendAsync(request);
                        }
                        catch (Exception ex)
                        {
                            Console.WriteLine(ex.Message);
                        }
                    });
                    tasks.Add(task);
                }
                await Task.WhenAll(tasks);
                if (i%10 == 0)
                    Console.WriteLine($"{i*n+1}-{(i+10)*n} request sent. Time:
{sw.ElapsedMilliseconds} ms.");
            }
            Console.WriteLine($"Time: {sw.ElapsedMilliseconds} ms.");
        }
    }
}
```

ДОДАТОК К

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи: Методи та програмний засіб комплексного кешування даних у високонавантажених інформаційних система

Тип роботи магістерська кваліфікаційна робот

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

Підрозділ Кафедра обчислювальної техніки

(кафедра, факультет (інститут), навчальна група)

Науковий керівник к.т.н., доц. Ткаченко О. М.

(прізвище, ініціали, посада)

Показники звіту подібності

Plagiat.pl (StrikePlagiarism)		Unicheck	
КП1			
КП2		Оригінальність	96,3
Тривога/Білі знаки	/	Схожість	3,7

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 _____ Горбачов Г. О.

(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)