Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно–інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних систем та мереж
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

Магістр
(назва освітнього ступеня)

на тему: Методи і засоби підвищення продуктивності веб–додатків з використанням Spring Boot

Виконав: студент(ка) 6 курсу, групи СІм–62
спеціальності 123 «Комп'ютерна інженерія»
(шифр і назва спеціальності)

_____ Шеремета В.З.
(підпис) (прізвище та ініціали)

Керівник _____ Жаровський Р.О.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Луцик Н.С.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Осухівська Г.М.
(підпис) (прізвище та ініціали)

Рецензент _____ Никитюк В.В.
(підпис) (прізвище та ініціали)

Тернопіль
2024

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно–інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних систем та мереж
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ *Осухівська Г.М.*
(підпис) (прізвище та ініціали)
«11» листопада 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня *магістр*
(назва освітнього ступеня)

за спеціальністю *123 «Комп'ютерна інженерія»*
(шифр і назва спеціальності)

студенту *Шереметі Василю Зіновійовичу*
(прізвище, ім'я, по батькові)

1. Тема роботи *Методи і засоби підвищення продуктивності веб–додатків з використанням* Spring Boot

Керівник роботи *Жаровський Руслан Олегович*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від « *29* » *листопада 2024* року № 4/7–1144
2. Термін подання студентом завершеної роботи 21.12.2024
3. Вихідні дані до роботи *Фреймворк Spring Boot, засоби тестування ПЗ.*

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ
1 Теоретичні основи продуктивності веб – додатків
2 Методи підвищення продуктивності веб–додатків з використанням Spring Boot
3 Практична реалізація оптимізації продуктивності веб – додатка на основі Spring Boot
4 Охорона праці та безпека в надзвичайних ситуаціях
Висновки
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1. Актуальність і мета дослідження.
2. Завдання, об'єкт і предмет, наукова новизна і практична цінність дослідження.
3.
4.
5.
6.
7.
8. Висновки

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	*Осухівська Г.М., зав.каф. КС*	*05.12.2024*	
Безпека в надзвичайних ситуаціях	*Стручок В.С. старший викладач*	*10.12.2024*	

7. Дата видачі завдання *11.11.2024*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	*Аналіз існуючих методів та засобів розв'язання науково–технічних проблем, що відповідають тематиці кваліфікаційної роботи.*	*20.11.2024р. – 30.11.2024р.*	*Викон.*
2.	*Теоретичні основи продуктивності веб–додатків*	*04..12.2024р.*	*Викон.*
3.	*Методи підвищення продуктивності веб–додатків з використанням Spring Boot*	*08.12.2024р.*	*Викон.*
4.	*Практична реалізація продуктивності веб–додатка на основі Spring Boot*	*15.12.2024р*	*Викон.*
5.	*Охорона праці та безпека в надзвичайних ситуаціях*	*17.12.2024р*	*Викон.*
6.	*Оформлення пояснювальної записки і графічного матеріалу*	*19.12.2024*	*Викон.*
7.	*Попередній захист кваліфікаційної роботи магістра*	*21.12.2024*	*Викон.*
8.	*Захист кваліфікаційної роботи магістра*	*26.12.2024*	*Викон.*

Студент _____ *Шеремета В.З.*
(підпис) (прізвище та ініціали)

Керівник роботи _____ *Жаровський Р.О.*
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Шеремета В.З. Методи і засоби підвищення продуктивності веб–додатків з використанням Spring Boot. Робота на здобуття кваліфікаційного ступеня магістра: спец. 123 — комп'ютерна інженерія / наук.кер. Жаровський Р.О. Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2024. — 73с.

Ключові слова: фреймоврк spring boot, тестування веб–додатків, оптимізація продуктивності, багатопоточність, асинхроні процеси.

Кваліфікаційна робота присвячена засобам по підвищенню продуктивності із використанням технології Spring Boot із взаємодією другорядних бібліотек.

У першому розділі проведено аналіз сучасних теоретичних основ продуктивності веб–додатків, стратегій та підходів до оптимізації продукту.

У другому розділі описано вибір інструментів, таких як кешування та налаштування запитів, налаштування веб–додатків і кластеризація, використання побічних бібліотек та моніторинг продуктивності.

Третій розділ присвячено практичні реалізації веб–додатку, опис предметної області, аналіз поточної продуктивності, застосування методів та техніки оптимізації,

Результати тестування та продуктивності після оптимізації.

ANNOTATION

Sheremeta V.Z. Methods and tools for improving the performance of web applications using Spring Boot. Master's qualification thesis: specialty 123 – Computer Engineering / Scientific advisor: Zharovski R.O. Ternopil: Ternopil Ivan Puluj National Technical University, 2024. — 73 p.

Keywords: spring boot framework, web application testing, performance optimization, multithreading, asynchronous processes.

The qualification work is dedicated to tools for improving productivity using Spring Boot technology with interaction of auxiliary libraries.

In the first chapter, an analysis of modern theoretical foundations for web application performance, strategies, and approaches to product optimization is conducted.

The second chapter describes the selection of tools such as caching and query tuning, web application configuration and clustering, the use of auxiliary libraries, and performance monitoring.

The third section is devoted to the practical implementation of the web application, description of the subject area, analysis of the current performance and application of optimization methods and techniques. The results of performance testing after optimization are presented.

ЗМІСТ

ВСТУП.....8
РОЗДІЛ 1 ТЕОРЕТИЧНІ ОСНОВИ ПРОДУКТИВНОСТІ ВЕБ–ДОДАТКІВ..... 11
1.1.Поняття продуктивності веб–додатків..... 11
1.2.Основні фактори, що впливають на продуктивність веб–додатків. 13
1.3.Стратегії та підходи до оптимізації продуктивності веб–додатків..... 15
1.4.Фреймворк Spring Boot і його можливостей для розробки ефективних веб–додатків……..... 19
РОЗДІЛ 2 МЕТОДИ ПІДВИЩЕННЯ ПРОДУКТИВНОСТІ ВЕБ–ДОДАТКІВ З ВИКОРИСТАННЯМ SPRING BOOT..... 24
2.1. Оптимізація роботи з базами даних 24
2.2. Використання асинхроних процесів та багатопоточность. 29
2.3.Налаштування масштабування веб–додатків (load balancing, кластеризація)..... 32
2.4. Профілювання та моніторинг продуктивності додатків у Spring Boot..... 37
2.5. Засоби тестування продуктивності..... 41
РОЗДІЛ 3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ОПТИМІЗАЦІЇ ПРОДУКТИВНОСТІ ВЕБ–ДОДАТКА НА ОСНОВІ SPRING BOOT 43
3.1. Розробка веб–додатку з використанням Spring Boot..... 43
3.3. Застосовані методи та техніки оптимізації..... 51
3.4. Результати тестування та порівняння продуктивності до і після оптимізації..... 53
3.5. Висновки щодо ефективності застосованих методів..... 55
РОЗДІЛ 4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ... 57
4.1.Охорона праці 57
4.2 Безпека в надзвичайних ситуаціях 59

ВИСНОВКИ 62

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 62

Додаток А… 67

ВСТУП

Актуальність роботи.

Веб–додатки сьогодні є ключовими елементами цифрової інфраструктури, забезпечуючи роботу електронної комерції, онлайн–сервісів та інформаційних платформ. Зростання кількості користувачів, збільшення обсягів даних і підвищення складності додатків створюють значні виклики, зокрема: уповільнення часу відгуку, неефективне використання серверних ресурсів, збої під час пікових навантажень, затримки при роботі з великими обсягами даних і проблеми масштабованості. Для вирішення цих питань необхідне впровадження сучасних технологій, таких як Spring Boot, що пропонує широкий спектр засобів для підвищення продуктивності через оптимізацію роботи з базами даних, кешування, асинхронну обробку та інші методи. Для вирішення цих питань необхідне впровадження сучасних технологій і підходів, таких як Spring Boot, що пропонує широкий спектр засобів для підвищення продуктивності веб–додатків. Ця платформа дозволяє оптимізувати роботу з базами даних, впроваджувати механізми кешування, використовувати асинхронну обробку запитів та реалізовувати інші ефективні методи, які сприяють зменшенню часу відгуку та підвищенню загальної стабільності системи. Завдяки цим можливостям, Spring Boot стає потужним інструментом для розробників, які прагнуть створювати масштабовані та надійні веб–додатки, здатні витримувати високі навантаження та забезпечувати безперебійну роботу.

Науковці, які займаються цією тематикою : Джош Лонг, Род Джонсон, Мартін Фаулер, Адам Бієн, Джим Гавін.

Серед українських дослідників можна виокремити:

– Ю. П. Зозуля – фахівець із розподілених систем і підвищення їхньої продуктивності.

– О. О. Коваленко – дослідник у галузі програмної інженерії та оптимізації інформаційних систем.

– І. В. Величко – спеціаліст із масштабованості програмного забезпечення та баз даних.

– Н. В. Гончарова – експертка у програмних системах із високими вимогами до продуктивності.

– О. М. Шевченко – дослідник у сфері мікросервісної архітектури та оптимізації веб-додатків.

Метою кваліфікаційної роботи. Методи підвищення продуктивності Spring Boot для обчислення та балансування трудомістких запитів користувачів. Це дозволить підвищити продуктивність та швидкодію веб–додатків, зменшити час очікування користувачів та забезпечити більш якісне обслуговування. Для досягнення цієї мети будуть використані сучасні технології веб–розробки та інструменти Spring Boot.

Завдання кваліфікаційної роботи:

– Ознайомлення з теоретичними основами продуктивності веб–додатків;

– Дослідження технік і методів підвищення продуктивності веб–додатків;

– Розробити веб-додаток з використанням Spring Boot для апробації запропонованих методів підвищення продуктивності;

– Показати ефективність запропонованих методів та оцінити їх параметри шляхом тестування розробленого веб-додатку.

Об'єкт дослідження веб-додаток розроблений з використанням Spring Boot. *Предмет дослідження:* методи і засоби розробки веб-додатків на базі фреймворку Spring Boot.

Методи дослідження: методи системного аналізу та дослідження операцій; теорії алгоритмів, теорії масового обслуговування, теорії мов програмування; методи автоматизованого тестування.

Наукова новизна дослідження. Полягає у використанні фреймворку Spring Boot при розробці веб–додатків, що дало можливість знизити навантаження на ресурси системи, а також механізмів масштабування, що дозволяють адаптувати систему до змінних навантажень, які виникають під час експлуатації веб–додатків.

Практичне значення результатів кваліфікаційної роботи. Було досягнуто в меті роботи найвищу продуктивність веб–додатку та виявлення і знешкодження наявних проблем з продуктивністю, та реалізація кількісного користування для користувачів високого трафіку після тестування веб–додатку .

Публікації. Результати дослідження апробовано на XI науково–технічній конференції Тернопільського національного технічного університету імені Івана Пулюя «Інформаційні моделі, системи та технології», XII міжнародній науково–технічній конференція молодих учених та студентів «Актуальні задачі сучасних технологій», у вигляді тез конференцій.

1. Шеремета В.З., Жаровський Р.О. Використання Spring Boot та інтеграція другорядних інструментів для створення сучасних веб–додатків . Матеріали XII міжнародної науково–практичної конференції молодих учених та студентів «Актуальні задачі сучасних технологій» (11–12 грудня 2023 року). Тернопіль: ТНТУ. 2024. С. 439.
2. Шеремета В.З., Жаровський Р.О. Тестування веб–додатків розробленими на основі Spring Boot за допомогою Testing. Матеріали XI науково–технічної конференції Тернопільського національного технічного університету імені Івана Пулюя «Інформаційні моделі, системи та технології» (18–19 грудня 2024 року). Тернопіль: ТНТУ. 2024. С. 162.

Структура роботи. Робота складається з пояснювальної записки та графічної частини. Пояснювальна записка складається із вступу, 4 розділів, висновків, списку використаних джерел та додатку (–ів). Обсяг роботи: пояснювальна записка – 73 арк. формату А4, графічна частина – 8 аркушів формату А1.

РОЗДІЛ 1 ТЕОРЕТИЧНІ ОСНОВИ ПРОДУКТИВНОСТІ ВЕБ–ДОДАТКІВ

1.1. Поняття продуктивності веб–додатків.

Дослідження різних способів ефективного використання наявних інформаційних ресурсів є важливим, щоб підвищити якість інформаційно середовища. З розвитком мережі Інтернет, мобільних пристроїв, все частіше програмні продукти переходять в сферу мобільних і веб – додатків. Сучасні веб–додатки пропонують широкий спектр функцій та інструментів, які раніше були доступні лише в локальних додатках. Водночас, веб–додатки незалежні від апаратного і програмного забезпечення, достатньо встановити веб браузер [19].

Також, веб–додатки забезпечують доступність у будь–якому місці та в будь–який час, що є важливою перевагою для користувачів, які працюють на комп'ютерах, планшетах або мобільних пристроях [17]. Це дозволяє бізнесу бути більш гнучким і адаптивним до змінюваних умов, що, безумовно, є важливим фактором у сучасному швидкоплинному світі.

Крім того, веб–додатки часто мають меншу сукупну вартість володіння в порівнянні з традиційними програмами, оскільки вони зазвичай не вимагають значних витрат на ліцензування, обслуговування та оновлення. Ці особливості роблять веб–технології надзвичайно привабливими для вирішення широкого спектра бізнес–завдань, від управління проектами до електронної комерції та клієнтського обслуговування.

Зростаюча популярність веб–додатків висуває високі вимоги до їх продуктивності. Для досягнення максимальної ефективності та швидкості роботи, веб–додатки повинні мінімізувати обсяг переданих даних і зменшувати кількість запитів до веб–сервера. Це може бути досягнуто шляхом оптимізації коду, використання технологій кешування, а також впровадження сучасних методів стиснення даних. У результаті, такі заходи не тільки покращують швидкість завантаження та загальну продуктивність, але й підвищують задоволеність

користувачів, що є критично важливим для успіху будь–якого бізнесу в умовах сучасної конкуренції [18].

Даний фактор характеристик відповідає за швидкість роботи веб–додатків і є важливим показником, що визначає не лише швидкість, з якою додаток завантажується та відображається для користувача, але й його здатність швидко реагувати на дії користувача. Ці параметри дозволяють веб–додаткам безпосередньо впливає на загальний досвід користувача, оскільки затримки або повільна реакція можуть призвести до розчарування та зниження задоволеності від використання програми.

Підвищення веб–продуктивності (WPO) — це спеціалізована галузь знань, що охоплює різноманітні методи, техніки та стратегії, спрямовані на оптимізацію роботи веб–додатків як з клієнтської, так і з серверної сторони [8]. Ця дисципліна включає в себе аналіз часу завантаження, зменшення обсягу (рис. 1.1) даних, що передаються, оптимізацію коду, а також використання кешування та інших технологій для покращення швидкості обробки запитів.

Рис. 1.1. Підвищення продуктивності веб–додатків

Дослідження підвищення продуктивності веб–додатків зосереджене на виявленні ефективних рішень для оптимізації та поліпшення загальної продуктивності.

Сам процес оцінки продуктивності веб–додатка є складним і багатогранним, що включає кілька ключових етапів та кроків. Основними завданнями для розробника є

вибір відповідних інструментів для проведення оцінки продуктивності, а також визначення критеріїв, які потрібно оптимізувати в конкретних умовах. Розглянемо які основні фактори впливають на продуктивність веб–додатків.

1.2. Основні фактори, що впливають на продуктивність веб–додатків.

В роботі [15] визначені основні критерії продуктивності. Кожен з показників, відповідає за певні функціональні елементи додатку, що дозволяє детально вивчити його роботу. Наприклад в роботі [18], виявлено, що існує проблема з часом виконання програмного коду, це може свідчити про те, що проблема локалізується в конкретних функціях або (рис. 1.2) алгоритмах, які мають значну кількість взаємодій з даними.

Рис. 1.2. Цикл моніторингу додатка

Основними факторами які характеризують і впливають на продуктивність веб додатків є:

–Ефективність і завантаження: досліджує швидкість роботи веб–додатку та його здатність обробляти велику кількість запитів одночасно. У цьому випадку відстежуються реакції на дії користувачів, час відгуку сервера та час завантаження

сторінок. Це дослідження дозволяє визначити, які частини додатку обробляють найдовше і як їх можна оптимізувати для швидкодії роботи.

– Оптимізація коду та архітектури: дослідження продуктивності аналізує код і архітектуру веб–додатку, щоб виявити слабкі місця і можливості для оптимізації. Результати можуть включати зменшення кількості запитів до сервера, кешування даних, використання стиснення для передачі даних та покращення алгоритмів обробки даних [12].

– Оптимізація ресурсів: дії, спрямовані на оптимізацію використання ресурсів, таких як CPU, пам'ять і мережевий трафік.

– Моніторинг та аналіз продуктивності під час роботи веб додатків: Це дозволяє збирати інформацію про продуктивність у режимі реального часу та аналізувати їх для виявлення проблем і покращення програмного забезпечення. Вимірювання часу відгуку, використання ресурсів сервера, навантаження на мережу та інші показники продуктивності будуть частиною моніторингу.

При виборі критеріїв та метрик для оцінки продуктивності веб–додатків можна врахувати [19] наступні аспекти:

– Продуктивність на боці клієнта. Для оцінки (рис. 1.3) продуктивності на стороні клієнта можна використовувати такі показники як, час відгуку на дії користувача, час відгуку анімації та переходів або період рендерингу контенту [13].

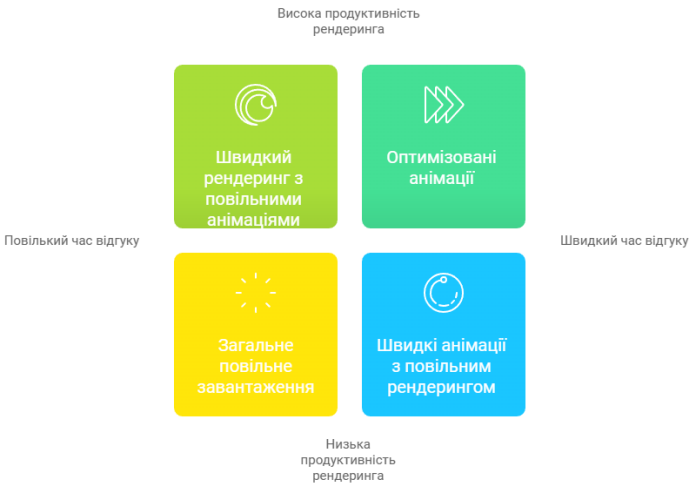

Рис. 1.3. Оцінка ефективності на кілєнтській стороні веб–додатку.

– Використання ресурсів. Здійснюється оцінка ефективності використання ресурсів яка оцінює використання ресурсів, таких як процесор, пам'ять і мережа використовуючи такі показники, як використання процесора і пам’яті, кількість запитів (рис. 1.6) до сервера і обсяг переданих даних.

Рис. 1.4. Використання ресурсів додатку.

Вибір конкретних критеріїв та метрик для оцінки продуктивності веб–додатків залежить від конкретного контексту та вимог проекту. Наприклад, якщо веб–додаток зосереджений на швидкості завантаження сторінок, то метрики, пов'язані з часом завантаження сторінки.[19] У разі масштабованих веб–додатків метрики, пов'язані з часом відгуку при великому навантаженні або пропускною здатністю мережі, можуть бути більш суттєвими.

1.3. Стратегії та підходи до оптимізації продуктивності веб–додатків

Впровадження заходів з оптимізації функціональних елементів програмного коду є критично важливим етапом у процесі розробки веб–додатків. Після проведення детального аналізу та визначення проблемних місць у веб–додатку, розробнику необхідно ретельно обрати методи для вирішення виявлених проблем продуктивності.

Повторна оцінка продуктивності веб–додатка дозволяє ретельно проаналізувати, наскільки ефективно були реалізовані заходи, спрямовані на підвищення продуктивності. Ця оцінка необхідна для визначення, чи досягнуті поставлені цілі за визначеними критеріями, а також для виявлення можливих змін у інших параметрах, які впливають на загальну ефективність інструменту оцінки.

У деяких випадках, обрані методи підвищення продуктивності можуть мати небажані побічні ефекти, які, в свою чергу, здатні негативно позначитися на інших показниках. Наприклад в роботі [16], проведений аналіз проблем з часом відображення сторінки, що може свідчити про наявність компонентів, які зазнають надмірного ре–рендерингу. У такій ситуації розробник може вирішити впровадити метод кешування даних для оптимізації роботи цих компонентів, зосередившись на покращенні їхньої продуктивності.

Проте, важливо враховувати, що, хоча час відображення сторінки може справді зменшитися в результаті таких змін, це не завжди означає загальне покращення продуктивності системи. В роботі, розглянуто випадок коли може виникнути ситуація, за якої разом із зменшенням часу завантаження спостерігається збільшення використання пам'яті. Це може свідчити про те, що кешування даних, яке було впроваджено для підвищення швидкості, вимагає значних обсягів пам'яті для зберігання кешованих даних, що, в свою чергу, може негативно вплинути на загальну продуктивність веб–додатка.

Існує ряд стратегій [16] які використовуються для оптимізації продуктивності. Як вже частково було згадано в підрозділі 1.1 вибір конкретних технологій і засобів залежить від специфіки проекту. Використовуються різні методи та техніки для оптимізації веб–продукту:

–Внутрішня оптимізація продуктивності передбачає впровадження стратегій для мінімізації затримок, підвищення ефективності та покращення масштабованості веб–додатків.

–Впровадження кешування може значно (рис. 1.4) змінити навантаження на сервер, і зменшує час відклику обробки запитів до API.

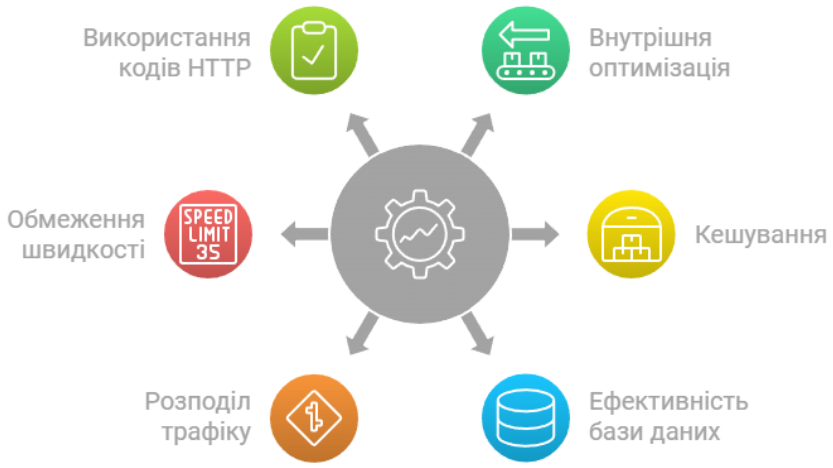

Рис. 1.4. Ефективність взаємодії API.

–Ефективність операцій з базами данних, таких як оптимізація запитів, впровадження індексів та використання представлень, з певною ймовірністю значно скоротить час на обробку даних сервера.

– Розподіл мережевого трафіку між серверами надає допомогу підвищити шкалу навантаження та запобігти витоку продуктивності.

–Обмеження швидкості допомагає запобігти перевантаження сервера, контроль певної кількості запитів, отриманих за короткий відлік часу. Це забезпечує стабільність системи і знижує низку помилок і низької продуктивності через зловживання одночасних запитів. Таким чином можна відтворювати балансування навантаження і налаштувати швидкість відклику при великих обсягах трафіку.

–Використання HTTP кодів забезпечує ефективний зв’язок з клієнтом і сервером та швидке вилучення несправностей.

–Оптимізаця продуктивності внутрішніх API. Швидкий відгук сервера зазначає приріст завантаження сторінок. Високопродуктивні веб–інтерфейси API можуть обробляти великі обсяги трафіку і забезпечувати стабільність виконання роботи при надвеликих навантаженях [15].

Окрім звичайних підходів до оптимізації веб–застосунків, потрібно також використати сучасні техніки для підвищення продуктивності, в роботі [9] за основу обирають:

– Архітектура мікросервісів підвищує масштабованість із великою ефективністю використовує ресурси за рахунок поділу великих додатків на менші, незалежні сервіси.

– Впровадження попередньої обробки сторінок на сервері покращує взаємодію з користувачем та значно скоротивши час початкового завантаження.

– Впровадження HTTP/2 ефективно підвищує продуктивність завдяки функціям, таких як стиснення заголовків, пріоритизація запитів і мультиплексування. Завдяки цим функціям зменшується затримка і одночасно (рис. 1.5) підвищується пересилання даних між клієнтом і сервером.

Рис. 1.5. Продуктивність програмного забезпечення

Продуктивність внутрішніх веб–застосунків має значний вплив, оскільки швидкість і ефективність роботи системи безпосередньо впливають на те, як користувачі взаємодіють з платформою. Повільні API можуть призвести до вищого показника відмов, що може спричинити нижчу активність користувачів, оскільки користувачі не отримують бажаного рівня взаємодії [13].

І навпаки, оптимізовані внутрішні API можуть значно покращити залучення користувачів. Коли система працює швидко та ефективно, це сприяє позитивному

користувацькому досвіду, що, в свою чергу, може призвести до підвищення коефіцієнта конверсії. Користувачі, які отримують швидкі та релевантні відповіді, з більшою ймовірністю завершують цільові дії, такі як покупки чи реєстрації.

Завдяки вбудованим інструментам для моніторингу, кешування та оптимізації роботи з базами даних, Spring Boot суттєво сприяє зменшенню затримок у обробці запитів, що, в свою чергу, підвищує стабільність системи. Ці функції дозволяють розробникам ефективно контролювати продуктивність додатків та оперативно виявляти і усувати можливі проблеми.

Spring Boot стає надійним вибором для створення продуктивних, гнучких і стійких веб–додатків, які можуть легко адаптуватися до змінюваних вимог бізнесу та технологій. Ця платформа забезпечує розробникам потужні інструменти для створення інноваційних рішень, що відповідають сучасним стандартам якості та продуктивності.

1.4. Фреймворк Spring Boot і його можливості для розробки ефективних веб–додатків

Spring Boot побудований на основі Spring Framework [10], який значно спрощує процес розробки Java–додатків. Основною метою Spring Boot є максимізація стабільності, продуктивності та швидкості розробки, що дозволяє зекономити час розробникам. Це досягається шляхом надання готових рішень та заздалегідь налаштованих компонентів, що дозволяє зосередитися на реалізації бізнес–логіки, а не на рутинних налаштуваннях.

Spring Boot є надбудовою над Spring Framework, що пропонує безліч можливостей для конфігурації «під капотом» застосунку. Це забезпечує широке поле для кастомізації та перевизначення конфігурацій, що робить його гнучким інструментом для розробників [23].

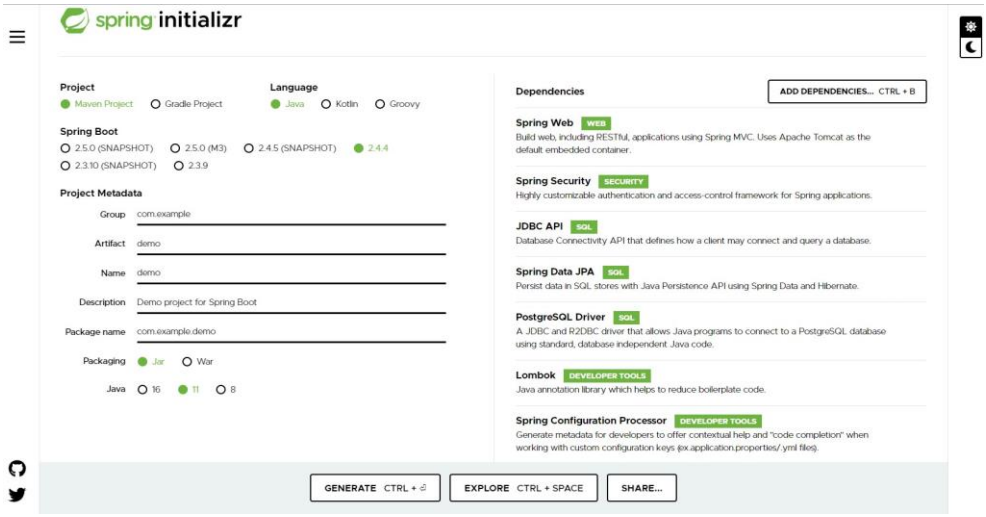

Рис. 1.6. Утиліта Spring Initializr.

На відміну від традиційного Spring Framework, який часто вимагає використання XML–конфігурацій, Spring Boot впроваджує механізм Java–анотацій для налаштування застосунку. Цей підхід спрощує (рис. 1.6) процес конфігурації, оскільки анотації дозволяють оголошувати (Bean), і задавати значення конфігурацій, працювати з профілями та виконувати багато інших операцій без необхідності написання громіздкого XML–коду.

Крім того, для створення та ініціалізації нового проекту на базі Spring Boot існує офіційна платформа Spring Initializr. Цей інструмент дозволяє розробникам швидко створити проект, автоматично підбираючи всі необхідні залежності, засоби автоматичного збирання (таких як Maven або Gradle), вибрану мову програмування (Java, Kotlin або Groovy), а також необхідну версію Spring Boot та метадані проекту. Завдяки цьому, розробники можуть швидко перейти до реалізації своїх ідей, не витрачаючи час на налаштування середовища.

Spring Boot забезпечує безперешкодну інтеграцію з різноманітними проектами Spring, що дозволяє створювати комплексні рішення для широкого спектра потреб додатків. Це включає в себе не лише доступ до даних та управління безпекою, але й підтримку обміну повідомленнями, мікросервісної архітектури та хмарного розгортання, що робить його ідеальним вибором для сучасних розробок [23].

Рис. 1.7. Архітектурні рівні Spring Boot

Проте, незважаючи на численні переваги, автоматизація може викликати певні проблеми з продуктивністю [17]. Зокрема, збільшене споживання пам’яті є одним із основних недоліків, оскільки автоматична конфігурація може призводити до завантаження різних модулів і ресурсів, (рис. 1.7) що створює додаткові додаткове навантаження на розподіл пам’яті. Це може негативно вплинути на загальну продуктивність системи. У складних програмах, де необхідно виконати обширний аналіз конфігурації та визначення шляхів до класів, запуск може затягуватися, що, в свою чергу, негативно позначається на часі старту програми та знижує її загальну ефективність. Тому важливо ретельно планувати архітектуру програми та оптимізувати використання ресурсів, щоб уникнути цих потенційних проблем.

Ще одним важливим аспектом у Spring Boot є можливість автоматично включаючи численні бібліотеки та модулі, може призвести до створення значних артефактів. Це, в свою чергу, впливає на час розгортання програми та споживання ресурсів сервера, що є критично важливим у випадках обмеженої інфраструктури. Чим більший артефакт, тим більше ресурсів він споживає, що може призвести до затримок у виконанні та зниження загальної продуктивності системи.

Щоб ефективно вирішити ці проблеми, надзвичайно важливо ретельно сформулювати профіль програми, визначити ключові області для вдосконалення та виявити потенційні вузькі місця продуктивності. Це передбачає вибіркову

оптимізацію окремих частин програми, уникнення включення невикористаних функцій, а також налаштування автоматичного запуску модулів лише за потреби. Регулярні оновлення Spring Boot, а також безперервний моніторинг продуктивності програми є основними методами, які допоможуть пом'якшити можливий негативний вплив на продуктивність.

Підтримка RESTful API у Spring Boot є однією з його основних можливостей, що дозволяє розробникам створювати гнучкі і легкі для усвоєння у використанні веб–сервіси, адаптовані до сучасних вимог бізнесу. Зазначений фреймворк надає простий та зручний спосіб створення API, зокрема для виконання ключових операцій CRUD таких як, створення, читання, оновлення, видалення, що є основою для будь–якої взаємодії з даними. Завдяки використанню анотацій Spring, таких як @RestController та @RequestMapping, розробники можуть легко визначати маршрути для HTTP–запитів,(рис. 1.8) що значно спрощує процес розробки [20].

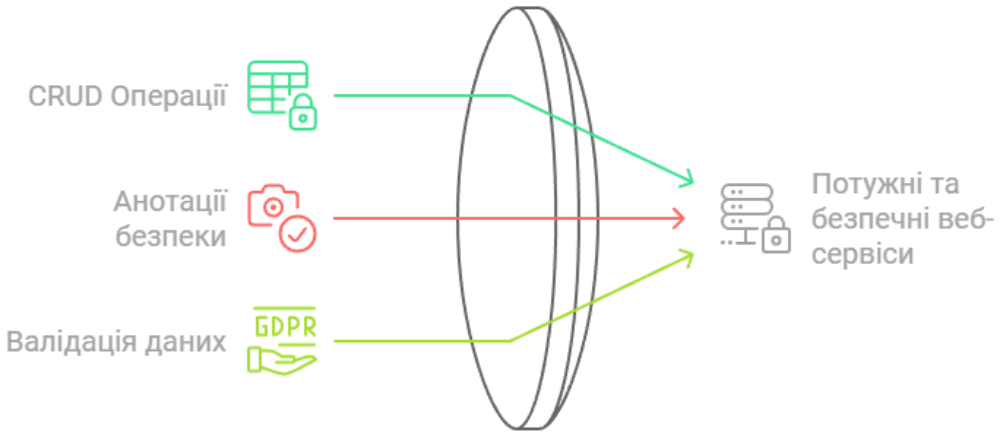

Рис. 1.8. Взаємодія API з Spring Boot.

Крім базових операцій CRUD, Spring Boot також підтримує широкий спектр технологічних можливостей, які дозволяють реалізувати розширену функціональності у веб–сервісах. Наприклад, вбудовані анотації Spring Security надають основні механізми для забезпечення аутентифікації та авторизації користувачів. Це є критично важливим аспектом для підтримання цілісності безпеки

API, адже без належного контролю доступу до ресурсів можуть виникати важливі загрози [9].

Крім того, Spring Boot надає можливість використовувати інструменти для підтвердження даних, таких як анотації, що належать (Bean Validation). Ці анотації дозволяють розробникам легко перевіряти вхідні дані та валідувати їх на предмет відповідності визначеним правилам і критеріям. Завдяки цьому API стає значно надійнішим, оскільки валідація даних допомагає виявляти та запобігати потенційним помилкам ще на етапі обробки запитів [20].

Висновки до першого розділу:

У першому розділі мого дослідження проведено аналіз продуктивності веб–додатків, що є критично важливим аспектом для успішної роботи сучасних цифрових рішень. Продуктивність, визначає не лише швидкість виконання завдань, але й загальний рівень зручності використання додатку для кінцевого користувача.

У процесі розгляду факторів, що впливають на продуктивність веб–додатків, я виділив кілька ключових аспектів. Серед них особливу увагу було приділено оптимізації на стороні сервера, що включає в себе правильну конфігурацію запитів до бази даних, використання механізмів кешування, а також балансування навантаження. Ці елементи є критично важливими для забезпечення стабільної та швидкої роботи додатку, особливо під час пікових навантажень. Крім того, я проаналізував загальну ефективність інтеграції фронт–енду та бек–енду, що також впливає на швидкість реагування системи.

Окрім цього, було підкреслено важливість архітектурних підходів та технічної інфраструктури для підвищення продуктивності веб–додатків.

В роботі буде проведено дослідження основних методів підвищення продуктивності веб–додатків з використанням Spring Boot.

РОЗДІЛ 2 МЕТОДИ ПІДВИЩЕННЯ ПРОДУКТИВНОСТІ ВЕБ–ДОДАТКІВ З ВИКОРИСТАННЯМ SPRING BOOT

В другому розділі буде проведений аналіз технології Spring Bot і інтеграцію Spring JPA та Hibernate. Дані технології містять важливі фактори для підвищення продуктивності веб–додатку, а завдяки згаданим технологіям з підпунктів 1.1–1.4 розробники використовують покращені підходи, такі як кешування та оптимізація запитів. Тобто ці техніки мінімізують кількість звернень до бази даних, скорочують час оброблених запитів та пришвидшують доступ до даних. Після інтеграції з цими інструментами застосунки стають масштабними та стабільними і можуть обробляти великі обсяги даних при високих навантаженях.

2.1. Оптимізація роботи з базами даних

Оптимізація роботи з базами даних є надзвичайно важливим аспектом у забезпеченні високої продуктивності та ефективності сучасних програмних рішень. Вона дозволяє не лише підвищити швидкість виконання запитів, але й зменшити навантаження на сервери, що в свою чергу сприяє поліпшенню загального користувацького досвіду. Використання таких потужних інструментів, як Spring Data JPA та Hibernate, надає можливість автоматизувати взаємодію з базами даних, що суттєво спрощує процес розробки [22].

Spring Data JPA – це один із модулів, що входить до складу Spring Framework, який допомагає забезпечити розробникам змогу легкого та ефективного взаємодії з базами даних із допомогою Java Persistence API (JPA). Наведений модуль спершу пропонує високо рівневий інтерфейс, що скорочує роботу з базами даних, що, в свою чергу, призводить до значного скорочення обсягу коду, необхідного для виконання операцій з даними.

Spring Data JPA реалізує підхід Repository, який визначає інтерфейси репозиторіїв з методами доступу до даних. Ці техніки автоматично трансформуються

в SQL–запити, які виконуються в контексті бази даних, що забезпечує високу продуктивність та зручність у використанні. Репозиторії надають можливість виконувати різноманітні операції з даними, такі як збереження, видалення, оновлення та пошук, використовуючи як стандартні методи, так і можливість створення власних запитів. Це дозволяє розробникам швидко адаптуватися (табл. 2.1) до змін у вимогах проекту та з легкістю інтегрувати нові функціональні можливості в існуючі системи.

Таблиця 2.1

Порівняння роботи з базами даних до і після використання Spring Data JPA

Аспект	До Spring JPA	Після Spring JPA
Обсяг коду	Великий обсяг коду для CRUID–Операцій	JPA генерує CRUID операції
Підтримка стандартів	Робота з даними які не відповідали стандартам JPA	JPA працює поверх Hibernate
Кастомні запити	Створення вимагало багато часу	Швидке створення через анотацію @Query

Завдяки потужним можливостям Spring Data JPA, розробка додатків, що потребують взаємодії з базами даних, стає більш ефективною, надійною та менш схильною до помилок, що, безумовно, підвищує якість кінцевого продукту.

Окрім того, Spring Data JPA надає можливість використовувати специфікації (Specifications) для динамічної генерації запитів до бази даних. Завдяки специфікаціям, розробники можуть створювати складні запити, які залежать від критеріїв, визначених під час виконання програми (runtime). Це забезпечує значну гнучкість при формулюванні запитів, адже дозволяє адаптувати їх до змінних умов і потреб бізнесу, що спрощує розробку складного функціоналу і знижує ризик помилок [22].

Spring Data JPA автоматично генерує SQL–запити на основі іменованих методів репозиторіїв або анотованих запитів, що виключає необхідність у ручному написанні

SQL–коду. Це не лише спрощує процес розробки, але й дозволяє розробникам зосередитися на бізнес–логіці додатка, а не на деталях реалізації запитів до бази даних.

Крім того, Spring Data JPA підтримує кешування результатів запитів, що суттєво покращує продуктивність додатка шляхом зменшення кількості запитів до бази даних. Це є особливо важливим для додатків, які обробляють великі обсяги даних або мають високі вимоги до швидкості реагування.

Для автоматизації щоденних процесів, таких як: керування транзакціями, створення та налаштування запитів і відображення об’єктів у реаліаційній таблиці можна скористатись поєднанням Spring JPA та Hibernate.

Hibernate, як ORM (Object–Relational Mapping) фреймворк, відіграє надзвичайно важливу роль у розробці сучасних інформаційних систем, особливо в контексті ефективної взаємодії з реляційними базами даних. Цей фреймворк надає розробникам можливість абстрагуватися від прямого написання SQL–коду, завдяки чому стає можливим мапінг об'єктно–орієнтованої моделі Java на реляційну модель бази даних [24].

Технічні особливості Hibernate включають його здатність до гнучкого налаштування, що дозволяє ефективно провести оптимізацію та взаємодію з базою даних для досягнення найвищої продуктивності. Однією з ключових функцій є підтримка лінивої загрузки (lazy loading), яка дозволяє завантажувати дані лише за потреби, зменшуючи витрати на пам'ять і підвищуючи швидкість роботи системи. Крім того, Hibernate забезпечує механізми кешування запитів (табл. 2.2) та асоціацій, що суттєво знижує загальне навантаження на базу даних і покращує відгук системи, особливо в умовах високих навантажень.

Таблиця 2.2

Порівняння взаємодії з базами даних до і після використання Spring JPA та Hibernate

Аспект	До	Після
Робота з сесіями	Необхідно було вручну створювати та закривати сесії Hibernate	Spring JPA автоматизує управління сесіями та їхнім життєвим циклом.
Підтримка платформ	Обмежена інтеграція з різними базами даних.	Hibernate підтримує роботу з MySQL, PostgreSQL
Виконання складних запитів	детального прописання SQL або ручної логіки.	Hibernate підтримує HQL та Criteria API

Окрім цього, Hibernate надає можливість використовувати складні запити HQL (Hibernate Query Language), які є об'єктно–орієнтованим аналогом SQL, що дозволяє розробникам писати запити, які більш інтуїтивно зрозумілі в контексті об'єктно–орієнтованого програмування. Також, завдяки (Criteria) API, можна створювати типобезпечені запити у програмному коді, що знижує ймовірність помилок під час виконання і підвищує загальну надійність застосунку [24].

В порівнянні з іншими ORM–бібліотеками, такими як Entity Framework, що широко використовується у спільноті .NET, Hibernate вирізняється своєю портативністю та гнучкістю, що робить його особливо привабливим вибором для розробників. Entity Framework, будучи тісно інтегрованим із середовищем Microsoft, обмежує свою функціональність лише платформою .NET, тоді як Hibernate надає можливість використовувати його в будь–якому середовищі, що підтримує Java. Це означає, що Hibernate може безперешкодно інтегруватися з різними базами даних, такими як MySQL, PostgreSQL, Oracle та інші, що забезпечує розробникам більшу свободу у виборі технологій та архітектури системи.

У традиційних сценаріях взаємодії Hibernate без Spring JPA розробниками доводилося написати код вручну для створення сесій, управління транзакціями та виконання операцій з базами даних.

Також варто відмітити що Spring JPA спрощує взаємодію з базою даних та проводить автоматизацію процесів завдяки вбудованим засобам: обробки транзакцій, управління сесіями та пулами [24,22].

Код презентує сервісний клас (UserService), який забезпечує логіку праці з користувачем. Основною функцією даного класу є метод (saveUser), який створює об'єкт User, задає ім'я і зберігає в базі даних за допомогою (UserRepository). (UserRepository) – це JPA компонент Spring Data, котрий спрощує взаємодію з базою даних. Анотація Service позначає цей клас як керований Spring – сервіс, що робить доступним в інших частинах програми.

До взаємодії з JPA розробка вимагала ручної конфігурації та повторюваного коду; після інтеграції Spring JPA управління взаємодією з Hibernate стало набагато гнучкішим та масштабованим. Це дозволяє зосередитися на бізнес–логіці, а не на внутрішніх технічних деталях.

Spring Data JPA є потужним інструментом, який часто використовується в комплексі з Hibernate, оскільки ці технології працюють синергічно в рамках стеку технологій, призначеного для роботи з базами даних у середовищі Spring. Hibernate виконує роль реалізації Java Persistence API (JPA), яка забезпечує стандартизований спосіб управління даними. Spring Data JPA, у свою чергу, надає додатковий рівень абстракції, спрощуючи розробку репозиторіїв та знижуючи обсяг коду, що необхідно написати для виконання стандартних операцій з базою даних, таких як збереження, оновлення, видалення та пошук даних. Spring JPA та Hibernate взаємодіють разом, для того щоб забезпечити ефективну взаємодію з базою даних за допомогою ORM. Технологія підсилює продуктивність, автоматизацію запитів JPA, зменшує кількість помилок, і генерує SQL–запити на основі Java–запитів [24,22].

Hibernate підтримує кешування першого рівня для запобігання повторюваних запитів до бази даних. Оптимізація вибірки використовує стратегії завантаження Lazy і Eager тобто можна мінімізувати (табл.2.3) непотрібні дані, що витягуються з бази даних.

Таблиця 2.3

Налаштування бази даних

Параметр	Інструмент	Опис
@Query	Spring Data JPA	Використовується для створення кастомних SQL/HQL запитів без написання додаткового коду.
spring.jpa.show–sql	Spring Boot	виведення SQL–запитів у консоль для моніторингу.
hibernate.cache.use_second_level_cache	Hibernate	кешування другого рівня для зменшення запитів до бази.

2.2. Використання асинхронних процесів та багатопоточність.

Асинхронні процеси та багатопоточність відіграють надзвичайно важливу роль у розробці сучасних додатків, особливо тих, які повинні ефективно працювати з великими обсягами даних або обслуговувати значну кількість користувачів одночасно. Ці технології дозволяють значно підвищити продуктивність системи, оскільки асинхронна обробка запитів дозволяє не блокувати виконання програми під час очікування завершення тривалих операцій, таких як доступ до бази даних або зовнішніх сервісів. Основні переваги асинхронності:

–Покращена продуктивність асинхронних процесів (табл. 2.4) дозволяють головному потоку продовжувати обчислення інших запитів, таким чином покращуючи загальну продуктивність програми [31].

–Мінімальна затримка транзакції обробляються у фоновому режимі ,тому користувачі отримують швидкі відповіді на запити, а час очікування скорочується;

–Оптимізація ресурсів асинхронної обробки гарантує, що потоки не блокуються під час довготривалих процесів, завдяки чому системні ресурси використовуються ефективно.

Таблиця 2.4

Порівняння роботи без асинхронності та з її використанням

Аспект	Без асинхронності	З асинхронністю
Швидкість обробки запитів	Висока затримка через послідовну обробку запитів.	Запити обробляються паралельно,
Реалізація обробки	вимагає ручного управління потоками та ресурсами	проста реалізацію асинхронних завдань без значних змін у логіці системи
Продуктивність програми	Основний потік блокується під час виконання тривалих завдань	Асинхронні методи виконуються в окремих потоках

Багатопоточність надає змогу програмам виконувати декілька завдань одночасно, використовуючи різні потоки. Це корисно для застосунків, які виконують велику кількість значних операцій або вимагають одночасного доступу до різних ресурсів. Основними перевагами багатопоточності є:

–Підвищення ефективності паралельно виконуючи завдання, програми які можуть обробляти запити та виконувати завдання швидко.

–Додаток можна розширювати, додаючи нові потоки для виконання завдань, що дозволяє ефективно обробляти великі обсяги даних.

–Кожен потік працює окремими ресурсами, що дозволяє з великою ефективністю використовувати доступні системні ресурси та зменшувати помилки при доступі до ресурсів.

Spring Boot підтримує багатопоточність (табл.2.5) за допомогою (ExecutorServi ce),дана функція забезпечує гнучкість та ефективність у керуванні ресурсами, дозволяючи створювати пули потоків для виконання завдань. Водночас Spring Boot підтримує інструменти синхронізації потоків для уникнення конфліктів при одночасному доступі до ресурсів. Це особливо важливо для багатопоточних додатків, щоб забезпечити цілісність даних та уникнути конфліктів [30].

Таблиця 2.5

Порівняння роботи без багатопоточності та з її використанням.

Аспекти	Було погано (без багато поточності)	Стало краще (з багато поточністю)
Масштабованість програми	Додаток складно масштабувати через відсутність механізмів паралельної обробки	Додавання нових потоків дозволяє ефективно обробляти зростаючі обсяги даних або запитів.
Цілісність даних	Конкуренція за ресурси може призводити до конфліктів	Інструменти синхронізації у Spring Boot гарантують, що дані залишаються цілісними навіть у багато потоковому середовищі.
Оптимізація ресурсів	Використовуються всі доступні ресурси повільно, що призводить до їхнього простою.	Розподіл завдань між потоками забезпечує ефективне використання процесорного часу та пам’яті.

Один з найпоширеніших сценаріїв використання асинхронних процесів є обробка файлів. Тобто якщо користувач завантажує великий за розміром файл на сервер, асинхронний процес використовується в фоновому режимі, зокрема не блокуючи основний потік і не змушує користувача очікувати на завершення процесу. Це надає плавну роботу користувача, і зменшить затримку та підвищить продуктивність програми.

Асинхронна обробка та багатопоточність є ключовими елементами для підвищення продуктивності сучасних веб–додатків, а Spring Boot інтегрує певні концепції за допомогою бібліотек та фреймворків, які дозволяють легко реалізувати паралельну обробку завдань. Одним з найпоширеніших способів реалізації асинхронності є використання анотації @Async, котра дозволяє методам виконуватися в окремих потоках, звільняючи основний потік для інших завдань.[31]

Асихроне виконання збільшує швидкість обробки запитів користувачів, оскільки процеси більше не блокують потік. Це особливо корисно для обробки електронної пошти, завантаження великих файлів і взаємодії із зовнішнім API.

2.3 Налаштування масштабування веб–додатків (load balancing, кластеризація)

У контексті розробки на платформі Spring Boot, масштабування може бути досягнуто через налаштування кластеризації та балансування навантаження. Ці методи дозволяють ефективно розподілити навантаження між кількома серверами, що не тільки мінімізує ризик збоїв у роботі (табл.2.6.) системи, але й забезпечує високу доступність додатку, що є критично важливим для користувачів.

Таблиця 2.6

Балансування навантаження до і після використання інструментів Spring Boot

Проблема	До навантаження	Після навантаження
Розподіл трафіку	Запити надходять на один сервер що викликало перезавантаження	Розподіл трафіку між серверами
Відмово стійкість	Зупинка сервера при виході з ладу сервера.	Доступність завдяки перенаправленню трафіку на інші сервери.
Швидкість обробки запитів	Затримки через перевантаження вузол	Метод розподілу трафіку

Балансування навантаження виконує функцію розподілу вхідного трафіку між різними екземплярами програми. Це є ключовим аспектом, оскільки допомагає уникнути перевантаження одного сервера, що може призвести до затримок у відповідях або навіть перебоїв у роботі сервісів. Для реалізації балансування навантаження часто використовуються такі популярні інструменти, як Nginx і HAProxy, які забезпечують ефективний розподіл трафіку. Крім того, сучасні хмарні

платформи, такі як AWS, пропонують вбудовані рішення, наприклад, (AWS Elastic) (Load Balancer) , які автоматично керують балансуванням навантаження[32].

У Spring Boot інтеграція механізмів балансування навантаження здійснюється за допомогою бібліотек, таких як Spring Cloud LoadBalancer. Ці бібліотеки спрощують процес роботи з великою кількістю вузлів, дозволяючи розробникам зосередитися на створенні функціоналу додатку, а не на управлінні інфраструктурою. Завдяки цьому, розробники можуть досягти більшої гнучкості та ефективності у масштабуванні своїх веб–додатків, що є важливим чинником у сучасному цифровому середовищі.

Також варто відмітити, що балансування навантаження використовує різноманітні стратегії, які дозволяють оптимізувати розподіл запитів між серверами. Однією з таких стратегій є (Round–Robin), яка передбачає рівномірний розподіл запитів між доступними вузлами в циклічному порядку. Це означає, що кожен сервер отримує запит по черзі, що дозволяє уникнути перевантаження окремих вузлів.

Крім того, існує стратегія розподілу за часом відгуку, при якій пріоритет надається серверам з меншим часом відгуку. Це забезпечує швидшу (табл.2.7.) реакцію системи на запити користувачів і покращує загальний користувацький досвід.

Таблиця 2.7

Кластеризація до і після налаштування Spring Boot

Проблема	До кластеризації	Після кластеризації
Збереження сесій	Сесія вимикається при перемиканні	Збігання сесії через SQL
Масштабування	Ручне налаштування серверів	кластеризація Spring
Надійність роботи додатку	Збій в роботі при навантаженні	Безперебійна робота серверів

Іншою важливою стратегією є зважений розподіл навантаження, де сервери з більшою пропускною здатністю отримують більше запитів. Це дозволяє максимально

використовувати ресурси потужніших серверів і забезпечити більш ефективну обробку запитів.

Щодо кластеризації, то цей функціонал дозволяє об'єднати кілька серверів або екземплярів додатків, які працюють у рамках єдиної системи. У контексті Spring Boot кластеризація підтримується через Spring Session, що використовує централізовані системи, такі як Redis або PostgreSQL, для зберігання користувацьких сесій.

Наприклад для Redis необхідно додати до файлу `pom.xml` залежності наведені на (рис. 2.1.)

```
<dependency>
    <groupId>org.springframework.session</groupId>
    <artifactId>spring-session-data-redis</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
```

Рис. 2.1. Додавання залежностей

Наступним кроком буде налаштування підключення до Redis, для цього у файлі `application.yml` проведемо наступні налаштування наведені на (рис. 2.2).

```
spring:
  redis:
    host: localhost
    port: 6379
  session:
    store-type: redis
```

Рис. 2.2. Налаштування конфігурації

Або у (рис. 2.3) файлі `application.properties`:

```
spring.redis.host=localhost
spring.redis.port=6379
spring.session.store-type=redis
```

Рис. 2.3. Налаштування конфігурації у файлі `application.properties`

Після чого створюємо клас конфігурації, щоб (рис. 2.4) увімкнути використання Redis для зберігання сесій.

```
@Configuration
@EnableRedisHttpSession(maxInactiveIntervalInSeconds = 1800) // Сесія активна 30 хвилин
public class RedisSessionConfig {}
```

Рис. 2.4. Конфігурація сесій

Запускаємо Redis через Docker. Для Docker використовуємо (рис. 2.5) таку команду:

```
docker run -d --name redis -p 6379:6379 redis
```

Рис. 2.5. Запуск команди

для перевірки роботи необхідно запустити Spring Boot програму і переконатися, що сесії користувачів (рис. 2.6) зберігаються в Redis. Використовуємо команду redis–cli для перевірки:

```
redis-cli
keys *
```

Рис. 2.6. Запуск команди для перевірки

Це рішення забезпечує безперебійну роботу користувачів, навіть якщо їх запити обробляються на різних серверах, гарантуючи, що сесії залишаються доступними незалежно від того, на якому сервері вони були ініційовані. Таким чином, система може забезпечити високу доступність та надійність, що є критично важливим для сучасних веб–додатків.

Масштабування веб–додатків є критично важливим аспектом сучасної розробки програмного забезпечення, оскільки воно забезпечує стабільну і ефективну роботу додатків під час пікових навантажень та високого трафіку [33]. У контексті використання Spring Boot, розробники мають можливість впроваджувати різноманітні методи масштабування, такі як балансування навантаження та кластеризація, які дозволяють оптимально розподілити робоче навантаження між кількома серверами або вузлами. Це, в свою чергу, забезпечує (табл.2.8) не лише високу продуктивність, але й відмово стійкість системи, що є надзвичайно важливим для збереження безперервності бізнес–процесів.

Таблиця 2.8

Порівняння стратегій балансування

Стратегія	Переваги	Недоліки
Round–Robin	Розподіл запитів між вузлами	Не враховує продуктивність окремих серверів
Зважене балансування	Розподіл потужності серверів	Потрібне налаштування серверів вручну
Розподіл за відгуком	Обробка запитів швидкими вузлами	Коливання часу відгуку на слабких серверах

Балансування навантаження є одним із найпоширеніших і найефективніших методів масштабування веб–додатків. Цей процес передбачає рівномірний розподіл вхідного трафіку між кількома екземплярами програми, які працюють на різних серверах. Завдяки інтеграції Spring Boot з такими інструментами, як Spring Cloud Load Balancer, розробники можуть реалізувати гнучке програмне балансування, яке адаптується до змінних умов навантаження. Балансування навантаження не лише допомагає уникнути перевантаження окремого сервера, але й суттєво зменшує ризик виникнення збою в роботі додатка, що, в свою чергу, підвищує загальну доступність і надійність програми. Це є особливо важливим для підприємств, які покладаються на

безперебійну роботу своїх веб–сервісів для підтримки конкурентоспроможності на ринку.

2.4. Профілювання та моніторинг продуктивності додатків у Spring Boot

Профілювання і моніторинг продуктивності додатків є основоположними аспектами у створенні та підтримці високоефективних систем, що здатні задовольнити вимоги сучасного користувача. У технологічному середовищі, де швидкість і ефективність є ключовими факторами успіху, забезпечення оптимальної продуктивності додатків стає критично важливим завданням для розробників та інженерів. Spring Boot, один із провідних фреймворків для створення Java–додатків, пропонує широкий спектр інструментів і механізмів, які допомагають у досягненні цієї мети.

Профілювання продуктивності є складною (табл.2.9) методикою, що передбачає детальний аналіз роботи додатка з метою виявлення та усунення вузьких місць, які можуть негативно впливати на його ефективність. Це дозволяє не лише покращити загальну продуктивність, але й підвищити задоволеність користувачів. Основні завдання профілювання включають:

–Процес ідентифікації конкретних компонентів або процесів, які сповільнюють роботу додатка.

–Оптимізація цих елементів може суттєво покращити загальну продуктивність системи. Наприклад, це може бути пов'язано з оптимізацією запитів до бази даних або вдосконаленням алгоритмів обробки даних [23].

Таблиця 2.9

Профілювання продуктивності додатків до і після впровадження інструментів Spring boot

Проблема	До застосування інструментів	Після застосування інструментів
Ідентифікація вузьких місць	Компоненти які сповільнюють роботу	Java Flight recorder для вузьких місць
Оптимізація використаних ресурсів	Надмірне використання CPU, що викликало збій	Моніторинг SB Acutator, оптимізація користування
Час відповіді системи	Затримки при обробці запитів	Зменшення затримок, кешування асинхронних методів
Задоволеність користувачів	Тривалий відклик в системі	Швидкість обробки запитів

Забезпечення оптимального використання таких системних ресурсів, як процесор, пам'ять, мережа та інші, є критично важливим для забезпечення стабільної роботи додатка під навантаженням. Це може включати в себе моніторинг використання пам'яті та CPU, а також налаштування параметрів конфігурації для досягнення максимальної ефективності.

Зменшення затримок у відповіді системи на запити користувачів є важливим аспектом, що впливає на загальне враження від використання додатка. Це може бути досягнуто шляхом оптимізації коду, використання кешування та впровадження асинхронних методів обробки запитів, що дозволяє значно підвищити швидкість обробки запитів[23].

Таким чином, профілювання і моніторинг продуктивності є невід'ємною частиною успішного розвитку програмного забезпечення, що дозволяє створювати системи, які відповідають найвищим стандартам ефективності та надійності.

Інструменти для профілювання Java–додатків є незамінними у процесі моніторингу та оптимізації їх продуктивності. Ось деякі з найпотужніших інструментів, які можуть суттєво полегшити цю задачу:

–Вбудований модуль, що надає розробникам можливість отримувати детальну інформацію про стан їхніх додатків. Завдяки Actuator, можна легко здійснювати моніторинг метрики, такі як завантаження процесора, використання пам'яті, кількість оброблених запитів, а також стан різних компонентів системи. Це дозволяє вчасно виявляти проблеми та оптимізувати роботу додатка.

–Інструмент Java Flight Recorder є потужним засобом для збору та аналізу детальної інформації про виконання Java–додатків. Він дозволяє відстежувати час виконання методів, використання системних ресурсів, а також інші важливі метрики, що допомагає виявляти "вузькі місця" у продуктивності та оптимізувати код. JFR є особливо корисним для тривалих операцій, оскільки може записувати дані без значного впливу на продуктивність [8].

–Графічний інструмент VisualVM, який надає можливості моніторингу та профілювання Java–додатків у реальному часі. VisualVM забезпечує детальну інформацію про використання пам'яті, завантаження процесора, кількість створених потоків та інші важливі показники. Завдяки зручному інтерфейсу, розробники можуть швидко ідентифікувати проблеми з продуктивністю та отримати візуалізацію даних, що значно спрощує процес аналізу.

Використання цих інструментів у поєднанні дозволяє досягти високої продуктивності Java–додатків, своєчасно виявляючи та усуваючи проблеми, що можуть негативно вплинути на їхню роботу.

Моніторинг продуктивності додатків є критично важливим процесом, що забезпечує можливість відслідковувати їх стан у реальному часі, а також відклик реагування на потенційні проблеми, які (табл.2.10) можуть виникнути під час експлуатації. Це дозволяє не лише підтримувати високий рівень обслуговування, але й забезпечувати безперебійну роботу системи.

Таблиця 2.10

Інструменти моніторингу продуктивності Java–додатків до і після їх використання

Інструмент	До використання	Після використання
Spring Acutator	Відсутність метрик для моніторингу	Надає ключові метрики такі як,CPU, стан компонентів
Java Flight Recorder	Використання ресурсів	Аналіз додатка без впливу на продуктивність
VisualVM	Немає графічного інструменту для продуктивності	Забезпечує графічний інтерфейс для аналізу використання пам’яті, потоків
Загальний підхід до моніторингу	Незнання стану додатка та проблемних зон	Усунення проблем інтегрованими засобами

Основні цілі моніторингу включають:

–Процес виявлення аномалій, що передбачає ідентифікацію нетипових значень метрик, які можуть свідчити про можливі проблеми в роботі додатка. Наприклад, різке підвищення часу відповіді або зниження продуктивності може сигналізувати про необхідність негайного втручання;

–Забезпечення стабільності важливо не лише відстежувати стан додатка, але й забезпечувати його стабільну роботу навіть під високими навантаженнями. Це включає в себе моніторинг ресурсів, таких як пам'ять і процесор, а також управління навантаженням для уникнення перевантаження системи;

–Зібрані дані моніторингу можуть бути використані для стратегічного планування масштабування додатка в залежності від зростаючого навантаження. Це дозволяє своєчасно адаптувати інфраструктуру до потреб бізнесу, забезпечуючи ефективність та конкурентоспроможність.

2.5. Засоби тестування продуктивності

За основу для дослідження інструментів тестування веб–додатків було оглянуто декілька варіацій інструментів і використано лише один в напрацюванні.

Перший інструмент – Lighthouse він використовується для аудиту продуктивності веб–додатків і містить декілька важливих критерій, такі як: продуктивність, доступність, SEO та складність. Lighthouse доступний як вбудований інстурмент у Chrome DevTools, і також доступний у браузері через утиліту CLI або другорядні сервіси. Під час аналізу генеруються дотичні звіти, які включають основні показники ефективності сторінки, такі як ключові життєво необхідні веб–показники (час завантаження першого байта, швидкість відображення контенту та взаємодія зі сторінкою). Lighthouse може допомогти зменшити розмір ресурсу, кешувати дані, видалити скрипти та вузькі місця котрі, негативно впилавють на продуктивність.

Другий інструмент по якому проводиться дослідження WebPageTest.

WebPageTest – сервіс для ретельного тестування продуктивності веб – додатків,що дозволяє оцінити швидке завантаження сторінки. Інструмент дозволяє тестувати веб–сайти з різноманітних пристроїв, браузерів і регіонів, для того щоб відобразити поведінку користувача. Також сервіс надає основні ознаки ефективності, такі як час відповіді сервера, загальний час завантаження веб–сторінки. Потрібно зазначити, що даний сервіс подібний до функціоналу попереднього і містить ідентичні функції. Але варто відмітити що WebPageTest містить унікальну діаграму водоспаду, котра виключно аналізує продуктивність ресурсів.

Було проведено огляд серед ресурсів і вибрано корисний сервіс для тестування JMeter, оскільки засіб має великий функціонал в порівнянню з попередніми.

За допомогою JMeter користувач може імітувати великий кількість одночасних користувачів і оцінювати продуктивність веб–додатків в умовах високих навантажень. Варто зазначити, що інструмент у своєму інструментарії містить широкий спектр протоколів, таких як HTTP, HTTPS, FTP, і API, що надає можливість бути придатним для тестування спектр сервісів; за допомогою Jmeter ви можете

аналізувати час відклику сервера, затримки обробки запитів і пропускну здатність системи. На додаток до тестування навантаження, Jmeter може використовувати системні функції для того, щоб зібрати показники продуктивності [12].

Висновки до другого розділу:

Було проведений аналіз технологій Spring Boot і інтеграція Spring JPA та Hibernate. Також із взаємодією я застосував методи та техніки кешування та оптимізацію запитів, які зменшують кількісне звернення до бази даних і скорочують час оброблених запитів. Щодо Hibernate, я не взаємодіяв з даною технологією у наступному розділі оскільки основним напрямком розробки та тестування були технології Spring Boot,Spring JPA, Jmeter

РОЗДІЛ 3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ОПТИМІЗАЦІЇ ПРОДУКТИВНОСТІ ВЕБ–ДОДАТКА НА ОСНОВІ SPRING BOOT

3.1. Розробка веб–додатку з використанням Spring Boot

Визначимо структуру досліджуваного веб – додатку. Spring Boot використовує архітектурний шаблон Model–View–Controller (MVC), де представлення (view) відображає дані для користувача, а контролер (рис. 3.1) обробляє запити та керує взаємодією між моделлю та представленням.

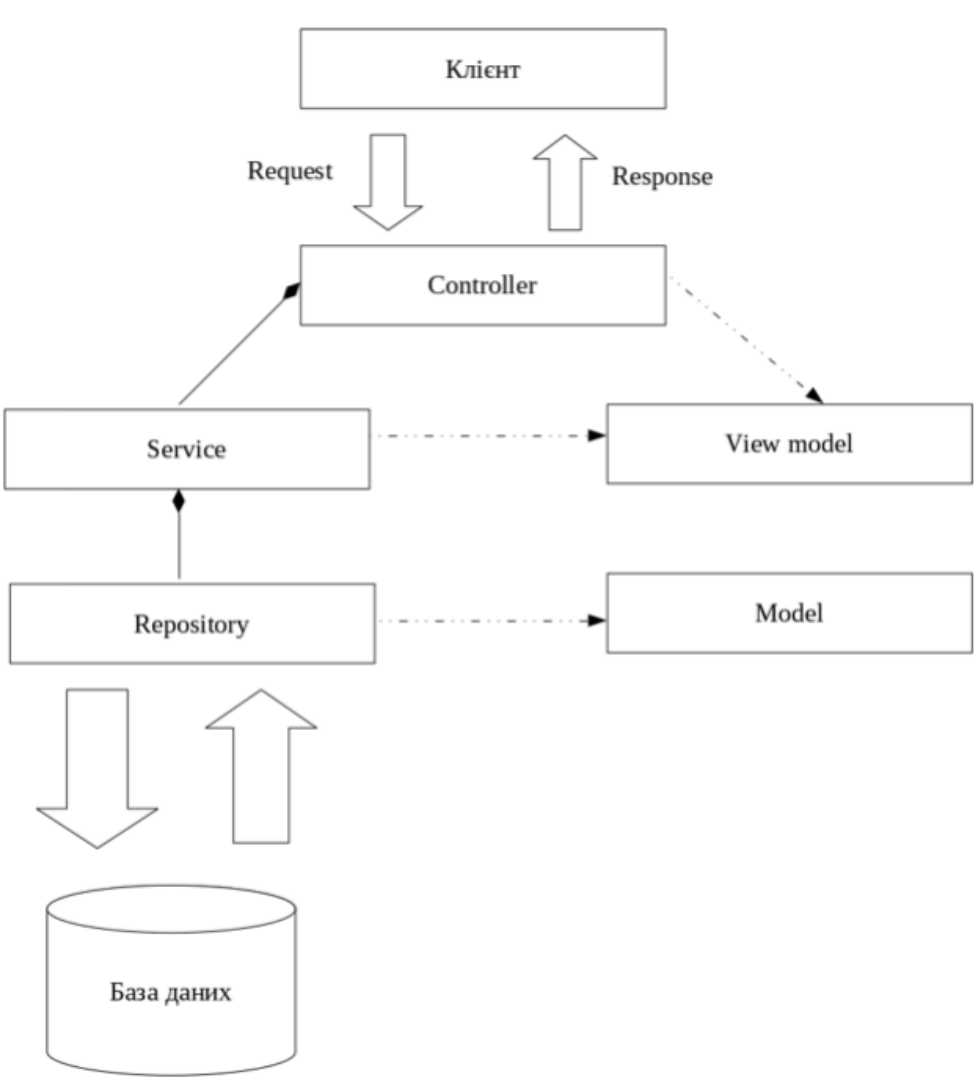

Рис. 3.1. Архітектура системи

Для початку розробки веб–застосунку з використанням Spring Boot необхідно створити новий проект. Можна скористатися офіційним інструментом Spring Initializr, який дозволяє швидко налаштувати проект і вибрати необхідні залежності.

Після створення проекту інструментом Spring Initializr можна завантажити проект в (Jar) або (War) пакеті. Автоматично (рис. 3.2) створюється клас Spring Boot додатка з методом (main) для його запуску

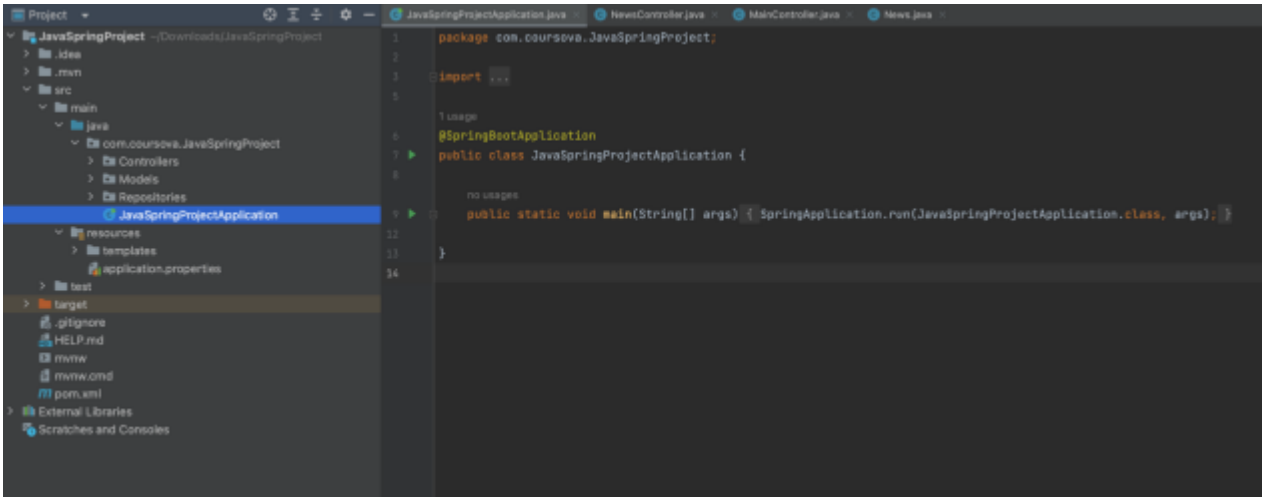

Рис. 3.2. Клас Проекту

У конфігураційних файлах, зокрема у файлі application.properties або application.yml, можна визначити налаштування для бази даних, сервера та інші аспекти. Наприклад, ви можете вказати налаштування для підключення до бази 15 даних, такі як URL, ім'я користувача та пароль. Також можна визначити порт, на якому буде працювати веб–сервер, а також інші (рис. 3.3) налаштування, пов'язані з безпекою, логуванням та кешуванням [29].

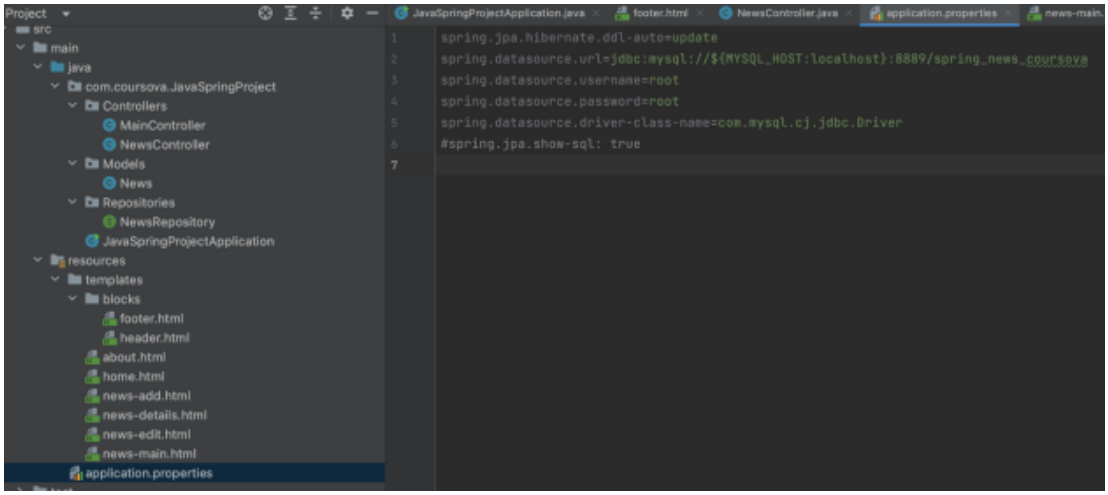

Рис. 3.3. Компоненти application.properties для використання бази даних

Одним за найважливіших конфігураційних файлів проекту Maven для проекту на основі Spring Boot є файл pom.xml (Project Object Model). POM файл (Project Object Model) використовується в Maven для опису проекту та його залежності. Він містить інформацію про проект, таку як назва, версія, автор (рис. 3.4.) та опис проекту. Крім того, POM файл визначає залежності проекту, які необхідні для його компіляції та виконання [31].

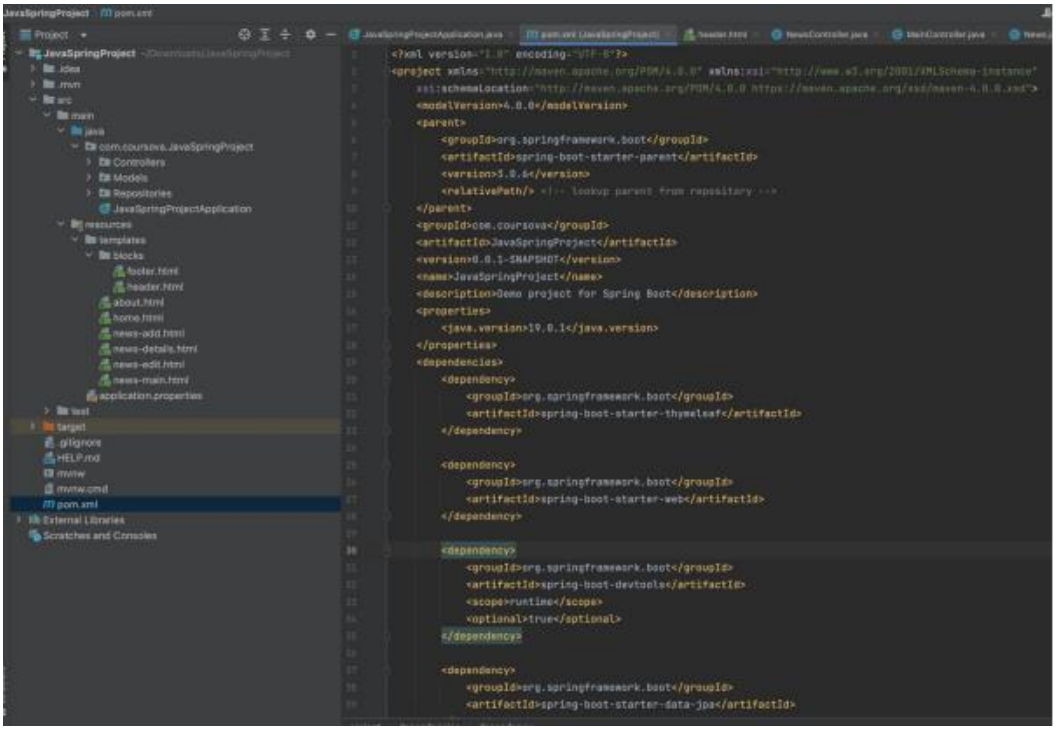

Рис. 3.4. Компоненти файлу pom.xml

У Spring Boot контролери використовуються для обробки HTTP–запитів [5]. Вони визначають методи, які будуть виконуватись при отриманні певного запиту. Контролери можуть повертати дані у різних форматах, таких як HTML–сторінки, JSON–відповіді або файли. Для розробки контролерів використовують анотації для програмування. Ви можете використовувати анотації, такі як @Controller, @RestController, @RequestMapping та інші, для визначення маршрутів, методів та обробки параметрів запиту.

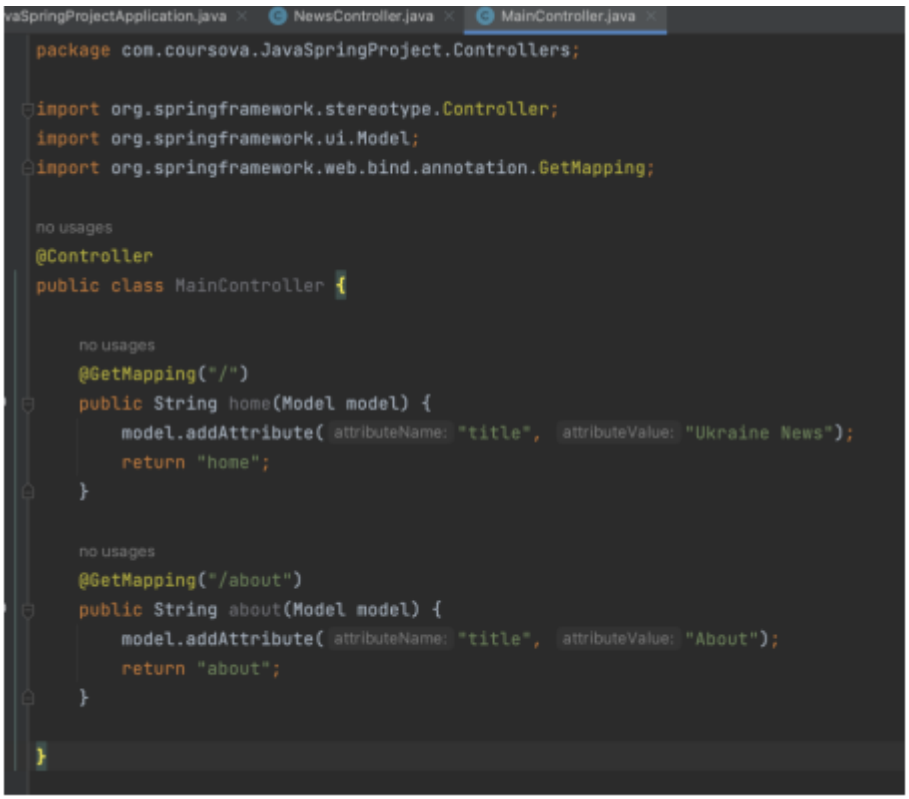

Рис. 3.5. MainController, для використання анотацій

Використовуючи анотації, такі як @Entity, @Table та інші, ви можете визначити моделі даних і налаштувати взаємодію з базою даних. Spring Boot автоматично налаштовує підключення до бази даних на основі конфігураційних налаштування.

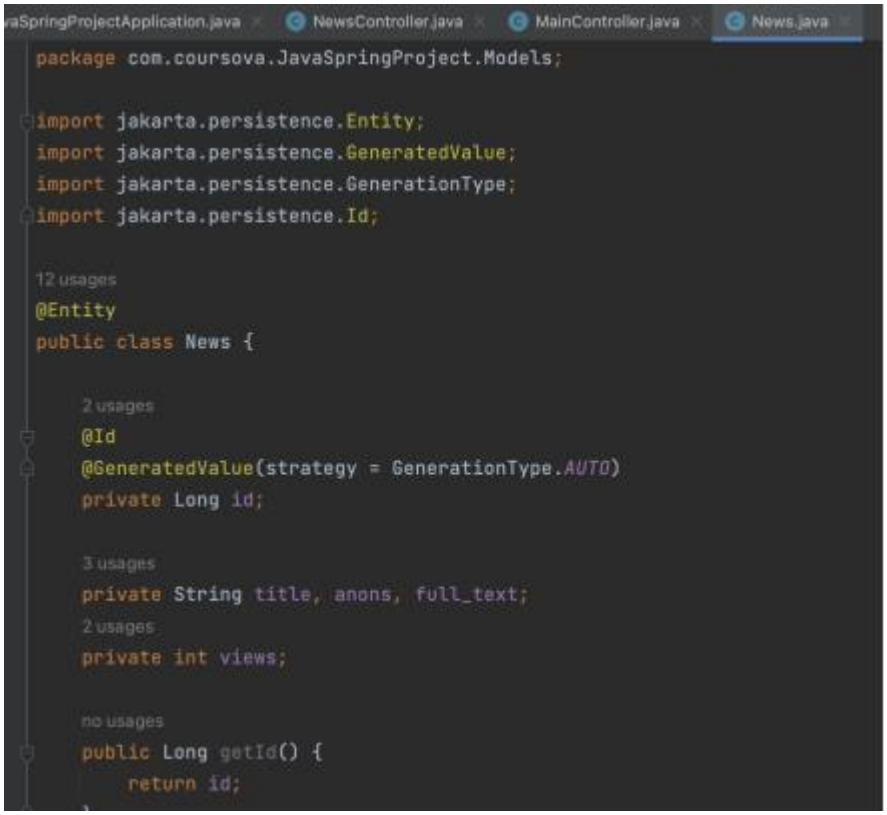

Рис. 3.6. Модель @ Entity

Для створення веб–додатку створено модель даних для сервісу.

Основними сутностями (рис.3.6.) моделі є Організація, Подія та Локація. Розглянемо (рис. 3.7.) кожну із сутностей. ER–діаграму сутності [32].

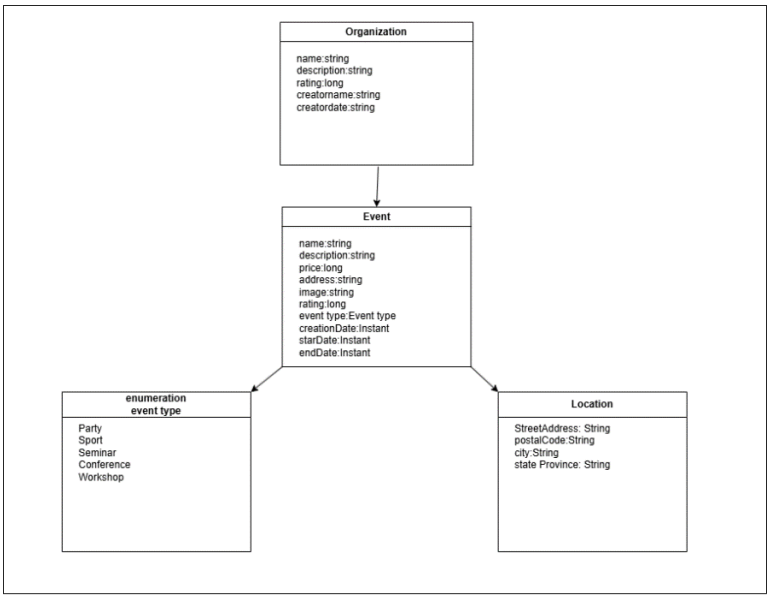

Рис. 3.7. ER Діаграма веб–додатку

Розглянемо докладніше кожну із сутностей.

Сутність Організація має такі поля.

– id – початковий ключ таблиці;

– name – поле, що містить ім'я організації;

– description – поле, що містить опис організації;

– rating – поле, що містить рейтинг організації;

– creatorName – поле, що містить назву засновника організації; – creationDate – поле, що містить дату створення організації.

Ця сутність являє собою компанію, що проводить певний захід.

Сутність Подія має такі поля:

– id – початковий ключ таблиці;

– name – поле, що містить назву події;

– description – поле, що містить опис події;

– price – поле, що містить ціну події;

– address – поле, що містить адресу події;

– image – поле, що містить шлях до картинки;

– rating – поле, що містить оцінку події;

– eventType – поле, що містить тип події;

– creationDate – поле, що містить дату створення події;

– startDate – поле, що містить дату початку події;

– endDate – поле, що містить дату закінчення події.

Ця сутність є основоположною в цьому мікросервісі.

Останньою сутністю є локація, яка зберігає в собі дані про розміщення події та містить такі поля:

– id – первинний ключ таблиці;

– streetAddress – поле, що містить адресу вулиці;

– postalCode – поле. Містить поштовий код;

– city – поле, що містить назву міста;

– stateProvince – поле, що містить назву району.

На основі цієї моделі даних можемо розробити програмну реалізацію цієї моделі використовуючи ORM (Object–relational mapping). ORM – це механізм, що дає змогу звертатися та керувати об'єктами, не зважаючи на те, як ці об'єкти зберігаються. Це ідея можливості писати запити в базу даних будь–якої складності, використовуючи об'єктно–орієнтовану парадигму бажаної мови програмування.

У цьому разі Spring надає зручний модуль для роботи з ORM – Spring Data. Цей модуль дає змогу за допомогою анотацій визначати сутності баз даних.

Як можна побачити на малюнку, необхідно створити так званий POJO клас і позначити його анотаціями @Entity і @Table, які позначають його як сутність і таблицю, і допоможуть (рис. 3.8.)модулю Spring Data відповідним чином обробити цей клас і створити на його основі відповідну таблицю в будь–якій базі даних.

```java
@Entity
@Table(name = "event")
@Cache(usage = CacheConcurrencyStrategy.NONSTRICT_READ_WRITE)
public class Event implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "name")
    private String name;

    @Column(name = "description")
    private String description;

    @Column(name = "price")
    private Long price;

    @Column(name = "address")
    private String address;

    @Column(name = "image")
    private String image;

    @Column(name = "rating")
    private Long rating;

    @Enumerated(EnumType.STRING)
    @Column(name = "event_type")
    private EventType eventType;

    @Column(name = "creation_date")
    private Instant creationDate;

    @Column(name = "start_date")
    private Instant startDate;

    @Column(name = "end_date")
    private Instant endDate;

    @OneToOne(fetch = FetchType.LAZY)

    @JoinColumn(unique = true)
    private Location location;

    @ManyToOne(fetch = FetchType.LAZY)
    @JsonIgnoreProperties("events")
    private Organization organization;
```

Рис. 3.8. Класи ORM

Усі ORM сутності знаходяться в пакеті (domain) і його загальна структура має такий вигляд:

–Для роботи з даними є модуль доступу до даних під назвою repository.

Для доступу до даних будемо (Рис.3.9.) використовувати Spring Data Repositories.

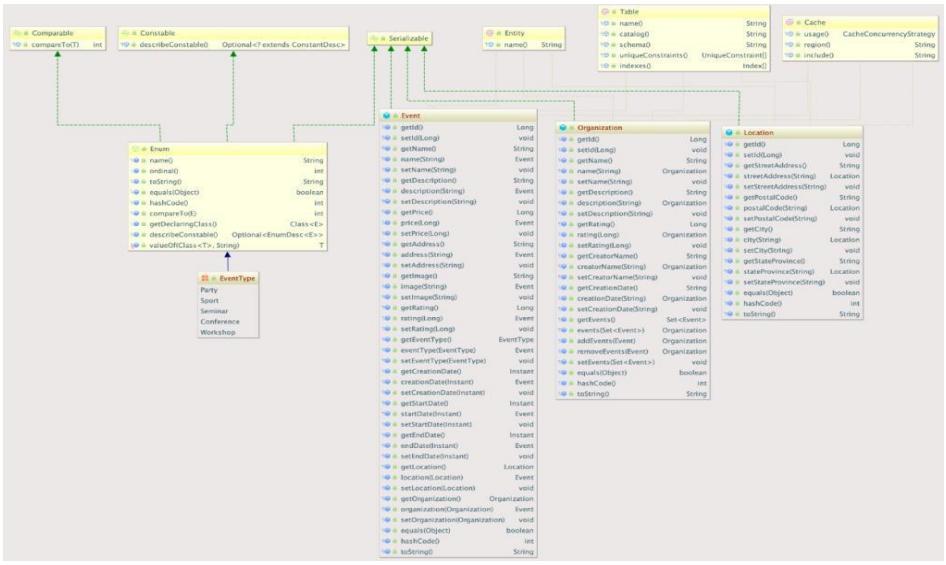

Рис. 3.9. Spring Data Repositories domain

Цей модуль надає змогу створювати запити до баз даних за допомогою програмного коду.

Для цього потрібно втілити інтерфейс JpaRepository. Після цього просто називаючи відповідним чином способи можна створювати запити до бази даних. Аналогом до наступного запиту до бази даних select e from Event e where e.name=?1 and e.address=?2 буде наступний програмний код зображений на (рис 3.10.)

```
@Repository
public interface EventRepository extends JpaRepository<Event, Long> {

    List<Event> findByNameAndAddress(final String name, final String address);

}
```

Рис. 3.10. Інтерфейс роботи JpaRepository.

3.2. Застосовані методи та техніки оптимізації

Щоб продемонструвати переваги тестування до оптимізація та після оптимізації, я обрав тестову систему і провів поточне порівняння веб–застосунку.

Я порівнював Lighthouse і Apache Jmeter для того щоб обрати певну систему тестування. У Lighthouse є тест на поведінку сайту при поганому як веб-з'єднання, а також при повній відсутності зв'язку. Це не дуже співвідноситься з перформанс-тестуванням як таким, проте, якщо задуматися, в деяких випадках веб-додаток сприймається повільним, а Apache Jmeter всі дії та супутні запити записуються є у вигляді скрипту. Отже я обрав саме Jmeter скільки він має більше переваг.

Веб–додаток, побудований на технології Spring Boot разом із взаємодією Spring Data JPA, Hibernate, Apache Jmeter. Кожна функція відповідає та взаємодіє з певною другорядною функцію, а додаток загалом був здатний обробляти велику кількість користувачів і трафіку завдяки своїй масштабованості та відмово стійкості [29].

Веб–додаток створений для демонстрації (рис. 3.11.) різноманітних актуальних новин.

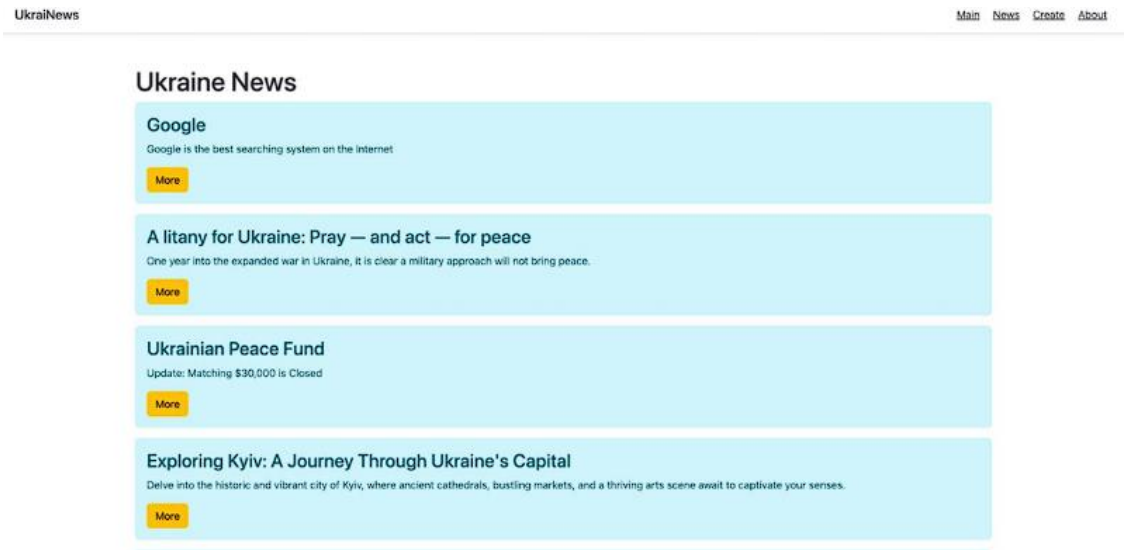

Рис. 3.11. Сторінка веб–додатку

Монолітний застосунок було побудовано як єдину велику кодову базу з усіма функціями в одному застосунку. Хоча це і спростило початкову розробку, це також ускладнило масштабування і підтримку застосунку в міру зростання його розміру і складності.

Для створення навантаження на систему було застосовано утиліту імітації відвідуваності Apache JMeter з рівномірним розподілом трафіку в часі та різними піками навантаження. Було встановлено обсяг трафіку 200 тис. запитів рівномірно розподілений на весь термін тестування з максимальним піковим навантаженням у 20 тис. запитів одночасно. У разі відмови системи виконувалося ручне перезавантаження і рестарт сервісу.

Метрики, які збиралися в процесі спостереження:

– число успішно оброблених запитів;

– кількість невдалих запитів до системи;

– час простою системи, коли вона не могла обробляти запити;

– пікове навантаження на систему

– максимальне (табл.3.1) число запитів перед відмовою системи, яке додаток зміг обробити.

Таблиця 3.1

Зібрана статистика роботи додатка до оптимізації за вказаний період

День	Средн. CPU, %	Средн. Memory, %	Успішні запити	Невдалі запити	Застій системи, мін.	Макс. число запитів перед відмовленням
1	78	86	14290	14930	29	9544
2	71	79	13849	13482	32	9391
3	75	85	14938	15731	35	9884
4	74	84	14831	14839	30	9102
5	76	81	13483	14470	25	9938
6	73	79	13382	14483	29	9543
7	76	84	13848	13444	28	9891
Разом:	–	–	98621	101379	208	–
Середній показник:	75	83	14089	14483	30	9613

3.3. Результати тестування та порівняння продуктивності після оптимізації

Для загального аналізу було використана попередня таблиця для підсумку тестування.

Було проведено тестування веб – додатка, і відображення наведеної статистики в таблиці (табл. 3.2.)

Таблиця 3.2

Зібрана статистика роботи додатка після оптимізації за вказаний період.

День	Средн. CPU, %	Средн. RAM, %	Успішні запити	Невдалі запити	Застій системи, мін.	Макс. число запитів перед відмовленням
1	44	52	22300	7391	15	16459
2	43	55	20844	7382	14	14688
3	41	51	19388	6990	9	15441
4	48	47	22013	6872	12	15970
5	51	55	20932	7644	11	14983
6	45	51	22561	7323	16	13081
7	49	46	21012	7348	10	15733
Разом:	–	–	149050	50950	87	–
Середнє:	46	51	21293	7279	12	15194

Список метрик, які були використанні в процесі розрахунку ефективності на підставі отриманої статистики:

–Час простою, який відноситься до часу, коли система не обробляла жодних запитів через недоступність;

–Частота відмов, яка показує відсоток запитів, які не були оброблені від загального числа запитів до системи;

–Використання ресурсів, що описує відсоток доступних ресурсів, використовуваних системою;

–Загальна кількість успішно (рис. 3.11) оброблених запитів за весь період тестування і спостереження [35].

Було зображено панель приладів графіків, які входять в Jmeter, для базової візуалізації метрик. Також Jmeter надає багатий інтерфейс користувача, що дозволяє досліджувати, створювати інформаційну панель.

Рис. 3.11. Детальний приклад тестів у вигляді інформаційної панелі до оптимізації

– пікове навантаження на систему, під яким розуміють максимальну кількість запитів, яку система може обробити (табл.3.3.) в будь–який момент часу.

Таблиця 3.3

Фінальний тест продуктивності після наведених тестів в реальному часі

Метрики Продуктивності	Архітектура мікро сервісу	Монолітна архітектура	Покращення
Період простою	~ 87 хв.	~ 208 хв.	На 58.2% менше простоїв системи
Коефіцієнт відмов	1.00%	4.00%	На 75% менше відмов системи
Використання ресурсів	48.00%	79.00%	На 39.24% ефективніше використання ресурсів
Сумарне число успішно оброблених запитів за весь період	Здатність обслуговувати сумарно 149 050 запитів без простоїв і збоїв	Здатність обслуговувати сумарно 98 621 запитів із періодичними простоями та відмовами	Поліпшення на 55.10% і відсутність простоїв
Пікове навантаження	15 194 запитів одночасно	9613 запитів одночасно	Поліпшення на 77.78%

Результати дослідження показали, що тестований (рис. 3.12) веб–додаток показав себе ефективно, в порівнянню з додатком який не проходив етапи тестування.

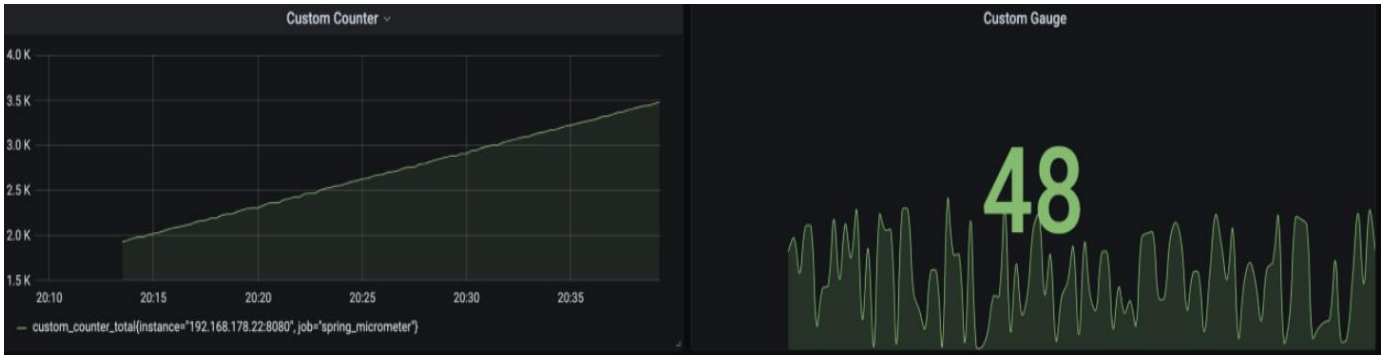

Рис 3.12. Тестування після оптимізації з відображеною поточною продуктивністю.

3.4. Висновки щодо ефективності застосованих методів.

Основні зміни після впровадження методів оптимізації веб–додатку:

–Веб–додаток мав на 58,2% менше простою у системі порівняно з не оптимізованим продуктом, що вказує на більш надійну та відмовостійку систему;

–Рівень відмов для архітектури мікросервісів також був значно нижчим – 1,00% порівняно з 4,00% для монолітної архітектури, що вказує на кращу ізоляцію та відновлення після збоїв в архітектурі мікросервісів;

–З точки зору використання ресурсів, мікросервісна архітектура була на 39,24% ефективнішою у використанні ресурсів порівняно з монолітною. Це можна пояснити тим, що кожен мікросервіс може масштабуватися незалежно залежно від попиту, тоді як у монолітній архітектурі масштабування здійснюється на рівні додатка;

–Додаток зміг успішно обробити 149 050 запитів без простоїв і збоїв, у той час як монолітна архітектура змогла обслужити тільки 98 621 запит із періодичними простоями та помилками, що свідчить про те, що архітектура продукту була надійнішою та ефективнішою;

–Оптимізація також змогла впоратися зі значно вищим піковим навантаженням, як порівняти з монолітною архітектурою, з покращенням на 77,78%.

Загалом, результати тестування обчислюють переваги веб–додатку з погляду масштабованості, відмово стійкості та використання ресурсів. Застосунок дає змогу розробляти і розгортати незалежні сервіси окремо, що полегшує обслуговування і масштабованість. Крім того, забезпечує відмово стійкість, за якої збій в одному сервісі не обов'язково призведе до відмови всього застосунку.

Під час тестування я виявили, що веб–додаток архітектура значно перевершує монолітну архітектуру за кількома ключовими параметрами. По–перше, забезпечує легку масштабованість і відмово стійкість, даючи змогу застосунку обробляти велику кількість користувачів і трафіку без простоїв і збоїв.

Крім того, застосунок був гнучкіший та адаптований, даючи змогу використовувати різні технології та бази даних для створення нових сервісів. Це полегшило додавання нових функцій у застосунок у міру необхідності та зменшило складність кодової бази.

Крім того, веб архітектура була економічним та ефективним з погляду використання ресурсів, оскільки розгортати та масштабувати потрібно було тільки необхідні сервіси. Це призвело до зниження витрат і підвищення продуктивності порівняно з монолітною архітектурою.

Загалом, наше порівняння продемонструвало явні переваги мікросервісної архітектури перед монолітною, особливо в плані масштабованості, гнучкості, відмово стійкості та використання ресурсів. Мікросервісна архітектура є більш надійним і ефективним рішенням для розробки складних, масштабованих веб–додатків.

Висновки до розділу:

Я розробив та протестував додаток за допомогою Spring Boot, Apache Jmeter і виявив, що оптимізація необхідна, оскільки зменшується завантаження веб–додатку, відбувається підвищення продуктивності, зменшується час відклику користувача який використовує повний функціонал розробленої та протестованої системи.

РОЗДІЛ 4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1. Охорона праці

Метою кваліфікаційної роботи магістра є забезпечення безпечних умов праці для фахівців, які займаються розробкою та тестуванням веб–додатків. Це є ключовим аспектом організації робочого процесу, оскільки безпека працівників безпосередньо впливає на якість та ефективність виконуваних завдань. Особливу увагу необхідно приділити дотриманню вимог охорони праці при роботі з екранними пристроями, які є невід'ємною частиною повсякденної діяльності програмістів і тестувальників. Важливо організувати електромережу відповідно до сучасних стандартів, забезпечити належний рівень пожежної безпеки, а також створити комфортне робоче середовище, яке сприятиме продуктивності працівників.

Відповідальність за виконання цих вимог покладається на керівництво організації, яке повинно забезпечити, щоб робочі місця відповідали нормативам ергономіки, освітлення, електробезпеки та санітарно–гігієнічних стандартів. Це не лише сприяє підвищенню продуктивності працівників, але й мінімізує ризики для їхнього здоров'я, що є вкрай важливим у контексті тривалого перебування за комп'ютером. Забезпечення безпеки праці також грає важливу роль у забезпеченні безперервності роботи над проектами, зокрема тими, що реалізуються за допомогою таких технологій, як Spring Boot, що вимагають високої концентрації та тривалої роботи.

Для забезпечення ефективної та безпечної діяльності колективу, що займається тестуванням програмного забезпечення, необхідно створити відповідні умови праці, які включають не лише фізичні аспекти, але й психологічний комфорт. Керівник організації несе пряму відповідальність за дотримання всіх вищезазначених вимог, що передбачає регулярний моніторинг умов праці, проведення навчань з охорони праці для співробітників, а також впровадження системи управління безпекою праці,

яка дозволить оперативно реагувати на виникаючі проблеми та покращувати умови роботи.

Керівник організації несе пряму відповідальність за дотримання норм і правил охорони праці, що є важливим аспектом забезпечення безпеки на робочому місці. Відповідно до вимог законодавства, на робочих місцях працівників мають виконуватись вимоги, затверджені наказом Міністерства соціальної політики України від 14 лютого 2018 року № 207 «Про затвердження Вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями». Цей наказ визначає основні принципи організації робочого процесу, що сприяють зменшенню ризиків для здоров'я працівників, які працюють з комп'ютерними технологіями.

Приміщення, де розташовані серверні операторські робочі місця, повинні бути обладнані системами пожежної сигналізації та засобами пожежогасіння, що відповідають вимогам Державних будівельних норм (ДБН) В.2.5–56:2014, а також нормативним актам НАПБ А.01.001–2014 і технічної документації, що регламентує ці питання. Це забезпечує не лише безпеку працівників, але й захист обладнання та приміщень від можливих пожежних загроз. Важливо, щоб доступ до засобів пожежогасіння залишався вільним, що дозволяє у разі необхідності оперативно реагувати на надзвичайні ситуації.

Електромережі для живлення комп’ютерів та периферійного обладнання слід організовувати як окрему три провідну групову мережу, яка включає фазовий, нульовий робочий та нульовий захисний провідники. Захисний провідник призначений для заземлення, тоді як робочий провідник використовується для передавання електричного струму. Важливо, щоб ці два типи провідників не були сполучені, оскільки це може призвести до небезпечних ситуацій. Розділення таких провідників має бути забезпечене як на розподільних щитах, так і в усіх точках підключення, що гарантує безпечну експлуатацію електричних систем і запобігає можливим аваріям.

У три провідних електричних мережах важливо дотримуватись певних технічних вимог, зокрема щодо сумарної площі перерізу нульових провідників. Ця

площа повинна бути еквівалентною площі перерізу фазового провідника. Це забезпечує належну роботу електромережі та безпеку її експлуатації. Провідники, які використовуються в таких мережах, повинні відповідати специфікаціям електромережі, включаючи навантаження, температурні умови та вимоги захисного обладнання, що регламентується нормативними документами, зокрема НПАОП 40.1–1.01–97 [34].

Крім того, у приміщеннях, де розміщено більше п’яти комп’ютерів, необхідно встановлювати аварійний вимикач. Цей пристрій дозволяє оперативно відключити електропостачання в усьому приміщенні, за винятком освітлення, що є критично важливим для забезпечення безпеки в разі надзвичайних ситуацій. Важливо також, щоб підключення комп’ютерної техніки до електромережі здійснювалося виключно через сертифіковані розетки та з'єднання. Використання перехідників для підключення до двопровідної мережі категорично заборонено, оскільки це може призвести до небезпечних ситуацій, пов'язаних з електричним струмом та можливими короткими замиканнями. Таким чином, дотримання цих вимог є запорукою безпечної та надійної роботи електричних систем у сучасних офісних умовах.

4.2. Безпека в надзвичайних ситуаціях

Особливої гостроти й актуальності на сучасному етапі розвитку суспільства набула проблема безпеки життєдіяльності.

Безпека життєдіяльності – це наука, покликана виявляти можливі причини та шляхи виникнення небезпеки, передбачати вірогідність її виникнення, а також захищати людей від цієї небезпеки, ліквідовувати наслідки її проявів тощо. Ця наука ідентифікує небезпечні та шкідливі чинники навколишнього середовища, розроблює заходи, пов’язані зі створенням сприятливих умов для існування людини.

Екологічна безпека окремої людини, нації, цивілізації залежить як від діяльності кожної людини, так і від впливу всього суспільства на природу – біосферу. Здоров’я людини є одним з найважливіших нормативних показників успішності

природокористування. Стан навколишнього середовища і здоров'я людини – тісно пов'язані між собою [6].

Адже негативні екологічні явища сприяють виникненню тих чи інших захворювань, зумовлюють підвищення смертності, скорочення середньої тривалості життя тощо. Європейський центр ВООЗ з навколишнього середовища і здоров'я для оцінки зв'язку між довкіллям і здоров'ям населення та створення відповідних інформаційних систем рекомендує основні групи індикаторів:

– стан здоров'я (смертність, захворюваність, поширеність хвороб);

фізичне середовище (індикатори стану і впливу)

– забезпечення житлом, якість питної води і атмосферного повітря, радіація, шум;

–умови праці (вплив чинників на організм);

–захист здоров'я (нормативне забезпечення якості продуктів харчування);

–служби охорони здоров'я.

Аналізуючи вплив негативних екологічних (антропогенних) факторів на основні показників здоров'я населення можна виділити наступні напрями:

– на соматичному рівні

– погіршення стану здоров'я в результаті несприятливої антропогенної екологічної ситуації, несприятливих умов трудової діяльності;

– на психічному рівні – погіршення стану здоров'я внаслідок тривалої соціально екологічної напруженості, стресових ситуацій, зумовлених техногенними аваріями і катастрофами;

– на соціальному рівні – невідповідність між обсягом і якістю доступних медичних послуг і реальним станом здоров'я населення, обумовленими впливом антропогенного екологічної ситуації [7].

погіршення демографічних показників – зниження тривалості і якості життя, зменшення народжуваності, зростання захворюваності і смертності. Основою збереження життя на планеті та покращення національного добробуту й потенційних можливостей держав світу сьогодні стає, насамперед, пошук шляхів і засобів

нейтралізації і подолання негативних тенденцій, які становлять реальну загрозу для безпечного існування суспільства.

Не менш важливе значення має також активна участь кожної держави в заходах з попередження та зменшення негативних наслідків екологічних загроз, підвищення дієвості й ефективності системи захисту населення і територій від надзвичайних ситуацій різного походження, всебічного використання наукових розробок, використання кращого досвіду розвинених країн у цих напрямках.

Основи екологічної безпеки в Україні закріплені на конституційному рівні, що підкреслює важливість цього аспекту для держави та суспільства. Зокрема, у Декларації про незалежність України чітко проголошується необхідність забезпечення екологічної безпеки, що є невід'ємною частиною державної політики. У статті 16 Конституції України зазначається, що екологічна безпека та екологічна рівновага на території країни, а також збереження генофонду народу, є обов'язком держави. Це свідчить про високий рівень відповідальності, яку несе держава перед своїми громадянами у питаннях охорони навколишнього середовища.

Крім того, статті 49 та 50 Конституції України гарантують право кожної людини на охорону здоров'я, медичну допомогу та безпечне для життя і здоров'я природне середовище. Ці статті підкреслюють важливість екологічної безпеки для забезпечення фізичного та психічного благополуччя населення. У разі порушення цих прав, Конституція передбачає можливість отримання компенсації за завдану шкоду, що створює правову основу для захисту прав громадян.

Для забезпечення екологічної безпеки та здоров'я населення особливу увагу слід приділяти переліку видів діяльності та об'єктів, які становлять підвищену екологічну небезпеку. Цей перелік є важливим інструментом для ідентифікації потенційно небезпечних об'єктів, які можуть негативно впливати на здоров'я людей. Відповідно до законодавства, біля таких об'єктів повинні бути встановлені санітарно–захисні зони, які покликані зменшити шкідливий вплив на навколишнє середовище та забезпечити безпечні умови для життя і діяльності населення.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи було проведено комплексне дослідження продуктивності веб-додатків та запропоновано ефективні підходи для його підвищення:

1.Проведено аналіз продуктивності веб-додатків, що дозволило визначити основні фактори які впливають не лише швидкість виконання завдань, але й на зручність використання для кінцевого користувача.

2.Виокремлено основні аспекти оптимізації продуктивності, серед яких налаштування запитів до бази даних, використання кешування та балансування навантаження, що сприяють стабільній роботі під час пікових навантажень.

3.Досліджено вплив фреймворку Spring Boot на продуктивність веб-додатків. Розглянуто методи оптимізації, зокрема ефективне управління базами даних, реалізацію асинхронних викликів та профілювання продуктивності додатків. Окрему увагу приділено перевагам мікросервісної архітектури, таким як масштабованість, гнучкість та відмовостійкість..

4.Проведено огляд інструментів тестування продуктивності веб-додатків, що дозволило підтвердити ефективність запропонованих рішень.

5.У практичній частині реалізовано веб-додаток із використанням Spring Boot, який використовувався для експериментальної перевірки методів оптимізації продуктивності.

6.Результати тестування показали наступне:

–Зменшено простій на 58,2% порівняно з не оптимізованим продуктом, що вказує на більш надійну та відмово стійку систему.

–Знижено рівень відмов до 1,00% у мікросервісній архітектурі порівняно з 4,00% для монолітної архітектури.

–З точки зору використання ресурсів, мікро сервісна архітектура була на 39,24% ефективнішою у використанні ресурсів порівняно з монолітною. Це можна пояснити тим, що кожен мікро сервіс може масштабуватися незалежно.

–Оброблено на 50% більше запитів без збоїв (149 050 проти 98 621).

–Пікова продуктивність зросла на 77,78%.

Таким чином впроваджені методи оптимізації дозволяють створювати сучасні, економічно ефективні веб-рішення, що відповідають актуальним вимогам бізнесу, зменшують навантаження на серверну частину та зменшують час відгуку для кінцевих користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Луцик Н.С., Луцків А.М., Осухівська Г.М., Тиш Є.В. Програма та методичні рекомендації з проходження практики за тематикою кваліфікаційної роботи для студентів спеціальності 123 «Комп'ютерна інженерія» другого (магістерського) рівня вищої освіти усіх форм навчання. Тернопіль: ТНТУ. 2024. 45 с.

2. Луцик Н.С., Луцків А.М., Осухівська Г.М., Тиш Є.В. Методичні рекомендації до виконання кваліфікаційної роботи магістра для студентів спеціальності 123 «Комп'ютерна інженерія» другого (магістерського) рівня вищої освіти усіх форм навчання. Тернопіль. 2024. 44 с.

3. Варавін А.В., Лещишин Ю.З., Чайковський А.В. Методичні вказівки до виконання курсового проєкту з дисципліни «Дослідження і проєктування комп'ютерних систем та мереж» для здобувачів другого (магістерського) рівня вищої освіти спеціальності 123 «Комп'ютерна інженерія» усіх форм навчання. Тернопіль: ТНТУ, 2024. 32 с.

4. Шеремета В.З., Жаровський Р.О. Використання Spring Boot та інтеграція другорядних ісінструментів для створення сучасних веб–додатків . Матеріали XII міжнародної науково–практичної конференції молодих учених та студентів «Актуальні задачі сучасних технологій» (11–12 грудня 2023 року). Тернопіль: ТНТУ. 2024. С. 439.

5. Шеремета В.З., Жаровський Р.О. Тестування веб–додатків розробленими на основі Spring Boot за допомогою Testing. Матеріали XI науково–технічної конференції Тернопільського національного технічного університету імені Івана Пулюя «Інформаційні моделі, системи та технології» (18–19 грудня 2024 року). Тернопіль: ТНТУ. 2024. С. 162.

6. Стручок, Володимир Сергійович. "Безпека в надзвичайних ситуаціях. Методичний посібник для здобувачів освітнього ступеня «магістр» всіх спеціальностей денної та заочної (дистанційної) форм навчання." Тернопіль: ТНТУ. 2022. С. 67.

7. Стручок, Володимир Сергійович. "Техноекологія та цивільна безпека. Частина «Цивільна безпека». Навчальний посібник." Тернопіль: ТНТУ. 2022. С. 42.

8. Слюз, І., & Жаровський, Р. О. (2022). Принципи та основні етапи комплексного тестування комп'ютерної інформаційної системи. Матеріали X науково–технічної конференції „Інформаційні моделі, системи та технології "Тернопільського національного технічного університету імені Івана Пулюя, 94–94.

9. Слюз, І., & Жаровський, Р. О. (2022). Критерії ефективності тестування комп'ютерної інформаційної системи. Матеріали XI Міжнародної науково–практичної конференції молодих учених та студентів „Актуальні задачі сучасних технологій ", 174–174.

10. Свергун, С., and Руслан Олегович Жаровський. "Тестування програмного забезпечення побудованого на мікросервісній архітектурі." Матеріали X науково–технічної конференції „Інформаційні моделі, системи та технології "Тернопільського національного технічного університету імені Івана Пулюя (2022): 92–92.

11. Свергун, С., and Руслан Олегович Жаровський. "Тестування програмного продукту, побудованого на мікросервісній архітектурі на основі BDD." Матеріали X науково–технічної конференції „Інформаційні моделі, системи та технології "Тернопільського національного технічного університету імені Івана Пулюя (2022): 93–93.

12. Yatsyshyn, V., Pastukh, O., Palamar, A., & Zharovskyi, R. (2023). Technology of relational database management systems performance evaluation during computer systems design. Вісник Тернопільського національного технічного університету, 109(1), 54–65.

13. ЯЦИШИН, В. В., et al. SOFTWARE TOOL FOR PRODUCTIVITY METRICS MEASURE OF RELATIONAL DATABASE MANAGEMENT SYSTEM. Математичне моделювання, 2023, 1 (48): 7–17.

14. Madhusudhan Konda. Just Hibernate: A Lightweight Introduction to the Hibernate Framework. Newton, Massachusetts, United States : O'Reilly Media, 2014. 193

р.

15. Baron Schwartz, Peter Zaitsev, Vadim Tkachenko High Performance MySQL. Gravenstein Highway North, Sebastopol: O'Reilly Media, Inc., 2012. 771 с.

16. Building an Application with Spring Boot [Електронний ресурс]. Режим доступу: https://spring.io/guides/gs/spring–boot/ (дата звертання 13.05.2021)

17. Spring best practices [Електронний ресурс]. Режим доступу: https://www.endoflineblog.com/spring–best–practices (дата звертання 11.05.2021)

18. Software Testing Methodologies [Електронний ресурс]. Режим доступу: https://smartbear.com/learn/automated–testing/softwaretesting–methodologies/ (дата звертання 12.04.2021)

19. The History of Software Testing [Електронний ресурс]. Режим доступу: http://www.testingreferences.com/testinghistory.php (дата звертання 03.04.2021)

20. Restfulapi – What is REST – REST API Tutorial [Електронний ресурс] https://restfulapi.net/

21. Web MVC framework docs.spring.io: вебсайт. URL: https://docs.spring.io/spring–framework/docs/3.2.x/spring–frameworkreference/html/mvc.html (дата звернення: 03.03.2023)

22. Defining JPA Entities baeldung.com: вебсайт. URL: https://www.baeldung.com/jpa–entities (дата звернення: 06.03.2023)

23. Spring–boot spring.io: вебсайт. URL: https://spring.io/projects/springboot (дата звернення: 04.03.2023)

24. Hibernate, Reference Documentation / Електронний ресурс. – Режим доступу: https://hibernate.org/orm/documentation/6.0/

25. Bionics: Оптимізація та масштабування веб–застосунків [Електронний ресурс] – Режим доступу до ресурсу: http://bionics.nure.ua/article/view/228458.

26. Spring boot – how thymeleaf works? – geeksforgeeks [Електронний ресурс] // GeeksforGeeks. – Режим доступу: https://www.geeksforgeeks.org/springboot–how–thymeleaf–works/ (дата звернення: 16.05.2024).

27. Spring Boot Tutorial [Електронний ресурс]. – Режим доступу: https://www.javatpoint.com/spring–boot–tutorial

28. Spring Boot Starter Web [Електронний ресурс]. – Режим доступу: https://www.javatpoint.com/spring–boot–starter–web

29.Building REST services with Spring [Електронний ресурс]. – Режим доступу: https://spring.io/guides/tutorials/rest/

30. The History of Software Testing [Електронний ресурс]. Режим доступу: http://www.testingreferences.com/testinghistory.php (дата звертання 03.04.2021)

31. Software Testing Methodologies [Електронний ресурс]. Режим доступу: https://smartbear.com/learn/automated–testing/softwaretesting–methodologies/ (дата звертання 12.04.2021)

32. Spring Boot – Best Practices [Електронний ресурс]. Режим доступу: https://www.e4developer.com/2018/08/06/spring–boot–best–practices/ (дата звертання 13.05.2021)

33. Building an Application with Spring Boot [Електронний ресурс]. Режим доступу: https://spring.io/guides/gs/spring–boot/ (дата звертання 13.05.2021)

34. Соколан, Ю. С. (2021). Проблематика забезпеченості спеціалізованим програмним забезпеченням в сфері охорони праці. Всеукраїнська науково–практична конференція «Проблеми та перспективи розвитку охорони праці». Львів, 16–17.

Додаток А

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Тернопільський національний технічний університет імені Івана Пулюя (Україна)
Університет імені П'єра і Марії Кюрі (Франція)
Маріборський університет (Словенія)
Технічний університет у Кошице (Словаччина)
Вільнюський технічний університет ім. Гедимінаса (Литва)
Наукове товариство ім. Т.Шевченка

АКТУАЛЬНІ ЗАДАЧІ СУЧАСНИХ ТЕХНОЛОГІЙ

Збірник
тез доповідей

XIII Міжнародної науково-практичної конференції молодих учених та студентів
11-12 грудня 2024 року

УКРАЇНА
ТЕРНОПІЛЬ – 2024

Матеріали XIII Міжнародної науково-практичної конференції молодих учених та студентів
«АКТУАЛЬНІ ЗАДАЧІ СУЧАСНИХ ТЕХНОЛОГІЙ» – Тернопіль, 11-12 грудня 2024 року

УДК 621.855

В.З. Шеремета, Р.О. Жаровський, к.т.н
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

ВИКОРИСТАННЯ SPRING BOOT ТА ІНТЕГРАЦІЯ ДРУГОРЯДНИХ ІНСТРУМЕНТІВ ДЛЯ СТВОРЕННЯ СУЧАСНИХ ВЕБ–ДОДАТКІВ

V. Z. Sheremeta, R.O. Zharovskyi, Ph.D

USING SPRING BOOT AND INTEGRATION OF 3RD-TIER TOOLS TO CREATE MODERN WEB APPLICATIONS

У сучасному світі веб-розробка є ключовою сферою інформаційних технологій, що забезпечує користувачам доступ до різноманітних послуг, додатків і ресурсів через Інтернет. Для створення ефективних веб-застосунків необхідні потужні інструменти та фреймворки, які спрощують процес розробки й підвищують продуктивність.

Одним із найпопулярніших рішень для розробки веб-додатків на Java є Spring Boot, що пропонує простий і зручний підхід до створення застосунків завдяки вбудованим функціям і готовим компонентам, які значно прискорюють розробку та покращують її якість.

Це дозволяє користувачам створювати самодостатні програми, які можуть працювати незалежно без будь-яких зовнішніх залежностей. Spring boot – це різновид Spring фреймворку. Spring Boot використовує переваги платформи Spring і тісно взаємодіє з нею. Він пропонує попередньо налаштовану структуру для розробки додатків на основі Spring з мінімальною конфігурацією, покладаючись на XML.

Spring Boot має низку переваг: він включає вбудовані сервери (Tomcat, Jetty, Undertow), що дозволяє створювати автономні Java-додатки із вбудованим веб-сервером, усуваючи потребу в окремих контейнерах; забезпечує параметри за замовчуванням для різних компонентів, таких як база даних, журналювання та безпека, що пришвидшує розробку; підтримує архітектури мікросервісів через функції Spring Cloud для створення розподілених систем; легко інтегрується з екосистемою Spring, дозволяючи використовувати її бібліотеки; має зручну зовнішню конфігурацію для адаптації без змін коду; надає інструменти для розробників, як-от Spring Initializr і Spring Boot CLI, для швидкого створення проектів; включає модуль Spring Boot Actuator для моніторингу та керування додатками; і спрощує тестування через вбудовані фреймворки для різних типів тестів.

Взаємодія з іншими API у Spring Boot є важливою складовою для створення розширених та інтегрованих сервісів. Фреймворк надає зручні інструменти для спрощення цього процесу, включаючи вбудовану підтримку для викликів RESTful API. Для цього можна використовувати бібліотеку Spring Web, яка дозволяє легко взаємодіяти з іншими веб-службами через HTTP-запити. Крім того, Spring Boot надає можливість інтеграції з іншими API за допомогою бібліотек, таких як Feign або Retrofit, що спрощують взаємодію з віддаленими сервісами та дозволяють швидко створювати клієнтські компоненти для взаємодії з іншими API. Це дозволяє розробникам легко інтегрувати свої застосунки з різними сервісами та джерелами даних, що розширює можливості їх функціональності та забезпечує більш гнучке та розширене використання. Таким чином, взаємодія з іншими API у Spring Boot сприяє створенню більш інтегрованих та функціональних застосунків.

Підтримка RESTful API у Spring Boot є однією з його ключових можливостей, що дозволяє розробникам створювати потужні та гнучкі веб-сервіси. Фреймворк надає простий та зручний спосіб створення API, зокрема для виконання операцій CRUD (створення, читання, оновлення, видалення).

Крім базових операцій CRUD, Spring Boot також підтримує різноманітні технологічні можливості для реалізації розширених функціональностей у веб-сервісах. Наприклад, вбудовані анотації Spring Security дозволяють забезпечити аутентифікацію та авторизацію користувачів, забезпечуючи безпеку API.

11. **Nadiia SKLIAROVA, Tymophii LANEVYCH** **408**
EVELOPMENT AND MANAGEMENT STRATEGIES FOR THE ADMIN WEBSITE
12. **А. В. Островський, Р. І. Королюк** **409**
ДОСЛІДЖЕННЯ ДАВАЧІВ ДЛЯ СИСТЕМИ ВИЯВЛЕННЯ РОЄВОГО СТАНУ БДЖОЛОСІМ'Ї
13. **А. Луцків, А. Люлька** **410**
ЗАСТОСУВАННЯ МЕТОДУ БЕЗПЕРЕРВНОГО ХЕШУВАННЯ У ПЛАНУВАННІ ВИКОНАННЯ ПОСЛІДОВНИХ ПРОЦЕСІВ
14. **А.В. Зінченко** **411**
ТЕЛЕРАДІОЛОГІЯ ЯК НЕВІДЄМНА СКЛАДОВА ТЕЛЕМЕДИЦИНИ
15. **А.Є. Олексяк; В.М. Семисюк, В.В. Левицький** **413**
ДОСЛІДЖЕННЯ АВТОМАТИЗОВАНОЇ СИСТЕМИ КОНДИЦІЮВАННЯ ПОВІТРЯ
16. **А.М. Ковтко; В.Б. Савків, І.Р. Козбур** **415**
АВТОМАТИЗОВАНА СИСТЕМИ ГЕНЕРАЦІЇ МОДУЛЬНИХ ТЕСТІВ НА ОСНОВІ ШТУЧНОГО ІНТЕЛЕКТУ ТА МУТАЦІЙНОГО ТЕСТУВАННЯ
17. **А.М. Паламар, І.І. Куляс, Ю.О. Тимошенко** **417**
РОЗРОБКА КОМП'ЮТЕРИЗОВАНОЇ IOT-СИСТЕМИ ДЛЯ ПІДТРИМКИ БЕЗПЕКИ ЛЮДЕЙ ПОХИЛОГО ВІКУ
18. **А.М. Паламар, О.Р. Вергун, М.Р. Франків** **418**
КОМП'ЮТЕРИЗОВАНА IOT-СИСТЕМА ДЛЯ МОНІТОРИНГУ ІОНІЗУЮЧОГО ВИПРОМІНЮВАННЯ
19. **О.В. Палка** **419**
ІНФОРМАЦІЙНІ ПАНЕЛІ ЯК ІНФОРМАЦІЙНО-ТЕХНОЛОГІЧНИЙ ІНСТРУМЕНТ ДЛЯ ВІДСТЕЖЕННЯ KPI У РОЗУМНОМУ МІСТІ
20. **А.С. Луговий, Є.В. Тиш** **420**
МЕТОДИ ТА ЗАСОБИ РОБОТИ ETL-ПРОЦЕСІВ В УМОВАХ ВИСОКИХ НАВАНТАЖЕНЬ
21. **Башняк В.Т., Дунець В.Л** **421**
ДОСЯГНЕННЯ ТА ВИКЛИКИ ВИЯВЛЕННЯ ТА КЛАСИФІКАЦІЇ БЕЗПІЛОТНИКІВ
22. **В.А. Готович, Д.В. Граб** **424**
АКТУАЛЬНІСТЬ ЗАДАЧІ РОЗРОБКИ МОДУЛЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ УПРАВЛІННЯ IT-ПРОЄКТАМИ
23. **В.А. Готович, В.С. Бондаренко** **426**
АКТУАЛЬНІСТЬ ЗАДАЧІ РОЗРОБКИ ДОДАТКУ ВІДЕОТРАНСЛЯЦІЇ ПІД МОБІЛЬНІ ПРИСТРОЇ НА БАЗІ ОПЕРАЦІЙНОЇ СИСТЕМИ ANDROID
24. **В.А. Лабчук** **428**
ДОСЛІДЖЕННЯ ПОКРАЩЕННЯ ШВИДКОДІЇ ОБРОБКИ ДАНИХ У PLINQ В ПОРІВНЯННІ З LINQ ДЛЯ РІЗНИХ РОЗМІРІВ ВХІДНИХ ДАНИХ
25. **В.Г. Хомишин** **430**
ЗАХИСТ ASP.NET CORE ВЕБ-ДОДАТКІВ ВІД ВПРОВАДЖЕННЯ SQL-КОДУ
26. **В.З. Шеремета, Р.О. Жаровський** **432**
МЕТОДИ І ІНСТРУМЕНТИ ПІДВИЩЕННЯ ПРОДУКТИВНОСТІ ВЕБ–ДОДАТКІВ З ВИКОРИСТАННЯМ SPRING BOOT
27. **В.З. Шеремета,Р.О. Жаровський** **434**
ВИКОРИСТАННЯ SPRING BOOT ТА ІНТЕГРАЦІЯ ДРУГОРЯДНИХ ІНСТРУМЕНТІВ ДЛЯ СТВОРЕННЯ СУЧАСНИХ ВЕБ–ДОДАТКІВ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
імені ІВАНА ПУЛЮЯ
НАУКОВЕ ТОВАРИСТВО ім. ШЕВЧЕНКА

М А Т Е Р І А Л И

XII НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

18-19 грудня 2024 року

ТЕРНОПІЛЬ
2024

Д. Романюк, І. Мудрик
ВПЛИВ СУЧАСНИХ ТЕХНОЛОГІЙ НА БУХГАЛТЕРСЬКИЙ ОБЛІК
D. Romaniuk, Ph.D.; I. Mudryk
IMPACT OF MODERN TECHNOLOGIES ON ACCOUNTING 183
О. А. Саган, К. Б. Шварло
ІННОВАЦІЙНІ СТРАТЕГІЇ АДАПТИВНОЇ МОБІЛЬНОЇ ВЕРСТКИ: ЗАБЕЗПЕЧЕННЯ ШВИДКОСТІ, ЗРУЧНОСТІ ТА ФУНКЦІОНАЛЬНОСТІ
O. A. Sahan, K. B. Shvyrlo
INNOVATIVE STRATEGIES OF ADAPTIVE MOBILE WEBSITE: PROVIDING SPEED, CONVENIENCE AND FUNCTIONALITY 185
В. Сарновський, Ю. Стоянов
ПРОЄКТУВАННЯ АВТОМАТИЗОВАНОЇ СИСТЕМИ УПРАВЛІННЯ ЗАЯВКАМИ ГРОМАДЯН З ВИКОРИСТАННЯМ ТЕХНОЛОГІЙ .NET
V. Sarnovskyi, Y. Stoyanov
DESIGNING AN AUTOMATED SYSTEM FOR MANAGING CITIZENS' APPLICATIONS USING .NET TECHNOLOGIES 186
М. Стрілецький
МЕТАМОДЕЛЬ ФОРМУВАННЯ ПАРНИКОВИХ ГАЗІВ МАЛИМИ ПІДПРИЄМСТВАМИ МАШИНОБУДІВНОГО СПРЯМУВАННЯ
M. Striletskyi
METAMODEL OF GREENHOUSE GAS FORMATION BY SMALL MACHINE-BUILDING ENTERPRISES 187
С. Сучков
СЕРВІСИ ДЛЯ МАШИННОГО ПЕРЕКЛАДУ КЛЮЧОВИХ СЛІВ ТА АНОТАЦІЇ НАУКОВИХ ТЕКСТІВ
S. Suchkov
SERVICES FOR MACHINE TRANSLATION OF KEYWORDS AND ANNOTATION OF SCIENTIFIC TEXTS 189
В. Сухарський, М. Петрик
РОЗРОБКА ANDROID-ЗАСТОСУНКУ ДЛЯ УПРАВЛІННЯ СКЛАДОМ З ВИКОРИСТАННЯМ JAVA ТА SQL SERVER
V. Sukharskyi, M. Petryk
DEVELOPMENT OF AN ANDROID APPLICATION FOR WAREHOUSE MANAGEMENT USING JAVA AND SQL SERVER 190
М. Франчевський, М. Петрик
РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ ДЛЯ ПЕРСОНАЛІЗОВАНОЇ АНАЛІТИКИ ПРОГРЕСУ ЧИТАННЯ
M. Franchevskyy, M. Petryk
DEVELOPMENT OF A MOBILE APPLICATION FOR PERSONALIZED ANALYTICS OF READING PROGRESS 191
А. Харлан, М. Петрик
РОЗРОБКА МОДУЛЬНИХ СИСТЕМ ЗВІТНОСТІ У ФІНАНСОВИХ ДОДАТКАХ З КОМПОНЕНТНОЮ АРХІТЕКТУРОЮ
A. Kharlan, M. Petryk
DEVELOPMENT OF MODULAR REPORTING SYSTEMS IN FINANCIAL APPLICATIONS WITH COMPONENT ARCHITECTURE 192
В. З. Шеремета, Р. О. Жаровський
ТЕСТУВАННЯ ВЕБ-ДОДАТКІВ РОЗРОБЛЕНИМИ НА ОСНОВІ SPRING BOOT ЗА ДОПОМОГОЮ TESTNG
V. Z. Sheremeta, R. O. Zharovskyi
TESTING WEB APPLICATIONS DEVELOPED ON SPRING BOOT WITH TESTNG 193