МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
УМАНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
Факультет економіки, підприємництва та інформаційних технологій

Кафедра інформаційних технологій

Допущено до захисту
"_____" _____ 2026 р.
Зав. кафедри, к.т.н., доцент
_____ **Ліщук Р.І.**

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

РОЗРОБКА ПІДСИСТЕМИ ДЛЯ ПОШУКУ, ЗБОРУ ТА АНАЛІЗУ КОРИСТУВАЦЬКИХ ВІДГУКІВ

Виконав – студент 4 курсу, групи 41 кн
освітньої програми "Комп'ютерні науки"
спеціальності 122 Комп'ютерні науки
Покотило Святослав Михайлович

Науковий керівник – к.е.н., доцент
Концеба Сергій Михайлович

Умань – 2026

Реферат
на кваліфікаційну роботу "Розробка підсистеми для пошуку, збору та аналізу користувацьких відгуків"
студента факультету економіки, підприємництва та інформаційних технологій
Уманського національного університету
Покотила Святослава Михайловича

Структура та обсяг роботи. Пояснювальна записка кваліфікаційної роботи складається з трьох розділів, містить 62 сторінки, 3 таблиці, 22 рисунки, 1 додаток, 47 джерел наукової літератури.

Метою роботи є розробка підсистеми для пошуку, збору та аналізу користувацьких відгуків, яка забезпечує ефективну індексацію текстових даних, швидкий повнотекстовий пошук та базову аналітичну обробку коментарів.

У першому розділі здійснено огляд систем індексації та аналітики інформації та існуючих систем обробки та аналітики коментарів. Здійснено постановку задач з розробки підсистеми для пошуку, збору та аналізу користувацьких відгуків.

У другому розділі проведено обгрунтування технологій та вибір методів розроблення підсистеми.

У третьому розділі реалізовано розробку підсистеми для пошуку, збору та аналізу користувацьких відгуків, яка забезпечує ефективну індексацію текстових даних, швидкий повнотекстовий пошук та базову аналітичну обробку коментарів.

Анотація

Досліджено підходи пошуку та аналізу коментарів на мові програмування Java, використовуючи Spring, Hibernate, PostgreSQL та Maven. Зручність системи полягає у можливості детального аналізу обраних коментарів і можлиовсті отримання рекомендацій що до роботи з цим коментарем. Розроблений модуль є гнучким у конфігурації, кроссплатформеним та відкритий для розширень.

Ключові слова: пошук, аналіз, коментарі, графічний інтерфейс, програмне забезпечення, бази даних.

Annotation

Approaches for searching and analyzing comments in the Java programming language using Spring, Hibernate, PostgreSQL and Maven have been investigated. The convenience of the system lies in the possibility of detailed analysis of selected comments and the possibility of receiving recommendations for working with this comment. The developed module is flexible in configuration, cross-platform and open to extensions.

Keywords: search, analysis, comments, graphical interface, software, databases.

ЗМІСТ

ВСТУП 4
РОЗДІЛ 1 ДОСЛІДЖЕННЯ ІСНУЮЧИХ ТЕХНОЛОГІЙ ІНДЕКСАЦІЇ ТА АНАЛІЗУ ІНФОРМАЦІЇ ТА КОМЕНТАРІВ 6
1.1. Огляд існуючих систем індексації та аналітики інформації 6
1.2. Огляд існуючих систем обробки та аналітики коментарів 19
1.3. Постановка задачі 23
РОЗДІЛ 2 ОБҐРУНТУВАННЯ ВИБОРУ ТЕХНОЛОГІЇ ТА МЕТОДІВ РОЗРОБКИ 25
2.1. Вибір мови програмування 25
2.2. Обґрунтування вибору інструментів розробки 30
2.3. Обґрунтування вибору архітектурних методів та підходів розробки 33
2.4. Обґрунтування вибору систем управління базами даних 36
2.5. Архітектурні шаблони 39
РОЗДІЛ 3 РОЗРОБКА ТА ТЕСТУВАННЯ ПІДСИСТЕМИ 44
3.1. Інтеграція інструменту автоматичної збірки проекту 44
3.2. Розробка програмного забезпечення 47
3.3. Тестування програмного забезпечення 51
ВИСНОВКИ 57
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 59
ДОДАТКИ 63

ВСТУП

У сучасних умовах стрімкого розвитку інформаційних технологій та цифрових сервісів обсяг користувацького контенту зростає надзвичайно швидкими темпами. Особливо це стосується текстових даних у вигляді відгуків, коментарів та оцінок, які формуються користувачами на різних вебплатформах, у соціальних мережах та сервісах електронної комерції. Такі дані містять цінну інформацію про досвід користувачів, якість продуктів і послуг, а також загальні тенденції сприйняття інформації.

Актуальність даної теми зумовлена необхідністю ефективної обробки великих обсягів неструктурованих текстових даних, що вимагає використання сучасних методів індексації, пошуку та аналізу. Традиційні підходи до роботи з текстовою інформацією є недостатньо ефективними при масштабуванні систем, тому виникає потреба у створенні спеціалізованих підсистем, здатних забезпечити швидкий пошук, структуровану обробку та базовий семантичний аналіз користувацьких відгуків. Використання сучасних технологій, таких як повнотекстовий пошук, обробка природної мови та реляційні бази даних, дозволяє значно підвищити ефективність роботи з такими даними.

Метою роботи є розробка підсистеми для пошуку, збору та аналізу користувацьких відгуків, яка забезпечує ефективну індексацію текстових даних, швидкий повнотекстовий пошук та базову аналітичну обробку коментарів.

Для досягнення поставленої мети необхідно виконати такі завдання**:**

- проаналізувати сучасні підходи та існуючі системи обробки текстових даних і користувацьких відгуків;
- дослідити методи індексації та повнотекстового пошуку інформації;
- обґрунтувати вибір технологій для реалізації системи;
- спроєктувати архітектуру підсистеми з урахуванням вимог масштабованості та модульності;
- реалізувати програмну систему на основі обраного технологічного стеку;
- провести тестування розробленої системи та оцінити її працездатність;

- визначити можливі напрями подальшого розвитку та вдосконалення системи.

Об’єктом дослідження є процеси збору, зберігання, індексації та обробки користувацьких текстових відгуків у інформаційних системах.

Предметом дослідження є методи та програмні засоби індексації, пошуку та аналізу текстових даних, а також їх реалізація у вигляді програмної підсистеми на основі сучасних технологій.

Таким чином, дана робота спрямована на розробку сучасного програмного рішення, яке поєднує механізми пошуку та аналізу текстових даних і може бути використане як основа для створення масштабованих інформаційно-аналітичних систем.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ІСНУЮЧИХ ТЕХНОЛОГІЙ ІНДЕКСАЦІЇ ТА АНАЛІЗУ ІНФОРМАЦІЇ ТА КОМЕНТАРІВ

1.1 Огляд існуючих систем індексації та аналітики інформації

Індексація є одним із ключових етапів роботи з інформаційними ресурсами, зокрема з користувацькими коментарями та відгуками. Її застосування забезпечує низку важливих переваг.

По-перше, індексація суттєво прискорює процес пошуку інформації. Завдяки створенню спеціалізованих структур даних система отримує можливість швидко знаходити необхідні відомості без необхідності перегляду всього масиву даних. У контексті розроблюваної системи це дозволить ефективно виконувати пошук серед великої кількості коментарів, що значно спростить роботу користувачів із ними.

По-друге, індексація сприяє впорядкуванню та структуризації даних. Вона допомагає організувати значні обсяги інформації та встановити взаємозв’язки між окремими її елементами. Це забезпечує більш системний підхід до роботи з даними та дозволяє підтримувати належний рівень організованості серед великої кількості коментарів.

Ще однією важливою перевагою є підвищення ефективності використання обчислювальних і мережевих ресурсів. Запити до попередньо проіндексованих даних обробляються значно швидше, що скорочує час пошуку та обробки інформації. Як наслідок, покращується продуктивність системи та загальний користувацький досвід.

Крім того, індексація створює необхідні умови для проведення якісного аналізу даних і підтримки процесу прийняття рішень. Завдяки швидкому доступу до структурованої інформації підвищується ефективність аналітичної обробки коментарів та відгуків користувачів.

Додатковою перевагою є підтримка поширених форматів документів, зокрема PDF та DOCX, що розширює можливості роботи системи з різними джерелами даних [3, 4].

Таким чином, індексація є фундаментальним компонентом сучасних систем обробки інформації. Вона забезпечує швидкий доступ до даних, їх ефективну організацію та створює основу для подальшого розвитку функціональних можливостей системи, зокрема для виконання інтелектуального аналізу даних.

У наступних розділах буде проведено огляд найбільш поширених платформ для індексації з метою визначення їхніх характеристик, переваг і функціональних можливостей у контексті створення та оптимізації систем пошуку, обробки та аналізу інформації.

OpenKM є сучасною системою електронного документообігу та управління документами (Document Management System, DMS), яка призначена для централізованого зберігання, організації, пошуку та обробки електронних документів. Система розроблена для підприємств і організацій, що працюють із великими обсягами інформації та потребують ефективних інструментів керування документацією. OpenKM підтримує створення єдиного сховища документів, що дозволяє забезпечити швидкий доступ до інформації, контроль її актуальності та безпечне зберігання.

Однією з ключових особливостей OpenKM є використання механізмів індексації документів. Після завантаження файлів до системи їхній вміст автоматично аналізується та додається до пошукового індексу. Завдяки цьому користувачі можуть здійснювати повнотекстовий пошук не лише за назвами документів, але й за інформацією, що міститься всередині них. Такий підхід значно скорочує час пошуку необхідних даних і підвищує ефективність роботи з інформаційними ресурсами.

Система підтримує широкий спектр форматів документів, серед яких PDF, DOCX, XLSX, PPTX, TXT, HTML, XML та інші. Крім того, OpenKM може інтегруватися з технологіями оптичного розпізнавання символів (OCR), що дозволяє індексувати текст із відсканованих документів та зображень. Це

особливо важливо для організацій, які працюють з архівними матеріалами або паперовими документами, переведеними в електронний формат.

Важливою функцією OpenKM є підтримка метаданих. Для кожного документа можуть бути визначені додаткові атрибути, такі як автор, дата створення, категорія, тип документа або інші параметри. Використання метаданих суттєво підвищує точність пошуку та дозволяє ефективно класифікувати інформацію. Завдяки цьому користувачі можуть швидко знаходити потрібні документи навіть у великих інформаційних сховищах.

Ще однією перевагою системи є механізм керування версіями документів. OpenKM автоматично зберігає історію змін файлів, що дозволяє відстежувати процес їх редагування та за необхідності повертатися до попередніх версій. Такий підхід мінімізує ризик втрати важливої інформації та забезпечує контроль над усіма змінами, які відбуваються в документах.

Відповідно до рисунку 1.1, OpenKM має веб-версію програмного забезпечення, що у певній мірі вирішує проблему кросплатформеності.

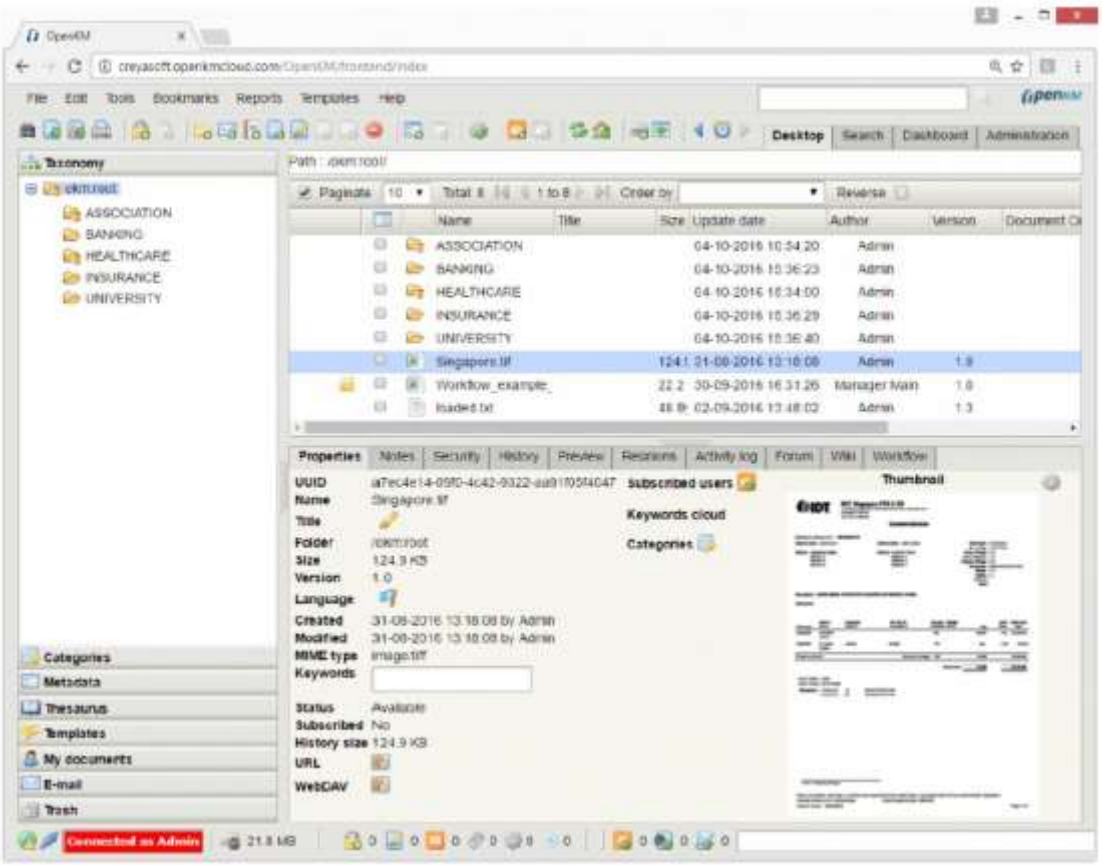


Рис. 1.1. Графічний інтерфейс OpenKM

Проте для роботи цієї системи необхідний доступ до Інтернету. Подані нижче рисунки (1.2 та 1.3) вказують на значну кількість функцій та інструментів у OpenKM, проте інтерфейс може виглядати перенавантаженим та містити додаткові вікна.

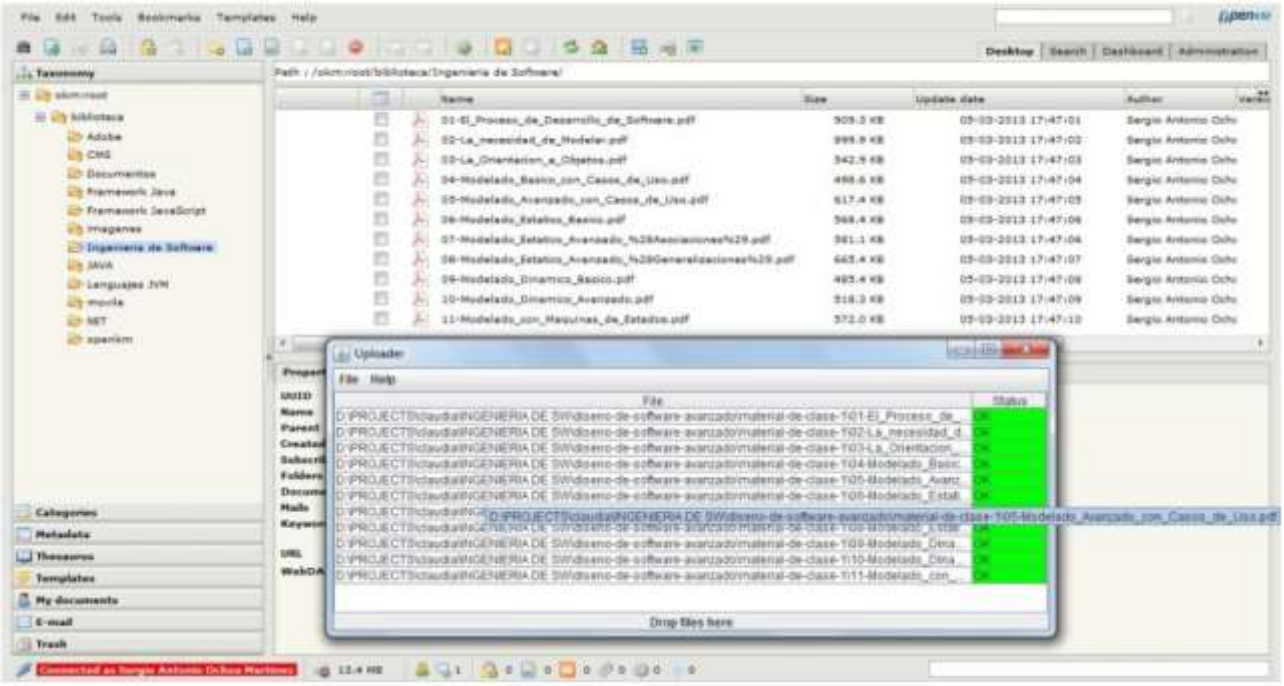


Рис. 1.2. Процес індексування в OpenKM

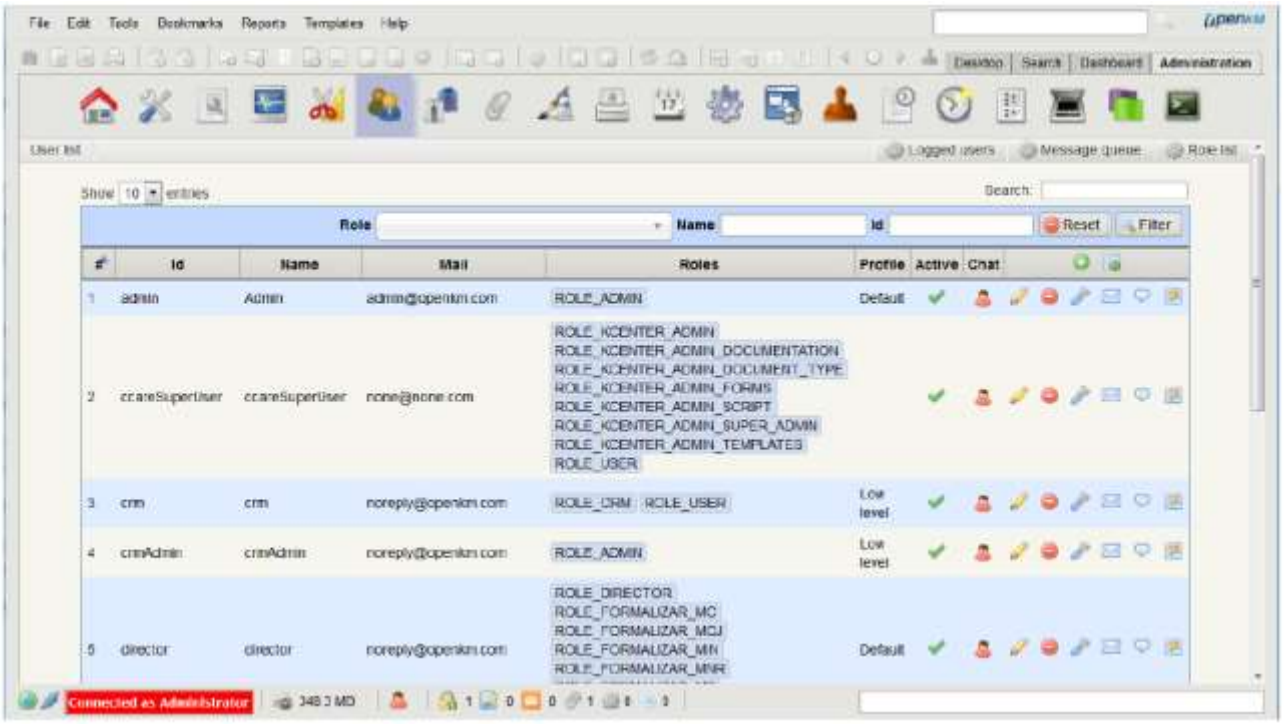


Рис. 1.3. Панель адміністратора в OpenKM

Система також пропонує розвинені засоби керування доступом. Адміністратор може налаштовувати права користувачів і груп користувачів на перегляд, редагування, видалення або завантаження документів. Це дозволяє забезпечити належний рівень інформаційної безпеки та захистити конфіденційні дані від несанкціонованого доступу. Додатково OpenKM веде журнал подій, у якому фіксуються всі дії користувачів, що спрощує аудит та контроль використання системи.

До переваг OpenKM належать:

– підтримка повнотекстового пошуку та індексації документів;

– можливість роботи з великою кількістю форматів файлів;

– використання метаданих для класифікації інформації;

– контроль версій документів та збереження історії змін;

– гнучке налаштування прав доступу;

– підтримка OCR для роботи зі сканованими документами;

– інтеграція з іншими інформаційними системами через API;

– підтримка автоматизації бізнес-процесів за допомогою механізмів Workflow;

– наявність безкоштовної Community Edition для навчальних і дослідницьких проєктів.

Незважаючи на значну кількість переваг, OpenKM має і певні недоліки. Одним із них є відносно складний процес встановлення та налаштування. Для повноцінного розгортання системи необхідні знання щодо роботи з серверними технологіями, базами даних та платформою Java. Це може створювати труднощі для невеликих організацій або окремих користувачів.

Іншим недоліком є високі вимоги до апаратних ресурсів при роботі з великими обсягами документів. Процеси індексації, пошуку та аналізу даних можуть потребувати значного обсягу оперативної пам'яті та продуктивних серверних компонентів. Крім того, деякі розширені можливості системи, такі як професійна технічна підтримка, розширені інструменти інтеграції та корпоративні модулі, доступні лише у платних версіях продукту.

Також слід зазначити, що інтерфейс OpenKM орієнтований переважно на корпоративних користувачів і може здатися складним для нових користувачів. Для ефективного використання всіх функціональних можливостей системи часто потрібне додаткове навчання персоналу.

Таким чином, OpenKM є потужною платформою для управління документами, яка поєднує засоби індексації, пошуку, класифікації та контролю доступу до інформації. Завдяки широким можливостям роботи з документами та підтримці автоматизації бізнес-процесів система може успішно використовуватися як основа для створення підсистем пошуку, збору та аналізу інформації. Водночас складність налаштування та значні вимоги до ресурсів можуть обмежувати доцільність її використання у невеликих проєктах.

SimpleSearch є програмним засобом для пошуку та перегляду документів, який входить до складу системи SimpleIndex або може використовуватися як окремий модуль для роботи з проіндексованими даними. Основне призначення SimpleSearch полягає у забезпеченні швидкого доступу до електронних документів шляхом виконання пошуку за індексними полями, метаданими та вмістом документів. Система орієнтована на організації, які працюють із великими обсягами електронних документів і потребують простого інструменту для їх пошуку та перегляду.

Принцип роботи SimpleSearch базується на попередній індексації документів. Під час індексації для кожного документа створюється набір атрибутів та індексних записів, які надалі використовуються для виконання пошукових запитів. Завдяки цьому система може швидко знаходити необхідні документи без потреби аналізувати весь інформаційний масив під час кожного запиту. Користувачеві достатньо ввести пошукові критерії, після чого система повертає список документів, які відповідають заданим умовам.

Однією з особливостей SimpleSearch є простота використання. Інтерфейс системи орієнтований на виконання пошукових операцій і не містить складних інструментів адміністрування або налаштування (рис. 1.4). Це дозволяє користувачам швидко освоїти роботу із системою та ефективно використовувати

її для пошуку необхідної інформації. Пошук може здійснюватися за різними параметрами, включаючи назву документа, дату створення, автора, категорію та інші індексні поля. Також підтримується частковий збіг та повнотекстовий пошук, що підвищує точність отриманих результатів.

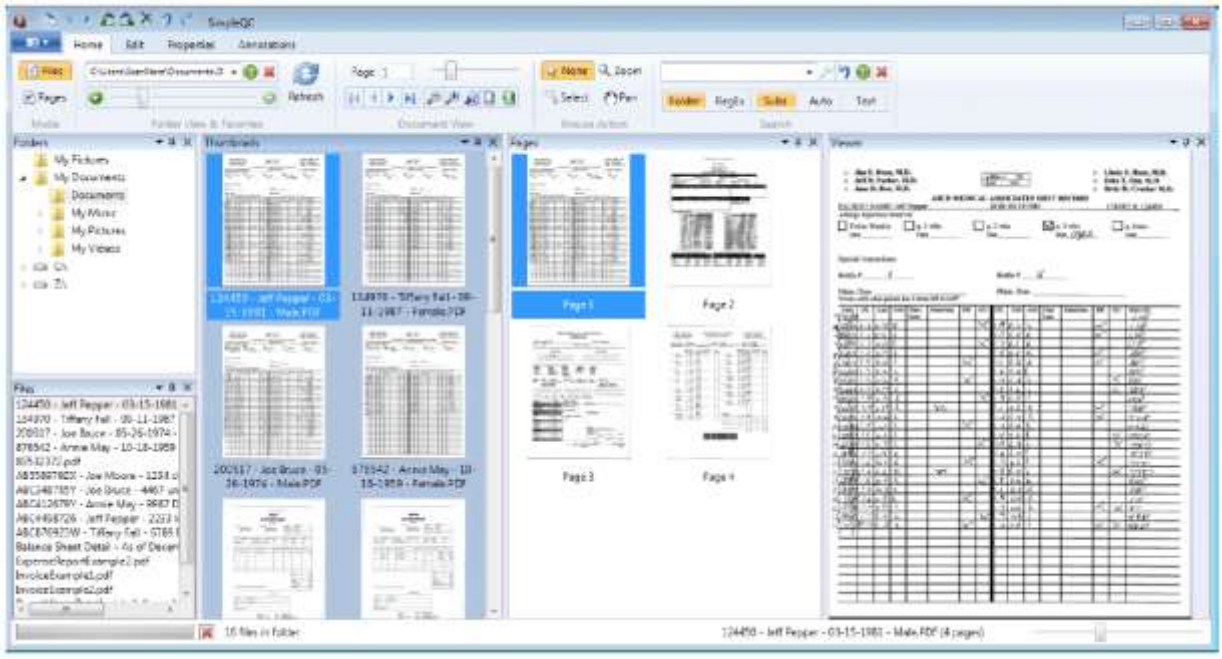


Рис. 1.4 Графічний інтерфейс SimpleSearch

В одному з функціональних елементів SimpleSearch, відображеному на рисунку 1.5, передбачено можливість зручного перегляду різних сторінок одного документа.

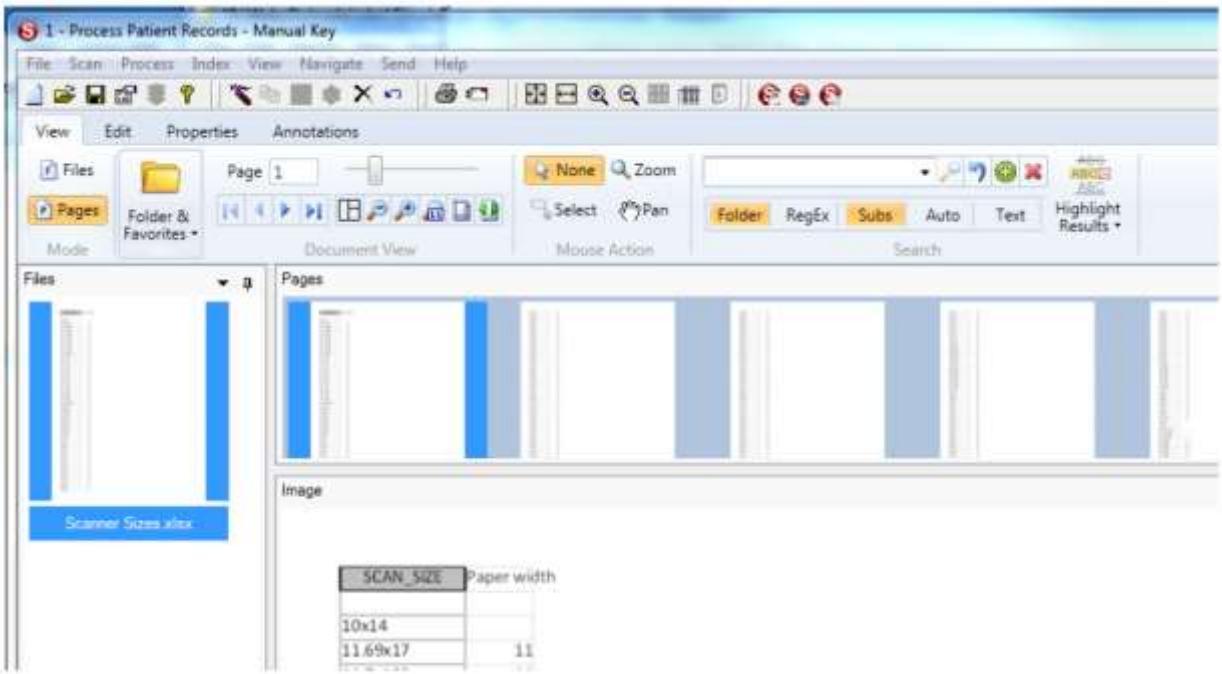


Рис. 1.5. Перегляд сторінок документу в SimpleSearch

Важливою перевагою SimpleSearch є можливість інтеграції з різними базами даних. Система може працювати як із власною базою даних, так і з зовнішніми сховищами інформації, що дозволяє використовувати її в уже існуючих інформаційних системах підприємства. Такий підхід забезпечує гнучкість впровадження та знижує витрати на модернізацію наявної IT-інфраструктури (рис. 1.6).

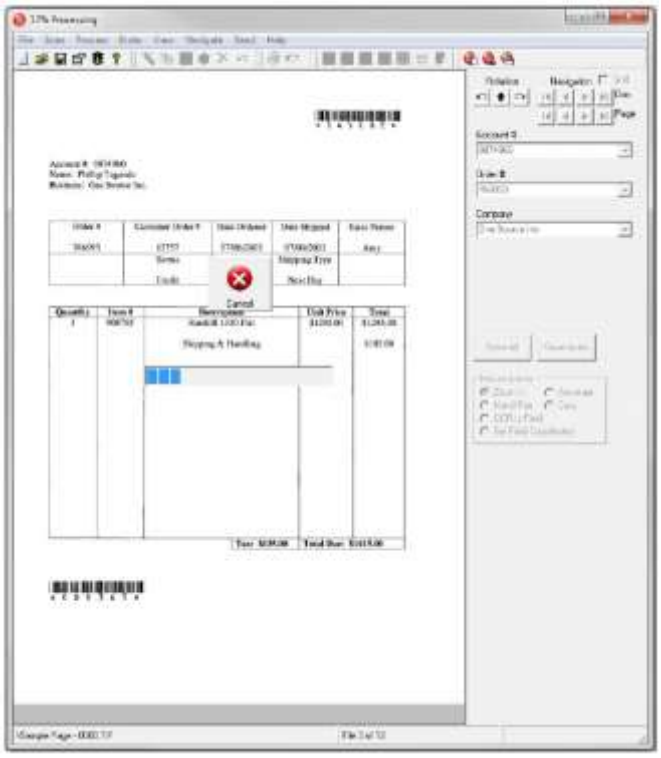


Рис. 1.6. Процес індексації в SimpleSearch

До основних переваг SimpleSearch належать:

– простота встановлення та використання;

– висока швидкість пошуку завдяки попередній індексації даних;

– підтримка повнотекстового пошуку;

– можливість пошуку за метаданими та індексними полями;

– інтеграція з різними базами даних і джерелами інформації;

– невисокі вимоги до апаратних ресурсів;

– можливість роботи з великими архівами документів;

– швидке навчання користувачів завдяки простому інтерфейсу.

Незважаючи на зазначені переваги, система має певні недоліки. Насамперед функціональні можливості SimpleSearch значно поступаються сучасним платформам інтелектуального пошуку, таким як Elasticsearch або OpenSearch. Система орієнтована переважно на класичний пошук за ключовими словами та індексними полями, тому не підтримує складні алгоритми семантичного аналізу тексту, машинне навчання та сучасні механізми обробки природної мови.

Ще одним недоліком є обмежені можливості аналітичної обробки даних. SimpleSearch забезпечує швидкий пошук документів, однак практично не надає інструментів для аналізу змісту документів, визначення тональності текстів, кластеризації інформації або побудови статистичних звітів. У зв'язку з цим для реалізації складних аналітичних задач необхідне використання додаткових програмних засобів.

Крім того, ефективність роботи системи значною мірою залежить від якості індексації та правильності заповнення метаданих. Помилки під час індексації або неповні атрибути документів можуть призвести до зниження точності пошуку та ускладнити доступ до потрібної інформації.

Отже, SimpleSearch є ефективним рішенням для організації швидкого пошуку та перегляду документів у великих електронних архівах. Основними перевагами системи є простота використання, швидкодія та підтримка індексованого пошуку. Водночас обмежені аналітичні можливості та відсутність сучасних механізмів інтелектуальної обробки даних роблять її менш придатною для складних систем аналізу користувацьких відгуків і великих неструктурованих масивів текстової інформації.

LogicalDOC є системою електронного документообігу та управління документами (Document Management System, DMS), призначеною для централізованого зберігання, організації, пошуку та контролю електронних документів. Платформа орієнтована на підприємства та організації, які працюють із великими обсягами документів і потребують ефективних інструментів для їх обробки, класифікації та швидкого пошуку. Система

підтримує як локальне розгортання, так і використання хмарних технологій, що забезпечує гнучкість впровадження відповідно до потреб користувачів.

Основою функціонування LogicalDOC є механізм індексації документів. Після завантаження файлу система автоматично аналізує його вміст і створює пошуковий індекс, який використовується для швидкого доступу до інформації. Завдяки цьому користувачі можуть виконувати повнотекстовий пошук не лише за назвами файлів, а й за словами та фразами, що містяться всередині документів. Такий підхід значно скорочує час пошуку необхідної інформації та підвищує ефективність роботи з великими масивами даних.

Система підтримує широкий спектр форматів документів, включаючи PDF, DOCX, XLSX, PPTX, TXT, HTML, XML та інші поширені типи файлів. Крім того, LogicalDOC може інтегруватися з технологіями оптичного розпізнавання символів (OCR), що дозволяє виконувати індексацію тексту зі сканованих документів, зображень та архівних матеріалів. Це розширює можливості використання системи в організаціях, які працюють як з електронними, так і з оцифрованими паперовими документами.

Аналізуючи рисунки 1.7, 1.8 та 1.9, можна зауважити, що у LogicalDOC присутній сучасний та естетичний інтерфейс, що відображається в його дизайні.

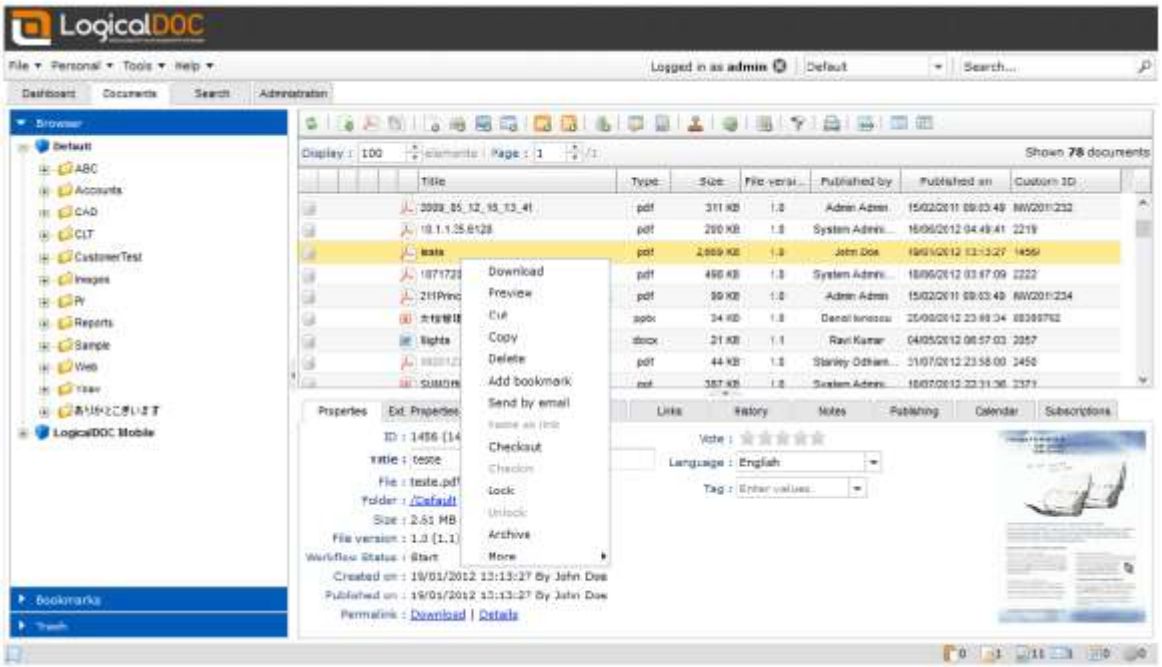


Рис. 1.7. Графічний інтерфейс LogicalDOC

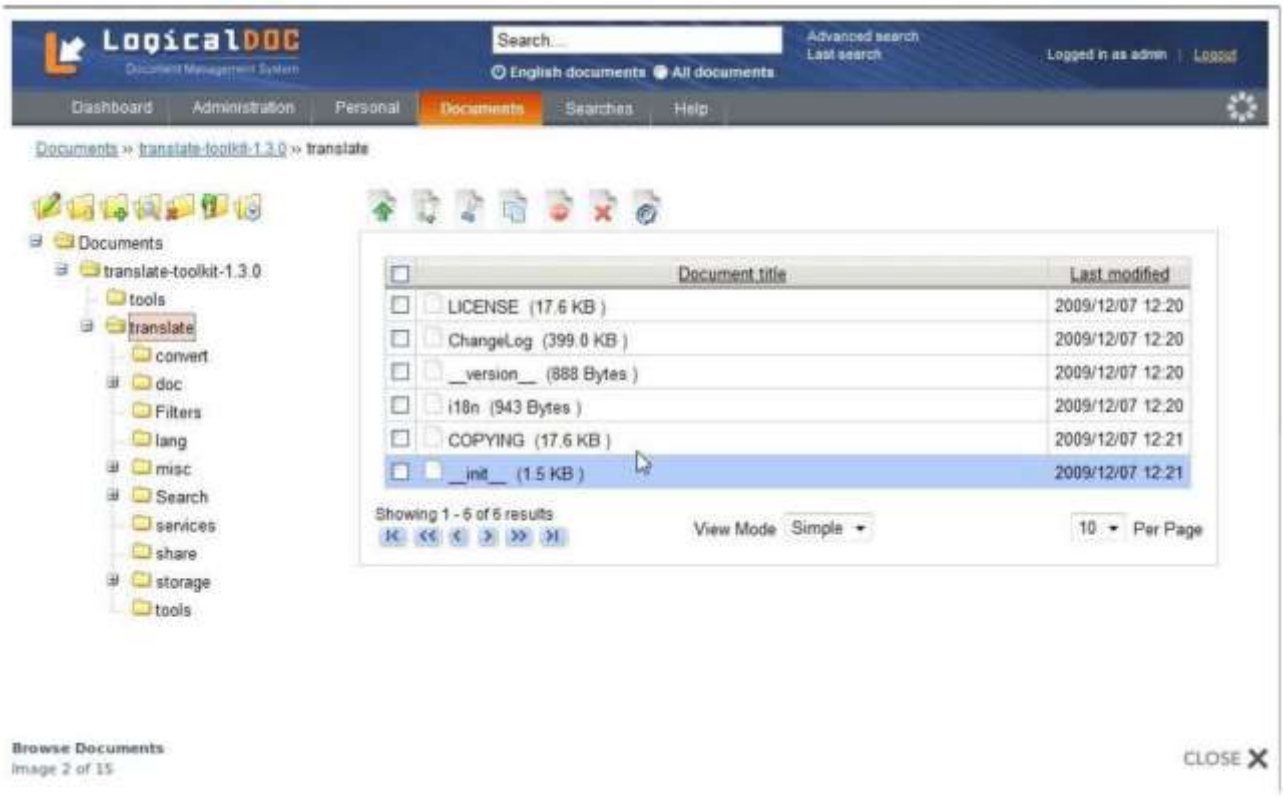


Рис. 1.8. Меню вибору декількох документів в LogicalDOC

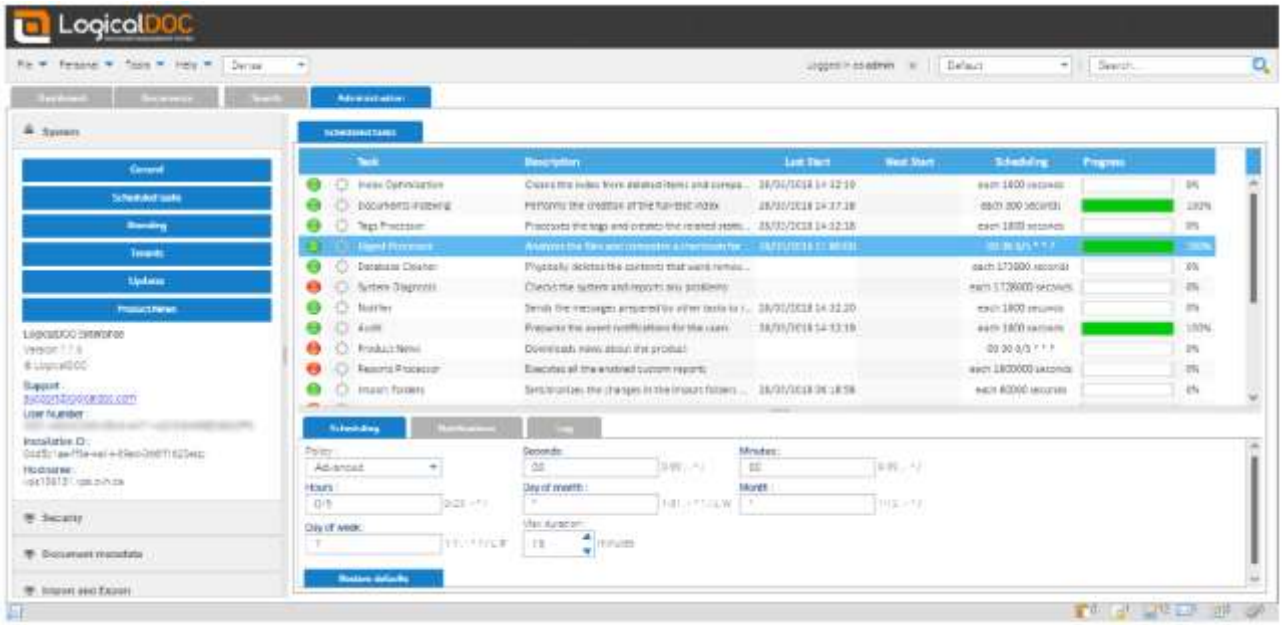


Рис. 1.9. Процес індексації LogicalDOC

Однією з важливих особливостей LogicalDOC є підтримка метаданих. Для кожного документа можуть бути визначені додаткові атрибути, такі як автор, дата створення, категорія, тип документа, ключові слова та інші характеристики.

Метадані дозволяють більш ефективно класифікувати інформацію та виконувати пошук за різними параметрами. У результаті користувачі отримують можливість швидко знаходити потрібні документи навіть у сховищах, що містять сотні тисяч файлів.

Система також реалізує механізм керування версіями документів. Кожна зміна файлу автоматично фіксується, а попередні версії зберігаються в архіві. Це дозволяє контролювати історію редагування документів, аналізувати внесені зміни та відновлювати попередні версії за потреби. Такий функціонал є особливо важливим для організацій, де над документами працюють кілька співробітників одночасно.

Значна увага в LogicalDOC приділяється забезпеченню інформаційної безпеки. Система підтримує багаторівневе керування правами доступу, що дозволяє визначати окремі дозволи для користувачів і груп користувачів. Адміністратор може обмежувати перегляд, редагування, видалення або завантаження документів залежно від ролі користувача. Крім того, система веде журнал подій, який містить інформацію про всі операції з документами, що забезпечує можливість аудиту та контролю дій користувачів.

Важливою перевагою LogicalDOC є підтримка автоматизації бізнес-процесів. Система дозволяє налаштовувати маршрути погодження документів, автоматичні сповіщення та сценарії обробки інформації. Це сприяє підвищенню продуктивності праці та скороченню часу виконання рутинних операцій.

До основних переваг LogicalDOC належать:

– потужні засоби повнотекстового пошуку та індексації документів;
– підтримка великої кількості форматів файлів;
– можливість використання OCR для обробки сканованих документів;
– гнучка система метаданих для класифікації інформації;
– підтримка контролю версій документів;
– розвинена система керування правами доступу;
– ведення журналів аудиту та моніторингу дій користувачів;
– можливість автоматизації бізнес-процесів;

– підтримка інтеграції з корпоративними інформаційними системами;
– наявність вебінтерфейсу для віддаленої роботи з документами.

Незважаючи на значну функціональність, система має і певні недоліки. Одним із них є складність налаштування та адміністрування. Для повноцінного впровадження LogicalDOC необхідні знання щодо конфігурації серверного програмного забезпечення, баз даних та систем управління документами. Це може збільшувати витрати на впровадження та супровід системи.

Ще одним недоліком є підвищені вимоги до апаратних ресурсів при роботі з великими сховищами документів. Під час індексації та виконання складних пошукових запитів система може споживати значний обсяг оперативної пам'яті та процесорного часу. Тому для забезпечення стабільної роботи необхідно використовувати продуктивне серверне обладнання.

Крім того, безкоштовна версія LogicalDOC має певні функціональні обмеження. Частина розширених можливостей, пов'язаних з автоматизацією бізнес-процесів, інтеграцією з корпоративними сервісами та технічною підтримкою, доступна лише в комерційних редакціях системи. Для невеликих організацій це може стати додатковим фактором при виборі програмного забезпечення.

Також слід зазначити, що LogicalDOC створювалася насамперед як система управління документами, а не як платформа для аналізу текстових даних. Тому її можливості щодо інтелектуального аналізу інформації, визначення тональності текстів, тематичного моделювання або обробки природної мови є обмеженими та потребують інтеграції з додатковими аналітичними інструментами.

Таким чином, LogicalDOC є потужною системою електронного документообігу, яка забезпечує ефективне зберігання, індексацію та пошук документів. Завдяки підтримці повнотекстового пошуку, контролю версій, розмежуванню прав доступу та автоматизації бізнес-процесів система може успішно використовуватися в корпоративному середовищі. Водночас складність налаштування, вимоги до ресурсів та обмежені можливості інтелектуального

аналізу текстових даних можуть бути суттєвими факторами при виборі цієї платформи для спеціалізованих інформаційно-аналітичних систем.

1.2. Огляд існуючих систем обробки та аналітики коментарів

Коментарі відіграють важливу роль у процесі прийняття рішень споживачами, виступаючи ефективним інструментом підвищення якості товарів і рівня обслуговування клієнтів. Їх аналіз є невід'ємною складовою вивчення зворотного зв'язку користувачів щодо продуктів і послуг, а також оптимізації бізнес-процесів. Завдяки цьому інструменту можна оцінювати суспільні настрої та виявляти актуальні тенденції, які формуються на маркетплейсах, у соціальних мережах та на інших цифрових платформах.

Серед основних можливостей аналізу коментарів варто виділити такі. По-перше, це виявлення трендів, що дозволяє визначати популярні теми серед користувачів, а також фіксувати домінування позитивних або негативних оцінок і своєчасно реагувати на них. По-друге, аналіз рівня задоволеності клієнтів, який дає змогу оцінити ефективність продуктів, сервісу, рекламних кампаній чи окремих подій. По-третє, виявлення проблемних аспектів шляхом пошуку ключових слів або фраз, що допомагає знаходити слабкі місця у продуктах чи обслуговуванні.

Окрім цього, аналіз коментарів сприяє вдосконаленню продукції, сервісів, контенту та маркетингових стратегій завдяки використанню конструктивного зворотного зв'язку. Важливим аспектом є також підвищення якості взаємодії з користувачами через оперативне реагування на їхні відгуки, надання додаткової інформації, вирішення проблем або висловлення вдячності за позитивні оцінки.

Позитивні відгуки можуть ефективно використовуватися у маркетингових і рекламних кампаніях як інструмент підвищення довіри до бренду. Крім того, аналіз коментарів у порівнянні з відгуками конкурентів дозволяє визначити сильні та слабкі сторони власних продуктів чи послуг. Не менш важливим є

виявлення нових можливостей для розвитку, оскільки коментарі часто містять ідеї та запити користувачів щодо вдосконалення або розширення функціоналу.

Збір і обробка таких даних зазвичай потребують застосування спеціалізованих інструментів для моніторингу соціальних мереж і аналізу текстової інформації. У підсумку, коментарі та відгуки виступають важливим джерелом інформації про сприйняття продуктів і послуг користувачами, а їх аналіз дозволяє підвищити якість сервісу та конкурентоспроможність компаній.

На прикладі сучасних маркетплейсів доцільно розглянути підходи до обробки та візуалізації аналітики коментарів. Хоча такі платформи не орієнтовані безпосередньо на глибокий аналіз відгуків, вони забезпечують базовий функціонал для роботи з ними. Зокрема, користувачам доступні інструменти перегляду, сортування та часткової фільтрації коментарів щодо конкретних товарів або продавців. Це дозволяє використовувати маркетплейси як приклад реалізації базових методів збору та представлення відгуків, а також як джерело даних для визначення загальних тенденцій і популярності продукції серед споживачів.

Одним із провідних представників українського ринку електронної комерції є інтернет-магазин Rozetka, заснований у 2005 році (рис.1.10). Платформа пропонує широкий асортимент товарів у різних категоріях — від електроніки до товарів повсякденного вжитку. Станом на сьогодні Rozetka активно впроваджує елементи персоналізації, рекомендаційні механізми та вдосконалені системи рейтингу, що базуються на поведінці користувачів і їхніх відгуках. Це дозволяє не лише підвищити зручність користування сервісом, але й покращити якість аналітики споживчих уподобань.

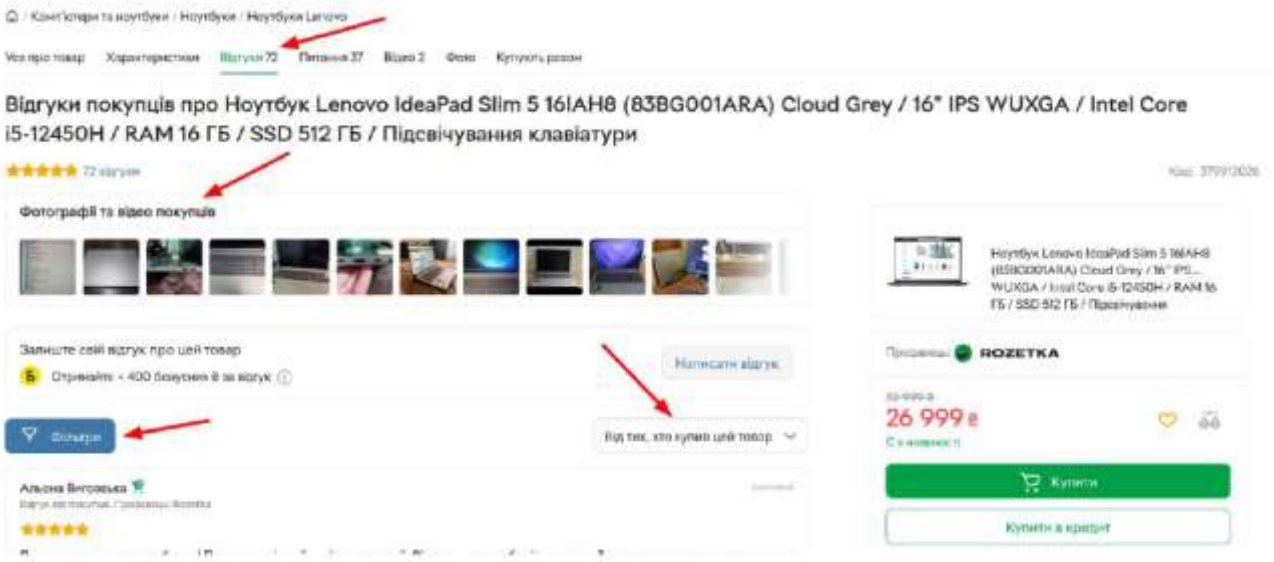


Рис. 1.10 Робота з коментарями на маркетплейсі Rozetka

Платформа Hotline виконує функцію агрегатора, що надає можливість порівняння цін і характеристик товарів у різних інтернет-магазинах (рис.1.11). Вона також містить систему відгуків і оцінок, яка допомагає користувачам приймати обґрунтовані рішення перед покупкою. У сучасних умовах такі сервіси дедалі частіше інтегрують елементи аналітики даних і візуалізації інформації, що дозволяє швидко оцінювати ринкові пропозиції та знаходити оптимальні варіанти.



Рис. 1.11 Робота з коментарями на маркетплейсі Hotline

Міжнародна платформа Amazon є одним із найбільш технологічно розвинених прикладів електронної комерції (рис.1.12). Окрім широкого асортименту товарів, вона активно використовує алгоритми машинного навчання для персоналізації контенту, рекомендацій товарів та аналізу відгуків користувачів. Сучасні рішення Amazon включають автоматизований аналіз текстів відгуків, виявлення ключових характеристик товарів, а також формування узагальнених оцінок (наприклад, виділення «найбільш корисних» коментарів). Це демонструє перехід від простого відображення відгуків до їх інтелектуальної обробки.

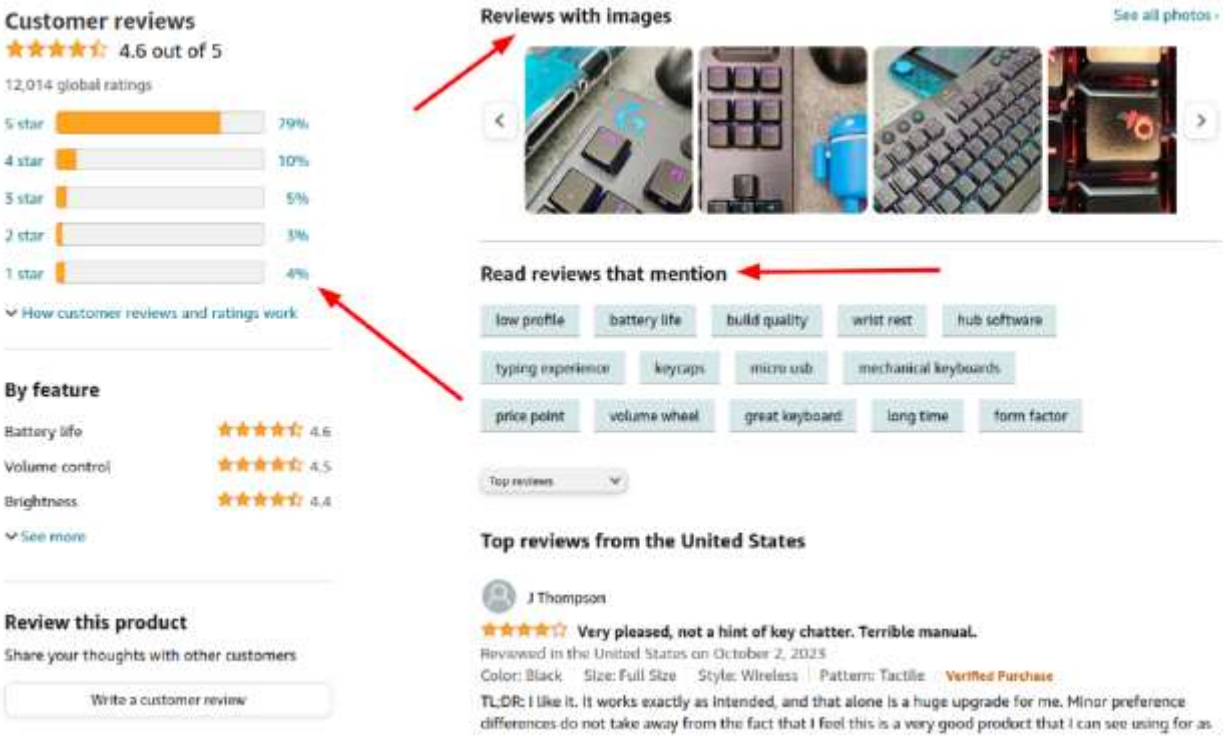


Рис. 1.12 Робота з коментарями на маркетплейсі Amazon

Загалом маркетплейси надають користувачам низку стандартних можливостей для роботи з коментарями. До них належать фільтрація та сортування відгуків за різними параметрами (оцінкою, датою, популярністю), пошук за ключовими словами, а також групування за категоріями товарів або окремими продуктами. Крім того, у більшості систем реалізовано механізми взаємодії у вигляді відповідей на коментарі, що формує додатковий рівень аналітики та дозволяє відстежувати діалог між користувачами.

Сучасною тенденцією є впровадження систем оцінювання корисності відгуків, де кожен коментар може отримувати рейтинг від інших користувачів. Це дозволяє виділяти найбільш релевантні та інформативні відгуки, що значно підвищує ефективність їх використання. Також активно застосовуються технології обробки природної мови (NLP), які дозволяють автоматично класифікувати коментарі, визначати їхню тональність та виявляти ключові теми.

Водночас функціональні можливості таких платформ залишаються обмеженими у контексті комплексного аналізу даних. Більшість інструментів працює лише в межах конкретного ресурсу та не забезпечує інтеграцію відгуків з різних джерел. Використання стандартних засобів браузера, таких як пошук по тексту, частково розширює можливості аналізу, однак не є зручним і масштабованим рішенням.

Таким чином, існуючі маркетплейси забезпечують базовий рівень роботи з коментарями та їх візуалізацією, проте для глибокого аналізу, інтеграції даних з різних платформ і формування інтелектуальних рекомендацій необхідне використання спеціалізованого програмного забезпечення, що базується на сучасних методах аналізу даних, машинного навчання та штучного інтелекту.

1.3. Постановка задачі

Сучасні вебресурси та онлайн-платформи генерують значні обсяги користувацьких відгуків, які містять цінну інформацію про якість продуктів, послуг та рівень задоволеності споживачів. Однак через велику кількість та неструктурований характер таких даних їх ручний збір і аналіз є трудомістким та малоефективним процесом. У зв'язку з цим виникає необхідність створення спеціалізованої підсистеми, яка забезпечуватиме автоматизований пошук, збір, зберігання та аналіз користувацьких відгуків з різних джерел інформації.

Метою роботи є розробка підсистеми для пошуку, збору та аналізу користувацьких відгуків, яка дозволить автоматизувати процес отримання інформації з вебресурсів, забезпечити її структуроване зберігання та надати інструменти для подальшого аналізу отриманих даних.

Для досягнення поставленої мети необхідно вирішити такі завдання:

1. Провести аналіз предметної області та дослідити особливості роботи з користувацькими відгуками як джерелом інформації.

2. Виконати огляд існуючих програмних рішень та платформ для збору, індексації та аналізу текстових даних, визначити їх переваги та недоліки.

3. Сформулювати функціональні та нефункціональні вимоги до розроблюваної підсистеми.

4. Розробити архітектуру програмної підсистеми та визначити взаємодію її основних компонентів.

5. Реалізувати модуль пошуку та автоматизованого збору користувацьких відгуків із зовнішніх джерел даних.

6. Розробити механізм збереження, структуризації та індексації отриманих відгуків для забезпечення швидкого пошуку інформації.

7. Реалізувати функціонал фільтрації та пошуку відгуків за різними параметрами, такими як дата, джерело, ключові слова та інші характеристики.

8. Розробити модуль аналізу текстових даних, який забезпечуватиме визначення основних характеристик відгуків, зокрема тональності, частоти використання ключових слів та тематичної спрямованості повідомлень.

9. Створити користувацький інтерфейс для взаємодії з підсистемою та візуалізації результатів аналізу.

10. Провести тестування розробленої підсистеми та оцінити ефективність її роботи за основними показниками продуктивності та точності пошуку.

11. Виконати аналіз отриманих результатів та сформулювати рекомендації щодо подальшого розвитку і вдосконалення системи.

Результатом виконання роботи повинна стати програмна підсистема, яка забезпечує автоматизований пошук, збір, індексацію та аналіз користувацьких відгуків, дозволяє швидко отримувати необхідну інформацію та формувати аналітичні висновки на основі зібраних даних.

РОЗДІЛ 2 ОБҐРУНТУВАННЯ ВИБОРУ ТЕХНОЛОГІЇ ТА МЕТОДІВ РОЗРОБКИ

2.1 Вибір мови програмування

Вибір мови програмування є одним із ключових етапів проєктування програмного забезпечення, оскільки саме від нього значною мірою залежать швидкість розробки, продуктивність системи, можливості масштабування та подальша підтримка проєкту. Сучасні мови програмування відрізняються не лише синтаксисом і парадигмами програмування, а й рівнем розвитку екосистеми, доступністю інструментів розробки, бібліотек, фреймворків та спільноти розробників.

Насамперед вибір мови програмування визначає набір доступних засобів розробки. Для популярних мов існують потужні інтегровані середовища розробки (IDE), системи автоматизованого тестування, профілювання та налагодження програмного коду. Використання сучасних інструментів дозволяє суттєво скоротити час розробки, спростити пошук помилок та підвищити якість програмного забезпечення.

Не менш важливим фактором є наявність бібліотек та фреймворків. Сучасні програмні проєкти рідко створюються виключно власними силами розробників. Значна частина функціональності реалізується за допомогою готових рішень, які дозволяють уникнути дублювання коду та зосередитися на вирішенні прикладних задач. Особливо це актуально для систем збору та аналізу користувацьких відгуків, де необхідно працювати з веб-запитами, базами даних, інструментами пошуку, засобами обробки природної мови та алгоритмами машинного навчання.

Важливим критерієм також є масштабованість програмного забезпечення. Обрана технологія повинна забезпечувати можливість подальшого розширення функціональності системи без необхідності повного перепроєктування архітектури. Для систем аналізу відгуків це особливо актуально, оскільки обсяг

текстових даних може постійно зростати, а вимоги до продуктивності та швидкодії збільшуватися.

Крім того, сучасні програмні системи дедалі частіше працюють у хмарних середовищах та використовують мікросервісну архітектуру. Тому важливими факторами є підтримка контейнеризації, асинхронного програмування, розподіленої обробки даних та інтеграції з хмарними сервісами. Наявність таких можливостей значно спрощує створення масштабованих та надійних інформаційних систем.

Таким чином, вибір мови програмування є стратегічним рішенням, яке впливає на ефективність розробки, продуктивність системи, можливість її масштабування та подальший супровід. Для реалізації підсистеми пошуку, збору та аналізу користувацьких відгуків доцільно розглянути найбільш поширені сучасні мови програмування, а саме Java, C# та Python.

Java є високорівневою об'єктно-орієнтованою мовою програмування, яка була представлена компанією Sun Microsystems у 1995 році та продовжує активно розвиватися. Головною особливістю Java є платформна незалежність, що реалізується за допомогою Java Virtual Machine (JVM). Це дозволяє запускати програмне забезпечення на різних операційних системах без необхідності зміни програмного коду.

На сьогодні Java залишається однією з провідних мов для розробки корпоративних інформаційних систем, вебзастосунків, фінансових платформ та хмарних сервісів. Велику популярність мова отримала завдяки потужній екосистемі фреймворків, серед яких особливе місце займають Spring Boot, Spring Cloud, Hibernate та Jakarta EE. Ці технології забезпечують створення надійних, масштабованих та високопродуктивних систем.

Серед переваг Java варто відзначити високу продуктивність, надійність, сувору типізацію, розвинуті засоби багатопотокового програмування та велику кількість готових бібліотек. Особливо важливим для систем аналізу даних є наявність інструментів для роботи з великими обсягами інформації, таких як Apache Kafka, Apache Spark та Elasticsearch.

C# є сучасною об'єктно-орієнтованою мовою програмування, створеною компанією Microsoft у 2000 році як складова платформи .NET. Завдяки розвитку платформи .NET Core та подальшому переходу до універсальної платформи .NET, мова стала повністю кросплатформною та може використовуватися не лише в операційній системі Windows, а й у Linux та macOS.

Сьогодні C# широко застосовується для розробки вебсервісів, корпоративних систем, настільних застосунків, хмарних сервісів та ігор. Основними перевагами мови є висока продуктивність, сучасний синтаксис, розвинуті засоби асинхронного програмування та тісна інтеграція з хмарною платформою Microsoft Azure. Для веброзробки активно використовується фреймворк ASP.NET Core, який забезпечує створення високонавантажених вебсистем та API.

Додатковою перевагою C# є наявність сучасного середовища розробки Visual Studio, яке вважається одним із найпотужніших інструментів програмування. Також платформа .NET пропонує широкий набір бібліотек для роботи з базами даних, мережевими сервісами, обробкою документів та аналітикою даних.

Python є високорівневою інтерпретованою мовою програмування, яка була створена Гвідо ван Россумом та вперше представлена у 1991 році. Завдяки простому синтаксису, високій читабельності коду та великій кількості бібліотек Python став однією з найпопулярніших мов програмування у світі.

Особливу популярність Python отримав у сферах аналізу даних, машинного навчання, штучного інтелекту та автоматизації. Для реалізації підсистеми аналізу користувацьких відгуків ця мова є особливо привабливою завдяки наявності великої кількості спеціалізованих бібліотек. Зокрема, бібліотеки NumPy та pandas використовуються для обробки даних, NLTK та spaCy — для обробки природної мови, а TensorFlow, Keras та PyTorch — для побудови моделей машинного навчання.

Крім того, Python має широкий набір інструментів для веброзробки, серед яких найпопулярнішими є Django, Flask та FastAPI. Це дозволяє використовувати

одну мову як для реалізації серверної частини системи, так і для виконання аналітичної обробки текстових даних.

Слід зазначити, що всі три розглянуті мови програмування мають автоматичне управління пам'яттю за допомогою механізму збирання сміття (Garbage Collection). Цей механізм автоматично звільняє пам'ять від невикористовуваних об'єктів, що підвищує надійність програмного забезпечення, зменшує ризик виникнення помилок, пов'язаних із керуванням пам'яттю, та спрощує процес розробки.

Отже, Java, C# та Python є сучасними мовами програмування, які мають розвинені екосистеми, підтримують створення масштабованих інформаційних систем та забезпечують широкий набір інструментів для реалізації підсистеми пошуку, збору та аналізу користувацьких відгуків. Водночас Python має певну перевагу для даного проєкту завдяки великій кількості бібліотек для аналізу текстів, машинного навчання та обробки природної мови, що є важливими складовими систем аналізу користувацьких відгуків.

Оцінки виставимо в таблиці 2.1. по 5–бальній шкалі де 1 це дуже погано, а 5 це дуже добре.

2.1 Порівняння мов програмування

Мова програмування	Java	C#	Python
Швидкість та ефективність	5	4	2
Кросплатформеність	5	3	4
Масштабованість та розширюваність	5	3	4
Кількість бібліотек та інструментів	4	3	5
Зручність та швидкість розробки	4	4	5
Універсальність та адаптивність під різні задачі	4	3	5
Безпека та надійність	5	5	3
Загальна кількість балів	32	25	28

Проведений аналіз показав, що Java найбільшою мірою відповідає функціональним і технічним вимогам розроблюваної підсистеми. Завдяки своїй зрілості, надійності та широким можливостям для створення масштабованих

інформаційних систем ця мова програмування є доцільним вибором для реалізації проєкту. Розглянемо основні переваги Java, які обґрунтовують її використання.

– Кросплатформність. Однією з ключових особливостей Java є принцип «Write Once, Run Anywhere» («Написав один раз — запускай будь-де»). Програмний код компілюється у байт-код, який виконується віртуальною машиною Java (JVM). Завдяки цьому програмні продукти можуть працювати на різних операційних системах без необхідності внесення змін до вихідного коду, що значно спрощує процес розробки та супроводу.

– Надійність і безпека. Java має вбудовані механізми захисту від багатьох типових помилок програмування. Автоматичне керування пам'яттю за допомогою механізму Garbage Collection дозволяє мінімізувати ризик виникнення витоків пам'яті, а сувора типізація та система перевірки коду під час компіляції сприяють підвищенню стабільності програмного забезпечення.

– Розвинена екосистема. Java володіє однією з найбільших екосистем серед сучасних мов програмування. Розробникам доступна велика кількість бібліотек, фреймворків та інструментів для реалізації практично будь-яких завдань. Особливо важливими для даного проєкту є такі технології, як Spring Boot, Hibernate, Elasticsearch, Apache Kafka та інші рішення для створення корпоративних інформаційних систем.

– Об'єктно-орієнтований підхід. Java повністю підтримує принципи об'єктно-орієнтованого програмування, що забезпечує високу структурованість програмного коду. Використання інкапсуляції, наслідування та поліморфізму спрощує розробку, тестування та подальший супровід програмних компонентів.

– Масштабованість і продуктивність. Java добре зарекомендувала себе у високонавантажених системах, які працюють із великими обсягами даних та значною кількістю одночасних користувачів. Розвинені засоби багатопотокового програмування та підтримка сучасних архітектурних підходів дозволяють ефективно масштабувати систему відповідно до зростаючих вимог.

– Підтримка командної розробки. Для Java сформовано широкий набір стандартів, практик та інструментів, які сприяють організації командної роботи. Системи керування залежностями, такі як Maven та Gradle, а також засоби автоматизованого тестування й безперервної інтеграції дозволяють ефективно підтримувати великі програмні проєкти та забезпечувати високу якість коду.

Таким чином, Java поєднує високу продуктивність, надійність, масштабованість та розвинену екосистему інструментів, що робить її оптимальним вибором для реалізації підсистеми пошуку, збору та аналізу користувацьких відгуків.

Водночас Python також заслуговує на увагу як перспективна технологія для окремих компонентів системи. Хоча за деякими характеристиками, зокрема продуктивністю та можливостями створення великих корпоративних систем, Python поступається Java, він має суттєві переваги у сфері аналізу даних, машинного навчання та обробки природної мови.

Зокрема, Python може бути ефективно використаний для реалізації допоміжних модулів, пов'язаних зі збором інформації з вебресурсів. Завдяки бібліотекам BeautifulSoup, Scrapy, Selenium та Requests він є одним із найпопулярніших інструментів для вебскрапінгу та автоматизованого отримання користувацьких відгуків із зовнішніх джерел. Крім того, Python може застосовуватися для аналізу тональності текстів, класифікації відгуків та виконання інших завдань, пов'язаних із штучним інтелектом і машинним навчанням.

Отже, основною мовою розробки доцільно обрати Java, тоді як Python може використовуватися як додатковий інструмент для реалізації спеціалізованих модулів збору та аналітичної обробки текстових даних.

2.2 Обґрунтування вибору інструментів розробки

У якості основного середовища розробки було обрано IntelliJ IDEA - сучасну інтегровану середу розробки (IDE), яка зарекомендувала себе як одна з

найпотужніших та найзручніших платформ для створення програмного забезпечення, зокрема на мові Java та інших JVM-сумісних мовах. IntelliJ IDEA широко використовується у професійній розробці завдяки високій продуктивності, інтелектуальним можливостям аналізу коду та глибокій інтеграції з сучасними технологіями розробки програмних систем.

Інтерфейс середовища відзначається логічною структурою та зручністю використання, що дозволяє ефективно організовувати робочий процес розробника. Приклад інтерфейсу IntelliJ IDEA наведено на рисунку 2.1.

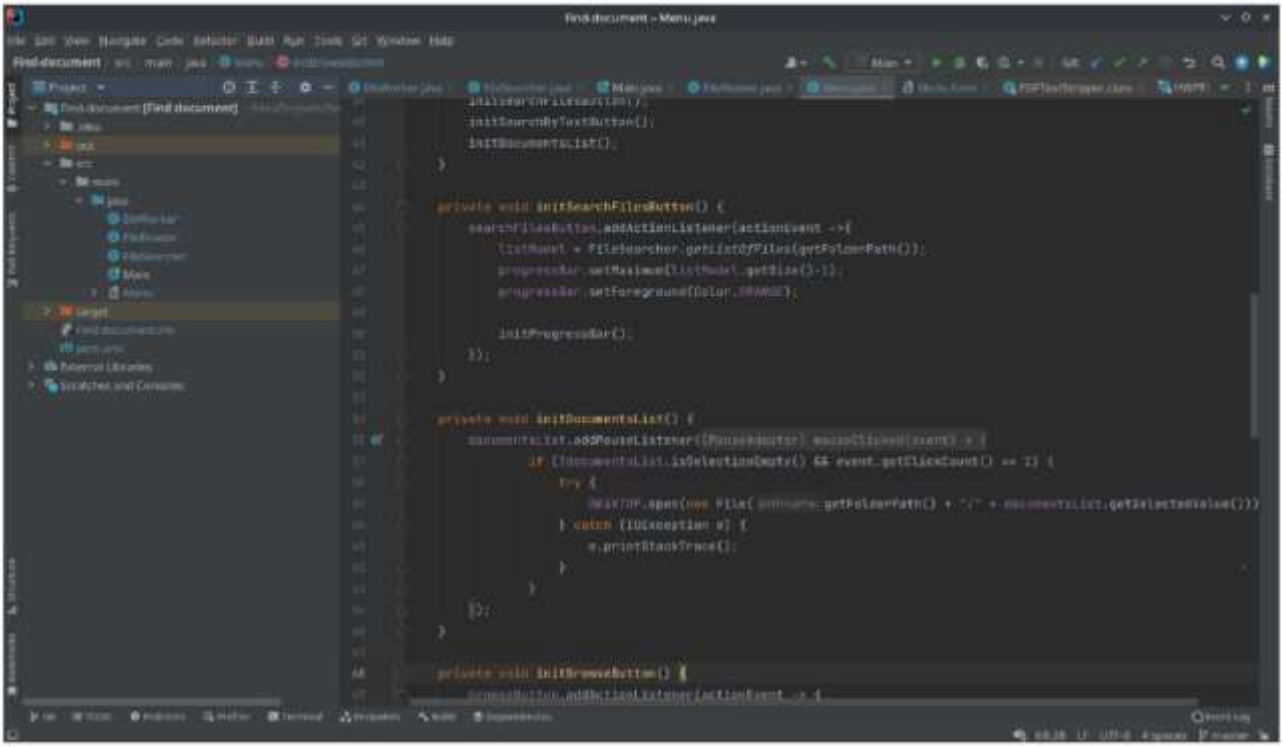


Рис. 2.1. Вигляд інтерфейсу Intellij IDEA

IntelliJ IDEA є потужною IDE, яка підтримує не лише Java, але й такі мови програмування, як Kotlin, Scala, Groovy, а також інші технології, що працюють на платформі JVM. Крім того, сучасні версії середовища забезпечують підтримку веброзробки, роботи з JavaScript, TypeScript, SQL, Python (через плагіни), а також інструментів для роботи з хмарними сервісами та контейнеризацією, такими як Docker і Kubernetes.

Серед ключових переваг IntelliJ IDEA можна виділити такі:

– Інтелектуальний редактор коду. Середовище забезпечує потужну систему автодоповнення, контекстно-залежні підказки, швидку навігацію по коду та автоматичне виявлення потенційних помилок ще на етапі написання програмного коду. Це значно підвищує продуктивність розробника та зменшує кількість синтаксичних і логічних помилок.

– Розвинуті інструменти рефакторингу. IntelliJ IDEA надає широкий набір інструментів для безпечного змінювання структури коду без порушення його функціональності. Автоматизований рефакторинг дозволяє покращувати архітектуру програмного забезпечення, підтримуючи його читабельність, масштабованість та супровід.

– Аналіз коду та виявлення помилок у реальному часі. Середовище виконує статичний аналіз коду під час його написання, що дозволяє оперативно виявляти потенційні помилки, антипатерни та проблеми продуктивності. Це сприяє підвищенню якості програмного забезпечення ще до етапу компіляції та тестування.

– Глибока інтеграція з сучасними інструментами розробки. IntelliJ IDEA підтримує інтеграцію з системами контролю версій (Git, GitHub, GitLab), системами автоматизації збірки (Maven, Gradle), фреймворками (Spring Boot, Hibernate), а також інструментами CI/CD. У сучасних умовах розробки це забезпечує повний цикл життєвого циклу програмного забезпечення в межах одного середовища.

– Підтримка тестування програмного забезпечення. IDE містить вбудовані інструменти для створення та виконання автоматизованих тестів (JUnit, TestNG), що сприяє впровадженню практик тестування на ранніх етапах розробки (TDD — Test Driven Development). Це підвищує стабільність та надійність програмних рішень.

– Підтримка сучасних технологій та хмарної розробки. Сучасні версії IntelliJ IDEA активно інтегруються з хмарними платформами (AWS, Azure, Google Cloud), а також підтримують роботу з контейнеризацією (Docker) і оркестрацією

(Kubernetes), що є актуальним для побудови масштабованих розподілених систем.

– Кросплатформеність. IntelliJ IDEA працює на операційних системах Windows, macOS та Linux, що забезпечує гнучкість використання у різних середовищах розробки.

Додатково слід зазначити, що IntelliJ IDEA активно використовує технології штучного інтелекту для підвищення ефективності розробки (наприклад, JetBrains AI Assistant), що дозволяє автоматизувати частину рутинних завдань, покращувати якість коду та прискорювати процес програмування.

Таким чином, IntelliJ IDEA є сучасним професійним середовищем розробки, яке поєднує високу продуктивність, інтелектуальні можливості аналізу коду та широку інтеграцію з актуальними технологіями програмної інженерії. Це робить її оптимальним вибором для розробки складних інформаційних систем, зокрема підсистеми пошуку, збору та аналізу користувацьких відгуків.

2.3 Обґрунтування вибору архітектурних методів та підходів розробки

Використання мови Java безпосередньо передбачає застосування об’єктно-орієнтованої парадигми програмування (ООП), яка є однією з найбільш поширених і ефективних методологій розробки сучасного програмного забезпечення. ООП розглядає програмну систему як сукупність взаємопов’язаних об’єктів, що поєднують у собі дані та методи їх обробки, а також взаємодіють між собою для виконання визначених завдань.

Сьогодні об’єктно-орієнтований підхід залишається домінуючою парадигмою у промисловій розробці програмного забезпечення, зокрема у створенні корпоративних систем, вебзастосунків, мікросервісних архітектур та хмарних рішень. Його широке застосування пояснюється високим рівнем структурованості, повторного використання коду та зручністю масштабування проєктів.

Основними принципами об’єктно-орієнтованого програмування є інкапсуляція, наслідування, поліморфізм та абстракція. Ці принципи формують основу побудови гнучких, розширюваних і підтримуваних програмних систем [21, 22].

Інкапсуляція забезпечує приховування внутрішньої реалізації об’єкта та надання доступу до його функціональності лише через визначений інтерфейс. Наслідування дозволяє створювати нові класи на основі вже існуючих, повторно використовуючи їх властивості та поведінку. Поліморфізм дає можливість використовувати єдиний інтерфейс для роботи з різними типами об’єктів, що підвищує гнучкість системи. Абстракція, у свою чергу, дозволяє виділяти лише суттєві характеристики об’єктів, приховуючи другорядні деталі реалізації.

Приклад реалізації ООП на практиці зображено на рисунку 2.2.

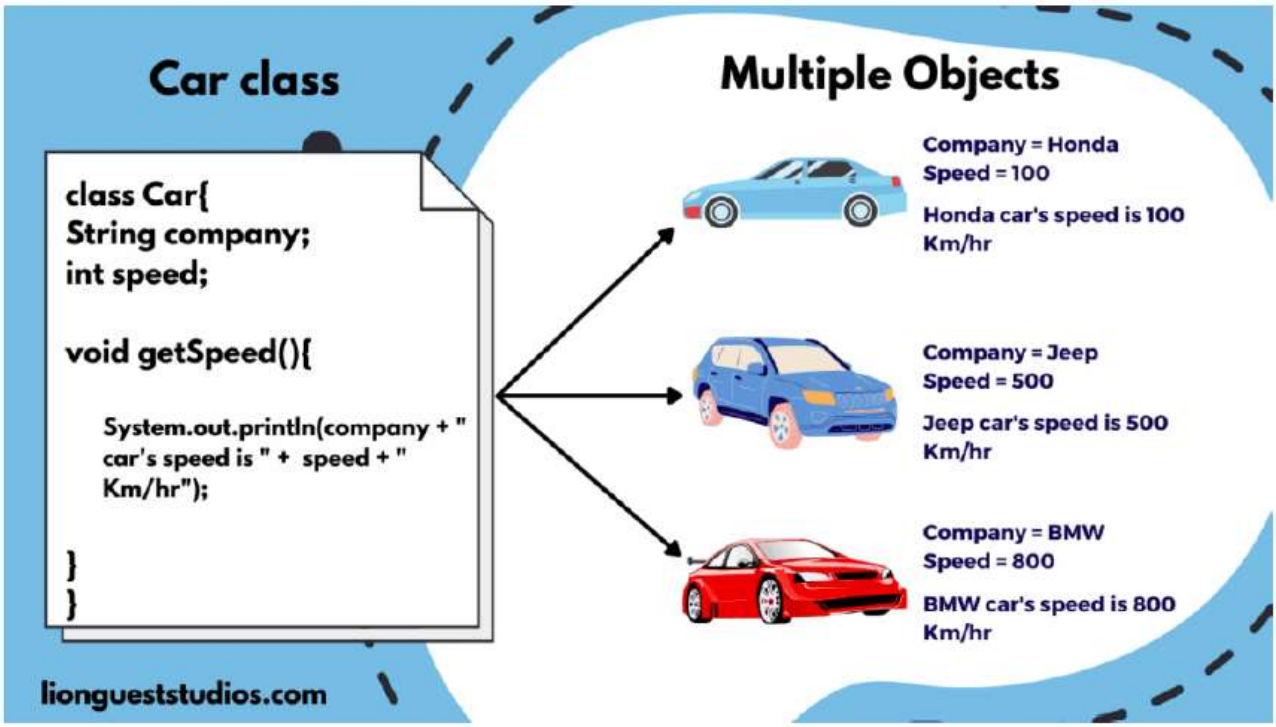


Рис. 2.2 Приклад реалізації ООП з класом Car

Однією з ключових переваг ООП є підвищення модульності програмного забезпечення. Завдяки поділу системи на окремі класи та об’єкти значно спрощується її розробка, тестування та подальший супровід. У сучасних великих програмних системах тисячі рядків процедурного коду можуть бути

структуровані у вигляді десятків або сотень взаємопов’язаних класів, що суттєво покращує читабельність та керованість коду.

У контексті сучасної розробки особливо важливою є здатність ООП підтримувати повторне використання коду. Класи та бібліотеки можуть використовуватися в різних частинах системи або навіть у різних проєктах, що зменшує час розробки та підвищує ефективність командної роботи. Це є особливо актуальним для масштабованих інформаційних систем, де необхідно забезпечити швидке розширення функціоналу.

Ще однією важливою перевагою є зниження складності програмних систем завдяки високому рівню абстракції. Розробник може зосереджуватися на логіці роботи системи, не заглиблюючись у деталі реалізації кожного компонента. Це значно спрощує проєктування складних інформаційних рішень.

Поліморфізм, як один із ключових механізмів ООП, у сучасній практиці широко використовується для реалізації гнучких архітектур, зокрема у мікросервісних системах та патернах проєктування (наприклад, Strategy, Factory, Observer). Це дозволяє створювати легко розширювані системи, які можуть адаптуватися до нових вимог без значних змін існуючої архітектури.

Крім того, ООП дозволяє ефективно моделювати реальні об’єкти та процеси, що є важливим при розробці інформаційних систем. Такий підхід сприяє більш інтуїтивному проєктуванню програмного забезпечення, оскільки структура коду наближається до реальних предметних областей.

У сучасній розробці програмного забезпечення ООП також тісно інтегрується з іншими підходами, зокрема функціональним програмуванням та подієво-орієнтованою архітектурою. У мовах нового покоління, таких як Java (починаючи з Java 8 і вище), активно використовуються лямбда-вирази, стріми та функціональні інтерфейси, що розширює можливості класичної об’єктної моделі.

Підсумовуючи, об’єктно-орієнтоване програмування забезпечує високу структурованість коду, його повторне використання, масштабованість і зручність супроводу. Саме ці властивості зробили ООП основною парадигмою у

сучасній програмній інженерії та ключовим підходом при розробці складних інформаційних систем.

Таким чином, Java як мова, що базується на принципах ООП, є ефективним інструментом для створення надійних, масштабованих та структурованих програмних рішень, зокрема підсистеми пошуку, збору та аналізу користувацьких відгуків.

2.4 Обґрунтування вибору систем управління базами даних

Для реалізації підсистеми пошуку, збору та аналізу користувацьких відгуків передбачається використання системи управління базами даних (СУБД) для зберігання індексів, метаданих та структурованої інформації. Серед найпоширеніших сучасних реляційних СУБД виділяють MySQL та PostgreSQL. Обидві системи є достатньо зрілими, продуктивними та широко використовуються у промисловій розробці, однак у контексті даного проєкту доцільнішим є вибір PostgreSQL, оскільки вона забезпечує більшу гнучкість, розширюваність та підтримку сучасних типів даних.

PostgreSQL є однією з найпотужніших відкритих об’єктно-реляційних СУБД, яка активно використовується у сучасних інформаційних системах, аналітичних платформах та вебзастосунках. Її популярність у 2020–2025 роках стабільно зростає завдяки високій надійності, підтримці складних запитів та можливостям роботи з неструктурованими даними, зокрема JSON/JSONB, що є особливо актуальним для систем аналізу текстових відгуків.

Схему роботи JDBC наведено на рис. 2.3.

До ключових переваг PostgreSQL належать:

– Висока надійність і безпека. СУБД забезпечує повну підтримку транзакцій і принципів ACID, що гарантує цілісність даних навіть у разі збоїв системи. Крім того, PostgreSQL підтримує розвинуту систему ролей, автентифікації, шифрування та контролю доступу, що дозволяє ефективно захищати інформацію.

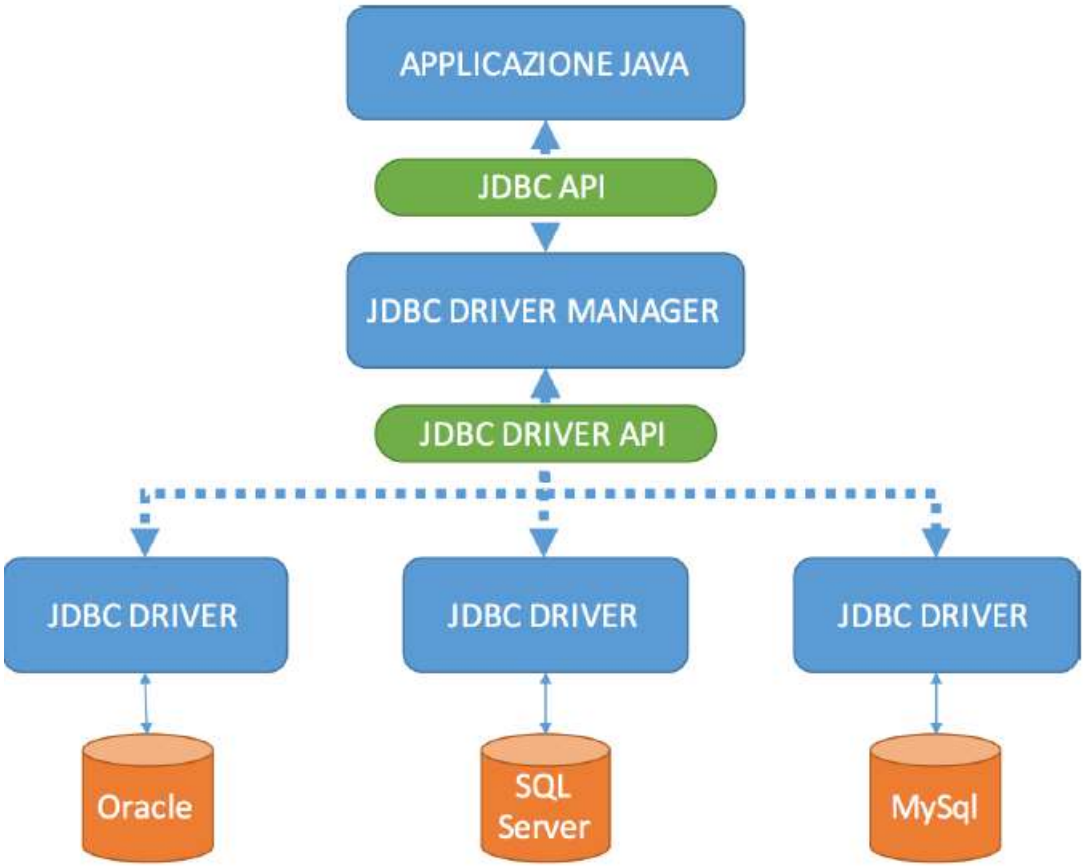


Рис. 2.3 Схема роботи JDBC

– Підтримка сучасних типів даних і SQL-можливостей. PostgreSQL реалізує розширений SQL-стандарт та підтримує роботу з геоданими (PostGIS), JSON/JSONB, масивами, hstore та іншими структурами. Це робить її особливо зручною для зберігання напівструктурованих даних, таких як користувацькі відгуки з додатковими метаданими.

– Висока розширюваність. Система дозволяє створювати власні типи даних, функції, оператори та індекси. Завдяки великій кількості розширень (наприклад, pg_trgm для текстового пошуку або PostGIS для геоданих) PostgreSQL легко адаптується під специфічні вимоги проєкту.

– Оптимізація продуктивності. PostgreSQL підтримує багаторівневе індексування (B-tree, GIN, GiST, BRIN), паралельне виконання запитів та сучасні механізми оптимізації виконання SQL-запитів. Це забезпечує високу швидкість

обробки великих обсягів даних, що є критично важливим для систем аналізу відгуків.

– Масштабованість і реплікація. СУБД підтримує як синхронну, так і асинхронну реплікацію, що дозволяє забезпечити високу доступність системи та горизонтальне масштабування. У сучасних хмарних архітектурах PostgreSQL часто використовується разом із контейнеризацією (Docker, Kubernetes), що додатково підвищує гнучкість розгортання.

– Активна спільнота та відкритий код. PostgreSQL є повністю відкритим програмним забезпеченням (Open Source) і має одну з найактивніших спільнот у сфері СУБД. Це забезпечує регулярні оновлення, швидке усунення помилок та постійний розвиток функціональності.

У поєднанні з PostgreSQL у проєкті доцільно використовувати фреймворк Hibernate, який забезпечує об’єктно-реляційне відображення (ORM) у середовищі Java. Основна мета Hibernate полягає у спрощенні взаємодії з базою даних шляхом роботи з об’єктами замість безпосереднього використання SQL-запитів.

Серед ключових можливостей Hibernate слід виділити:

– ORM (Object-Relational Mapping). Hibernate забезпечує автоматичне відображення Java-об’єктів на таблиці реляційної бази даних, що значно спрощує роботу з даними та зменшує обсяг ручного SQL-коду.

– Апаратна незалежність від СУБД. Завдяки абстракції доступу до даних, застосунок може працювати з різними СУБД (PostgreSQL, MySQL, Oracle тощо) без суттєвих змін у коді.

– HQL (Hibernate Query Language). Об’єктно-орієнтована мова запитів дозволяє працювати з сутностями, а не з таблицями, що підвищує читабельність і підтримуваність коду.

– Кешування даних. Hibernate підтримує рівні кешування (first-level і second-level cache), що дозволяє значно підвищити продуктивність за рахунок зменшення кількості звернень до бази даних.

– Управління транзакціями та сесіями. Фреймворк забезпечує надійне управління транзакціями, що гарантує узгодженість даних та спрощує контроль життєвого циклу об’єктів.

– Інтеграція із сучасними технологіями. Hibernate повністю сумісний із екосистемою Java, зокрема з Spring Boot, Spring Data JPA та Jakarta Persistence API (JPA), що робить його стандартом де-факто для сучасної Java-розробки.

Важливо зазначити, що Hibernate функціонує поверх JDBC (Java Database Connectivity) - стандартного API для взаємодії Java-додатків із реляційними базами даних. JDBC входить до складу Java SE та забезпечує базовий рівень роботи з SQL-запитами, тоді як Hibernate додає високорівневу абстракцію, значно спрощуючи процес розробки.

Отже, поєднання PostgreSQL як надійної та масштабованої СУБД і Hibernate як ORM-фреймворку дозволяє створити продуктивну, гнучку та легко підтримувану систему зберігання та обробки даних, що є особливо важливим для підсистеми аналізу користувацьких відгуків у сучасних інформаційних системах.

2.5 Архітектурні шаблони

Етап проєктування програмного забезпечення є критично важливим у процесі розробки сучасних інформаційних систем, оскільки саме на цьому етапі визначається загальна архітектура, структура компонентів та взаємодія між ними. Якісне проєктування дозволяє мінімізувати кількість помилок на етапі реалізації, забезпечити масштабованість системи та спростити її подальший супровід і розвиток. У сучасній програмній інженерії існує велика кількість перевірених архітектурних підходів, які використовуються для побудови надійних і гнучких систем.

Для розроблюваної підсистеми пошуку, збору та аналізу користувацьких відгуків доцільно розглянути два поширені архітектурні підходи: Model–View–Controller (MVC) та Representational State Transfer (REST). Ці підходи часто

використовуються спільно, доповнюючи один одного у побудові сучасних вебзастосунків. Загальна концепція взаємодії компонентів MVC наведена на рисунку 2.4.

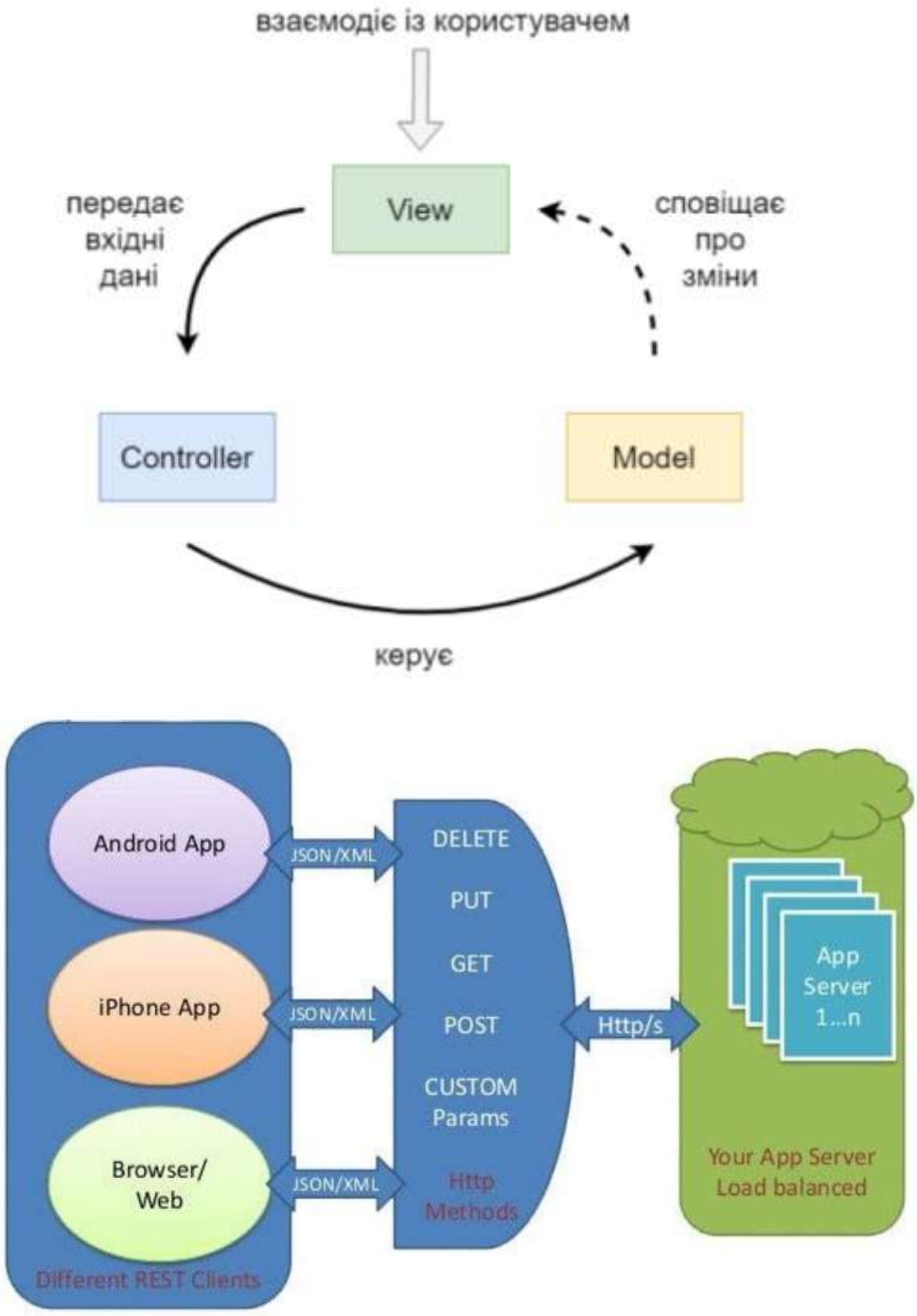


Рис. 2.3 Схема роботи MVC

Архітектурний шаблон MVC передбачає розділення програмної системи на три основні компоненти: модель (Model), представлення (View) та контролер

(Controller). Такий поділ забезпечує чітку структуризацію коду та розмежування відповідальностей між різними рівнями системи.

Model відповідає за зберігання та обробку даних, а також реалізацію бізнес-логіки. View забезпечує відображення інформації користувачу, тоді як Controller виступає посередником між користувацьким інтерфейсом та бізнес-логікою, обробляючи запити та координуючи взаємодію компонентів.

Основною перевагою MVC є підвищення модульності програмного забезпечення та розділення відповідальностей, що дозволяє змінювати інтерфейс користувача без впливу на бізнес-логіку і навпаки. Це сприяє повторному використанню компонентів та спрощує командну розробку.

У сучасних умовах MVC продовжує широко використовуватися у веброзробці, зокрема у фреймворках Spring MVC, ASP.NET MVC та подібних рішеннях. Водночас із розвитком складних розподілених систем його класична форма має певні обмеження.

До основних недоліків MVC можна віднести:

– Складність масштабування. При зростанні обсягу системи контролери та моделі можуть ставати перевантаженими, що ускладнює підтримку архітектури.

– Перевантаження контролерів. У великих системах контролери часто починають містити надмірну кількість логіки, що суперечить принципам чистої архітектури.

– Розмитість відповідальностей. У складних проєктах іноді виникає нечітке розмежування між бізнес-логікою, логікою представлення та обробкою запитів.

– Проблеми тестування та супроводу. Через можливе змішування рівнів абстракції ускладнюється модульне тестування та повторне використання компонентів.

– Обмежена придатність для сучасних розподілених систем. MVC історично орієнтований на монолітні або напівмонолітні застосунки, що може бути недостатньо гнучким для мікросервісної архітектури.

Паралельно з MVC широкого застосування набув архітектурний стиль REST (Representational State Transfer), який є одним із базових підходів до

створення сучасних вебсервісів та API. REST був формалізований Роєм Філдінгом у 2000 році та став стандартом де-факто для побудови веборієнтованих систем.

REST базується на наборі архітектурних принципів, які забезпечують масштабованість, простоту та незалежність компонентів системи. Основними з них є:

– Клієнт-серверна архітектура. Розділення відповідальності між клієнтом і сервером дозволяє незалежно розвивати обидві частини системи.

– Stateless (відсутність стану). Кожен запит містить усю необхідну інформацію для його обробки, що підвищує масштабованість та спрощує балансування навантаження.

– Кешування. Підтримка кешування дозволяє зменшити навантаження на сервер і підвищити продуктивність системи.

– Уніфікований інтерфейс. Використання стандартних HTTP-методів (GET, POST, PUT, DELETE) та уніфікованих ресурсів (URI) спрощує інтеграцію між різними системами.

– Багаторівнева архітектура. Система може складатися з декількох шарів, що підвищує її масштабованість та безпеку.

– Гіпермедіа як рушій стану (HATEOAS). Клієнт взаємодіє з системою через гіперпосилання, що дозволяє динамічно змінювати поведінку API.

У сучасних умовах REST широко використовується для побудови вебAPI, мікросервісних систем та хмарних застосунків. Його перевагами є простота реалізації, сумісність із HTTP-стандартами, висока масштабованість та легкість інтеграції з іншими системами. Особливо актуальним є використання REST у поєднанні з JSON як основним форматом обміну даними, що є стандартом у сучасних вебзастосунках 2020-х років.

Водночас REST має і певні обмеження. Зокрема, відсутність стану ускладнює реалізацію складних бізнес-процесів, які потребують збереження контексту між запитами. Також складні операції можуть вимагати великої

кількості запитів, що впливає на продуктивність у високонавантажених системах.

Попри це, REST залишається одним із найбільш поширених архітектурних стилів завдяки своїй універсальності та сумісності з сучасними вебтехнологіями. У контексті розроблюваної системи REST є більш доцільним вибором для реалізації серверного API, тоді як MVC може використовуватися на рівні побудови окремих компонентів застосунку або адміністративного інтерфейсу.

Таким чином, поєднання MVC-підходу для структуризації внутрішньої логіки та REST-архітектури для взаємодії між компонентами дозволяє створити гнучку, масштабовану та сучасну систему, яка відповідає вимогам розподілених інформаційних систем.

РОЗДІЛ 3 РОЗРОБКА ТА ТЕСТУВАННЯ ПІДСИСТЕМИ

3.1 Інтеграція інструменту автоматичної збірки проекту

Для автоматизації процесу збірки програмного забезпечення, а також керування залежностями та версіями підключених бібліотек у проєкті використовується інструмент Apache Maven. Він є одним із найбільш поширених засобів автоматизації збірки в екосистемі Java та активно застосовується у сучасних корпоративних і вебзастосунках.

Основною метою Maven є спрощення процесу управління життєвим циклом проєкту. Він забезпечує стандартизований підхід до компіляції коду, запуску тестів, пакування застосунку та його розгортання. Крім того, Maven автоматизує підключення зовнішніх бібліотек, самостійно завантажуючи необхідні залежності з централізованих репозиторіїв (зокрема Maven Central Repository), що суттєво спрощує налаштування проєкту та зменшує кількість ручних операцій.

У сучасній розробці Maven також відіграє важливу роль у забезпеченні відтворюваності збірки, що означає можливість отримати однаковий результат компіляції на різних середовищах без додаткових налаштувань. Це є особливо важливим у командній розробці та при використанні CI/CD-процесів (Continuous Integration / Continuous Deployment), які стали стандартом у сучасній індустрії програмного забезпечення.

Серед ключових переваг Maven можна виділити:

– Управління залежностями. Maven автоматично завантажує необхідні бібліотеки та їхні транзитивні залежності, що значно зменшує ручну роботу розробника.

– Стандартизований життєвий цикл проєкту. Інструмент визначає чіткі етапи збірки (compile, test, package, install, deploy), що забезпечує уніфікований процес розробки.

– Підтримка плагінної архітектури. Maven дозволяє розширювати функціональність за допомогою плагінів, наприклад для запуску тестів (Surefire), аналізу коду або створення документації.

– Інтеграція з сучасними CI/CD системами. Maven широко використовується разом із Jenkins, GitHub Actions, GitLab CI та іншими інструментами автоматизації, що дозволяє реалізувати безперервну інтеграцію та доставку програмного продукту.

– Відтворюваність збірки. Опис проєкту у вигляді конфігураційного файлу pom.xml дозволяє відтворити однакове середовище збірки на різних машинах.

Сучасні версії IntelliJ IDEA забезпечують повну інтеграцію з Maven, що значно спрощує процес створення та налаштування проєктів. Під час створення нового Java-проєкту користувач може обрати Maven як систему автоматизації збірки, після чого IDE автоматично генерує необхідну структуру проєкту, включаючи файл pom.xml, стандартні директорії (src/main/java, src/test/java) та базову конфігурацію.

На рисунках 3.1 та 3.2 наведено приклади діалогових вікон IntelliJ IDEA, де здійснюється вибір Maven як інструменту збірки, а також налаштування основних параметрів нового проєкту, таких як groupId, artifactId та версія застосунку.

У результаті налаштування створюється автоматично згенерований файл конфігурації pom.xml, який є центральним елементом Maven-проєкту. Саме в цьому файлі визначаються всі залежності, плагіни та параметри збірки. Розробник може додавати необхідні бібліотеки шляхом опису відповідних залежностей, після чого Maven автоматично завантажує їх із репозиторіїв.

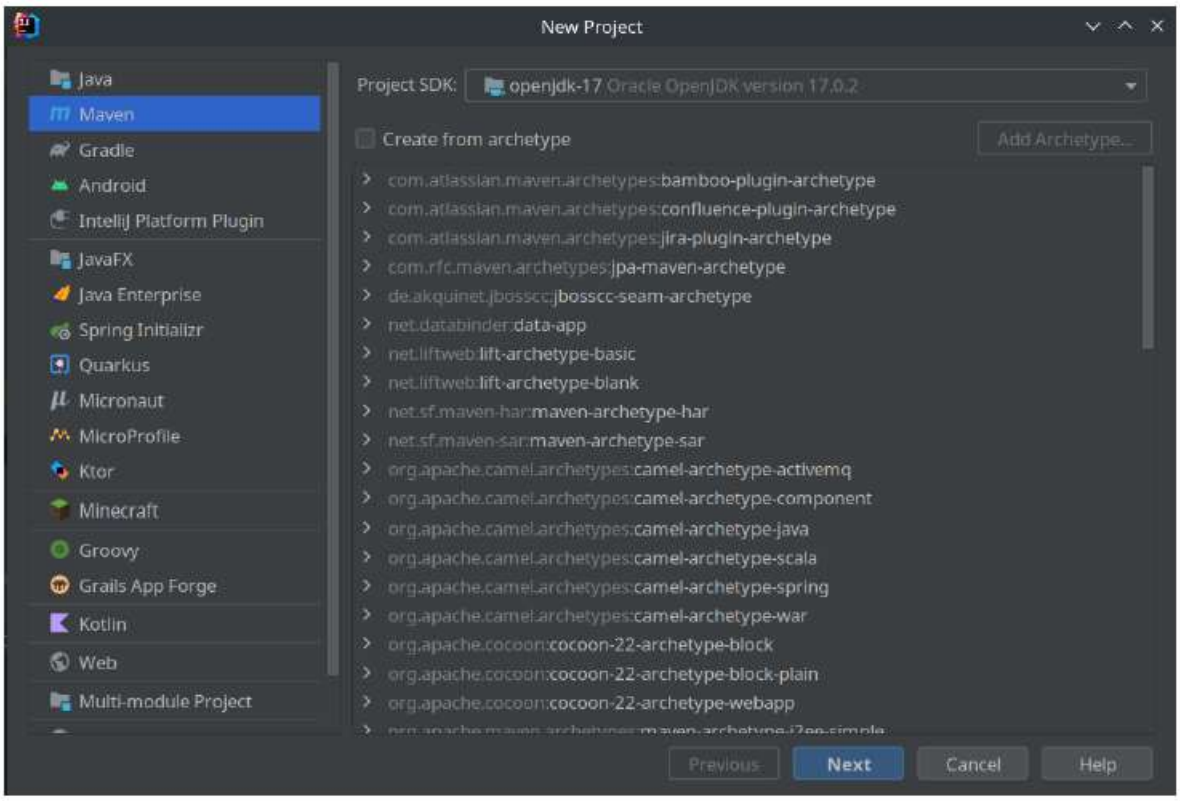


Рис. 3.1 Вибір Maven при створенні нового проекту

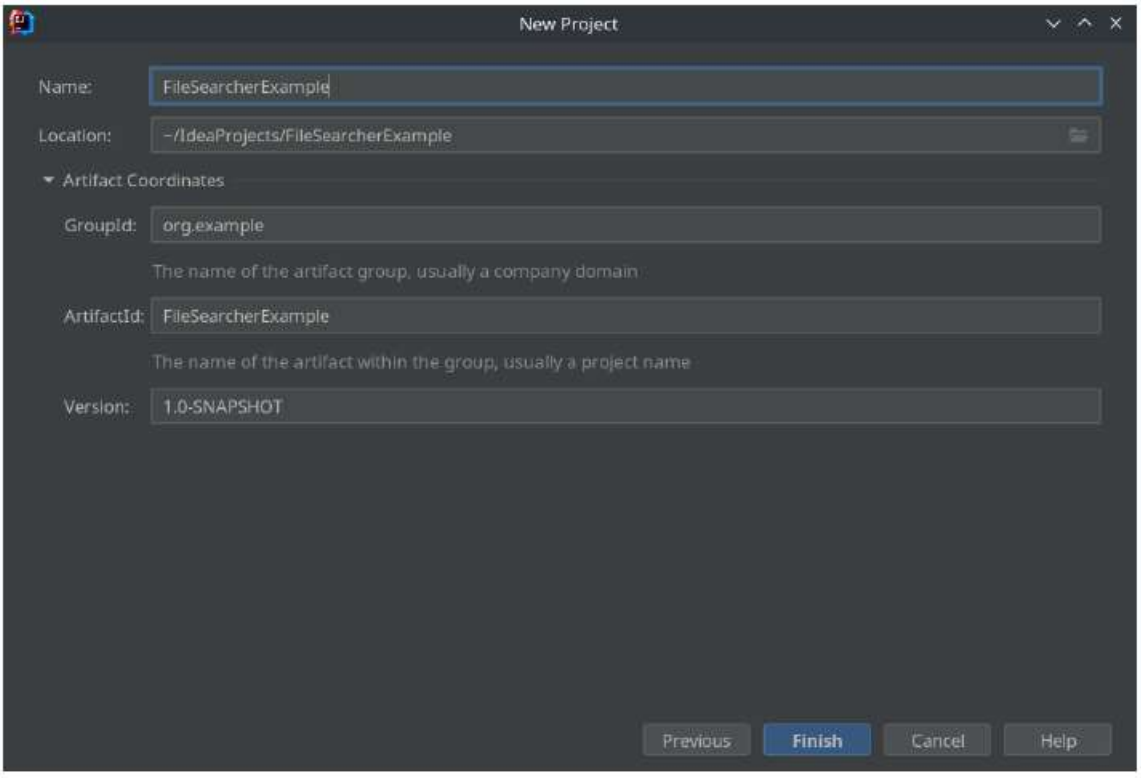


Рис. 3.2 Задання параметрів для Maven та підтвердження вибору

Таким чином, використання Maven у поєднанні з IntelliJ IDEA дозволяє значно підвищити ефективність розробки, автоматизувати процеси збірки та управління залежностями, а також забезпечити стабільність і відтворюваність програмного середовища, що є важливим для сучасних масштабованих інформаційних систем.

3.2 Розробка програмного забезпечення

Розробка програмного забезпечення є багатоетапним і складним процесом, який вимагає дотримання певних методологічних підходів, принципів і правил. У сучасній програмній інженерії ефективність створення програмних систем значною мірою залежить від правильного організування процесу розробки, якості комунікації в команді та використання сучасних інструментів автоматизації.

Однією з ключових передумов успішної розробки є чітке розуміння вимог до програмного продукту. На сучасному етапі широко застосовуються підходи збору вимог із використанням user stories, use cases та інструментів моделювання (UML, BPMN), що дозволяє більш точно формалізувати очікування замовника та зменшити ризик неправильного трактування задач.

Важливим аспектом є також поетапна організація розробки та ведення технічної документації. Сучасні підходи передбачають безперервне документування (continuous documentation), коли документація оновлюється разом із кодом у репозиторії, часто у форматі Markdown або у спеціалізованих системах (Confluence, Notion, GitBook). Це дозволяє забезпечити актуальність інформації протягом усього життєвого циклу проєкту.

Не менш важливим є використання систем контролю версій, серед яких стандартом де-факто є Git. У поєднанні з платформами GitHub, GitLab або Bitbucket він дозволяє організувати командну роботу, відстежувати зміни коду, керувати гілками розробки та забезпечувати можливість відкату до попередніх

версій системи. У сучасній практиці активно застосовуються Git Flow або trunk-based development як моделі роботи з репозиторіями.

Окрему роль відіграє тестування програмного забезпечення. У сучасних умовах воно інтегрується безпосередньо в процес розробки (shift-left testing) і включає unit-тести, інтеграційні та end-to-end тести. Автоматизоване тестування є стандартом у CI/CD-конвеєрах, що дозволяє своєчасно виявляти помилки та підвищувати якість коду.

Безпека програмного забезпечення також стала одним із ключових аспектів розробки. Сучасні практики включають DevSecOps-підхід, коли перевірки безпеки інтегруються безпосередньо в процес розробки. Це включає аналіз залежностей (dependency scanning), статичний аналіз коду (SAST) та перевірку вразливостей у рантаймі.

Ефективна командна робота та комунікація сьогодні забезпечуються не лише внутрішніми процесами, але й сучасними інструментами (Jira, Trello, Slack, Microsoft Teams), а також гнучкими методологіями управління проєктами.

Ще одним важливим принципом є спрощення архітектури та коду. Сучасна інженерна практика базується на принципах SOLID, KISS, DRY та YAGNI, які спрямовані на зменшення складності системи, підвищення читабельності коду та полегшення його супроводу.

Ітеративний підхід до розробки сьогодні є стандартом у більшості проєктів. Він передбачає поступове вдосконалення системи з регулярним отриманням зворотного зв’язку від користувачів або замовника. Це дозволяє швидко адаптувати продукт до змінних вимог ринку.

Серед сучасних підходів до розробки програмного забезпечення можна виділити:

– Agile-методології (Scrum, Kanban). Орієнтовані на швидку адаптацію до змін, короткі ітерації та постійний зворотний зв’язок.

– Waterfall-модель. Класичний послідовний підхід, який застосовується у проєктах із чітко визначеними вимогами, хоча сьогодні використовується значно рідше.

– DevOps-підхід. Інтеграція процесів розробки та експлуатації з автоматизацією CI/CD, що дозволяє швидше доставляти оновлення користувачам.

– V-подібна модель (V-Model). Передбачає симетричне співвідношення між етапами розробки та тестування, що забезпечує високу контрольованість процесу.

– Мікросервісна архітектура. Сучасний підхід до побудови систем як набору незалежних сервісів, що полегшує масштабування, оновлення та підтримку системи.

У межах даної розробки обґрунтовано використання V-подібної моделі життєвого циклу програмного забезпечення, оскільки вона забезпечує структурованість процесу, чітке співвідношення між етапами розробки та тестування, а також дозволяє ефективно контролювати якість на кожному етапі реалізації системи.

Основними перевагами V-моделі є:

– Чітка структурованість процесу. Кожному етапу розробки відповідає відповідний рівень тестування, що забезпечує логічну послідовність робіт.

– Підвищена передбачуваність результатів. Завдяки формалізованим етапам легше оцінювати прогрес і відповідність вимогам.

– Раннє виявлення помилок. Планування тестування на ранніх етапах дозволяє зменшити кількість критичних дефектів у фінальному продукті.

– Зручність управління проєктом. Чіткий поділ на етапи спрощує планування, контроль та розподіл завдань у команді.

– Зменшення ризиків невідповідності вимогам. Формалізація процесу дозволяє краще відстежувати відповідність реалізації початковим вимогам.

Таким чином, сучасна розробка програмного забезпечення базується на поєднанні гнучких методологій, автоматизації процесів та структурованих моделей життєвого циклу. Використання V-подібної моделі у поєднанні з сучасними практиками (Agile, DevOps, CI/CD) дозволяє забезпечити баланс між передбачуваністю, якістю та гнучкістю розробки.

Принципи, методи та їх візуальний вигляд наведено на рисунку 3.3

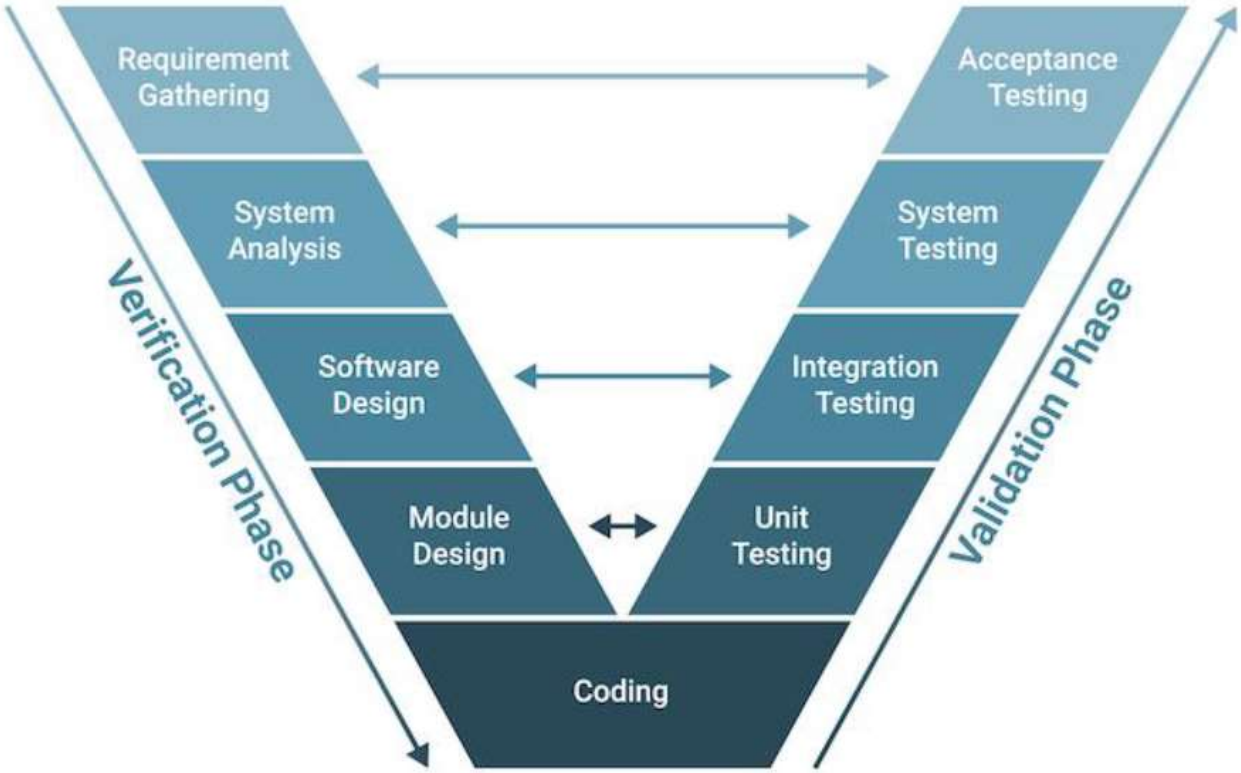


Рис. 3.3 Структура V-образу

На рисунку 3.4 зображено розроблений метод для спілкування з PaLM API.

```
@SneakyThrows
public String curlToGoogleModel(String prompt) {
    URL url = new URL( spec: "https://generativelanguage.googleapis.com/v1beta3/models/text-bison-001:generateText?

    HttpURLConnection httpConnection = (HttpURLConnection) url.openConnection();
    httpConnection.setRequestMethod("POST");
    httpConnection.setRequestProperty("Content-Type", "application/json");
    httpConnection.setDoOutput(true);

    OutputStreamWriter writer = new OutputStreamWriter(httpConnection.getOutputStream());
    writer.write( str: "{ \"prompt\": { \"text\": \"" + prompt + "\" } }");
    writer.flush();
    writer.close();
    httpConnection.getOutputStream().close();

    InputStream responseStream = httpConnection.getResponseCode() / 100 == 2
            ? httpConnection.getInputStream()
            : httpConnection.getErrorStream();
    Scanner scanner = new Scanner(responseStream).useDelimiter( pattern: "\\A");
    return scanner.hasNext() ? scanner.next() : "";
}

public CommentResponse getComment(Long id) { return CommentResponse.builder().build(); }
}
```

Рис. 3.4 Код методу для спілкування з PaLM API

3.3 Тестування програмного забезпечення

Тестування програмного забезпечення є невід'ємним і критично важливим етапом життєвого циклу розробки, оскільки воно спрямоване на виявлення дефектів, перевірку відповідності функціоналу вимогам та забезпечення стабільної і передбачуваної роботи системи. У сучасній програмній інженерії тестування розглядається не як завершальний етап, а як безперервний процес, інтегрований у всі стадії розробки (підхід shift-left testing). Це дозволяє виявляти помилки на ранніх етапах і значно зменшувати вартість їх виправлення.

Основною метою тестування є підтвердження коректності реалізації функціональних і нефункціональних вимог, включаючи продуктивність, безпеку, масштабованість та зручність використання. Завдяки тестуванню підвищується надійність програмного продукту та зменшується ризик виникнення критичних збоїв після його впровадження.

У сучасній практиці розробки програмного забезпечення виділяють ряд ключових принципів організації тестування:

– Визначення цілей і рівнів тестування. На кожному етапі життєвого циклу чітко визначаються завдання тестування, його обсяг та очікувані результати (unit, integration, system, acceptance).

– Максимальне покриття коду тестами (test coverage). Використовуються метрики покриття, що дозволяють оцінити якість тестового набору та виявити неперевірені ділянки коду.

– Повторне використання тестів. Створені тестові сценарії можуть застосовуватися повторно, що підвищує ефективність тестування та забезпечує стабільність перевірок.

– Використання реалістичних тестових даних. Тестування проводиться на даних, максимально наближених до реальних сценаріїв використання системи.

– Автоматизація тестування. Сучасні проєкти активно використовують автоматизовані тестові фреймворки та CI/CD-пайплайни для регулярного виконання тестів при кожній зміні коду.

– Моніторинг і аналіз дефектів. Виявлені помилки систематизуються, аналізуються та використовуються для покращення якості як коду, так і процесу розробки.

Сучасні підходи до тестування включають кілька основних рівнів:

– Модульне тестування (Unit Testing). Перевірка окремих компонентів або методів програми на коректність роботи.

– Інтеграційне тестування (Integration Testing). Перевірка взаємодії між модулями або сервісами системи.

– Системне тестування (System Testing). Оцінка роботи всієї системи як єдиного цілого відповідно до вимог.

– Приймальне тестування (Acceptance Testing). Перевірка відповідності програмного продукту вимогам замовника або кінцевих користувачів.

– Автоматизоване тестування. Використання спеціалізованих інструментів для автоматичного виконання тестів і перевірки результатів.

– Тестування безпеки (Security Testing). Аналіз системи на наявність вразливостей та перевірка захисту даних відповідно до сучасних стандартів кібербезпеки.

У межах розроблюваної системи передбачається комплексне використання різних рівнів тестування з акцентом на модульне, інтеграційне та системне тестування, а також активне застосування автоматизації. Такий підхід відповідає сучасним практикам DevOps та CI/CD, де тестування є безперервною частиною процесу розробки.

Модульне тестування забезпечує перевірку окремих компонентів системи ізольовано від інших частин, що дозволяє швидко виявляти помилки на рівні логіки. Системне тестування спрямоване на перевірку роботи всієї системи в цілому, включаючи взаємодію між компонентами та відповідність функціональним вимогам. Автоматизоване тестування дозволяє виконувати перевірки регулярно та без участі розробника, що значно підвищує швидкість та надійність процесу контролю якості.

На рисунку 3.5 наведено класичну піраміду тестування, яка відображає оптимальне співвідношення між різними рівнями тестів: найбільша кількість припадає на модульні тести, менша — на інтеграційні, і мінімальна — на системні та UI-тести. Такий підхід є стандартом у сучасній розробці програмного забезпечення.

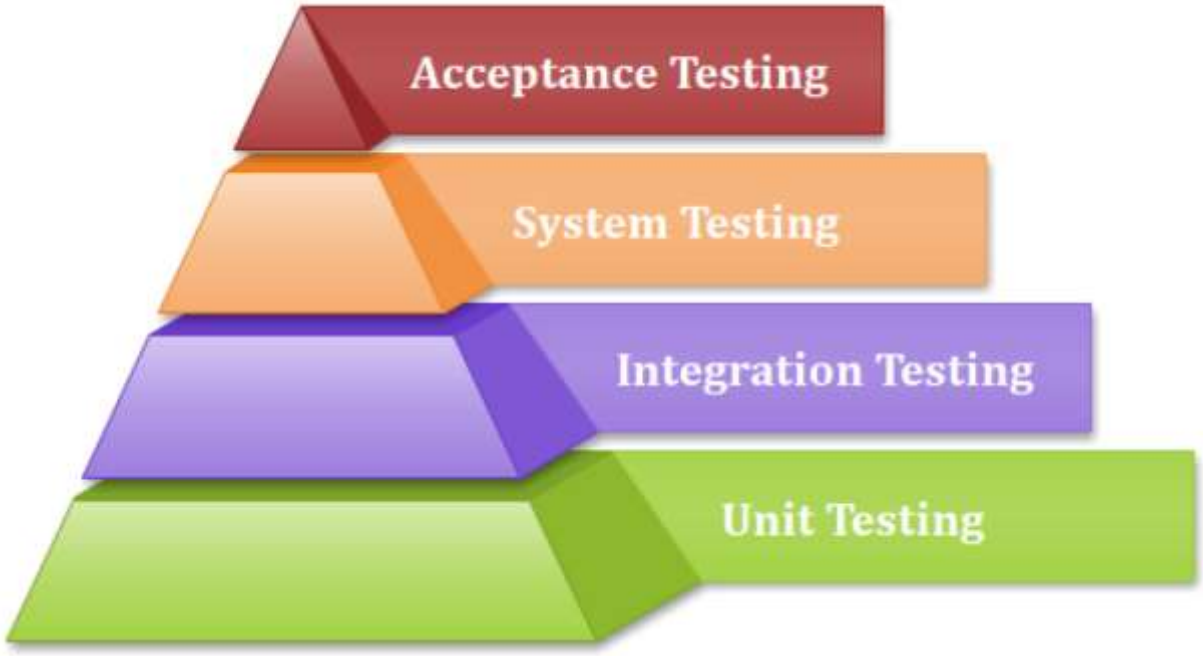


Рис. 3.5 Піраміда етапів тестування

Для реалізації модульного тестування у Java-проєктах використовується фреймворк JUnit, який є одним із найпоширеніших інструментів автоматизованого тестування в екосистемі Java та стандартом де-факто у сучасній розробці.

JUnit забезпечує:

– Анотаційний підхід. Використання анотацій, таких як @Test, @BeforeEach, @AfterEach, для організації життєвого циклу тестів.

– Механізм перевірок (assertions). Набір методів для перевірки очікуваних результатів, що дозволяє формалізувати процес тестування.

– Підтримку фіксації стану. Можливість налаштування середовища перед виконанням тестів і очищення після них.

– Групування тестів. Організацію тестів у набори (test suites) для комплексного запуску.

– Параметризовані тести. Використання різних наборів вхідних даних для одного тестового сценарію, що підвищує покриття та гнучкість перевірок.

У сучасній практиці JUnit активно використовується разом із інструментами CI/CD (Jenkins, GitHub Actions, GitLab CI), що дозволяє автоматично запускати тести при кожному оновленні коду. Це значно підвищує стабільність програмного продукту та зменшує ризик появи регресійних помилок.

Таким чином, комплексний підхід до тестування, який поєднує автоматизацію, багаторівневу структуру тестів та сучасні інструменти (зокрема JUnit), забезпечує високу якість, стабільність і надійність розроблюваної системи.

Приклад написання тестів та приклад успішного тестування наведено нижче на рисунках 3.6 та 3.7.

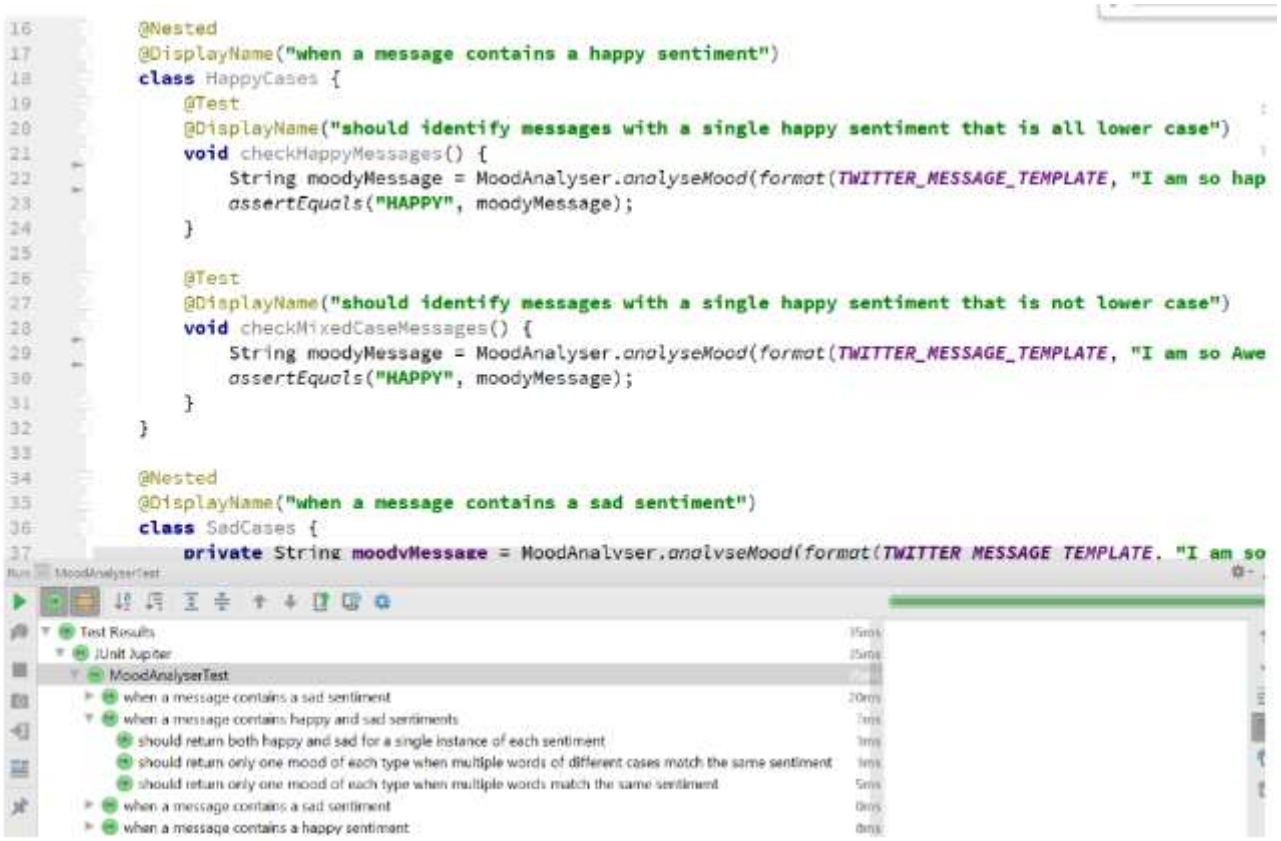


Рис. 3.6 Написання тестів

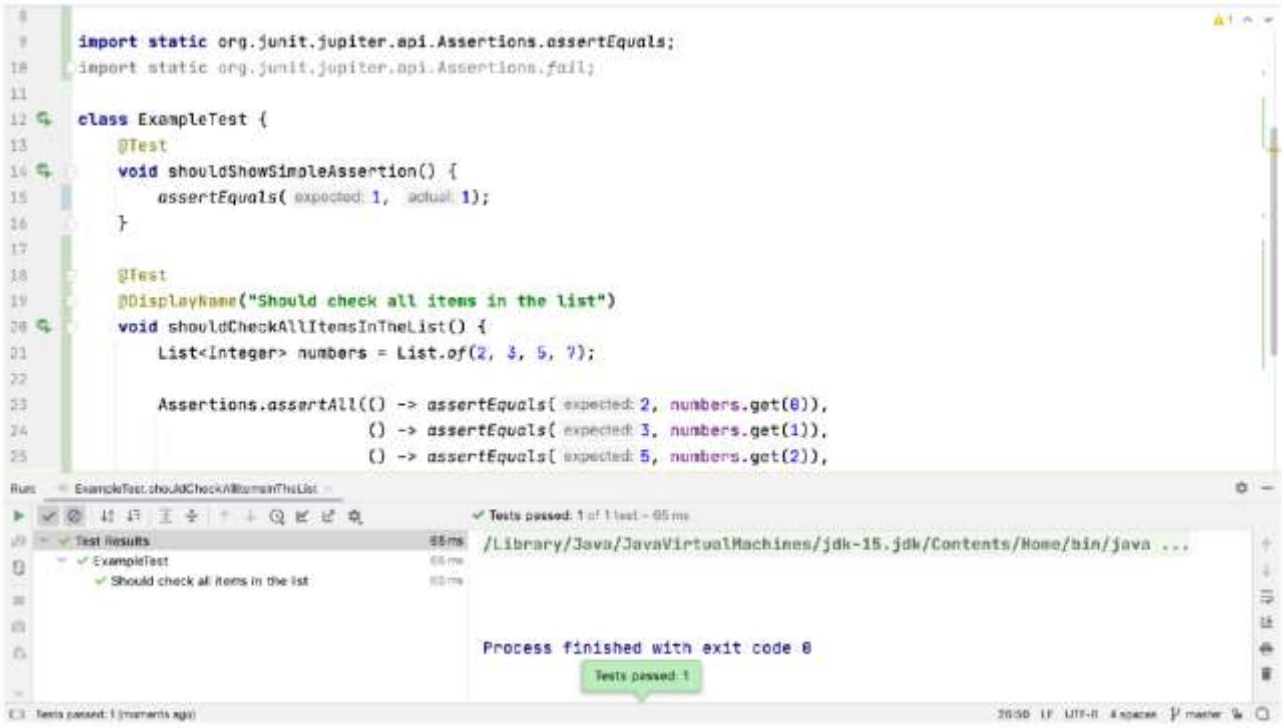


Рис. 3.7 Успішне виконання тестів

Порівняння роботи полягає в тому, що запропоноване програмне забезпечення для аналізу та пошуку коментарів, яка за рахунок використання мови програмування Java та таких фрейморків як Spring, Hibernate, а також мовній моделі PaLM, на відміну від існуючих засобів та інструментів, дозволяє розширити функціональні можливості програмного забезпечення, а саме надати кросплатформеність, суттєво спростити, розширити та зробити більш точним аналіз коментарів, а також надати поради користувачам як взаємодіяти з коментарями. Для цього було проведено порівняння розробленого програмного забезпечення з аналогічними засобами та інструментами. Результати порівняння наведено в таблиці 2.2.

2.2 Результати порівняння розробленої системи з аналогами

Фунціонал	Можливості браузера	Можливості OpenKM	Можливості Repustate	Розроблена система
Гнучкість налаштування	X	✓	✓	✓
Простота налаштування	✓	X	X	✓
Кросс-платформеність	✓	X	✓	✓
Підтримка читання великих обсягів даних	X	✓	✓	✓
Мовна модель в якості асистенту	X	X	X	✓
Робота з великим навантаженням	X	✓	✓	✓
Пошук серед конкретних коментарів	✓	✓	X	✓
Можливість аналізу кожного коментаря окремо	X	X	X	✓

Як можна побачити, мета роботи була досягнута через отримання підтвердження шляхом порівняння функціоналу розробленої системи з аналогами. Переважна кількість аналогів має неповний функціонал для вирішення задачі валідації подій.

ВИСНОВКИ

У результаті виконання даної кваліфікаційної роботи було розширено функціональні можливості існуючих підходів до індексації та обробки текстових даних, а також спроєктовано й реалізовано підсистему пошуку, збору та аналізу користувацьких коментарів. Розроблена система орієнтована на роботу з великими обсягами текстової інформації, має модульну архітектуру, проста у розгортанні та підтримує подальше розширення функціоналу без суттєвих змін у базовій структурі.

Запропоноване рішення реалізовано з використанням сучасного стеку технологій, що включає Java як основну мову програмування, Spring Framework для побудови серверної логіки, Hibernate для роботи з базою даних через ORM-підхід, Apache Lucene для індексації та повнотекстового пошуку, Apache OpenNLP для базової обробки природної мови, PostgreSQL як основну систему управління базами даних, а також Maven для автоматизації збірки та керування залежностями. Середовище розробки реалізовано на базі IntelliJ IDEA, яке забезпечує ефективну підтримку розробки, тестування та налагодження програмного коду.

У першому розділі роботи проведено аналіз сучасного стану проблеми, досліджено предметну область та сформульовано основні вимоги до системи. Також виконано огляд існуючих аналогів і рішень у сфері індексації та аналізу текстових даних, на основі чого визначено їхні переваги та недоліки, що вплинуло на формування архітектури запропонованої системи.

У другому розділі обґрунтовано вибір технологічного стеку, бібліотек та інструментів розробки. Особливу увагу приділено критеріям масштабованості, продуктивності, підтримки обробки текстових даних та інтеграції з сучасними вебтехнологіями. Представлено процес проєктування архітектури системи. Основний акцент зроблено на побудові модульної структури, яка забезпечує незалежність компонентів, легкість їх модифікації та можливість подальшого розширення функціоналу. Окрему увагу приділено забезпеченню

масштабованості системи, її стабільності та зручності взаємодії користувача з інтерфейсом. Також розглянуто використані архітектурні підходи, що дозволили реалізувати гнучку та підтримувану систему.

У третьому розділі описано процес розробки, тестування та забезпечення якості програмного продукту. Розглянуто основні принципи сучасного тестування, включаючи модульне, інтеграційне та системне тестування, а також використання автоматизованих інструментів перевірки. Проведене тестування підтвердило коректність роботи системи, її стабільність та здатність функціонувати в умовах підвищеного навантаження та непередбачуваних сценаріїв використання. Окремо розглянуто можливості подальшого вдосконалення системи шляхом розширення функціоналу, зокрема інтеграції більш сучасних моделей аналізу природної мови та алгоритмів машинного навчання.

Загалом у межах роботи було спроєктовано та реалізовано сучасну систему пошуку та аналізу користувацьких коментарів із використанням технологій Java, Spring, Hibernate, Apache Lucene, Apache OpenNLP, PostgreSQL та Maven. На відміну від існуючих аналогів, запропонована система поєднує в одному рішенні механізми індексації, повнотекстового пошуку та базового семантичного аналізу тексту, що забезпечує більш ефективну обробку користувацьких відгуків. Завдяки веб-орієнтованій архітектурі система може бути використана на різних платформах та легко інтегрується в існуючі інформаційні середовища.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Share of households with a computer at home worldwide from 2005 to 2019. Statista URL: https://www.statista.com/statistics/748551/worldwide-households-with-computer/

2. 53 Important Statistics About How Much Data Is Created Every Day. FinancesOnline URL: https://financesonline.com/how-much-data-is-created-every-day/

3. Format race 2018: DOCX over DOC and ODT? ONLYOFFICE URL: https://www.onlyoffice.com/blog/2018/09/format-race-2018-docx-over-doc-and-odt/

4. PDF statistics – the universe of electronic documents. PDF Association URL: https://www.pdfa.org/wp-content/uploads/2018/06/1330_Johnson.pdf

5. Document managment. OpenKM URL: https://www.openkm.com/en/document-management.html

6. Designing Your Document Managment Solution. SimpleIndex URL: https://www.simpleindex.com/features/document-management/

7. Full-text search of documents. LogicalDOC URL: https://www.logicaldoc.com/software-features/full-text-indexing

8. AI-powered Text Analytics API URL: https://www.repustate.com/text-analytics-api/

9. Історія компанії: як Rozetka «вмикалася» в лідери URL: https://new.finance.ua/ua/30-rokiv-nezalezhnosti/rozetka

10. The Amazon Marketplaces Worldwide in Focus URL: https://www.blankspace.eu/blog-posts-en/amazon-marketplaces-worldwide

11. Participatory Culture, Community, and Play: Learning from Reddit URL: https://www.researchgate.net/publication/284186822_Participatory_Culture_Community_and_Play_Learning_from_Reddit

12. Індексування. Вікіпедія URL: https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D0%B4%D0%B5%D0%BA%D1%81%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F

13. Пошуковий індекс. Вікіпедія URL: https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D1%88%D1%83%D0%BA%D0%BE%D0%B2%D0%B8%D0%B9_%D1%96%D0%BD%D0%B4%D0%B5%D0%BA%D1%81

14. Інвертований індекс. Вікіпедія URL: https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D0%B2%D0%B5%D1%80%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B9_%D1%96%D0%BD%D0%B4%D0%B5%D0%BA%D1%81

15. Tokenization in NLP: Types, Challenges, Examples, Tools URL: https://neptune.ai/blog/tokenization-in-nlp

16. Lemmatization and Stemming URL: https://medium.com/@alec.mccabe93/lemmatization-and-stemming-5b6b3718b49

17. Tokenizing, POS Tagging, Lemmatizing and Parsing UD 2.0 with UDPipe URL: https://www.researchgate.net/publication/318739283_Tokenizing_POS_Tagging_Lemmatizing_and_Parsing_UD_20_with_UDPipe

18. TIOBE Index for December 2023 URL: https://www.tiobe.com/tiobe-index/

19. Аналіз переваг та недоліків розробки програмного забезпечення за допомогою мови Java. Науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації URL: https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2022/paper/view/15429

20. Intellij IDEA. JetBrains URL: https://www.jetbrains.com/idea/

D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F – Режим доступу:

22. Мартін Р. Чиста архітектура – Фабула, 2019 – 368 с.

23. Порівняльний аналіз систем управління реляційними базами даних MySQL та PostgreSQL. Науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації: веб-сайт URL: https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2022/paper/view/15430 (дата звернення: 19.03.2023)

24. Lesson: JDBC Basics. Oracle Java Documentation URL: https://docs.oracle.com/javase/tutorial/jdbc/basics/index.html

25. The system of indexing documents and searching for their indexes stored in database written in Java using Gradle build automation tool. Молодь в науці: дослідження, проблеми, перспективи (МН-2022) URL: https://conferences.vntu.edu.ua/index.php/mn/mn2022/paper/view/13938

26. Review of build tool Gradle for Java. Молодь в науці: дослідження, проблеми, перспективи (МН-2021) URL: https://conferences.vntu.edu.ua/index.php/allvntu/mn2021/paper/view/13095

27. Mark Richards, Neal Ford Fundamentals of Software Architecture: An Engineering Approach 1st Edition. O'Reilly Media, 2020. 419 p.

28. Модель-вид-контролер. Вікіпедія URL: https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C-%D0%B2%D0%B8%D0%B4-%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D0%B5%D1%80

29 Gamma E., Helm R., Johnson R., Vlissides J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

30 Fowler M. *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2002.

31 Fielding R. T. *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000.

32 Oracle. *The Java Tutorials*. URL: https://docs.oracle.com/javase/tutorial/

33 Oracle. *Java SE Documentation*. URL: https://docs.oracle.com/en/java/

34 Spring Framework Documentation. URL: https://docs.spring.io/spring-framework/docs/current/reference/html/

35 Spring Boot Reference Guide. URL: https://docs.spring.io/spring-boot/docs/current/reference/html/

36 Hibernate ORM Documentation. URL: https://hibernate.org/orm/documentation/

37 Apache Software Foundation. *Apache Lucene Documentation*. URL: https://lucene.apache.org/core/

38 Apache OpenNLP Documentation. URL: https://opennlp.apache.org/

39 PostgreSQL Global Development Group. *PostgreSQL Documentation*. URL: https://www.postgresql.org/docs/

40 Silberschatz A., Korth H. F., Sudarshan S. *Database System Concepts*. McGraw-Hill, 2019.

41 Elmasri R., Navathe S. B. *Fundamentals of Database Systems*. Pearson, 2016.

42 Martin R. C. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall, 2017.

43 Martin R. C. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall, 2008.

44 Sommerville I. *Software Engineering*. Pearson, 2016.

45 Pressman R. S., Maxim B. R. *Software Engineering: A Practitioner's Approach*. McGraw-Hill, 2019.

46 Newman S. *Building Microservices*. O'Reilly Media, 2021.

47 Richardson L., Ruby S. *RESTful Web APIs*. O'Reilly Media, 2013.