МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет кораблебудування імені адмірала Макарова

Навчально-науковий інститут комп'ютерних наук та управління проєктами

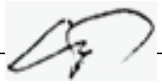
Кафедра програмного забезпечення автоматизованих систем

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма «Інженерія програмного забезпечення»

«Допущений до захисту»

Завідувач кафедри

_  _ проф. Приходько С.Б.

«___» _____ 2026 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти «бакалавр»

на тему: ***Розробка програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK***

Виконав: студент групи 4151

 Давидов А.І.

(підпис, ПІБ)

Керівник роботи:

Доцент каф. ПЗАС, к.т.н., доц.____

(посада, науковий ступінь вчене звання)

 Пухалевич А.В.

(підпис, ПІБ)

Миколаїв – 2026 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет кораблебудування імені адмірала Макарова

Навчально-науковий інститут комп'ютерних наук та управління проєктами
Кафедра програмного забезпечення автоматизованих систем
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма «Інженерія програмного забезпечення»

«ЗАТВЕРДЖУЮ»
Гарант освітньої програми
_____ доц. Макарова Л.М.
(підпис)
«_07_» ___04___ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ
на здобуття ступеня вищої освіти «бакалавр»

Студенту Давидову Артему Ігоровичу _____
(Прізвище, ім'я, по батькові)

1. Тема роботи: Розробка програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK

Керівник роботи Пухалевич Андрій Володимирович

Затверджені наказом ректора №275-уч від 07.04.2026 р.

2. Термін подання роботи: 01.06.2026 р. _____

3. Вихідні дані по роботі: _____

_____

4. Перелік питань, що належать до розробки (найменування розділів)_____

Титульний аркуш; Завдання на кваліфікаційну роботу; Анотація (українською та англійською мовами); Зміст; Перелік умовних позначень, символів, одиниць та термінів (при необхідності); Вступ; Аналіз практичної задачі інженерії програмного забезпечення; Проєкт програмного забезпечення; Результати розробки програмного забезпечення; Розділ з охорони праці; Висновки; Список використаних джерел; Додатки (технічне завдання, текст програми, опис програми, інструкція користувача, програма та методика випробувань програмного забезпечення).

5. Перелік презентаційних матеріалів 1. Титульний слайд; 2. Мета та завдання кваліфікаційної роботи; 3. Аналіз вимог до програмного забезпечення; 4. Архітектура програмного забезпечення; 5. Діаграма варіантів використання; 6. Діаграма діяльності; 7. Діаграма класів; 8. Діаграма послідовності; 9. Логічна модель даних; 10. Фізична модель даних; 11. Результати розробки; 12. Висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
4	Гурець Н.В., ст. викладач		

7. Дата видачі завдання 07.04.2026 р.

К А Л Е Н Д А Р Н И Й П Л А Н

Номер	Назва етапів роботи	Терміни виконання	Примітка
1.	Підготовка розділу ВСТУП	30.03.2026	ПП*
2.	Аналіз практичної задачі інженерії ПЗ	06.04.2026	ПП*
3.	Аналіз вимог до ПЗ	13.04.2026	ПП*
4.	Розробка архітектури ПЗ	20.04.2026	
5.	Розробка проєкту ПЗ	04.05.2026	
6.	Реалізація та тестування ПЗ	11.05.2026	
7.	Підготовка розділу Результати розробки ПЗ	15.05.2026	
8.	Підготовка розділу з охорони праці	18.05.2026	ПП*
9.	Підготовка розділу ВИСНОВКИ	22.05.2026	
10.	Оформлення списку використаних джерел та додатків	25.05.2026	
11.	Подання на кафедру ПЗАС тексту остаточного варіанту роботи, підписаного її керівником, у роздрукованому та електронному форматі разом із заявами щодо самостійності виконання роботи та ідентичності друкованої та електронної версій роботи (Додатки 1 і 2 «Порядку здійснення заходів з перевірки робіт на наявність текстових збігів/ідентичності/схожості із використанням програмно-технічних засобів», який введений в дію наказом ректора НУК за №20 від 20.01.2020 р.)	01.06.2026	не пізніше, ніж за 14 діб до захисту (згідно п.4.1 зазначеного Порядку
12.	Підготовка презентації та доповіді	05.06.2026	
13.	Попередній захист роботи на засіданні кафедри ПЗАС	08.06.2026	
14.	Подання на кафедру ПЗАС електронних версій наступних документів у форматі pdf: кваліфікаційної роботи; файлу-опису кваліфікаційної роботи (згідно Додатку до наказу ректора НУК за №287-уч від 19.05.2020 р.); презентації доповіді, а також Авторського договору з додатком	15.06.2026	

* - за результатами переддипломної практики (ПП), яка була з 02.02.2026 до 22.03.2026 р.

Студент _____ Давидов А.І.
(підпис) (ПІБ)

Керівник роботи _____ Пухалевич А.В.
(підпис) (ПІБ)

АНОТАЦІЯ

Кваліфікаційна робота присвячена вирішенню практичної задачі інженерії програмного забезпечення, яка характеризується комплексністю та невизначеністю умов, що полягає у розробці програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK.

У роботі розглянуто особливості використання GitHub-репозиторіїв як джерела програмного коду для подальшого аналізу, визначено можливості утиліти CK для статичного обчислення метрик Java-проєктів, а також обґрунтовано доцільність створення власного програмного забезпечення для автоматизації цього процесу. Виконано аналіз наявних програмних рішень і сформульовано постановку задачі на розробку десктопного ПЗ для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK. Проведено аналіз вимог до програмного забезпечення, розроблено архітектурні рішення та виконано проєкт програмного забезпечення. Реалізовано тестування й випробування розробленого програмного забезпечення та описано необхідну документацію для програми.

Кваліфікаційна робота викладена на 144 сторінках друкованого тексту, містить 23 рисунки, 2 таблиці, список використаних джерел із 15 найменувань та 5 додатків.

ABSTRACT

This qualification work focuses on solving a practical software engineering problem, which involves developing software to automate the collection of GitHub metrics for Java projects using the CK utility, a task characterized by complexity and uncertain conditions.

This work examines the features of using GitHub repositories as a source of source code for further analysis, identifies the capabilities of the CK utility for the static calculation of Java project metrics and provides justification for developing custom software to automate this process. The existing software solutions were analyzed and the problem statement for developing desktop software to automate the collection of GitHub metrics for Java projects using the CK utility was formulated. The software requirements were analyzed, architectural solutions were developed and the software project was implemented. Testing and trials of the developed software were performed, and the necessary documentation for the program was described.

The qualification work consists of 144 pages, 23 figures, 2 tables, 15 references, and 5 appendices.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

БД – база даних, яка потрібна для організованого зберігання та опрацювання даних.

ПЗ – програмне забезпечення, тобто сукупність програм для виконання задач.

ПК – персональний комп’ютер, тобто пристрій для запуску та роботи з ПЗ.

API – набір правил і засобів, за допомогою яких одна програмна складова може взаємодіяти з іншою.

CK – набір метрик Чидамбера й Кемерера для оцінки об’єктно-орієнтованого коду, а також назва утиліти для статичного аналізу Java-коду та збору метрик.

CRUD – набір базових операцій над даними таких як створення, читання, оновлення та вилучення.

CSV – текстовий формат подання табличних даних, у якому значення зазвичай розділяються комами або іншими розділювачами.

DAO – підхід до організації доступу до даних, згідно якого операції з базою даних виносяться в окремі класи.

JPA – специфікація Java, що описує правила збереження об’єктів програми у реляційній базі даних.

JVM – середовище виконання Java-програм, яке забезпечує запуск байт-коду на різних операційних системах.

MVC – архітектурний підхід, згідно якого дані, інтерфейс користувача та логіка керування розділяються між окремими частинами програми.

ORM – підхід до роботи з базою даних, згідно якого об’єкти програми зіставляються з таблицями реляційної бази даних.

SQL – мова для створення, зміни, вибірки та опрацювання даних у реляційних базах даних.

UI – частина програмного забезпечення, через яку користувач взаємодіє із застосунком.

URL – адреса ресурсу в мережі Інтернет, яка використовується для доступу до веб-сторінок або репозиторіїв.

ВСТУП

Ця кваліфікаційна робота присвячена розв’язанню практичної задачі інженерії програмного забезпечення, що характеризується комплексністю та невизначеністю умов, а саме розробці десктопного ПЗ для автоматизації аналізу Java-репозиторіїв, тобто основним завданням є створення інструменту, що спрощує збір метрик коду через інтеграцію з GitHub та автоматизовану роботу з утилітою CK [1, 2].

Традиційний підхід до аналізу Java-проєктів без засобів автоматизації є малоефективним. Він вимагає від користувача послідовного виконання рутинних кроків: отримання вихідного коду через Git-клієнт, запуск розрізнених скриптів для отримання метрик, ручне копіювання даних. Процес обчислення середньостатистичних показників для кожного проєкту ще більше сповільнює роботу. Розроблене програмне рішення оптимізує послідовність виконання операцій шляхом автоматизації кожного кроку, ініціюючи процес від етапу клонування репозиторію за певним тегом до генерації структурованого звіту у форматі XLSX з агрегованими значеннями метрик, виключаючи необхідність ручного втручання на проміжних етапах.

Метою кваліфікаційної роботи є розробка програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- виконати аналіз практичної задачі інженерії програмного забезпечення щодо розробки програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK.

- проаналізувати існуючі програмні рішення та обґрунтувати доцільність розробки власного програмного забезпечення.

- сформулювати постановку задачі на розробку програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK.

- провести аналіз вимог до програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK.

- розробити архітектурні рішення програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK.

- зробити проєкт програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK.

- реалізувати тестування та випробування розробленого програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK.

- описати всю необхідну документацію для програми.

Об’єктом кваліфікаційної роботи є процес розробки програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK.

1 АНАЛІЗ ПРАКТИЧНОЇ ЗАДАЧІ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ПОСТАНОВКА ЗАДАЧІ НА РОЗРОБКУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОМАТИЗАЦІЇ ЗБОРУ МЕТРИК GITHUB ПРОЄКТІВ НА МОВІ JAVA УТИЛІТОЮ CK

1.1 Аналіз практичної задачі інженерії програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK

У наш час мова програмування Java й надалі залишається однією з провідних платформ розробки програмного забезпечення та основою для створення масштабних корпоративних систем, які можуть обробляти значні обсяги даних і велику кількість транзакцій.

Протягом тривалого часу Java еволюціонувала від платформи для корпоративних застосунків до універсального середовища, яке використовується для розробки серверних, десктопних, мобільних, фінансових та аналітичних застосунків, тобто її поширення обумовлене поєднанням таких переваг як можливість побудови систем великого масштабу, кросплатформеність, наявність потужної екосистеми та постійний розвиток самої мови [1].

Через широке використання Java у складних програмних системах виникає потреба в регулярному аналізі їх програмного коду. Для цього доцільно застосовувати статичний аналіз, бо він надає змогу досліджувати внутрішні характеристики ПЗ без виконання програми. Одним із таких засобів є утиліта CK, яка призначена для обчислення метрик Java-коду засобами статичного аналізу [2]. Тому задача автоматизації збору метрик для Java-проєктів має практичну цінність для інженерії програмного забезпечення.

Існуючий підхід до аналізу Java-репозиторіїв з GitHub є надто повільним і вимагає постійного втручання з боку користувача. Послідовність дій цього підходу починається від ручного завантаження користувачем потрібних репозиторіїв на

локальний пристрій і запуску утиліти CK для цих репозиторіїв та закінчується копіюванням потрібних даних у вихідний результуючий файл із метриками.

Такий підхід ускладнює роботу користувача, особливо якщо потрібно аналізувати не один, а багато репозиторіїв. Клонування є базовою операцією, що створює локальну копію віддаленого репозиторію [3], однак при ручному виконанні кожного етапу збільшуються часові витрати та кількість помилок під час підготовки вхідних даних і формування вихідних результатів. Тому відсутність єдиного автоматизованого процесу ускладнює повторне проведення аналізу та порівняння отриманих показників.

Актуальність даної теми визначається тим, що статичний аналіз дозволяє ще на ранніх стадіях розробки оцінити якість коду, полегшити його подальшу підтримку та своєчасно виявити потенційні структурні дефекти й вразливості, тобто це забезпечує створення більш надійних і безпечних програм із мінімальними витратами на подальше налагодження. Оскільки системне застосування таких методів є фундаментальною умовою розробки високоякісного ПЗ, задача автоматизованого збору статичних метрик набуває значної практичної цінності [4].

У межах цієї практичної задачі інженерії програмного забезпечення необхідно не лише отримати метрики за допомогою CK, а також організувати повний процес їх виокремлення. Це означає, що ПЗ повинно забезпечувати роботу з GitHub-репозиторіями як джерелом початкових даних, підготовку локальних копій проєктів, запуск утиліти аналізу та збереження результатів у зручному вигляді для подальшого опрацювання. Тобто, мова не про окрему дію, а про створення цілісного застосунку, який автоматизує послідовність повторюваних операцій зі збору метрик.

З цього випливає, що розробка програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK спростить процес аналізу програмних репозиторіїв, зменшить кількість операцій зі сторони користувача, скоротить ймовірність помилок під час збирання результатів, покращить швидкість обробки проєктів та забезпечить зручне формування результатів.

1.2 Аналіз існуючих програмних рішень

«JArchitect» це перший досліджуваний desktop-застосунок, який призначений для вимірювання, розуміння та вдосконалення якості Java коду. Він орієнтований на статичний аналіз Java-проєктів та JVM-проєктів, побудову залежностей, отримання архітектурних звітів, аналіз технічних складових та перегляд метрик.

До переваг «JArchitect» належать розвинені можливості візуалізації за допомогою графіків, засоби пошуку проблем у структурі програмного коду та підтримка аналітичних сценаріїв для великих систем [5].

До недоліків «JArchitect» можна віднести відсутність відкритого коду програми й надлишкову для поставленої задачі функціональність, а також наявність ліцензійних обмежень та відсутність підтримки GitHub-репозиторіїв.

Графічний вигляд інформаційної панелі ПЗ «JArchitect» з даними про метрики проєкту представлено на рисунку 1.1.

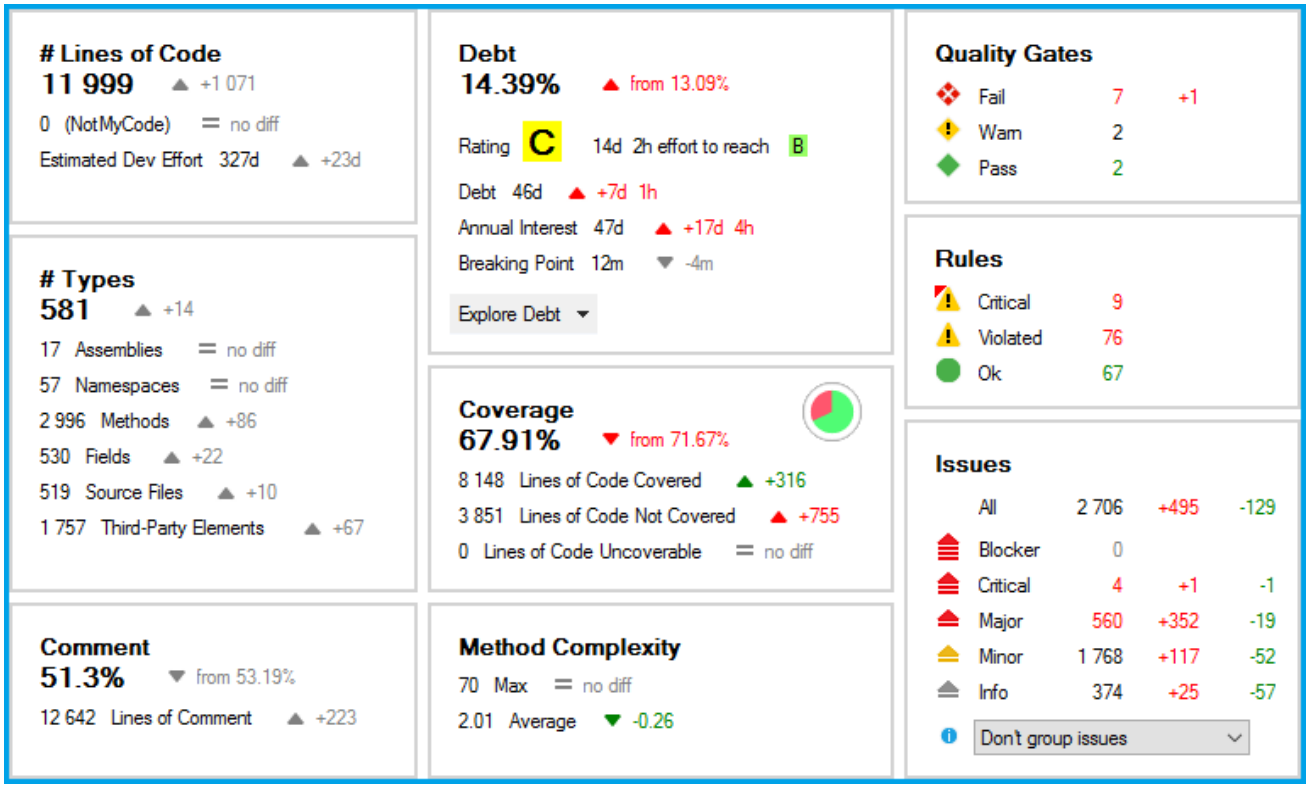


Рисунок 1.1 – Графічний вигляд інформаційної панелі ПЗ «JArchitect» з даними про метрики проєкту

Приклад графіку тенденцій ПЗ «JArchitect» з різними показниками щодо метрик проєкту представлено на рисунку 1.2.

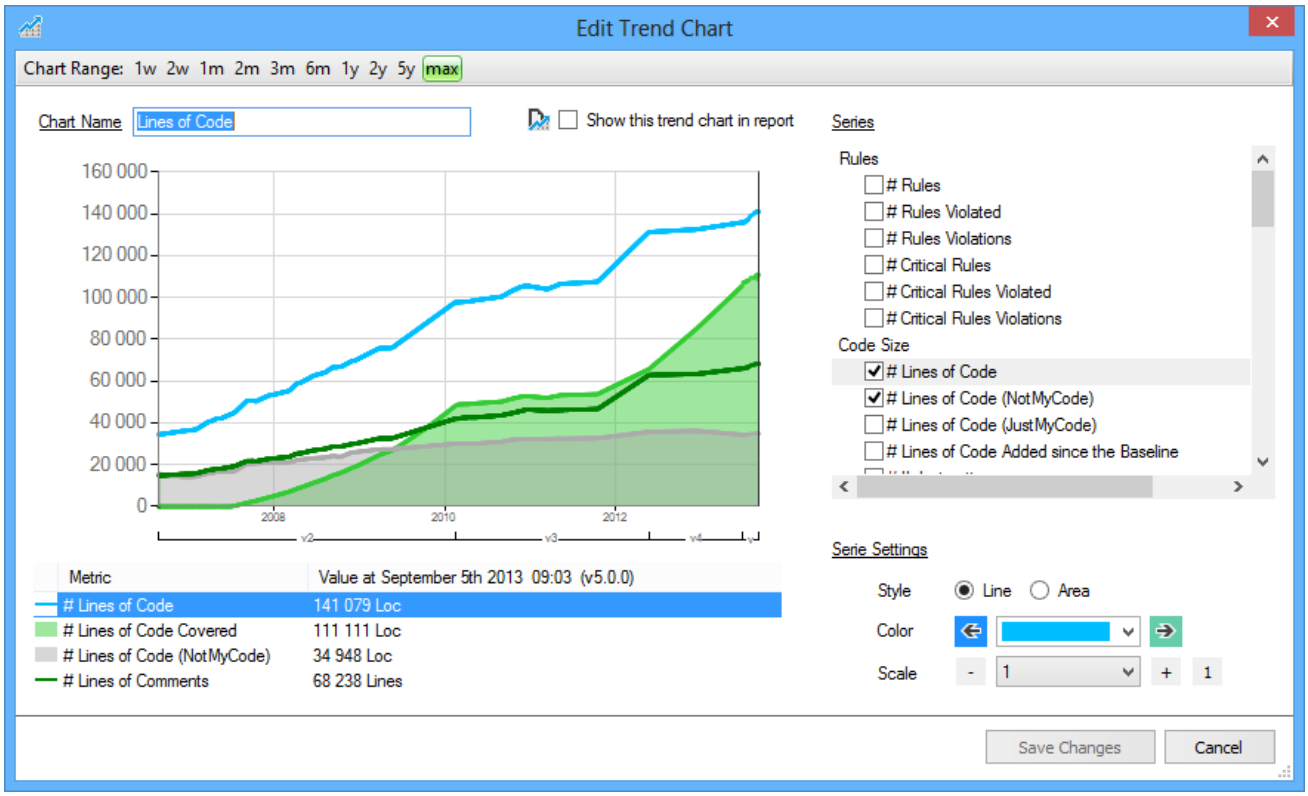


Рисунок 1.2 – Приклад графіку тенденцій ПЗ «JArchitect» з різними показниками щодо метрик проєкту

Другим проаналізованим ПЗ є застосунок «Understand» від компанії SciTools, головне завдання якого це статичний аналіз коду програм. Цей інструмент підтримує аналіз програмних систем, навігацію по їх кодовій базі, виявлення залежностей, побудову графів і розрахунок метрик [6].

До переваг «Understand» належать масштабованість, підтримка декількох мов програмування таких як Java, Python, C++, C#, Fortran, а також наявність API та орієнтація на професійне використання.

До недоліків «Understand» можна віднести те, що це пропрієтарне програмне забезпечення для комплексного дослідження програмного коду, а не для конкретної задачі з статичного аналізу GitHub-репозиторіїв. Також це ПЗ поширюється виключно через ліцензію, ціна якої досить недешева порівняно із тим, що

розроблюється у нашій задачі, де застосунок буде з відкритим вихідним кодом. Тому цей інструмент також не повністю відповідає предмету розробки і є складнішим, ніж той що потрібен для поставленої задачі.

Графічний вигляд вікна-оглядача ПЗ «Understand» для зручного подання метрик проєкту представлено на рисунку 1.3.

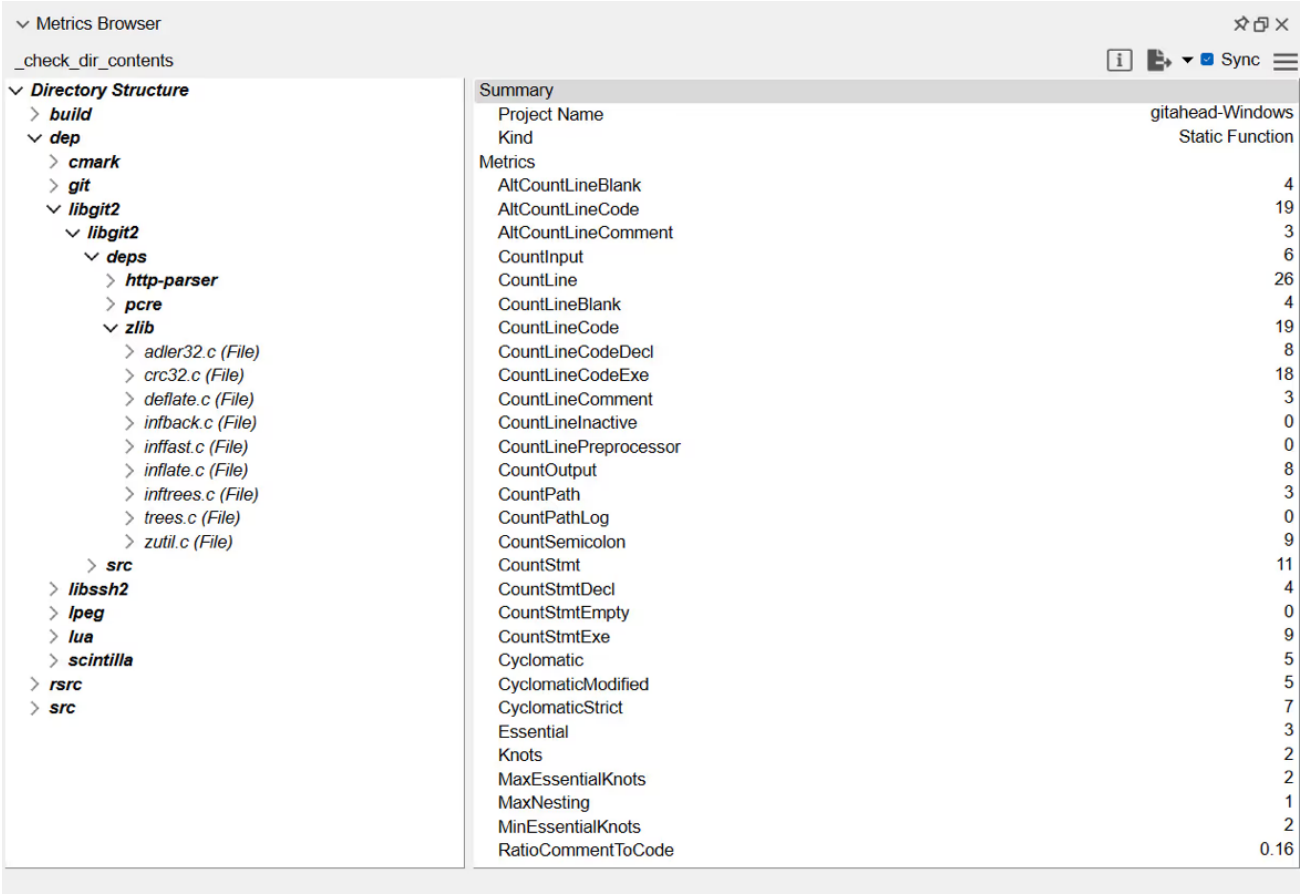


Рисунок 1.3 – Графічний вигляд вікна-оглядача ПЗ «Understand» для зручного подання метрик проєкту

Третім розглянутим ПЗ є «OpenClover» від компанії Atlassian. Це програмне забезпечення орієнтоване переважно на аналіз покриття коду тестами, а також на отримання деяких додаткових метрик та формування звітів.

До переваг «OpenClover» належать інтеграція з інструментами збирання та автоматизації, підтримка популярних середовищ розробки і наявність історії звітів.

До недоліків «OpenClover» можна віднести надмірний фокус на вимірюванні test coverage, аніж на автоматизації статичного збору метрик з репозиторії на GitHub

та агрегації результатів у підсумкових вихідний файл, тобто його функціональність лише частково перетинається з нашою поставленою задачею [7].

Графічний вигляд довідкової вкладки ПЗ «OpenClover» з інформацією про метрики проєкту представлено на рисунку 1.4.

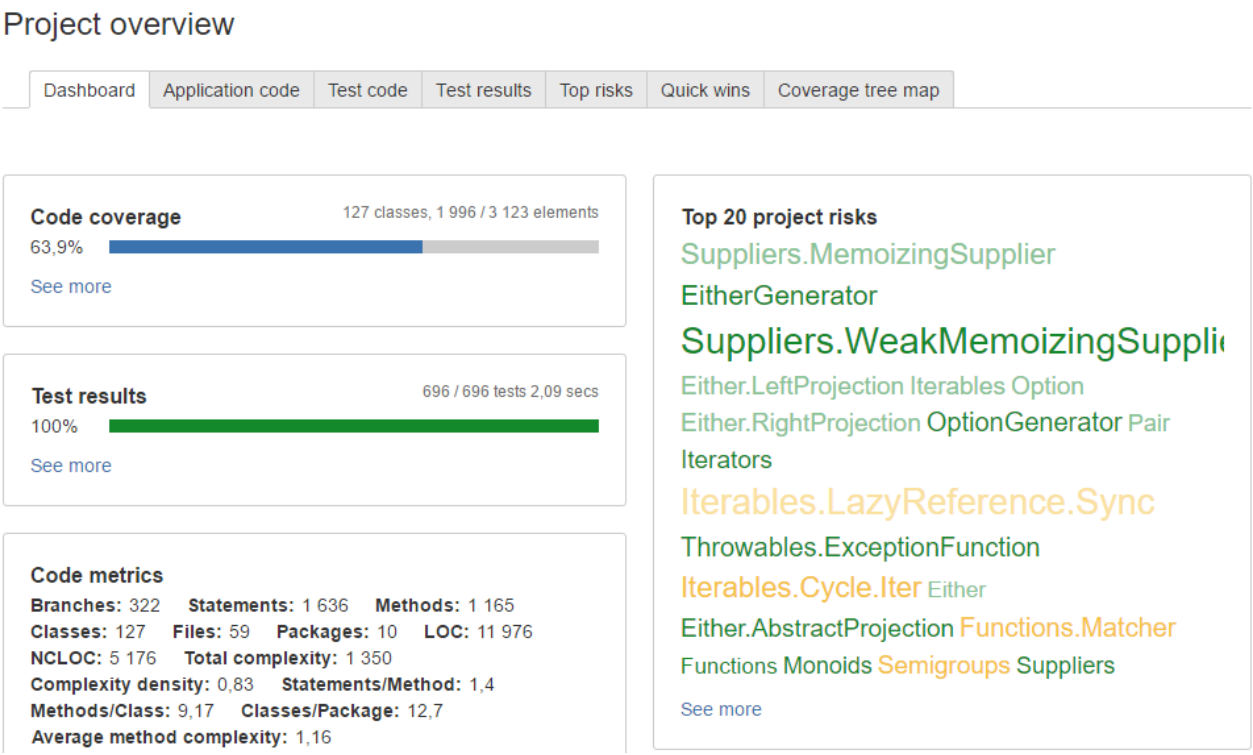


Рисунок 1.4 – Графічний вигляд довідкової вкладки ПЗ «OpenClover» з інформацією про метрики проєкту

Четвертим розглянутим ПЗ є утиліта «JOSSeph». Цей інструмент призначений для статичного аналізу програмного коду, дослідження внутрішньої структури проєктів та автоматизації процесів перевірки архітектурних елементів застосунків [8].

До переваг «JOSSeph» можна віднести відкритий вихідний код, підтримку роботи з GitHub-репозиторіями, використання метрик CK та можливість обробки великої кількості Java-проєктів, а також дане ПЗ орієнтоване на повторюваний збір даних і може використовуватися в дослідницьких задачах, пов’язаних з аналізом якості програмного коду.

До недоліків «JOSSeph» можна віднести те, що це не desktop-застосунок із графічним інтерфейсом, а саме засіб, який потребує запуску в контейнерному середовищі та попереднього налаштування через файл. Також у цій утиліті відсутній функціонал для фільтрування каталогів, які не потрібно аналізувати під час збору метрик.

Проведений аналіз існуючих програмних рішень дозволяє зробити висновок, що всі вони мають такі спільні недоліки відповідно до теми роботи:

- вони орієнтовані на ширші задачі аналізу коду, ніж автоматизація збору метрик GitHub-проєктів утилітою CK.

- частина з них має надлишковий функціонал, який не є критично необхідним для розв’язання поставленої задачі.

- існуючі рішення не забезпечують у готовому вигляді всі необхідні функції для нашого застосунку.

- окремі програмні рішення є комерційними, що знижує їх доступність.

Наведена вище інформація обґрунтовує доцільність розробки власного програмного забезпечення, бо це дозволить реалізувати необхідний функціонал, уникнути надлишковості та забезпечити зручність використання в межах кваліфікаційної роботи.

1.3 Постановка задачі на розробку програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK

Згідно з аналізом практичної задачі інженерії програмного забезпечення та результатами дослідження існуючих аналогів, для розв’язання поставленої задачі необхідно розробити програмне забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK.

Вимоги до складу виконуваних функцій

Програмне забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK повинно забезпечити можливість виконання таких функцій як:

- Можливість додавання, корегування, вилучення інформації про тип проєкту та відображення інформації про типи проєктів, тобто наявність дій CRUD для даної сутності.

- Можливість додавання, корегування, вилучення інформації про проєкт та відображення інформації про проєкти, тобто наявність дій CRUD для даної сутності.

- Можливість додавання, корегування, вилучення інформації про виключений каталог та відображення інформації про виключені каталоги, тобто наявність дій CRUD для даної сутності.

- Формування метрик.

Вимоги до організації вхідних та вихідних даних

- Для додавання інформації про тип проєкту:

Вхідні дані це назва типу проєкту.

Вихідні дані це новий запис про тип проєкту у таблиці БД «Типи» та системне повідомлення про успішне додавання.

- Для корегування інформації про тип проєкту:

Вхідні дані це обраний рядок з таблиці «Типи» і заповнені нові дані щодо його назви.

Вихідні дані це оновлений запис про тип проєкту у таблиці БД «Типи» зі зміненою інформацією та системне повідомлення про успішне корегування.

- Для вилучення інформації про тип проєкту:

Вхідні дані це обраний рядок з таблиці «Типи» і підтвердження дії в діалоговому вікні.

Вихідні дані це вилучений запис про тип проєкту у таблиці БД «Типи» та системне повідомлення про успішне вилучення, якщо даний тип проєкту ще не належить жодному проєкту.

- Для відображення інформації про типи проєктів:

Вхідні дані це відкрита вкладка «Типи».

Вихідні дані це відображення таблиці зі списком типів проєктів та заповненими даними.

- Для додавання інформації про проєкт:

Вхідні дані це посилання проєкту та вибір типу проєкту зі списку або його відсутність.

Вихідні дані це новий запис про проєкт у таблицях БД «Проєкти» і «Типи проєктів» та системне повідомлення про успішне додавання.

- Для корегування інформації про проєкт:

Вхідні дані це обраний рядок з таблиці «Проєкти» і заповнені нові дані щодо його посилання та типу проєкту або його відсутності.

Вихідні дані це оновлений запис про проєкт у таблицях БД «Проєкти» і «Типи проєктів» та системне повідомлення про успішне корегування.

- Для вилучення інформації про проєкт:

Вхідні дані це обраний рядок з таблиці «Проєкти» і підтвердження дії в діалоговому вікні.

Вихідні дані це вилучений запис про проєкт у таблицях БД «Проєкти» і «Типи проєктів» та системне повідомлення про успішне вилучення.

- Для відображення інформації про проєкти:

Вхідні дані це відкрита вкладка «Проєкти».

Вихідні дані це відображення таблиці зі списком проєктів та заповненими даними.

- Для додавання інформації про виключений каталог:

Вхідні дані це назва виключеного каталогу.

Вихідні дані це новий запис про виключений каталог у таблиці БД «Виключені каталоги» та системне повідомлення про успішне додавання.

- Для корегування інформації про виключений каталог:

Вхідні дані це обраний рядок з таблиці «Виключені каталоги» і заповнені нові дані щодо його назви.

Вихідні дані це оновлений запис про виключений каталог у таблиці БД «Виключені каталоги» зі зміненою інформацією та системне повідомлення про успішне корегування.

- Для вилучення інформації про виключений каталог:

Вхідні дані це обраний рядок з таблиці «Виключені каталоги» і підтвердження дії в діалоговому вікні.

Вихідні дані це вилучений запис про виключений каталог у таблиці БД «Виключені каталоги» та системне повідомлення про успішне вилучення.

- Для відображення інформації про виключені каталоги:

Вхідні дані це відкрита вкладка «Виключені каталоги».

Вихідні дані це відображення таблиці зі списком виключених каталогів та заповненими даними.

- Для формування метрик:

Вхідні дані це обраний тип проєкту, з яким пов’язано хоча б один проєкт, задані необхідні виключені каталоги або їх відсутність, визначена гілка коду, вказаний шлях до директорії для збереження вихідного файлу з агрегованими даними.

Вихідні дані це згенерований файл метрик формату XLSX, що містить у перших трьох стовпцях No проєкту, назву і вендора Project та його Commit Hash, а потім стовпці із такими метриками як NC, cbo, cboModified, fanin, fanout, wmc, dit, noc, rfc, lcom, totalMethodsQty, staticMethodsQty, publicMethodsQty, privateMethodsQty, protectedMethodsQty, defaultMethodsQty, visibleMethodsQty, abstractMethodsQty, finalMethodsQty, synchronizedMethodsQty, totalFieldsQty, staticFieldsQty, publicFieldsQty, privateFieldsQty, protectedFieldsQty, defaultFieldsQty, finalFieldsQty, synchronizedFieldsQty, nosi, loc, returnQty, loopQty, comparisonsQty, tryCatchQty, parenthesizedExpsQty, stringLiteralsQty, numbersQty, assignmentsQty, mathOperationsQty, variablesQty, maxNestedBlocksQty, anonymousClassesQty,

innerClassesQty, lambdasQty, uniqueWordsQty, logStatementsQty, у вказаній директорії.

Вимоги до складу та параметрів технічних засобів

Операційна система: macOS, Linux та Windows.

Процесор: 4 ядра або більше із частотою щонайменше 3 ГГц.

Оперативна пам’ять: 4 ГБ.

Жорсткий диск: не менше ніж 1 ГБ вільного простору для тимчасового збереження клонованих репозиторіїв.

Інтернет: мінімум 5 Мбіт/с для стабільної роботи.

Отже, розроблене ПЗ повинно автоматизувати збір метрик від завантаження коду до формування вихідного файлу, забезпечуючи зручну взаємодію користувача із системою та аналіз значної кількості Java-репозиторіїв із GitHub.

Детальна постановка задачі виконана у технічному завданні, яке наведено у додатку А.

2 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОМАТИЗАЦІЇ ЗБОРУ МЕТРИК GITHUB ПРОЄКТІВ НА МОВІ JAVA УТИЛІТОЮ CK

2.1 Аналіз вимог до програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK

При розробці якісного ПЗ потрібно не тільки створити працездатний кінцевий варіант деякого застосунку, а також і заздалегідь продумати та спроєктувати його архітектуру й логіку взаємодії з користувачем. Через це одразу після етапу постановки задачі доцільно перейти далі до етапу планування, на якому вже отримується повноцінне бачення майбутнього проєкту, визначаються його ключові компоненти і зв’язки між ними та описуються специфікації варіантів використання, бо саме такий підхід утворює надійне підґрунтя для реалізації застосунку в подальшому, а також для його тестування та покращення.

Однією з практичних дій для цього етапу можна виділити процес побудови діаграми варіантів використання та наведення специфікацій тих варіантів використання, що на ній відображено. Наприклад, варіант використання використовується для опису послідовності дій, відповідно до якої актор взаємодіє із системою для досягнення певної конкретної мети, а вже специфікація варіанту використання являє собою текстовий опис взаємодії актора із системою, який складається з таких елементів як ім’я варіанту використання, його призначення, учасники, передумови, стартова точка, процедура, результати, умови після завершення використання.

В даному ПЗ передбачено наявність лише одного актора такого як користувач, що має доступ до всіх існуючих варіантів використання таких як додавання/корегування/вилучення інформації про тип проєкту, проєкт і виключений каталог та відображення інформації про типи проєктів/проєкти/виключені каталоги, а також формування метрик.

Діаграма варіантів використання це графічне зображення відношень між акторами та варіантами використання.

Діаграма варіантів використання програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK представлена на рисунку 2.1.

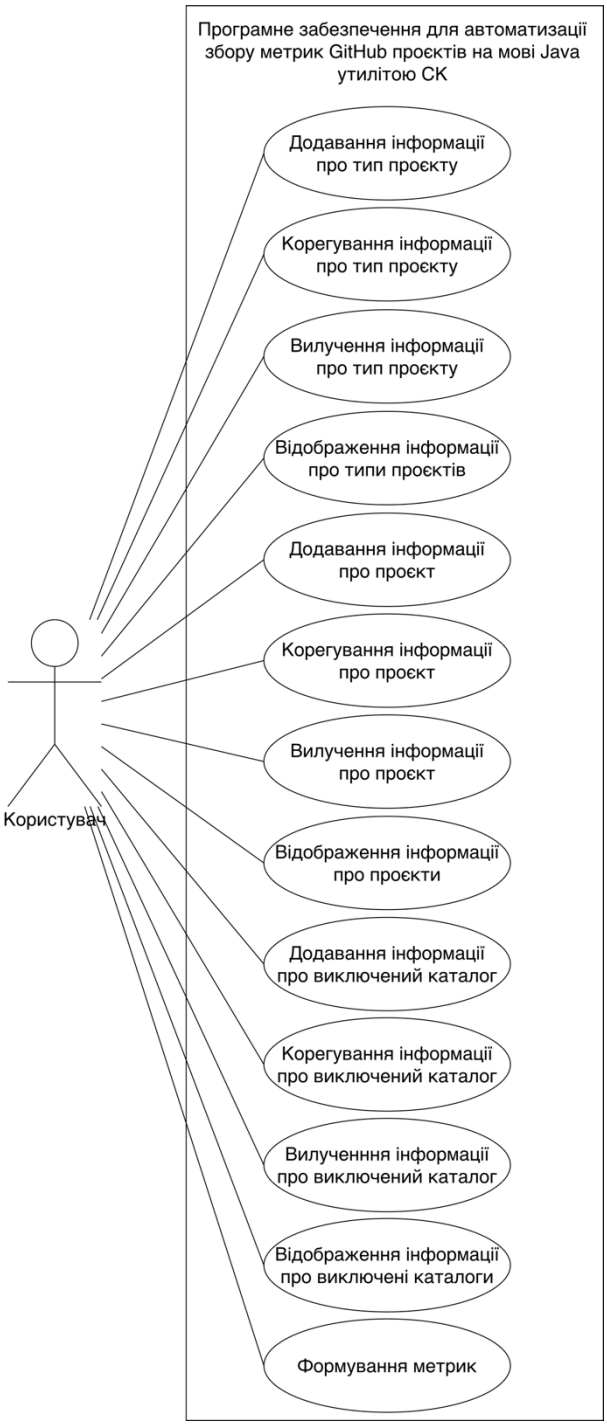


Рисунок 2.1 – Діаграма варіантів використання програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK

Специфікація варіанту використання «Додавання інформації про тип проєкту»

- Ім’я: Додавання інформації про тип проєкту.

- Призначення: Додавання нового типу проєкту під час його створення.

- Учасники: Користувач.

- Передумови: Тип проєкту на етапі створення і немає визначеного йому назви в базі даних.

- Стартова точка: Відкрита вкладка «Типи» із списком типів проєктів.

- Процедура:

1. Користувач ініціює перегляд списку типів проєктів.

2. Система виводить список типів проєктів.

3. Користувач заповнює, у текстовому полі для даних під списком, інформацію про назву типу проєкту.

4. Користувач підтверджує задані зміни натисненням кнопки додавання.

5. Система перевіряє дані на коректність і відсутність дублювання, якщо введені дані коректні і не дублюючі, тоді система зберігає їх, але якщо назва для типу проєкту некоректна згідно формату або дублююча, тоді система відображає спливаюче повідомлення з текстовою інформацією про некоректність або дублювання даних із проханням ввести інші.

6. Система відображає оновлений список типів проєктів у вкладці «Типи» зі зміненою інформацією.

- Результати: В таблицю бази даних «Типи» додано запис про тип проєкту.

- Умови після завершення використання: Система виводить повідомлення про успіх додавання нового запису про тип проєкту і оновлює вміст вкладки «Типи», де відображено список типів проєктів із доданою інформацією.

Специфікація варіанту використання «Корегування інформації про тип проєкту»

- Ім’я: Корегування інформації про тип проєкту.

- Призначення: Корегування існуючого типу проєкту.

- Учасники: Користувач.

- Передумови: Тип проєкту створено і визначено йому назву в базі даних.

- Стартова точка: Відкрита вкладка «Типи» із списком типів проєктів.

- Процедура:

1. Користувач ініціює перегляд списку типів проєктів.

2. Система виводить список типів проєктів.

3. Користувач обирає на поточній вкладці потрібний тип проєкту.

4. Користувач змінює, у текстовому полі для даних під списком, інформацію про назву типу проєкту.

5. Користувач підтверджує задані зміни натисненням кнопки корегування.

6. Система перевіряє дані на коректність і відсутність дублювання, якщо введені дані коректні і не дублюючі, тоді система зберігає їх, але якщо назва для типу проєкту некоректна згідно формату або дублююча, тоді система відображає спливаюче повідомлення з текстовою інформацією про некоректність або дублювання даних із проханням ввести інші.

7. Система відображає оновлений список типів проєктів у вкладці «Типи» зі зміненою інформацією.

- Результати: В таблиці бази даних «Типи» змінено запис про тип проєкту.

- Умови після завершення використання: Система виводить повідомлення про успіх корегування існуючого запису про тип проєкту і оновлює вміст вкладки «Типи», де відображено список типів проєктів із зміненою інформацією.

Специфікація варіанту використання «Вилучення інформації про тип проєкту»

- Ім’я: Вилучення інформації про тип проєкту.

- Призначення: Вилучення існуючого типу проєкту.

- Учасники: Користувач.

- Передумови: Тип проєкту створено і визначено йому назву в базі даних.

- Стартова точка: Відкрита вкладка «Типи» із списком типів проєктів.

- Процедура:

1. Користувач ініціює перегляд списку типів проєктів.

2. Система виводить список типів проєктів.

3. Користувач обирає на поточній вкладці потрібний тип проєкту.

4. Користувач підтверджує операцію видалення натисненням відповідної кнопки для вилучення.

5. Система перевіряє чи було обрано щонайменше один тип проєкту зі списку для подальшого вилучення, якщо так було обрано тип проєкту, тоді система відкриває діалогове вікно з двома діями як-от підтвердження або відхилення дії щодо видалення обраного типу проєкту зі списку, але якщо ні не було обраного жодного типу проєкту, тоді система відображає спливаюче повідомлення про відсутність обраного типу проєкту із проханням обрати щонайменше один тип проєкту у списку.

6. Користувач підтверджує дію щодо видалення типу проєкту через вибір відповідної кнопки в діалоговому вікні.

7. Система відображає повідомлення про неможливість вилучення обраного типу проєкту, тому що він присвоєний щонайменше одному з існуючих проєктів.

- Результати: В таблицях бази даних «Типи» і «Типи проєктів» залишається без змін запис про тип проєкту.

- Умови після завершення використання: Система виводить повідомлення про помилку вилучення існуючого запису про тип проєкту, тому що даний тип проєкту вже належить деякому проєкту або проєктам і повертає користувача назад на вкладку «Типи», де відображено список типів проєктів без змін.

Специфікація варіанту використання «Відображення інформації про типи проєктів»

- Ім’я: Відображення інформації про типи проєктів.

- Призначення: Перегляд існуючих типів проєктів.

- Учасники: Користувач.

- Передумови: Типи проєктів створено і визначено інформацію про них в базі даних.

- Стартова точка: Відкрита вкладка «Типи» із списком типів проєктів.

- Процедура:

1. Користувач ініціює перегляд списку типів проєктів.

2. Система перевіряє наявність записів для обраного списку, якщо вони є там, тоді система виводить список типів проєктів, але якщо в обраному списку дані відсутні, тоді система відображає порожню таблицю з пустим списком типів проєктів.

- Результати: Отримано потрібні дані з таблиці бази даних «Типи» для відображення списку типів проєктів.

- Умови після завершення використання: Система відображає на вкладці «Типи» список типів проєктів у вигляді таблиці.

Специфікація варіанту використання «Додавання інформації про проєкт»

- Ім’я: Додавання інформації про проєкт.

- Призначення: Додавання нового проєкту під час його створення.

- Учасники: Користувач.

- Передумови: Проєкт на етапі створення і немає визначеного йому посилання, типу проєкту в базі даних.

- Стартова точка: Відкрита вкладка «Проєкти» із списком проєктів.

- Процедура:

1. Користувач ініціює перегляд списку проєктів.

2. Система виводить список проєктів.

3. Користувач заповнює, у текстовому полі для даних під списком, інформацію про посилання, а також обирає тип проєкту з випадаючого списку або вказує на його відсутність.

4. Користувач підтверджує задані зміни натисненням кнопки додавання.

5. Система перевіряє дані на коректність і відсутність дублювання, якщо введені дані коректні і не дублюючі, тоді система зберігає їх, але якщо посилання

для проєкту некоректне згідно формату або дублююче, тоді система відображає спливаюче повідомлення з текстовою інформацією про некоректність або дублювання даних із проханням ввести інші.

6. Система відображає оновлений список проєктів у вкладці «Проєкти» зі зміненою інформацією.

- Результати: В таблицях бази даних «Проєкти» і «Типи проєктів» додано запис про проєкт.

- Умови після завершення використання: Система виводить повідомлення про успіх додавання нового запису про проєкт і оновлює вміст вкладки «Проєкти», де відображено список проєктів із доданою інформацією.

Специфікація варіанту використання «Корегування інформації про проєкт»

- Ім’я: Корегування інформації про проєкт.

- Призначення: Корегування існуючого проєкту.

- Учасники: Користувач.

- Передумови: Проєкт створено і визначено йому посилання, тип проєкту в базі даних.

- Стартова точка: Відкрита вкладка «Проєкти» із списком проєктів.

- Процедура:

1. Користувач ініціює перегляд списку проєктів.

2. Система виводить список проєктів.

3. Користувач обирає на поточній вкладці потрібний проєкт.

4. Користувач змінює, у текстовому полі для даних під списком, інформацію про посилання, а також обирає тип проєкту з випадаючого списку або вказує на його відсутність.

5. Користувач підтверджує задані зміни натисненням кнопки корегування.

6. Система перевіряє дані на коректність і відсутність дублювання, якщо введені дані коректні і не дублюючі, тоді система зберігає їх, але якщо посилання для проєкту некоректне згідно формату або дублююче, тоді система відображає

спливаюче повідомлення з текстовою інформацією про некоректність або дублювання даних із проханням ввести інші.

7. Система відображає оновлений список проєктів у вкладці «Проєкти» зі зміненою інформацією.

- Результати: В таблицях бази даних «Проєкти» і «Типи проєктів» змінено запис про проєкт.

- Умови після завершення використання: Система виводить повідомлення про успіх корегування існуючого запису про проєкт і оновлює вміст вкладки «Проєкти», де відображено список проєктів із зміненою інформацією.

Специфікація варіанту використання «Вилучення інформації про проєкт»

- Ім’я: Вилучення інформації про проєкт.

- Призначення: Вилучення існуючого проєкту.

- Учасники: Користувач.

- Передумови: Проєкт створено і визначено йому посилання, тип проєкту в базі даних.

- Стартова точка: Відкрита вкладка «Проєкти» із списком проєктів.

- Процедура:

1. Користувач ініціює перегляд списку проєктів.

2. Система виводить список проєктів.

3. Користувач обирає на поточній вкладці потрібний проєкт.

4. Користувач підтверджує операцію видалення натисненням відповідної кнопки для вилучення.

5. Система перевіряє чи було обрано щонайменше один проєкт зі списку для подальшого вилучення, якщо так було обрано проєкт, тоді система відкриває діалогове вікно з двома діями як-от підтвердження або відхилення дії щодо видалення обраного проєкту зі списку, але якщо ні не було обраного жодного проєкту, тоді система відображає спливаюче повідомлення про відсутність обраного проєкту із проханням обрати щонайменше один проєкт у списку.

6. Користувач підтверджує дію щодо видалення проєкту через вибір відповідної кнопки в діалоговому вікні.

7. Система зберігає ці зміни.

8. Система відображає оновлений список проєктів у вкладці «Проєкти» зі зміненою інформацією.

- Результати: В таблицях бази даних «Проєкти» і «Типи проєктів» вилучено запис про проєкт.

- Умови після завершення використання: Система виводить повідомлення про успіх вилучення існуючого запису про проєкт і оновлює вміст вкладки «Проєкти», де відображено список проєктів без врахування вилученої інформації.

Специфікація варіанту використання «Відображення інформації про проєкти»

- Ім’я: Відображення інформації про проєкти.

- Призначення: Перегляд існуючих проєктів.

- Учасники: Користувач.

- Передумови: Проєкти створено і визначено інформацію про них в базі даних.

- Стартова точка: Відкрита вкладка «Проєкти» із списком проєктів.

- Процедура:

1. Користувач ініціює перегляд списку проєктів.

2. Система перевіряє наявність записів для обраного списку, якщо вони є там, тоді система виводить список проєктів, але якщо в обраному списку дані відсутні, тоді система відображає порожню таблицю з пустим списком проєктів.

- Результати: Отримано потрібні дані з таблиці бази даних «Проєкти» для відображення списку проєктів.

Умови після завершення використання: Система відображає на вкладці «Проєкти» список проєктів у вигляді таблиці.

Специфікація варіанту використання «Додавання інформації про виключений каталог»

- Ім’я: Додавання інформації про виключений каталог.

- Призначення: Додавання нового виключеного каталогу під час його створення.

- Учасники: Користувач.

- Передумови: Виключений каталог на етапі створення і немає визначеного йому назви в базі даних.

- Стартова точка: Відкрита вкладка «Виключені каталоги» із списком виключених каталогів.

- Процедура:

1. Користувач ініціює перегляд списку виключених каталогів.

2. Система виводить список виключених каталогів.

3. Користувач заповнює, у текстовому полі для даних під списком, інформацію про назву виключеного каталогу.

4. Користувач підтверджує задані зміни натисненням кнопки додавання.

5. Система перевіряє дані на коректність і відсутність дублювання, якщо введені дані коректні і не дублюючі, тоді система зберігає їх, але якщо назва для виключеного каталогу некоректна згідно формату або дублююча, тоді система відображає спливаюче повідомлення з текстовою інформацією про некоректність або дублювання даних із проханням ввести інші.

6. Система відображає оновлений список виключених каталогів у вкладці «Виключені каталоги» зі зміненою інформацією.

- Результати: В таблицю бази даних «Виключені каталоги» додано запис про виключений каталог.

- Умови після завершення використання: Система виводить повідомлення про успіх додавання нового запису про виключений каталог і оновлює вміст вкладки «Виключені каталоги», де відображено список виключених каталогів із доданою інформацією.

Специфікація варіанту використання «Корегування інформації про виключений каталог»

- Ім’я: Корегування інформації про виключений каталог.

- Призначення: Корегування існуючого виключеного каталогу.

- Учасники: Користувач.

- Передумови: Виключений каталог створено і визначено йому назву в базі даних.

- Стартова точка: Відкрита вкладка «Виключені каталоги» із списком виключених каталогів.

- Процедура:

1. Користувач ініціює перегляд списку виключених каталогів.

2. Система виводить список виключених каталогів.

3. Користувач обирає на поточній вкладці потрібний виключений каталог.

4. Користувач змінює, у текстовому полі для даних під списком, інформацію про назву виключеного каталогу.

5. Користувач підтверджує задані зміни натисненням кнопки корегування.

6. Система перевіряє дані на коректність і відсутність дублювання, якщо введені дані коректні і не дублюючі, тоді система зберігає їх, але якщо назва для виключеного каталогу некоректна згідно формату або дублююча, тоді система відображає спливаюче повідомлення з текстовою інформацією про некоректність або дублювання даних із проханням ввести інші.

7. Система відображає оновлений список виключених каталогів у вкладці «Виключені каталоги» зі зміненою інформацією.

- Результати: В таблиці бази даних «Виключені каталоги» змінено запис про виключений каталог.

- Умови після завершення використання: Система виводить повідомлення про успіх корегування існуючого запису про виключений каталог і оновлює вміст вкладки «Виключені каталоги», де відображено список виключених каталогів із зміненою інформацією.

Специфікація варіанту використання «Вилучення інформації про виключений каталог»

- Ім’я: Вилучення інформації про виключений каталог.

- Призначення: Вилучення існуючого виключеного каталогу.

- Учасники: Користувач.

- Передумови: Виключений каталог створено і визначено йому назву в базі даних.

- Стартова точка: Відкрита вкладка «Виключені каталоги» із списком виключених каталогів.

- Процедура:

1. Користувач ініціює перегляд списку виключених каталогів.

2. Система виводить список виключених каталогів.

3. Користувач обирає на поточній вкладці потрібний виключений каталог.

4. Користувач підтверджує операцію видалення натисненням відповідної кнопки для вилучення.

5. Система перевіряє чи було обрано щонайменше один виключений каталог зі списку для подальшого вилучення, якщо так було обрано виключений каталог, тоді система відкриває діалогове вікно з двома діями як-от підтвердження або відхилення дії щодо видалення обраного виключеного каталогу зі списку, але якщо ні не було обраного жодного виключеного каталогу, тоді система відображає спливаюче повідомлення про відсутність обраного виключеного каталогу із проханням обрати щонайменше один виключений каталог у списку.

6. Користувач підтверджує дію щодо видалення виключеного каталогу через вибір відповідної кнопки в діалоговому вікні.

7. Система зберігає ці зміни.

8. Система відображає оновлений список виключених каталогів у вкладці «Виключені каталоги» зі зміненою інформацією.

- Результати: В таблиці бази даних «Виключені каталоги» вилучено запис про виключений каталог.

- Умови після завершення використання: Система виводить повідомлення про успіх вилучення існуючого запису про виключений каталог і оновлює вміст вкладки «Виключені каталоги», де відображено список виключених каталогів без врахування вилученої інформації.

Специфікація варіанту використання «Відображення інформації про виключені каталоги»

- Ім’я: Відображення інформації про виключені каталоги.

- Призначення: Перегляд існуючих виключених каталогів.

- Учасники: Користувач.

- Передумови: Виключені каталоги створено і визначено інформацію про них в базі даних.

- Стартова точка: Відкрита вкладка «Виключені каталоги» із списком виключених каталогів.

- Процедура:

1. Користувач ініціює перегляд списку виключених каталогів.

2. Система перевіряє наявність записів для обраного списку, якщо вони є там, тоді система виводить список виключених каталогів, але якщо в обраному списку дані відсутні, тоді система відображає порожню таблицю з пустим списком виключених каталогів.

- Результати: Отримано потрібні дані з таблиці бази даних «Виключені каталоги» для відображення списку виключених каталогів.

- Умови після завершення використання: Система відображає на вкладці «Виключені каталоги» список виключених каталогів у вигляді таблиці.

Специфікація варіанту використання «Формування метрик»

- Ім’я: Формування метрик.

- Призначення: Генерування метрик проєктів.

- Учасники: Користувач.

- Передумови: Проєкти, типи проєктів, виключені каталоги створено і визначено інформацію про них в базі даних, шлях до директорії існує та доступний для запису.

- Стартова точка: Відкрита вкладка «Checkout» із списком проєктів і налаштувань метрик.

- Процедура:

1. Користувач ініціює перегляд списку проєктів.

2. Система виводить список проєктів.

3. Користувач обирає на поточній вкладці потрібні налаштування для генерації метрик як-от тип проєкту, виключені каталоги, гілку коду проєктів, директорію для збереження файлу метрик.

4. Система перевіряє задані користувачем налаштування, якщо всі налаштування коректні, тоді система зберігає ці зміни налаштувань, якщо вказаний некоректний або недоступний шлях до директорії, тоді система відображається спливаюче повідомлення про недоступну директорію із проханням змінити шлях, якщо задані налаштування коректні, але проєкти не було завантажено з GitHub репозиторіїв, тоді система відображає повідомлення про помилку технічного характеру з проханням перевірити коректність заданих налаштувань, якщо задані налаштування коректні, проєкти було завантажено, але метрики не створено, тоді система відображає повідомлення про помилку технічного характеру з проханням перевірити налаштування утиліти CK.

5. Система створює на збережених останніх налаштувань метрики.

6. Система зберігає у вказану директорію файл із результатом роботи.

- Результати: Отримано потрібні дані з таблиць бази даних «Типи», «Проєкти», «Типи проєктів» і «Виключені каталоги» потрібні для генерації метрик проєктів.

- Умови після завершення використання: Усі проєкти відповідно до вказаного типу проєкту було завантажено і проаналізовано за допомогою утиліти CK, отримані на основі них метрики були успішно збережені у файл по шляху заданої директорії.

2.2 Архітектура програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK

Програмне забезпечення має багатошарову архітектуру з елементами десктопного застосунку MVC, де ролі Controller частково виконують панелі Swing та окремі класи Service.

Кожна функціональна область поділена на окремі шари з чітким розподілом відповідальності, тобто цей підхід працює за принципом розмежування обов'язків, а саме розроблюване ПЗ поділяється на шари, які взаємодіють між собою, але при цьому кожен із них виконує власну визначену роль або завдання [9].

Розроблюваний застосунок має чітко розділені групи класів, наприклад, UI-панелі на Swing, сервісні класи для реалізації бізнес-логіки, DAO-класи для доступу до БД та JPA-сутності для моделювання таблиць, що в загальному вигляді утворює 4 основні шари такі як інтерфейс користувача, бізнес-логіку, доступ до даних та моделі.

Такий поділ застосунку на окремі шари дозволяє змінювати та розширювати окремі частини системи з мінімальним впливом на інші, підвищуючи зручність підтримки, масштабування й подальшого розвитку, а саме:

- Класи UI зосереджують відображення інтерфейсу, обробку дій користувача та виклики сервісного або DAO-рівня, тобто для десктопного застосунку це реалізовано через панелі TypesPanel, ProjectsPanel, ExcludedFoldersPanel, CheckoutPanel, а також допоміжні UI-компоненти, які разом фактично поєднують ролі View і частково Controller.

- Класи Service як-от CheckoutService, GitHubService і CkService реалізують основні бізнес-процеси застосунку, а також спеціалізовані компоненти, наприклад, CkRunner, CkCsvParser, MetricsAggregator, ExcelReportWriter відповідальні за отримання проєктів із репозиторіїв, checkout потрібного стану, запуск CK-аналізу, обробку CSV-результатів та формування підсумкового Excel-звіту, що в черговий раз підтверджує ґрунтовність обраного архітектурного стилю, бо основна частини бізнес-логіки саме винесена з UI в окремий сервісний шар.

- Класи DAO, такі як ProjectDAO, TypeDAO, ExcludedFolderDAO, SettingDAO, забезпечують інкапсуляцію доступу до бази даних й ізолюють механізми збереження та читання даних від інших частин застосунку, тобто для простих CRUD-операцій та окремих локальних правил, пов’язаних безпосередньо з роботою з даними, цей шар також містить частину прикладної логіки.

- Класи Entity у застосунку, наприклад, Project, Type, ExcludedFolder, Setting, виконують роль Model, тому що вони представляють структуру даних предметної області та відображають відповідні таблиці бази даних.

Діаграма компонентів програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK представлена на рисунку 2.2.

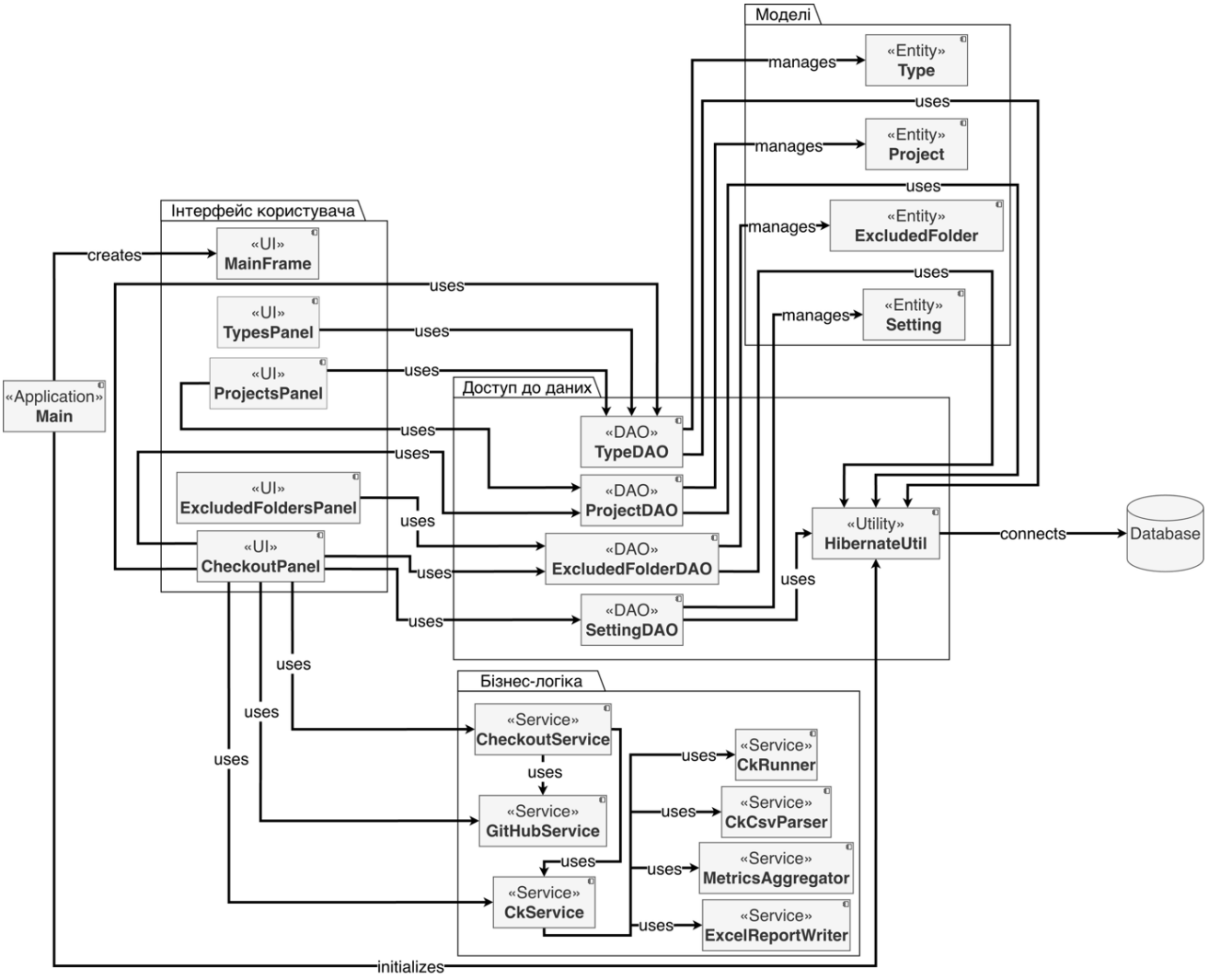


Рисунок 2.2 – Діаграма компонентів програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK

Специфікація компонентів згідно діаграми:

- Main (Application): запускає застосунок, ініціалізує графічний інтерфейс і створює MainFrame, не містить бізнес-логіки застосунку.

- MainFrame (UI): організовує вкладки з панелями такими як TypesPanel, ProjectsPanel, ExcludedFoldersPanel, CheckoutPanel, а також відповідає за загальний життєвий цикл головного вікна та при закритті застосунку завершує роботу HibernateUtil.

- TypesPanel, ProjectsPanel, ExcludedFoldersPanel, CheckoutPanel (UI): відображають користувачу таблиці й форми згідно даних БД, обробляють події кнопок та інших елементів інтерфейсу і викликають відповідні класи DAO та Service, і при цьому ще CheckoutPanel додатково координує роботу сервісів, пов’язаних із checkout, та збором метрик.

- CheckoutService, GitHubService, CkService (Service): реалізують основні бізнес-процеси застосунку, де CheckoutService координує сценарій checkout і збору метрик, GitHubService відповідає за клонування репозиторіїв і checkout потрібного стану, а CkService керує запуском аналізу CK та формуванням підсумкового результату.

- CkRunner, CkCsvParser, MetricsAggregator, ExcelReportWriter (спеціалізовані компоненти сервісного шару): використовуються класом CkService для виконання окремих підзадач, а саме запуску ck.jar, зчитування та парсингу CSV-файлів, агрегації метрик і створення Excel-звіту.

- TypeDAO, ProjectDAO, ExcludedFolderDAO, SettingDAO (DAO): інкапсулюють доступ до БД, працюють через HibernateUtil і надають методи для збереження, читання, пошуку, оновлення та видалення даних залежно від потреб конкретної сутності.

- Type, Project, ExcludedFolder, Setting (Entity, Model): представляють сутності предметної області та відображають відповідні таблиці бази даних, а також використовуються як об’єкти моделі в DAO, сервісах і UI.

- HibernateUtil (Utility): єдина точка конфігурації та управління SessionFactory, через яку всі DAO отримують сесії для роботи з базою даних.

2.3 Проєкт програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK

2.3.1 Ескізний проєкт програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK

Ескізний проєкт програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK містить базові сценарії взаємодії користувача з ПЗ, організацію даних та прототипи користувацького інтерфейсу. Було розроблено діаграми діяльності для двох варіантів використання, які пов’язані з обробкою інформації про проєкти, а також спроєктовано концептуальну модель для подання структури даних та ескізи інтерфейсу користувача.

Для розробки концептуальної моделі необхідно виконати декілька етапів, першим з яких буде визначення сутностей предметної галузі та їх атрибутів.

Сутність це реальний або уявний об’єкт, інформація про який повинна зберігатися в проєктованій системі. Атрибут повинен мати ім’я, що є унікальним в межах сутності.

В проєкті нашого ПЗ можна виділити наступні сутності та атрибути:

- Типи: ідентифікатор типу проєкту, назва типу проєкту.
- Проєкти: ідентифікатор проєкту, посилання проєкту.
- Виключені каталоги: ідентифікатор каталогу, назва виключеного каталогу.
- Налаштування: ідентифікатор налаштування, назва налаштування, значення налаштування

Продовженням розробки концептуальної моделі є визначення та опис зв’язків між сутностями. Ступінь зв’язку це число екземплярів однієї сутності, які можуть бути пов’язані з екземплярами іншої сутності. Концептуальна модель даних це графічне подання про уявлення системи, що досліджується із її сутностями, атрибутами та зв’язки між ними.

Зв’язки між сутностями представлені в таблиці 2.1.

Таблиця 2.1 – Зв’язки між сутностями

Назва сутності	Назва сутності	Тип	Пояснення
Проєкти	Типи	0…1:0…N	Кожен запис у таблиці «Проєкти» може бути пов’язаний із жодним або лише одним записом у таблиці «Типи», а кожен запис у таблиці «Типи» може бути пов’язаний із жодним, одним або багатьма записами у таблиці «Проєкти»

ER-діаграма концептуальної моделі даних представлена на рисунку 2.3.

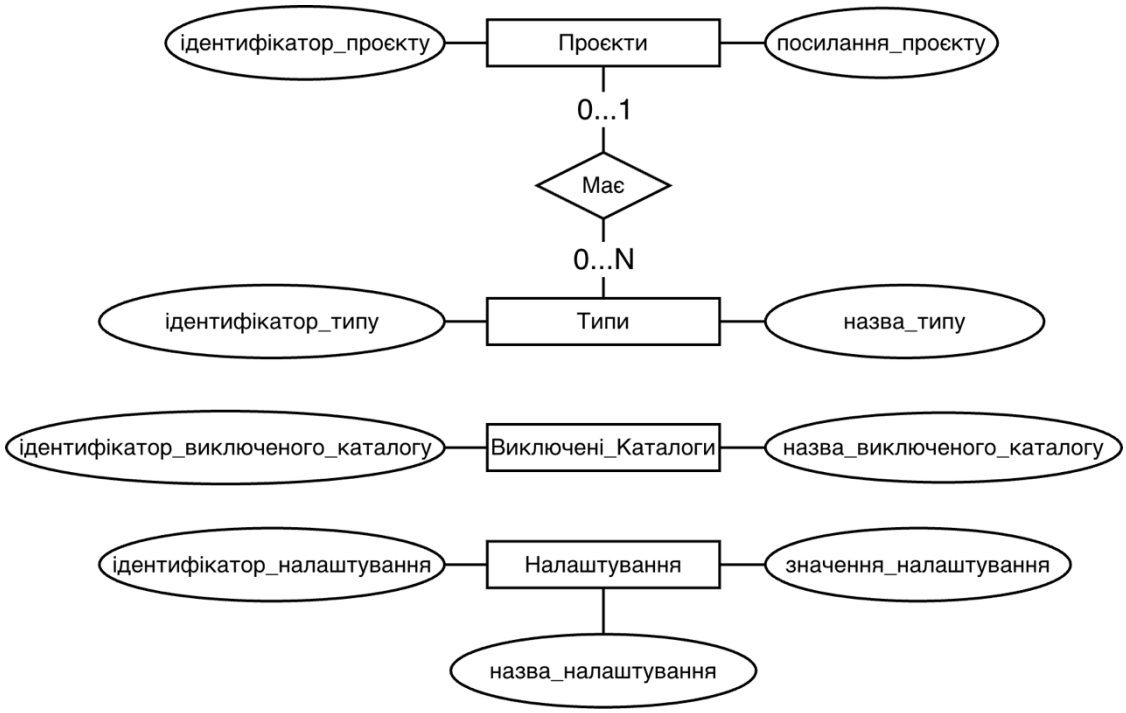


Рисунок 2.3 – ER-діаграма концептуальної моделі даних

Діаграма діяльності це розширене представлення блок-схеми, яке слугує для того, щоб візуалізувати процеси системи та показати потік виконання в системі від однієї діяльності до іншої.

В даній роботі було наведено лише дві діаграми діяльності згідно варіантів використання однієї з сутностей під назвою «Проєкти» такі як додавання

інформації про проєкт та відображення інформації про проєкти, тому що вигляд і спосіб подання інформації на діаграмах використання для варіантів використання щодо сутностей «Типи» й «Виключені каталоги» буде аналогічним, але зі зміненими назвами атрибутів.

Діаграми діяльності варіантів використання представлено на рисунках 2.4, 2.5.

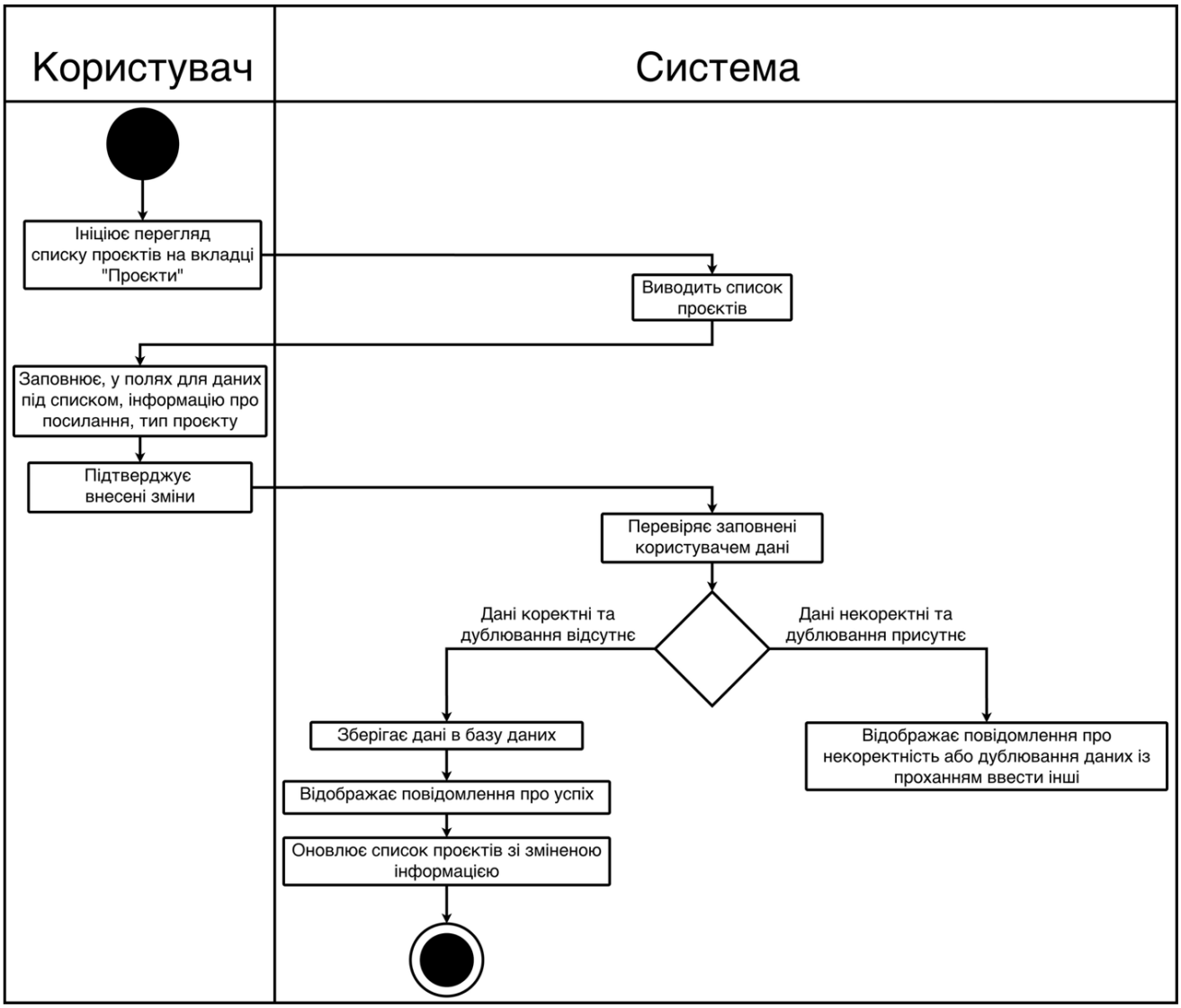


Рисунок 2.4 – Діаграма діяльності варіанту використання «Додавання інформації про проєкт»

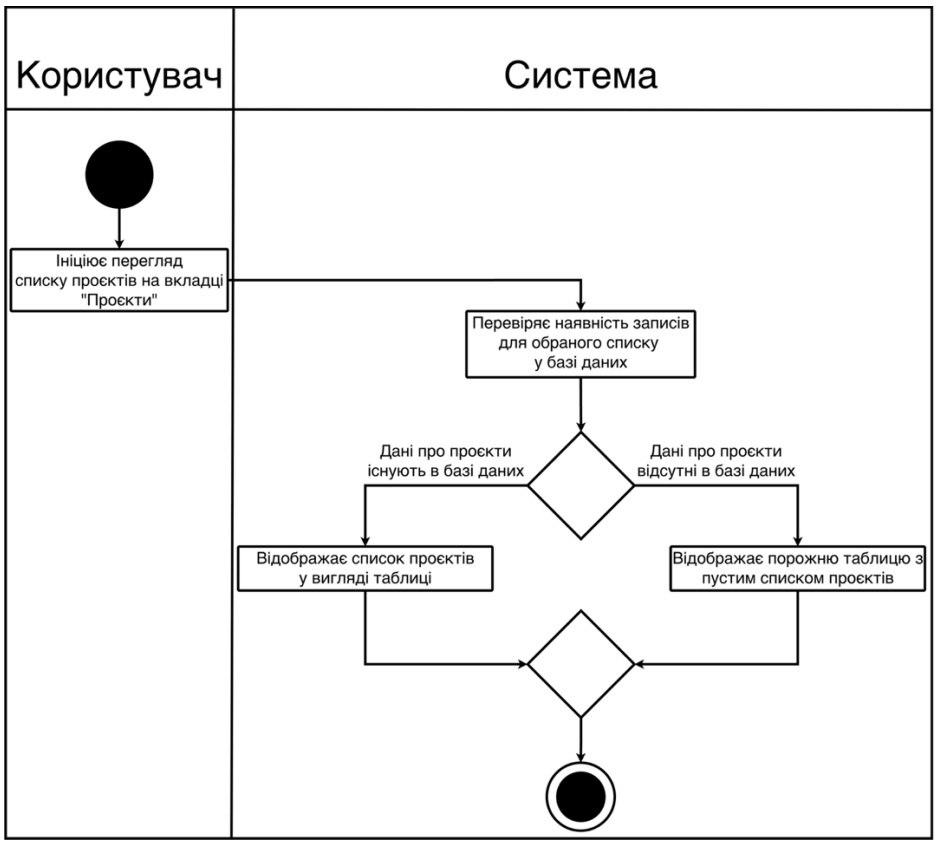


Рисунок 2.5 – Діаграма діяльності варіанту використання «Відображення інформації про проєкти»

Ескізи інтерфейсу користувача представлено на рисунках 2.6-2.9.

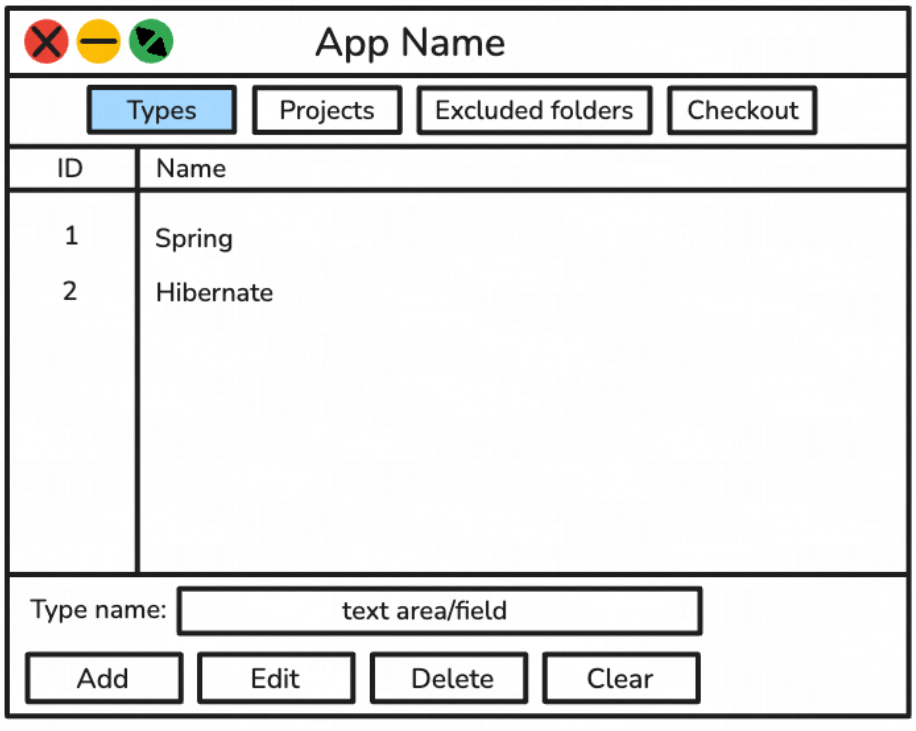


Рисунок 2.6 – Ескіз інтерфейсу вкладки зі списком типів проєктів

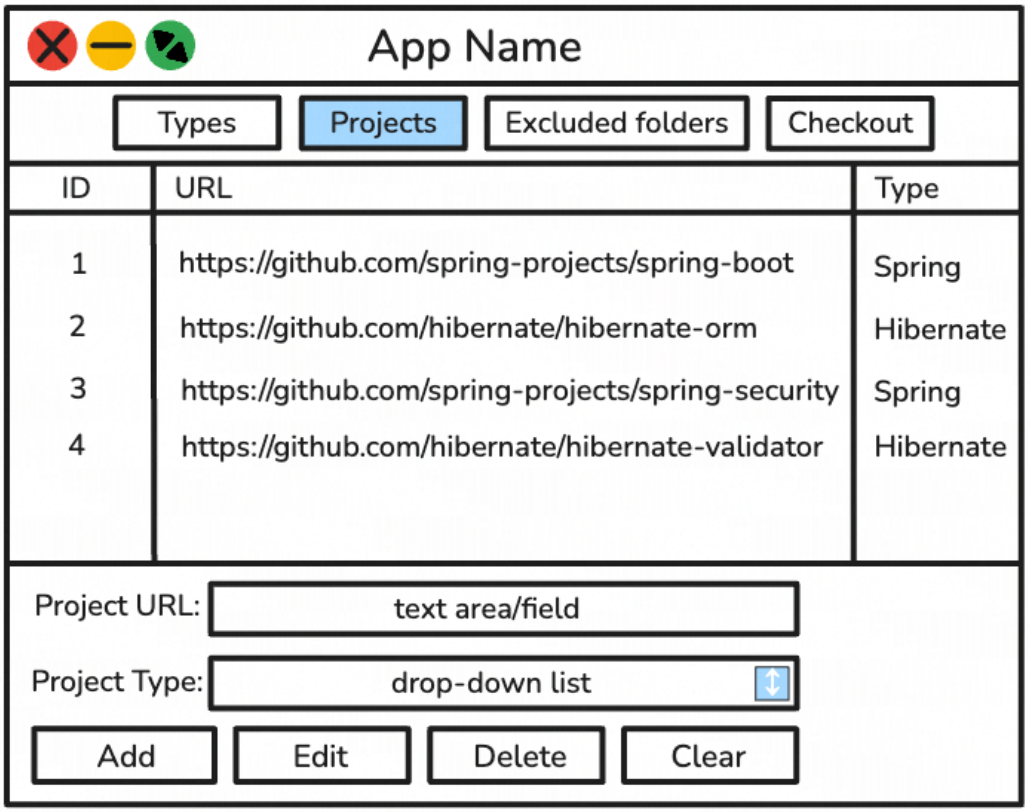


Рисунок 2.7 – Ескіз інтерфейсу вкладки зі списком проєктів

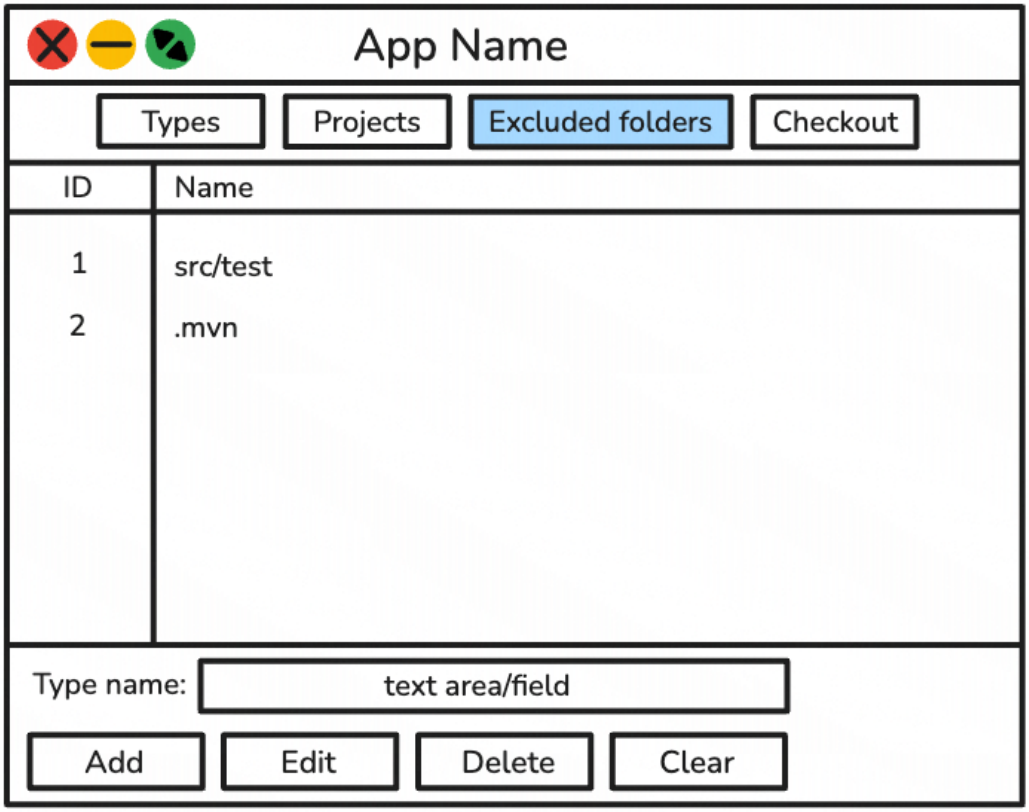


Рисунок 2.8 – Ескіз інтерфейсу вкладки зі списком виключених каталогів

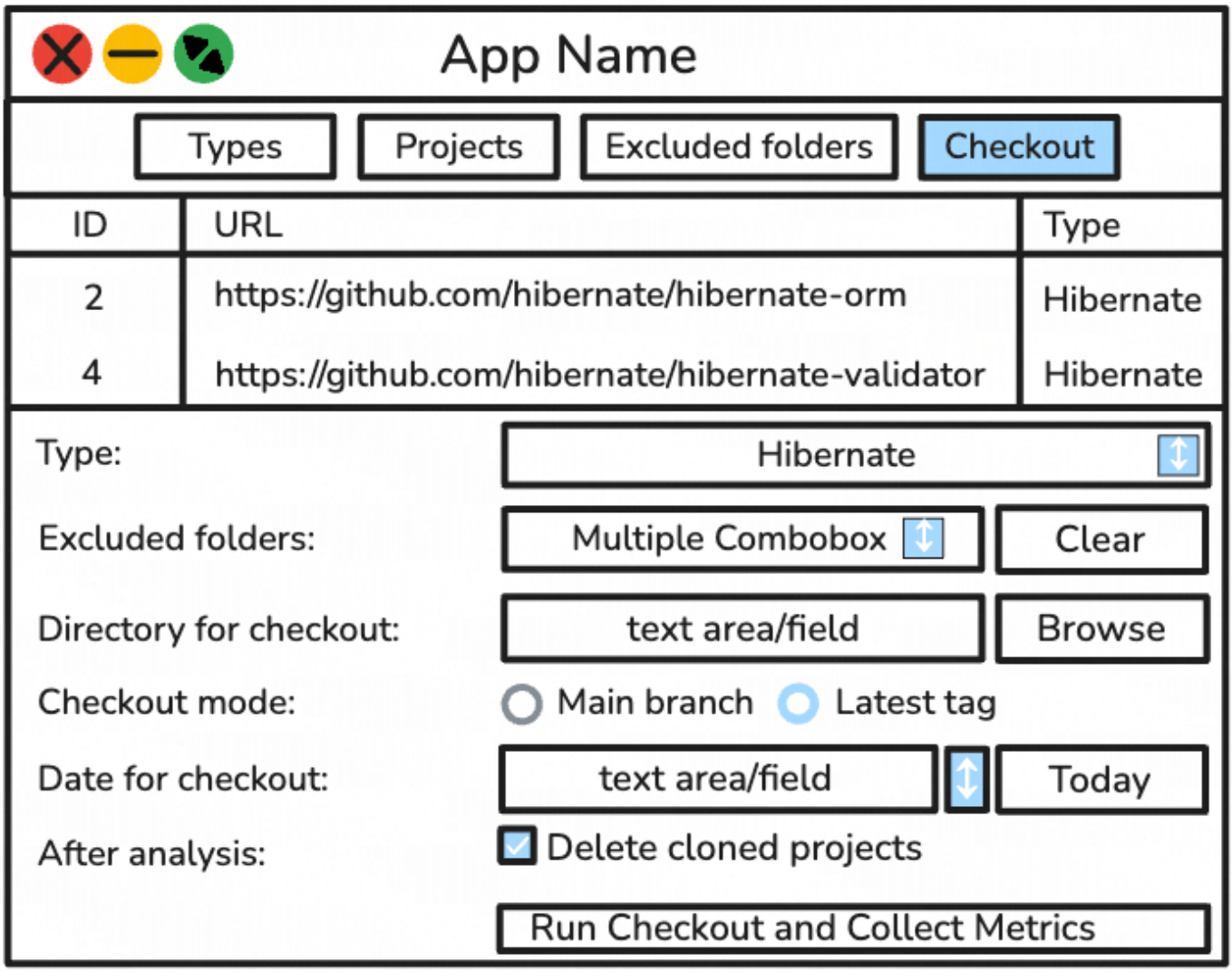


Рисунок 2.9 – Ескіз інтерфейсу вкладки «Checkout»

2.3.2 Технічний проєкт програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK

Технічний проєкт програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK містить основу для реалізації ПЗ, а саме візуалізацію основних класів застосунку на діаграмі та представлення логічної моделі даних.

Діаграма класів програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK представлена на рисунку 2.10.

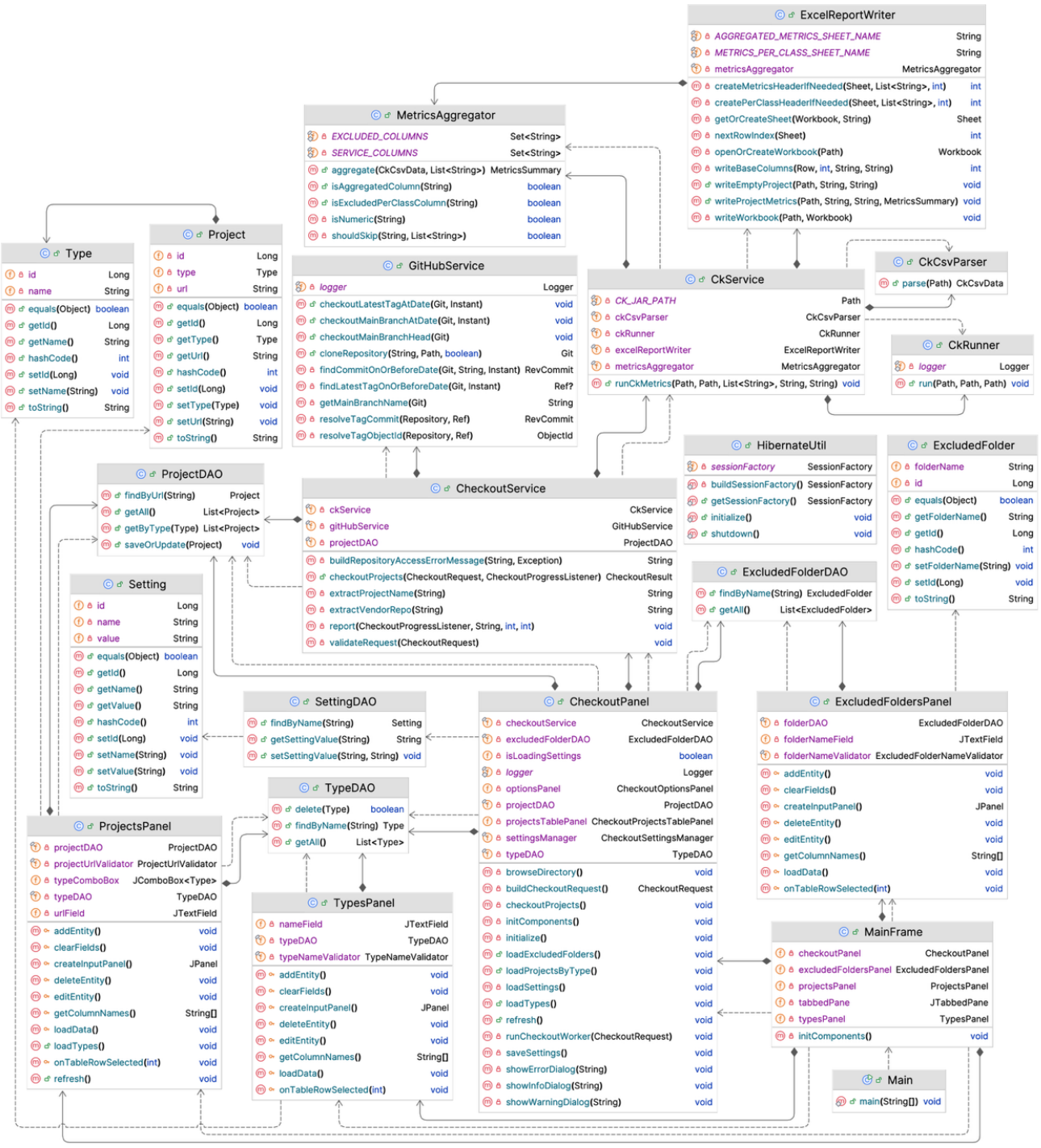


Рисунок 2.10 – Діаграма класів програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK

Специфікація класу «ExcludedFolderDAO»

Ім’я: ckmetrics.dao.ExcludedFolderDAO.

Відповідальність: клас доступу до даних для сутності ExcludedFolder, який забезпечує операції читання та пошуку виключених каталогів у базі даних через Hibernate.

Атрибути:

- Відсутні, тому що клас виконує роль DAO-обгортки над сесією Hibernate та не зберігає власний стан між викликами.

Методи:

- ExcludedFolderDAO() створює DAO для сутності ExcludedFolder.

- getAll() повертає список усіх виключених каталогів, відсортованих за ідентифікатором.

- findByName(String name) повертає виключений каталог згідно його назви або null, якщо виключений каталог не було знайдено.

Специфікація класу «ProjectDAO»

Ім’я: ckmetrics.dao.ProjectDAO.

Відповідальність: клас доступу до даних для сутності Project, який забезпечує CRUD-операції, пошук за URL та фільтрацію згідно типу проєкту із урахуванням зв’язку з Type.

Атрибути:

- Відсутні, тому що клас працює як DAO-шар над Hibernate і використовує стан сесії, а не власні поля логіки.

Методи:

- ProjectDAO() створює DAO для сутності Project.

- saveOrUpdate(Project project) зберігає або оновлює проєкт, а також перед merge перевантажує Type із поточної сесії для коректної роботи зі зв’язаною сутністю.

- getAll() повертає всі проєкти, відсортовані за ідентифікатором.

- findByUrl(String url) повертає проєкт за URL або null, якщо проєкт не було знайдено.

- getByType(Type type) повертає список проєктів заданого типу проєкту, відсортований за ідентифікатором.

Специфікація класу «SettingDAO»

Ім’я: ckmetrics.dao.SettingDAO.

Відповідальність: клас доступу до даних для сутності Setting, який забезпечує пошук налаштувань, отримання значення згідно ключа та операцію upsert значення.

Атрибути:

- Відсутні, тому що клас реалізує DAO-операції через Hibernate і не зберігає окремий стан.

Методи:

- SettingDAO() створює DAO для сутності Setting.

- findByName(String name) повертає налаштування за назвою або null, якщо налаштування не було знайдено.

- getSettingValue(String name) повертає значення налаштування за назвою або null, якщо значення налаштування не було знайдено.

- setSettingValue(String name, String value) створює або оновлює налаштування за ключем і зберігає зміни в базі даних.

Специфікація класу «TypeDAO»

Ім’я: ckmetrics.dao.TypeDAO.

Відповідальність: клас доступу до даних для сутності Type, який забезпечує CRUD-операції з перевіркою використання типу проєкту в проєктах перед видаленням.

Атрибути:

- Відсутні, тому що клас виконує запити через Hibernate-сесію та не накопичує власні дані.

Методи:

- TypeDAO() створює DAO для сутності Type.

- getAll() повертає всі типи проєктів, відсортовані за ідентифікатором.

- findByName(String name) повертає тип проєкту за назвою або null, якщо тип проєкту не було знайдено.

- delete(Type type) видаляє тип проєкту тільки якщо він не використовується жодним проєктом, а інакше повертає false.

Специфікація класу «ExcludedFolder»

Ім’я: ckmetrics.model.ExcludedFolder.

Відповідальність: JPA-сутність виключеного каталогу, яка представляє запис таблиці Excluded_Folder.

Атрибути:

- id: Long це первинний ключ excluded_folder_id.

- folderName: String це унікальна назва виключеного каталогу excluded_folder_name, яка є обов’язковим полем.

Методи:

- Конструктори ExcludedFolder() і ExcludedFolder(String folderName) необхідні для створення порожнього або ініціалізованого об’єкта.

- getId()/setId(...) використовуються для доступу до ідентифікатора.

- getFolderName()/setFolderName(...) потрібні для доступу до назви каталогу.

- toString() повертає назву виключеного каталогу.

- equals(...) і hashCode() застосовуються для порівняння і хешування за id та folderName.

Специфікація класу «Project»

Ім’я: ckmetrics.model.Project.

Відповідальність: JPA-сутність проєкту, яка представляє запис таблиці Projects і зв’язок із типом проєкту.

Атрибути:

- id: Long це первинний ключ project_id.

- url: String це унікальний URL проєкту project_url, який є обов'язковим полем.

- type: Type це зв'язок ManyToOne через таблицю Types_Projects з обмеженням унікальності project_id для моделі «0 або 1 тип проєкту на проєкт».

Методи:

- Конструктори Project() і Project(String url, Type type) необхідні для створення об'єкта.

- getId()/setId(...), getUrl()/setUrl(...), getType()/setType(...) використовуються для доступу до полів.

- toString() повертає URL проєкту.

- equals(...) і hashCode() застосовуються для порівняння і хешування за id та url.

Специфікація класу «Setting»

Ім'я: ckmetrics.model.Setting.

Відповідальність: JPA-сутність налаштування застосунку, яка представляє пару ключ-значення в таблиці Settings.

Атрибути:

- id: Long це первинний ключ setting_id.

- name: String це назва параметра setting_name, яка є унікальним обов'язковим полем.

- value: String це значення параметра setting_value, яке допускає null.

Методи:

- Конструктори Setting() і Setting(String name, String value) необхідні для створення об'єкта.

- Гетери/сетери id, name, value використовуються для доступу до полів.

- toString() повертає рядок у форматі «name: value».

- equals(...) і hashCode() застосовуються для порівняння і хешування за id та name.

Специфікація класу «Type»

Ім’я: ckmetrics.model.Type.

Відповідальність: JPA-сутність типу проєкту, яка представляє запис таблиці Types.

Атрибути:

- id: Long це первинний ключ type_id.

- name: String це назва типу проєкту type_name, яка є унікальним обов’язковим полем.

Методи:

- Конструктори Type() і Type(String name) необхідні для створення об’єкта.

- Гетери/сетери id, name використовуються для доступу до полів.

- toString() повертає назву типу проєкту.

- equals(...) і hashCode() застосовуються для порівняння і хешування за id та name.

Специфікація класу «CheckoutService»

Ім’я: ckmetrics.service.checkout.CheckoutService.

Відповідальність: сервіс для виконання процесу checkout проєктів, який координує вибір проєктів згідно типу проєкту, клонування, checkout гілки або тегу, запуск CK та формування результату з переліком помилок.

Атрибути:

- projectDAO: ProjectDAO це компонент доступу до даних, який використовується для отримання переліку проєктів згідно обраного типу проєкту.

- gitHubService: GitHubService це сервіс, який потрібен для виконання git-операцій clone і checkout.

- ckService: CkService це сервіс, який запускає CK-аналіз та записує метрики в звіт.

Методи:

- CheckoutService() і CheckoutService(ProjectDAO, GitHubService, CkService) необхідні для створення сервісу зі стандартними або переданими залежностями.

- checkoutProjects(CheckoutRequest request, CheckoutProgressListener progressListener) використовується як основний сценарій для обробки проєктів, запуску CK та повернення CheckoutResult.

- validateRequest(...) перевіряє обов'язкові параметри запуску.

- report(...) передає стан прогресу слухачу.

- extractProjectName(...) і extractVendorRepo(...) формують службові ідентифікатори з URL.

- buildRepositoryAccessErrorMessage(...) нормалізує помилки доступу до репозиторію для підсумкового звіту.

Специфікація класу «CkService»

Ім'я: ckmetrics.service.ck.CkService.

Відповідальність: координує повну послідовність процесів CK як-от запуск утиліти, обробку class.csv, агрегацію метрик та запис у файл формату XLSX.

Атрибути:

- CK_JAR_PATH: Path це шлях до jar утиліти CK.

- ckRunner: CkRunner це компонент запуску зовнішнього процесу CK.

- ckCsvParser: CkCsvParser це компонент читання CSV.

- metricsAggregator: MetricsAggregator це компонент обчислення агрегованих метрик.

- excelReportWriter: ExcelReportWriter це компонент запису XLSX-звіту.

Методи:

- CkService() і CkService(CkRunner, CkCsvParser, MetricsAggregator) необхідні для створення сервісу.

- runCkMetrics(...) використовується для виконання CK-аналізу проєкту, обробки результатів та дописування даних у підсумковий файл.

Специфікація класу «CkRunner»

Ім’я: ckmetrics.service.ck.CkRunner.

Відповідальність: виконує зовнішній запуск jar утиліти CK й обробляє її консольний вивід.

Атрибути:

- logger це логер для виводу повідомлень CK.

Методи:

- run(Path ckJar, Path projectDirectory, Path outputDirectory) використовується для запуску процесу CK, читання об’єднаного консольного виводу, контролю коду завершення та генерації помилки у випадку неуспішного виконання.

Специфікація класу «CkCsvParser»

Ім’я: ckmetrics.service.ck.CkCsvParser.

Відповідальність: читає файл class.csv та перетворює його в структурований об’єкт CkCsvData.

Атрибути:

- Відсутні, тому що клас виконує парсинг вхідних даних без збереження стану між викликами.

Методи:

- parse(Path csvFile) необхідний для зчитування заголовків й рядків CSV та повернення структури з даними для подальшої агрегації.

Специфікація класу «MetricsAggregator»

Ім’я: ckmetrics.service.ck.MetricsAggregator.

Відповідальність: агрегує значення метрик з class.csv, відфільтровує службові та виключені колонки, а також класи з виключених каталогів.

Атрибути:

- SERVICE_COLUMNS: Set<String> це службові колонки, що не агрегуються.

- EXCLUDED_COLUMNS: Set<String> це колонки метрик, що виключаються із звіту.

Методи:

- aggregate(CkCsvData csvData, List<String> excludedFolders) використовується для формування MetricsSummary з сумами числових метрик і рядками per-class.

- isAggregatedColumn(String header) визначає включення колонки до аркуша агрегованих метрик.

- isExcludedPerClassColumn(String header) визначає виключення колонки з аркуша per-class.

- shouldSkip(...) перевіряє, чи потрібно пропустити рядок класу за списком виключень.

- isNumeric(String value) перевіряє, чи значення можна інтерпретувати як число.

Специфікація класу «ExcelReportWriter»

Ім’я: ckmetrics.service.ck.ExcelReportWriter.

Відповідальність: формує та дописує підсумковий звіт у файл формату XLSX з двома аркушами такими як «Aggregated Metrics» й «Metrics Per Class».

Атрибути:

- AGGREGATED_METRICS_SHEET_NAME це назва аркуша з агрегованими метриками.

- METRICS_PER_CLASS_SHEET_NAME це назва аркуша з метриками по класах.

- metricsAggregator: MetricsAggregator це компонент фільтрації колонок під час запису.

Методи:

- writeProjectMetrics(...) записує для одного проєкту агрегований рядок і набір рядків per-class.

- writeEmptyProject(...) записує порожній summary-рядок, якщо class.csv порожній або відсутній.

- openOrCreateWorkbook(...) відкриває існуючий файл або створює новий workbook.

- writeWorkbook(...) зберігає workbook у файл.

- getOrCreateSheet(...) повертає або створює аркуш.

- nextRowIndex(...) визначає індекс наступного рядка для запису.

- createMetricsHeaderIfNeeded(...) створює заголовок агрегованого аркуша.

- createPerClassHeaderIfNeeded(...) створює заголовок аркуша per-class.

- writeBaseColumns(...) записує службові колонки No, Project, Commit Hash.

Специфікація класу «GitHubService»

Ім’я: ckmetrics.service.GitHubService.

Відповідальність: виконує git-операції з репозиторіями такі як клонування, checkout головної гілки, checkout на дату за комітом, checkout за тегом на дату.

Атрибути:

- logger це логер для службових повідомлень і резервних сценаріїв.

Методи:

- cloneRepository(String url, Path directory, boolean fetchTags) клонує репозиторій з тегами або без тегів залежно від режиму.

- checkoutMainBranchHead(Git git) перемикає на поточну HEAD головної гілки.

- checkoutMainBranchAtDate(Git git, Instant date) перемикає на останній коміт, що не пізніше заданої дати.

- checkoutLatestTagAtDate(Git git, Instant date) перемикає на останній тег на/до дати, а за відсутності тегів виконує fallback на головну гілку за датою.

- getMainBranchName(...) визначає назву головної гілки (main, master або перша доступна).

- findCommitOnOrBeforeDate(...) знаходить відповідний коміт за датою.

- findLatestTagOnOrBeforeDate(...) знаходить відповідний тег за датою або fallback на найраніший.

- resolveTagCommit(...) і resolveTagObjectId(...) визначає коміт, на який вказує тег.

Специфікація класу «CheckoutPanel»

Ім’я: ckmetrics.ui.CheckoutPanel.

Відповідальність: керує UI-вкладкою Checkout, який задає параметри запуску, синхронізує налаштування, запускає checkout у фоні та відображає прогрес і результати виконання процесу зі збору метрик.

Атрибути:

- projectDAO, typeDAO, excludedFolderDAO це компоненти для отримання типів проєктів, проєктів і виключених каталогів з БД та завантаження їх у форму.

- checkoutService це сервіс запуску checkout-логіки.

- settingsManager це менеджер збереження або відновлення налаштувань.

- projectsTablePanel, optionsPanel це UI-компоненти вкладки.

- isLoadingSettings це службовий прапорець, який тимчасово вимикає автозбереження під час завантаження налаштувань.

Методи:

- initialize() і initComponents() потрібні для ініціалізації UI та даних.

- loadTypes(), loadProjectsByType(), loadExcludedFolders() завантажують у форму список типів проєктів, список проєктів згідно обраного типу проєкту, а також список виключених каталогів.

- browseDirectory() використовується під час вибору директорії для збереження через JFileChooser.

- checkoutProjects() є початком сценарію checkout із обробкою помилок запуску.

- buildCheckoutRequest() формує CheckoutRequest з поточних значень UI.

- runCheckoutWorker(...) запускає checkout у фоновому потоці SwingWorker з подальшим відображенням результату в інтерфейсі.

- loadSettings()/saveSettings()/refresh() необхідний для синхронізації стану форми з БД.

- showInfoDialog()/showWarningDialog()/showErrorDialog() застосовується для відображення повідомлень користувачу.

Специфікація класу «ExcludedFoldersPanel»

Ім’я: ckmetrics.ui.ExcludedFoldersPanel.

Відповідальність: CRUD-панель для сутності ExcludedFolder, яка забезпечує додавання, корегування, вилучення та валідацію виключених каталогів.

Атрибути:

- folderNameField: JTextField це поле вводу назви виключеного каталогу.

- folderDAO: ExcludedFolderDAO це компонент доступу до даних, який використовується для роботи з виключеними каталогами у базі даних.

- folderNameValidator: ExcludedFolderNameValidator це компонент перевірки, який використовується для валідації коректності назви виключеного каталогу.

Методи:

- getColumnNames() повертає конфігурацію колонок таблиці.

- createInputPanel() створює форму вводу.

- onTableRowSelected(...) переносить значення вибраного рядка у форму.

- loadData() завантажує дані з БД у таблицю.

- clearFields() очищує форму.

- addEntity() додає новий каталог після валідації і перевірки дублювання. editEntity() редагує вибраний каталог після перевірок.

- deleteEntity() видаляє вибраний каталог після підтвердження.

Специфікація класу «MainFrame»

Ім’я: ckmetrics.ui.MainFrame.

Відповідальність: головне вікно застосунку, яке ініціалізує вкладки та координує оновлення панелей при перемиканні між ними.

Атрибути:

- tabbedPane: JTabbedPane це контейнер вкладок.

- typesPanel, projectsPanel, excludedFoldersPanel, checkoutPanel це основні UI-панелі застосунку.

Методи:

- MainFrame() налаштовує параметри вікна і підключає обробник закриття для HibernateUtil.shutdown().

- initComponents() створює вкладки, додає панелі та обробник зміни активної вкладки.

Специфікація класу «ProjectsPanel»

Ім’я: ckmetrics.ui.ProjectsPanel.

Відповідальність: CRUD-панель для сутності Project, яка забезпечує додавання, корегування, вилучення та валідацію проєктів.

Атрибути:

- urlField: JTextField це поле URL проєкту.

- typeComboBox: JComboBox<Type> це комбобокс для вибору типу проєкту.

- projectDAO: ProjectDAO це компонент доступу до даних, який використовується для отримання і збереження проєктів.

- typeDAO: TypeDAO це компонент доступу до даних, який використовується для завантаження типів проєктів у комбобокс.

- projectUrlValidator: ProjectUrlValidator це компонент перевірки, який використовується для валідації URL проєкту.

Методи:

- Конструктори ProjectsPanel() і пакетний конструктор з ін’єкцією залежностей створюють панель проєктів і ініціалізують потрібні компоненти.

- getColumnNames()/createInputPanel()/onTableRowSelected(...) задають колонки таблиці, створюють поля форми та заповнюють форму даними вибраного рядка.

- loadData() завантажує проєкти у таблицю.

- clearFields() очищує форму.

- loadTypes() завантажує типи проєктів у комбобокс.

- refresh() оновлює типи проєктів, таблицю і стан форми.

- addEntity() додає проєкт після валідації URL і перевірки унікальності.

- editEntity() редагує проєкт після перевірки вибору, валідації, змін і дублювання URL.

- deleteEntity() видаляє проєкт після підтвердження.

Специфікація класу «TypesPanel»

Ім’я: ckmetrics.ui.TypesPanel.

Відповідальність: CRUD-панель для сутності Type, яка забезпечує додавання, корегування, вилучення та валідацію типів проєктів.

Атрибути:

- nameField: JTextField це поле назви типу проєкту.

- typeDAO: TypeDAO це компонент доступу до даних, який використовується для роботи з типами проєктів у базі даних.

- typeNameValidator: TypeNameValidator це компонент перевірки, який використовується для валідації назви типу проєкту.

Методи:

- getColumnNames()/createInputPanel()/onTableRowSelected(...) задають колонки таблиці, створюють поля форми та заповнюють форму даними вибраного рядка.

- loadData() завантажує типи проєктів у таблицю.

- clearFields() очищує форму.

- addEntity() додає тип проєкту після валідації і перевірки дублювання.

- editEntity() редагує тип проєкту із перевіркою змін і унікальності.

- deleteEntity() видаляє тип проєкту після підтвердження користувача.

Специфікація класу «HibernateUtil»

Ім’я: ckmetrics.util.HibernateUtil.

Відповідальність: утилітний клас, який керує життєвим циклом SessionFactory Hibernate.

Атрибути:

- sessionFactory: SessionFactory це статично ініціалізована фабрика сесій для всього застосунку.

Методи:

- buildSessionFactory() створює SessionFactory з конфігурації Hibernate.

- getSessionFactory() повертає поточну фабрику сесій.

- initialize() виконує ранню перевірку успішної ініціалізації.

- shutdown() коректно закриває SessionFactory.

- HibernateUtil() є приватним конструктором із забороною створення екземплярів.

Специфікація класу «Main»

Ім’я: ckmetrics.Main.

Відповідальність: точка входу в застосунок, яка ініціалізує всю інфраструктуру та запускає головний UI.

Атрибути:

- Відсутні, тому що клас використовується лише як стартовий контейнер із методом main й не зберігає стан застосунку.

Методи:

- main(String[] args) ініціалізує Hibernate, налаштовує системний Look and Feel та запускає MainFrame у потоці інтерфейсу користувача.

Логічна модель даних програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK представлена на рисунку 2.11.

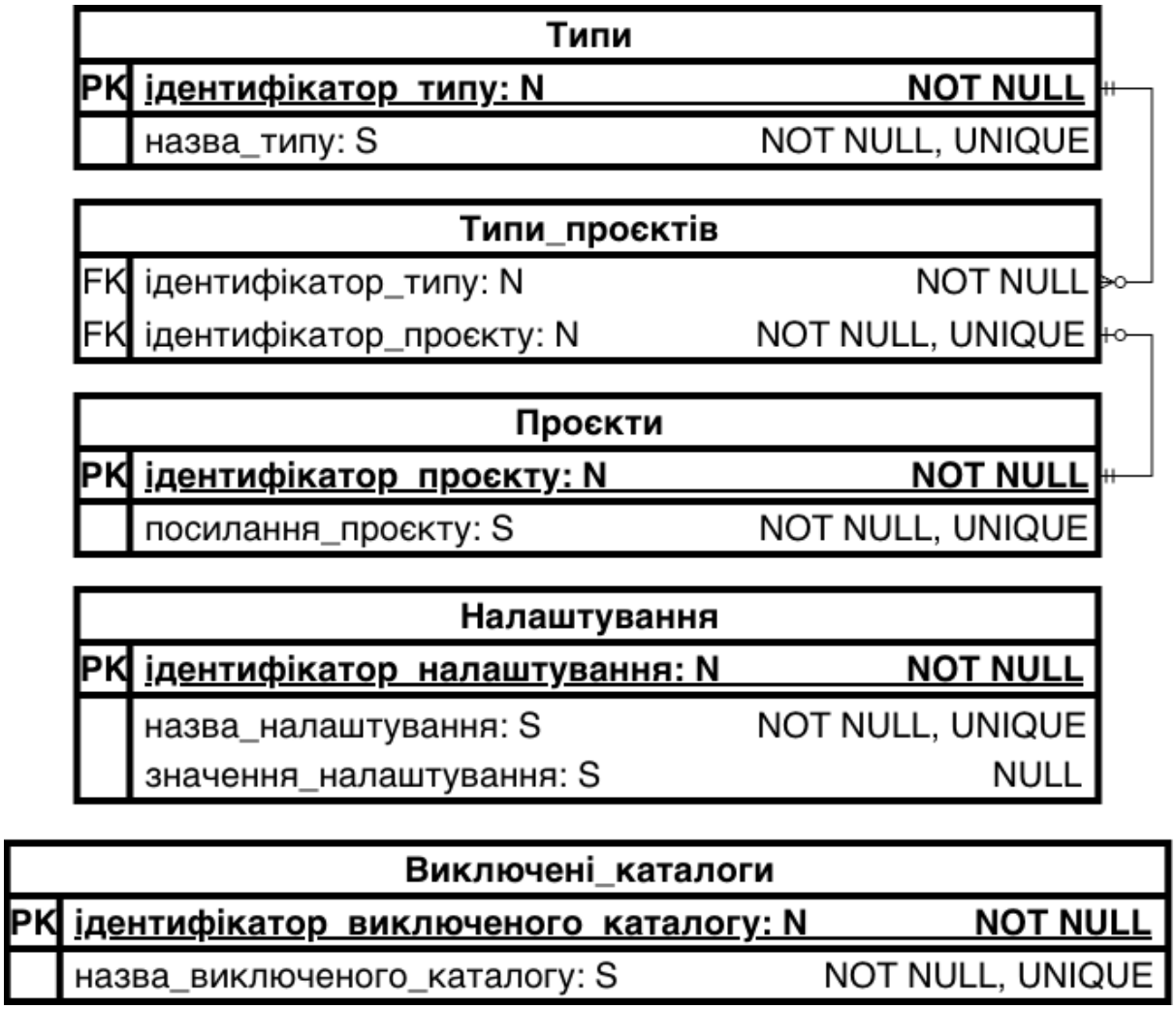


Рисунок 2.11 – Логічна модель даних програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK

2.3.3 Робочий проєкт програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK

Вибір засобів розробки було здійснено з урахуванням того, що ПЗ має бути десктопним, працювати локально без окремого серверного середовища та забезпечувати зручну взаємодію з базою даних, вихідними файлами з результатами аналізу й утилітою обчислення метрик, а також обрані програмні засоби повинні

підтримувати стабільну роботу з Java-кодом і забезпечувати можливість реалізації всіх основних функцій застосунку в межах єдиного середовища програми.

Для реалізації нашого ПЗ було обрано мову програмування Java 21, тому що вона залишається однією з провідних платформ розробки програмного забезпечення та широко використовується для побудови масштабних і надійних систем, тому використання Java у даному проєкті є доцільним через її розвинену екосистему, наявність стандартних засобів для роботи з файлами, мережею, колекціями, а також для обробки винятків і побудови графічного інтерфейсу, що дозволяє реалізувати повноцінний застосунок відповідно до поставлених вимог [1].

У ролі інструменту для обчислення метрик було використано утиліту CK, бо вона дозволяє виконати статичний аналіз Java-проєктів і визначити метрики на рівні класів та методів без потреби попередньої компіляції вихідного коду, а також її інтеграція у вигляді окремої утиліти дозволяє відокремити шар аналізу метрик від прикладної логіки нашого застосунку для автоматизації запуску аналізатора та подальшої обробки згенерованих результатів [2].

Систему керування базами даних SQLite було обрано як компактне вбудоване рішення, що не потребує окремого серверного процесу і зберігає всю базу даних у локальному файлі. Для нашого застосунку це є достатнім, тому що SQLite підтримує SQL-запити, має формат файлів, який розпізнає більшість сучасних операційних систем і добре підходить для зберігання даних про наші сутності [10].

Бібліотеку Hibernate ORM використано як ORM-рівень, що спрощує взаємодію між Java-об’єктами та таблицями бази даних, тобто цей підхід дозволяє працювати з сутностями предметної області на рівні класів, зменшуючи обсяг шаблонного SQL-коду та забезпечуючи зручніше керування збереженням, вибором й оновленням даних у локальній базі SQLite [11].

Побудову користувацького інтерфейсу було реалізовано із застосуванням бібліотеки Swing через те, що саме ця бібліотека надає стандартний набір компонентів, який є достатнім для створення подібних застосунків, а її використання у цьому проєкті є цілком виправданим, бо вона дозволяє створювати вікна, вкладки, таблиці, кнопки, поля введення та інші елементи інтерфейсу, що

необхідні для керування списками типів проєктів, проєктів та виключеними каталогами, а також параметрами checkout і запуском процесу зі збору метрик [12].

Для роботи з GitHub-репозиторіями було застосовано бібліотеку JGit, яка є Java-реалізацією системи керування версіями Git і дозволяє виконувати клонування, отримання тегів та перемикання між станами репозиторію без звернення до зовнішнього Git-клієнта, що значно підвищує автономність і надійність застосунку, бо вся логіка взаємодії з репозиторіями переноситься у код програми та краще контролюється на рівні обробки винятків і сценаріїв checkout.

Фізична модель даних програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK представлена на рисунку 2.12.

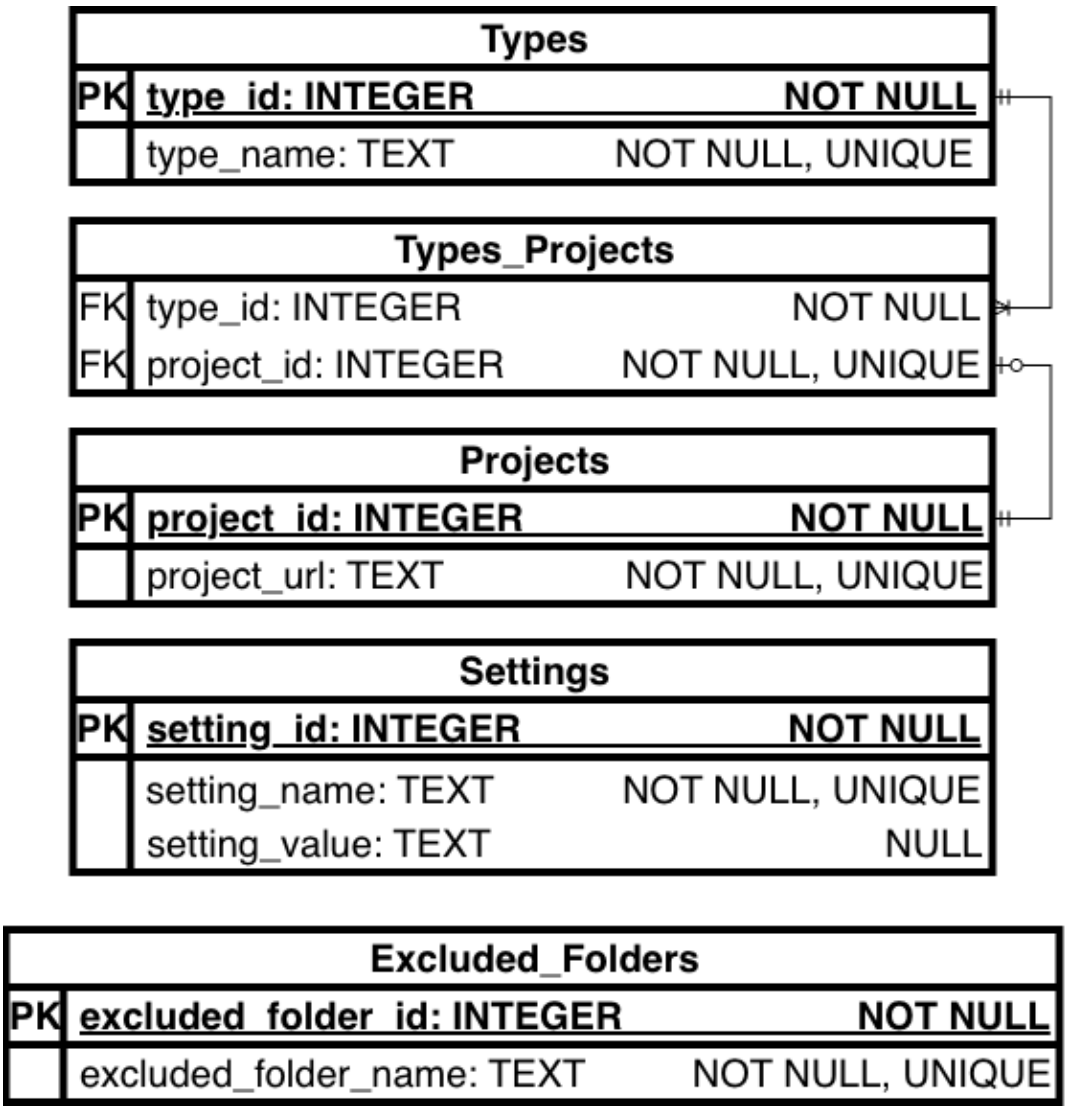


Рисунок 2.12 – Фізична модель даних програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK

Далі наведено фрагмент програми, що реалізує пошук останнього тегу репозиторію, дата коміту якого не перевищує вказану користувачем дату.

Фрагмент коду файлу GitHubService.java:

```
private Ref findLatestTagOnOrBeforeDate(Git git, Instant date)
        throws GitAPIException, IOException {
    List<Ref> tags = new ArrayList<>(git.tagList().call());
    if (tags.isEmpty()) {
        return null;
    }

    Ref earliestTag = null;
    long earliestTime = Long.MAX_VALUE;
    Ref latestEligibleTag = null;
    long latestEligibleTime = Long.MIN_VALUE;

    for (Ref tag : tags) {
        long tagCommitTime = resolveTagCommit(git.getRepository(),
tag).getCommitTime() * 1000L;
        if (tagCommitTime < earliestTime) {
            earliestTime = tagCommitTime;
            earliestTag = tag;
        }
        if (tagCommitTime <= date.toEpochMilli() && tagCommitTime >
latestEligibleTime) {
            latestEligibleTime = tagCommitTime;
            latestEligibleTag = tag;
        }
    }

    if (latestEligibleTag == null && earliestTag != null) {
        logger.warn("All tags are later than {}. The earliest tag will be
used.", date);
        return earliestTag;
    }

    return latestEligibleTag;
}
```

У цьому фрагменті коду реалізовано пошук останнього тегу GitHub-репозиторію, дата коміту якого не перевищує дату, вказану користувачем, тобто спочатку програма отримує список усіх тегів репозиторію, а потім в циклі визначає час коміту, на який посилається кожен тег і на основі цього встановлює найновіший допустимий тег та найбільш ранній тег у репозиторії. Якщо всі наявні теги виявляються пізнішими за вказану дату, метод не завершується помилкою, а використовує резервний сценарій, згідно якого повертається найбільш ранній доступний тег і формується відповідне попередження у логах програми, що

підвищує надійність роботи застосунку та забезпечує можливість подальшого виконання процесу зі збору метрик.

Тестування програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK було проведено через такі методи як:

- Складання тестових прикладів за методами «чорної скриньки» відповідно до тестових вимог на основі виділення класів еквівалентності для правильних і неправильних вхідних даних.

- Розроблення тестових прикладів для тестування класу GitHubService через його розбивку на категорії по функціональності, які були згруповані у три функціональні категорії такі як операції клонування, операції checkout головної гілки та операції checkout останнього тегу згідно вказаної дати.

Результати обох методів тестування продемонстрували наступне:

- Тести за специфікаціями достатньо добре перевіряють функціональні вимоги, але не гарантують повного проходження всіх внутрішніх шляхів виконання коду.

- Тести за методом розбивки на категорії по функціональності є ефективним інструментом для перевірки класів з кількома чітко виокремленими функціональними ролями, а розроблений набір тестів демонструє надійність класу GitHubService як складової підсистеми при підготовці даних у процесі checkout проєктів перед подальшим збором метрик.

Випробування програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK було виконано відповідно до програми та методики випробувань згідно додатку Д.

Результат проведення випробувань для кожної з функцій ПЗ повністю очікуваний і коректний. Отже, це свідчення того, що розроблене програмне забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK є повністю функціонуючим та готовим до використання, а також воно відповідає всім поставленим вимогам у технічному завданні.

3 РЕЗУЛЬТАТИ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОМАТИЗАЦІЇ ЗБОРУ МЕТРИК GITHUB ПРОЄКТІВ НА МОВІ JAVA УТИЛІТОЮ CK

Результат виконання кваліфікаційної роботи це розв’язання практичної задачі інженерії програмного забезпечення, що характеризується комплексністю та невизначеністю умов, а саме було розроблено програмне забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK.

Для досягнення даного результату було виконано аналіз предметної області, досліджено існуючі підходи та програмні засоби для збору метрик програмних проєктів, а також сформульовано постановку задачі на розробку відповідного ПЗ. Після цього було визначено вимоги до ПЗ, обрано архітектурні рішення та спроєктовано структуру майбутнього десктопного застосунку.

Програмне забезпечення написано на мові програмування Java з використанням відповідних бібліотек і засобів розробки, після чого виконано його тестування та випробування відповідно до програми й методики випробувань.

Розмір програмного забезпечення складає 3681 рядок коду, а розмір скомпільованого jar-файлу становить 63,7 МБ.

Згенерований XLSX-звіт із метриками для обраного типу проєкту представлено на рисунку 3.1.

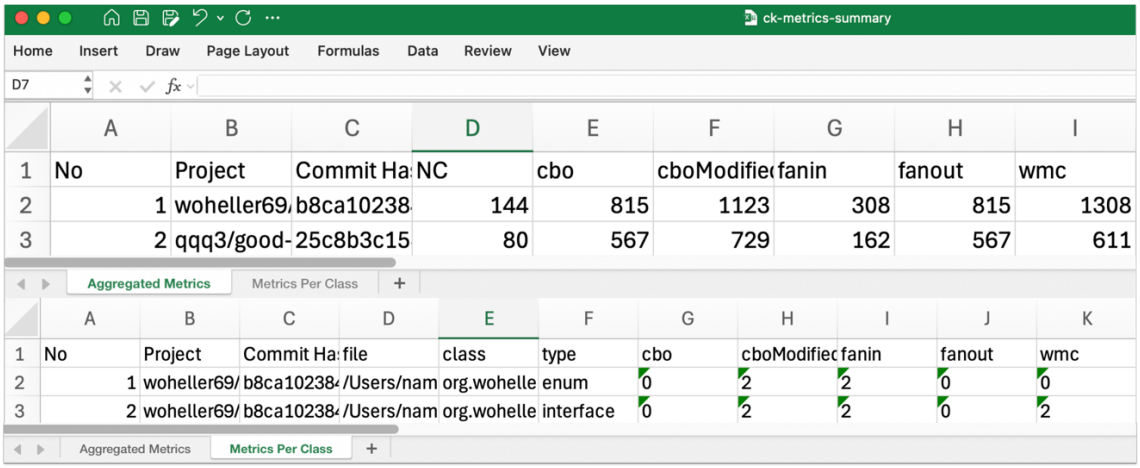


Рисунок 3.1 – Згенерований XLSX-звіт із метриками

4 ОХОРОНА ПРАЦІ

4.1 Правові та організаційні основи охорони праці під час розробки ПЗ

Згідно із Законом України «Про охорону праці», охорона праці є системою правових, соціально-економічних, організаційно-технічних, санітарно-гігієнічних і лікувально-профілактичних заходів та засобів, спрямованих на збереження життя, здоров’я і працездатності людини у процесі трудової діяльності [13]. Під час розробки програмного забезпечення питання охорони праці мають важливе значення, через те що діяльність програміста пов’язана з тривалою роботою за ПК, високою концентрацією уваги та постійним навантаженням.

Відповідно до Кодексу законів про працю України, на всіх підприємствах, в установах та організаціях повинні створюватися безпечні й нешкідливі умови праці [14]. Це стосується і робочого місця розробника програмного забезпечення, який працює з комп’ютерною технікою, локальними базами даних, мережевими сервісами та програмними інструментами.

Організація охорони праці в межах такої діяльності повинна включати проведення інструктажів, забезпечення належного технічного стану обладнання, контроль умов праці, підтримання відповідного рівня освітлення, мікроклімату та електробезпеки. Важливим є також навчання працівників безпечним методам роботи та організація режиму праці й відпочинку.

Закон України «Про охорону праці» визначає, що його дія поширюється на всіх юридичних і фізичних осіб, які використовують найману працю, а також на всіх працюючих [13]. Це означає, що вимоги охорони праці є обов’язковими і для тих видів діяльності, які не пов’язані з важким фізичним навантаженням, але передбачають тривалу роботу з технічними засобами. У випадку розробки програмного забезпечення основна увага приділяється умовам праці на робочому місці, безпечному користуванню обладнанням, запобіганню перевтомі та підтриманню належного виробничого середовища.

Окремо законодавство покладає обов’язок щодо створення належних умов праці саме на роботодавця. Відповідно до статті 13 Закону України «Про охорону праці», роботодавець повинен створити на робочому місці умови праці згідно з нормативно-правовими актами, а також забезпечити дотримання прав працівників у галузі охорони праці [13]. Для цього роботодавець має організувати систему управління охороною праці, призначити відповідальних осіб, розробити необхідні інструкції, вживати профілактичних заходів, контролювати технічний стан обладнання та своєчасно усувати причини, що можуть призводити до погіршення умов праці.

З цього всього можна зробити висновок, що правові та організаційні основи охорони праці під час розробки програмного забезпечення полягають у створенні таких умов, за яких забезпечується безпечне використання ПК, знижується вплив шкідливих факторів, дотримуються права працівника та підтримується належний рівень його працездатності.

4.2 Вимоги до робочого місця під час роботи з ПК

Робоче місце працівника, який розробляє програмне забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK, повинно відповідати вимогам щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями [15].

До складу такого робочого місця входять персональний комп’ютер, монітор, клавіатура, маніпулятор такий як «миша», робочий стіл, крісло, освітлення, а також навколишнє виробниче середовище. Робочий стіл повинен забезпечувати достатню площу для розміщення обладнання, а крісло повинно підтримувати правильну робочу позу.

Екран монітора має бути розташований саме так, щоб користувач міг працювати без надмірного напруження зору. Зображення на екрані повинно бути чітким, стабільним і достатньо контрастним. Необхідно уникати появи відблисків,

а освітлення робочого місця має бути організоване так, щоб не створювати додаткового навантаження на очі.

Особливу увагу слід приділяти ергономіці. Клавіатура має бути зручною для введення, а розташування всіх елементів робочого місця повинно мінімізувати зайве навантаження на кисті, руки, шию та спину. Робоча поза користувача має бути природною, а положення тіла має не призводити до швидкого стомлення [15].

Згідно з вимогами щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями, робоче місце потрібно проєктувати із огляду на те, що працівник повинен мати достатній простір для зміни положення тіла та виконання необхідних рухів. Усі елементи робочої станції та їх розміщення мають відповідати ергономічним, антропологічним і психофізіологічним вимогам, а також характеру виконуваної роботи. Для діяльності програміста це особливо важливо, тому що значна частина робочого часу проходить у сидячому положенні та потребує тривалого зосередження уваги.

Робоче місце повинно бути обладнане відповідно до вимог безпеки, ергономіки та зручності виконання роботи. Робочий стіл або робоча поверхня мають бути достатнього розміру та придатними для раціонального розміщення екрана, клавіатури, документів і допоміжного устаткування, а робоче крісло повинно бути стійким і забезпечувати можливість зміни положення тіла шляхом регулювання основних параметрів. Екранний пристрій має забезпечувати чітке, стабільне і контрастне зображення, можливість налаштування яскравості та контрастності, а також не створювати відблисків і інших факторів, що можуть викликати дискомфорт або підвищене зорове навантаження [15].

Підсумовуючи, правильна організація робочого місця є необхідною умовою безпечної та продуктивної роботи під час розробки програмного забезпечення, у зв'язку з тим що саме від неї залежить збереження здоров'я працівника, зниження втоми та підтримання належної працездатності.

4.3 Заходи зменшення впливу шкідливих факторів при роботі за ПК

Під час роботи за персональним комп’ютером на працівника можуть впливати такі шкідливі фактори, як напруження зору, статичне навантаження, монотонність праці, психоемоційне перевантаження та вплив несприятливих умов виробничого середовища. Для зменшення їх впливу необхідно реалізувати комплекс організаційних і технічних заходів.

Перш за все необхідно дотримуватися раціонального режиму праці та відпочинку, тобто перерви під час роботи з екранними пристроями дозволяють зменшити втому, знизити навантаження на зір і змінити положення власного тіла. Під час таких перерв доцільно виконувати вправи для очей, кистей рук, шиї та спини [15].

Важливим профілактичним заходом є підтримання належних параметрів освітлення та мікроклімату у приміщенні. Регулярне провітрювання, вологе прибирання, підтримання комфортної температури та вологості позитивно впливають на самопочуття працівника і зменшують втому.

Не менш важливо забезпечити електробезпеку. Все обладнання повинно працювати коректно, а користувач не повинен працювати з пошкодженими кабелями, розетками чи іншими неробочими елементами електромережі. У разі виявлення таких дефектів, робота повинна бути припинена до усунення проблеми.

Відповідно до Наказу № 207, роботодавець повинен проінформувати працівників щодо підпису про умови праці, наявні небезпечні та шкідливі фактори на робочому місці, а також про можливі наслідки їх впливу на здоров’я [15]. Окрім цього, до початку роботи з екранними пристроями працівник має пройти навчання та перевірку знань з питань охорони праці і безпечного використання такого обладнання. Такий підхід дає змогу виконати формальні вимоги законодавства, а також знизити ризик неправильного користування з технікою.

Суттєвим заходом профілактики є організація внутрішніх регламентованих перерв у межах робочої зміни [15], бо вони мають надаватися за рахунок робочого часу та спрямовані на зниження напруженості трудового процесу. Для програміста

цей пункт особливо актуальний, тому що робота за комп’ютером часто поєднує високу концентрацію уваги, обробку значного обсягу інформації та тривале перебування в одному положенні. Крім того, у разі потреби роботодавець має організовувати дослідження умов праці для виявлення факторів, які можуть негативно впливати на зір, фізичний стан чи психоемоційне навантаження працівника [15].

До щоденних практичних заходів безпеки належать очищення екранних пристроїв від пилу перед початком роботи, відключення обладнання від електромережі після завершення роботи та негайне знеструмлення у разі аварійної ситуації. Не допускається виконувати технічне обслуговування або ремонт екранних пристроїв безпосередньо під час роботи, самовільно змінювати їх конструкцію чи працювати з обладнанням, у якого з’являється нестабільне зображення або інші ознаки несправності [15]. Дотримання цих вимог є обов’язковим елементом безпечної експлуатації техніки.

З урахуванням положень Закону України «Про охорону праці» та Кодексу законів про працю України можна зробити висновок, що зменшення впливу шкідливих факторів залежить не лише від поведінки працівника, а й від належної організації охорони праці з боку роботодавця [13, 14]. Безпечні умови праці, справне обладнання, інформування працівника, навчання, перерви та контроль факторів виробничого середовища повинні розглядатися як єдина система профілактики.

Отже, впровадження заходів щодо зменшення впливу шкідливих факторів під час роботи за персональним комп’ютером є обов’язковою складовою безпечної розробки та експлуатації програмного забезпечення. Реалізація таких заходів сприяє збереженню здоров’я працівника, зниженню рівня втоми, підтриманню працездатності та підвищенню загальної ефективності роботи.

ВИСНОВКИ

У кваліфікаційній роботі розв’язано практичну задачу інженерії програмного забезпечення, що характеризується комплексністю та невизначеністю умов, а саме було розроблено програмне забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK.

Для розв’язання поставленої задачі було виконано наступні завдання:

- виконано аналіз практичної задачі інженерії програмного забезпечення щодо розробки програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK;
- проаналізовано існуючі програмні рішення та обґрунтовано доцільність розробки власного програмного забезпечення;
- сформульовано постановку задачі на розробку програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK;
- проведено аналіз вимог до програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK;
- розроблено архітектурні рішення програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK;
- виконано проєкт програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK;
- реалізовано тестування та випробування розробленого програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK;
- описано всю необхідну документацію для програми.

Результати тестування та випробувань підтвердили коректність реалізації основних функціональних вимог програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK, його повну працездатність і відповідність вимогам технічного завдання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Andrea Griffiths. Why Java endures: The foundation of modern enterprise development. URL: https://github.blog/developer-skills/why-java-endures-the-foundation-of-modern-enterprise-development/ (дата звернення 12.02.2026)
2. Code metrics for Java code by means of static analysis. URL: https://github.com/mauricioaniche/ck (дата звернення 12.02.2026)
3. Git Clone. URL: https://github.com/git-guides/git-clone (дата звернення 12.02.2026)
4. Gal Elbaz. Static Code Analysis. URL: https://www.oligo.security/academy/static-code-analysis (дата звернення 12.02.2026)
5. JArchitect. URL: https://www.codergears.com/ (дата звернення 25.03.2026)
6. Understand by SciTools. URL: https://scitools.com/ (дата звернення 25.03.2026)
7. OpenClover. URL: https://openclover.org/ (дата звернення 25.03.2026)
8. A reproducible and extensible pipeline for large-scale extraction of software quality metrics from open-source Java projects. URL: https://github.com/chesnokoff/josseph (дата звернення 25.03.2026)
9. Anand Butani. 5 essential patterns of software architecture. URL: https://www.redhat.com/en/blog/5-essential-patterns-software-architecture (дата звернення 25.03.2026)
10. Kent C. Dodds. Why you should probably be using SQLite. URL: https://www.epicweb.dev/why-you-should-probably-be-using-sqlite (дата звернення 25.03.2026)
11. Hibernate ORM. URL: https://hibernate.org/orm/ (дата звернення 25.03.2026)
12. Valli Kumari Vatsavayi. Basics of Java Swing. URL: https://pressbooks.pub/javaprogramming/chapter/swing-java-programming/ (дата звернення 25.03.2026)
13. Закон України «Про охорону праці». URL: https://zakon.rada.gov.ua/laws/show/2694-12/ed20250912#Text (дата звернення 14.04.2026)

14. Кодекс законів про працю України. URL: https://zakon.rada.gov.ua/laws/show/322-08#Text (дата звернення 14.04.2026)
15. Наказ №207 від 14.02.2018 «Про затвердження Вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями». URL: https://zakon.rada.gov.ua/laws/show/z0508-18#Text (дата звернення 14.04.2026)

ДОДАТОК А – ТЕХНІЧНЕ ЗАВДАННЯ НА РОЗРОБКУ ДЕСКТОПНОГО ПЗ ДЛЯ АВТОМАТИЗАЦІЇ ЗБОРУ МЕТРИК GITHUB ПРОЄКТІВ НА МОВІ JAVA УТИЛІТОЮ CK

1 Вступ

1.1 Назва програмного забезпечення

Програмне забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK.

1.2 Сфера застосування

Програмне забезпечення призначене для використання в галузі інженерії програмного забезпечення.

2 Підстава для розробки програмного забезпечення

Розробка ПЗ виконується на підставі завдання на кваліфікаційну роботу, яке видано кафедрою ПЗАС НУК імені адмірала Макарова.

3 Призначення розробки

3.1 Функціональне призначення

Функціональним призначенням ПЗ є автоматизація процесу збору метрик GitHub проєктів на мові Java за допомогою утиліти CK. Програмне забезпечення призначене для клонування репозиторіїв, вибору потрібного стану для коду проєктів, запуску утиліти CK, обробки отриманих метрик, а також формування результуючого вихідного файлу та забезпечення зручного користувацького інтерфейсу для взаємодії із застосунком.

3.2 Експлуатаційне призначення

Це ПЗ призначене для використання на персональних комп’ютерах.

4 Технічні вимоги до програми

4.1 Вимоги до функціональних характеристик

4.1.1 Вимоги до переліку виконуваних функцій

ПЗ повинно забезпечувати виконання описаних нижче функцій:

- Можливість додавання, корегування, вилучення інформації про тип проєкту та відображення інформації про типи проєктів, тобто наявність дій CRUD для даної сутності.

- Можливість додавання, корегування, вилучення інформації про проєкт та відображення інформації про проєкти, тобто наявність дій CRUD для даної сутності.

- Можливість додавання, корегування, вилучення інформації про виключений каталог та відображення інформації про виключені каталоги, тобто наявність дій CRUD для даної сутності.

- Формування метрик.

4.1.2 Вимоги до організації вхідних та вихідних даних

- Для додавання інформації про тип проєкту:

Вхідні дані це назва типу проєкту.

Вихідні дані це новий запис про тип проєкту у таблиці БД «Типи» та системне повідомлення про успішне додавання.

- Для корегування інформації про тип проєкту:

Вхідні дані це обраний рядок з таблиці «Типи» і заповнені нові дані щодо його назви.

Вихідні дані це оновлений запис про тип проєкту у таблиці БД «Типи» зі зміненою інформацією та системне повідомлення про успішне корегування.

- Для вилучення інформації про тип проєкту:

Вхідні дані це обраний рядок з таблиці «Типи» і підтвердження дії в діалоговому вікні.

Вихідні дані це вилучений запис про тип проєкту у таблиці БД «Типи» та системне повідомлення про успішне вилучення, якщо даний тип проєкту ще не належить жодному проєкту.

- Для відображення інформації про типи проєктів:

Вхідні дані це відкрита вкладка «Типи».

Вихідні дані це відображення таблиці зі списком типів проєктів та заповненими даними.

- Для додавання інформації про проєкт:

Вхідні дані це посилання проєкту та вибір типу проєкту зі списку або його відсутність.

Вихідні дані це новий запис про проєкт у таблицях БД «Проєкти» і «Типи проєктів» та системне повідомлення про успішне додавання.

- Для корегування інформації про проєкт:

Вхідні дані це обраний рядок з таблиці «Проєкти» і заповнені нові дані щодо його посилання та типу проєкту або його відсутності.

Вихідні дані це оновлений запис про проєкт у таблицях БД «Проєкти» і «Типи проєктів» та системне повідомлення про успішне корегування.

- Для вилучення інформації про проєкт:

Вхідні дані це обраний рядок з таблиці «Проєкти» і підтвердження дії в діалоговому вікні.

Вихідні дані це вилучений запис про проєкт у таблицях БД «Проєкти» і «Типи проєктів» та системне повідомлення про успішне вилучення.

- Для відображення інформації про проєкти:

Вхідні дані це відкрита вкладка «Проєкти».

Вихідні дані це відображення таблиці зі списком проєктів та заповненими даними.

- Для додавання інформації про виключений каталог:

Вхідні дані це назва виключеного каталогу.

Вихідні дані це новий запис про виключений каталог у таблиці БД «Виключені каталоги» та системне повідомлення про успішне додавання.

- Для корегування інформації про виключений каталог:

Вхідні дані це обраний рядок з таблиці «Виключені каталоги» і заповнені нові дані щодо його назви.

Вихідні дані це оновлений запис про виключений каталог у таблиці БД «Виключені каталоги» зі зміненою інформацією та системне повідомлення про успішне корегування.

- Для вилучення інформації про виключений каталог:

Вхідні дані це обраний рядок з таблиці «Виключені каталоги» і підтвердження дії в діалоговому вікні.

Вихідні дані це вилучений запис про виключений каталог у таблиці БД «Виключені каталоги» та системне повідомлення про успішне вилучення.

- Для відображення інформації про виключені каталоги:

Вхідні дані це відкрита вкладка «Виключені каталоги».

Вихідні дані це відображення таблиці зі списком виключених каталогів та заповненими даними.

- Для формування метрик:

Вхідні дані це обраний тип проєкту, з яким пов’язано хоча б один проєкт, задані необхідні виключені каталоги або їх відсутність, визначена гілка коду, вказаний шлях до директорії для збереження вихідного файлу з агрегованими даними.

Вихідні дані це згенерований файл метрик формату XLSX, що містить у перших трьох стовпцях No проєкту, назву і вендора Project та його Commit Hash, а потім стовпці із такими метриками як NC, cbo, cboModified, fanin, fanout, wmc, dit, noc, rfc, lcom, totalMethodsQty, staticMethodsQty, publicMethodsQty, privateMethodsQty, protectedMethodsQty, defaultMethodsQty, visibleMethodsQty, abstractMethodsQty, finalMethodsQty, synchronizedMethodsQty, totalFieldsQty,

staticFieldsQty, publicFieldsQty, privateFieldsQty, protectedFieldsQty, defaultFieldsQty, finalFieldsQty, synchronizedFieldsQty, nosi, loc, returnQty, loopQty, comparisonsQty, tryCatchQty, parenthesizedExpsQty, stringLiteralsQty, numbersQty, assignmentsQty, mathOperationsQty, variablesQty, maxNestedBlocksQty, anonymousClassesQty, innerClassesQty, lambdasQty, uniqueWordsQty, logStatementsQty, у вказаній директорії.

4.2 Вимоги до надійності

Дане ПЗ має працювати стабільно під час обробки великої кількості проєктів і не переривати загальний процес у разі локальних збоїв, а також система повинна коректно обробляти помилки клонування GitHub-репозиторіїв, невалідні URL та збої під час запуску Git, JGit, і ck.jar, надаючи зрозуміле текстове повідомлення користувачу й продовжуючи обробку інших репозиторіїв.

4.3 Умови експлуатації

4.3.1 Кліматичні умови експлуатації

Кліматичні умови експлуатації повністю відповідають кліматичним умовам експлуатації технічних засобів.

4.3.2 Умови до чисельності та кваліфікації користувачів

Програмне забезпечення розраховане на одного або декількох користувачів, які мають базові навички роботи з ПК, вміють працювати з файлами та каталогами. Спеціальних вимог до кваліфікації не висувається, бо інтерфейс ПЗ має бути інтуїтивно зрозумілим. Однак є перевагою, якщо користувачі мають загальне уявлення про GitHub-репозиторії і метрики програмного коду.

4.4 Вимоги до складу і параметрів технічних засобів

Операційна система: macOS, Linux та Windows.

Процесор: 4 ядра або більше із частотою щонайменше 3 ГГц.

Оперативна пам’ять: мінімум 4 ГБ.

Жорсткий диск: не менше 1 ГБ вільного простору для тимчасового збереження клонованих репозиторіїв і формування підсумкових звітів.

Інтернет: зі швидкістю прийнамні 5Мбіт/с для стабільної роботи.

Монітор: з роздільною здатністю не менше 1920×1080 для зручної роботи з графічним інтерфейсом програмного забезпечення.

Клавіатура та миша: стандартні пристрої введення, що необхідні для повноцінної взаємодії з desktop-застосунком.

4.5 Вимоги до інформаційної та програмної сумісності

Операційна система: macOS (Sequoia або Tahoe), Linux (Ubuntu 22.04 або більш нова версія) чи Windows (Windows 10 версії 20H2 або новіша версія) із підтримкою роботи Java 21.

Середовище виконання: JRE / JDK версії 21.

Мова програмування: Java версії 21.

Система керування збіркою проєкту: Apache Maven.

Утиліта збору метрик: CK версії 0.7.1-SNAPSHOT у форматі jar-файлу.

Утиліта для роботи з GitHub-репозиторіями: JGit версії 7.3.0.202506031305-r.

Система керування базами даних: SQLite.

JDBC-драйвер для роботи з базою даних: sqlite-jdbc версії 3.51.1.0.

ORM-фреймворк для взаємодії з базою даних: Hibernate ORM версії 7.2.6.Final.

Бібліотека для роботи з Excel-файлами формату XLSX: Apache POI та Apache POI OOXML версії 5.5.1.

4.6 Вимоги до маркування та упаковки

Вимоги до маркування та упаковки відсутні.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання відсутні.

4.8 Спеціальні вимоги

Спеціальні вимоги відсутні.

5 Вимоги до програмної документації

Склад програмної документації має включати в себе такі частини як технічне завдання, текст програми, текст із керівництвом користувача, текст опису програми, саму програму та методику її випробувань.

6 Техніко-економічні показники

У кваліфікаційній роботі техніко-економічні показники не розраховуються.

7 Стадії та етапи розробки

Етапи розробки ПЗ наведено в таблиці А.1.

Таблиця А.1 – Етапи розробки програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK

Номер	Назва етапу роботи	Термін виконання
1	Аналіз практичної задачі інженерії ПЗ	06.04.2026
2	Аналіз вимог до ПЗ	13.04.2026
3	Розробка архітектури ПЗ	20.04.2026
4	Розробка проєкту ПЗ	04.05.2026
5	Реалізація та тестування ПЗ	11.05.2026

8 Порядок контролю та приймання

Контроль здійснюється керівником кваліфікаційної роботи на кожному з етапів розробки ПЗ. Приймання виконується за програмою та методикою випробувань. Випробування проводяться в зазначений термін.

ДОДАТОК Б – ТЕКСТ ПЗ ДЛЯ АВТОМАТИЗАЦІЇ ЗБОРУ МЕТРИК GITHUB ПРОЄКТІВ НА МОВІ JAVA УТИЛІТОЮ CK

Текст файлу Main.java

```
package ckmetrics;

import ckmetrics.ui.MainFrame;
import ckmetrics.util.HibernateUtil;

import javax.swing.*;

/**
 * Запускає застосунок для управління збором метрик GitHub-проєктів
 */
public class Main {

    /**
     * Головний метод для запуску застосунку
     *
     * @param args аргументи командного рядка
     */
    public static void main(String[] args) {
        // Ініціалізація Hibernate
        HibernateUtil.initialize();

        // Встановлення системного стилю інтерфейсу
        try {

UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch (Exception e) {
            e.printStackTrace();
        }

        // Створення та відображення головного вікна
        SwingUtilities.invokeLater(() -> {
            MainFrame frame = new MainFrame();
            frame.setVisible(true);
        });
    }
}
```

Текст файлу dao/ExcludedFolderDAO.java

```
package ckmetrics.dao;

import ckmetrics.model.ExcludedFolder;

import java.util.List;
```

```
/**
 * Виконує операції доступу до даних для сутності ExcludedFolder
 */
public class ExcludedFolderDAO extends BaseDAO<ExcludedFolder> {

    /**
     * Створює DAO для виключених каталогів
     */
    public ExcludedFolderDAO() {
        super(ExcludedFolder.class);
    }

    /**
     * Отримує всі виключені каталоги з бази даних
     *
     * @return список виключених каталогів
     */
    public List<ExcludedFolder> getAll() {
        return executeQuery(session ->
                session.createQuery("FROM ExcludedFolder ef ORDER BY
ef.id", ExcludedFolder.class)
                        .getResultList()
        );
    }

    /**
     * Шукає виключений каталог за назвою
     *
     * @param name назва каталогу
     * @return виключений каталог, якщо знайдено, або null
     */
    public ExcludedFolder findByName(String name) {
        return executeQuery(session ->
                session.createQuery("FROM  ExcludedFolder  ef  WHERE
ef.folderName = :name", ExcludedFolder.class)
                        .setParameter("name", name)
                        .uniqueResultOptional()
                        .orElse(null)
        );
    }
}
```

Текст файлу dao/ProjectDAO.java

```
package ckmetrics.dao;

import ckmetrics.model.Project;
import ckmetrics.model.Type;

import java.util.List;

/**
 * Виконує операції доступу до даних для сутності Project
 */
```

```
public class ProjectDAO extends BaseDAO<Project> {

    /**
     * Створює DAO для проєктів
     */
    public ProjectDAO() {
        super(Project.class);
    }

    /**
     * Зберігає або оновлює проєкт у базі даних
     * Перевизначає метод для обробки зв'язку з Type
     *
     * @param project проєкт для збереження або оновлення
     */
    @Override
    public void saveOrUpdate(Project project) {
        executeInTransaction(session -> {
            // Перезавантаження Type, якщо він є
            if (project.getType() != null && project.getType().getId()
!= null) {
                Type      type      =      session.find(Type.class,
project.getType().getId());
                project.setType(type);
            }
            session.merge(project);
        });
    }

    /**
     * Отримує всі проєкти з бази даних
     *
     * @return список проєктів
     */
    public List<Project> getAll() {
        return executeQuery(session ->
                session.createQuery("FROM Project p ORDER BY p.id",
Project.class)
                        .getResultList()
        );
    }

    /**
     * Шукає проєкт за URL
     *
     * @param url URL проєкту
     * @return проєкт, якщо знайдено, або null
     */
    public Project findByUrl(String url) {
        return executeQuery(session ->
                session.createQuery("FROM Project p WHERE p.url =
:url", Project.class)
                        .setParameter("url", url)
```

```
                        .uniqueResultOptional()
                        .orElse(null)
        );
    }

    /**
     * Отримує проєкти за типом
     *
     * @param type тип проєкту
     * @return список проєктів
     */
    public List<Project> getByType(Type type) {
        return executeQuery(session ->
                session.createQuery("FROM Project p WHERE p.type.id =
:typeId ORDER BY p.id", Project.class)
                        .setParameter("typeId", type.getId())
                        .getResultList()
        );
    }

}
```

Текст файлу dao/SettingDAO.java

```
package ckmetrics.dao;

import ckmetrics.model.Setting;

/**
 * Виконує операції доступу до даних для сутності Setting
 */
public class SettingDAO extends BaseDAO<Setting> {

    /**
     * Створює DAO для налаштувань
     */
    public SettingDAO() {
        super(Setting.class);
    }

    /**
     * Шукає налаштування за назвою
     *
     * @param name назва налаштування
     * @return налаштування, якщо знайдено, або null
     */
    public Setting findByName(String name) {
        return executeQuery(session ->
                session.createQuery("FROM Setting s WHERE s.name =
:name", Setting.class)
                        .setParameter("name", name)
                        .uniqueResultOptional()
                        .orElse(null)
        );
```

```
    }

    /**
     * Повертає значення налаштування за назвою
     *
     * @param name назва налаштування
     * @return значення налаштування, якщо знайдено, або null
     */
    public String getSettingValue(String name) {
        Setting setting = findByName(name);
        return setting != null ? setting.getValue() : null;
    }

    /**
     * Встановлює значення налаштування
     * Створює нове налаштування, якщо воно не існує
     *
     * @param name назва налаштування
     * @param value значення налаштування
     */
    public void setSettingValue(String name, String value) {
        Setting setting = findByName(name);
        if (setting == null) {
            setting = new Setting(name, value);
        } else {
            setting.setValue(value);
        }
        saveOrUpdate(setting);
    }
}
```

Текст файлу dao/TypeDAO.java

```
package ckmetrics.dao;

import ckmetrics.model.Type;

import java.util.List;

/**
 * Виконує операції доступу до даних для сутності Type
 */
public class TypeDAO extends BaseDAO<Type> {

    /**
     * Створює DAO для типів проєктів
     */
    public TypeDAO() {
        super(Type.class);
    }

    /**
     * Отримує всі типи з бази даних
     *
```

```
     * @return список типів
     */
    public List<Type> getAll() {
        return executeQuery(session ->
                session.createQuery("FROM Type t ORDER BY t.id",
Type.class)
                        .getResultList()
        );
    }

    /**
     * Шукає тип за назвою
     *
     * @param name назва типу
     * @return тип, якщо знайдено, або null
     */
    public Type findByName(String name) {
        return executeQuery(session ->
                session.createQuery("FROM Type t WHERE t.name =
:name", Type.class)
                        .setParameter("name", name)
                        .uniqueResultOptional()
                        .orElse(null)
        );
    }

    /**
     * Видаляє тип, якщо він не використовується жодним проєктом
     *
     * @param type тип для видалення
     * @return true, якщо видалення успішне
     */
    @Override
    public boolean delete(Type type) {
        Long projectCount = executeQuery(session ->
                session.createQuery(
                                "SELECT COUNT(p.id) FROM Project p
WHERE p.type.id = :typeId", Long.class)
                        .setParameter("typeId", type.getId())
                        .uniqueResult()
        );

        if (projectCount != null && projectCount > 0) {
            return false;
        }
        return super.delete(type);
    }
}
```

Текст файлу model/ExcludedFolder.java

```
package ckmetrics.model;

import jakarta.persistence.*;
import java.util.Objects;

/**
 * Представляє сутність виключеного каталогу
 */
@Entity
@Table(name = "Excluded_Folders")
public class ExcludedFolder {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "excluded_folder_id")
    private Long id;

    @Column(name = "excluded_folder_name", nullable = false, unique =
true)
    private String folderName;

    /**
     * Створює порожній виключений каталог
     */
    public ExcludedFolder() {
    }

    /**
     * Конструктор з параметром
     *
     * @param folderName назва виключеного каталогу
     */
    public ExcludedFolder(String folderName) {
        this.folderName = folderName;
    }

    /**
     * Повертає ідентифікатор виключеного каталогу
     *
     * @return ідентифікатор каталогу
     */
    public Long getId() {
        return id;
    }

    /**
     * Встановлює ідентифікатор виключеного каталогу
     *
     * @param id ідентифікатор каталогу
     */
    public void setId(Long id) {
        this.id = id;
```

```
}

/**
 * Повертає назву виключеного каталогу
 *
 * @return назва каталогу
 */
public String getFolderName() {
    return folderName;
}

/**
 * Встановлює назву виключеного каталогу
 *
 * @param folderName назва каталогу
 */
public void setFolderName(String folderName) {
    this.folderName = folderName;
}

/**
 * Повертає рядкове представлення виключеного каталогу
 *
 * @return назва каталогу
 */
@Override
public String toString() {
    return folderName;
}

/**
 * Порівнює цей об'єкт з іншим на рівність
 *
 * @param o об'єкт для порівняння
 * @return true, якщо об'єкти рівні
 */
@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;

    ExcludedFolder excludedfolder = (ExcludedFolder) o;

    if (!Objects.equals(id, excludedfolder.id)) return false;
    return Objects.equals(folderName, excludedfolder.folderName);
}

/**
 * Обчислює хеш-код для об'єкта
 *
 * @return хеш-код
 */
@Override
```

```
    public int hashCode() {
        int result = id != null ? id.hashCode() : 0;
        result = 31 * result + (folderName != null ?
folderName.hashCode() : 0);
        return result;
    }
}
```

Текст файлу model/Project.java

```
package ckmetrics.model;

import jakarta.persistence.*;
import java.util.Objects;

/**
 * Представляє сутність GitHub-проєкту
 */
@Entity
@Table(name = "Projects")
public class Project {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "project_id")
    private Long id;

    @Column(name = "project_url", nullable = false, unique = true)
    private String url;

    @ManyToOne(fetch = FetchType.EAGER)
    @JoinTable(
            name = "Types_Projects",
            joinColumns = @JoinColumn(name = "project_id"),
            inverseJoinColumns = @JoinColumn(name = "type_id"),
            uniqueConstraints = @UniqueConstraint(columnNames =
{"project_id"})
    )
    private Type type;

    /**
     * Створює порожній проєкт
     */
    public Project() {
    }

    /**
     * Створює проєкт з URL і типом
     *
     * @param url URL GitHub-репозиторію
     * @param type тип проєкту
     */
    public Project(String url, Type type) {
        this.url = url;
```

```
    this.type = type;
}

/**
 * Повертає ідентифікатор проєкту
 *
 * @return ідентифікатор проєкту
 */
public Long getId() {
    return id;
}

/**
 * Встановлює ідентифікатор проєкту
 *
 * @param id ідентифікатор проєкту
 */
public void setId(Long id) {
    this.id = id;
}

/**
 * Повертає URL проєкту
 *
 * @return URL GitHub-репозиторію
 */
public String getUrl() {
    return url;
}

/**
 * Встановлює URL проєкту
 *
 * @param url URL GitHub-репозиторію
 */
public void setUrl(String url) {
    this.url = url;
}

/**
 * Повертає тип проєкту
 *
 * @return тип проєкту
 */
public Type getType() {
    return type;
}

/**
 * Встановлює тип проєкту
 *
 * @param type тип проєкту
 */
```

```
    public void setType(Type type) {
        this.type = type;
    }

    /**
     * Повертає рядкове представлення проєкту
     *
     * @return URL проєкту
     */
    @Override
    public String toString() {
        return url;
    }

    /**
     * Порівнює цей об'єкт з іншим на рівність
     *
     * @param o об'єкт для порівняння
     * @return true, якщо об'єкти рівні
     */
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        Project project = (Project) o;

        if (!Objects.equals(id, project.id)) return false;
        return Objects.equals(url, project.url);
    }

    /**
     * Обчислює хеш-код для об'єкта
     *
     * @return хеш-код
     */
    @Override
    public int hashCode() {
        int result = id != null ? id.hashCode() : 0;
        result = 31 * result + (url != null ? url.hashCode() : 0);
        return result;
    }
}
```

Текст файлу model/Setting.java

```
package ckmetrics.model;

import jakarta.persistence.*;
import java.util.Objects;

/**
 * Представляє сутність налаштування застосунку
```

```
 */
@Entity
@Table(name = "Settings")
public class Setting {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "setting_id")
    private Long id;

    @Column(name = "setting_name", nullable = false, unique = true)
    private String name;

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

    /**
     * Створює порожнє налаштування
     */
    public Setting() {
    }

    /**
     * Створює налаштування з назвою і значенням
     *
     * @param name назва налаштування
     * @param value значення налаштування
     */
    public Setting(String name, String value) {
        this.name = name;
        this.value = value;
    }

    /**
     * Повертає ідентифікатор налаштування
     *
     * @return ідентифікатор налаштування
     */
    public Long getId() {
        return id;
    }

    /**
     * Встановлює ідентифікатор налаштування
     *
     * @param id ідентифікатор налаштування
     */
    public void setId(Long id) {
        this.id = id;
    }

    /**
     * Повертає назву налаштування
```

```
     *
     * @return назва налаштування
     */
    public String getName() {
        return name;
    }

    /**
     * Встановлює назву налаштування
     *
     * @param name назва налаштування
     */
    public void setName(String name) {
        this.name = name;
    }

    /**
     * Повертає значення налаштування
     *
     * @return значення налаштування
     */
    public String getValue() {
        return value;
    }

    /**
     * Встановлює значення налаштування
     *
     * @param value значення налаштування
     */
    public void setValue(String value) {
        this.value = value;
    }

    /**
     * Повертає рядкове представлення налаштування
     *
     * @return рядок у форматі "назва: значення"
     */
    @Override
    public String toString() {
        return name + ": " + value;
    }

    /**
     * Порівнює цей об'єкт з іншим на рівність
     *
     * @param o об'єкт для порівняння
     * @return true, якщо об'єкти рівні
     */
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
```

```
        if (o == null || getClass() != o.getClass()) return false;

        Setting setting = (Setting) o;

        if (!Objects.equals(id, setting.id)) return false;
        return Objects.equals(name, setting.name);
    }

    /**
     * Обчислює хеш-код для об'єкта
     *
     * @return хеш-код
     */
    @Override
    public int hashCode() {
        int result = id != null ? id.hashCode() : 0;
        result = 31 * result + (name != null ? name.hashCode() : 0);
        return result;
    }
}
```

Текст файлу model/Type.java

```
package ckmetrics.model;

import jakarta.persistence.*;
import java.util.Objects;

/**
 * Представляє сутність типу проєкту
 */
@Entity
@Table(name = "Types")
public class Type {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "type_id")
    private Long id;

    @Column(name = "type_name", nullable = false, unique = true)
    private String name;

    /**
     * Створює порожній тип проєкту
     */
    public Type() {
    }

    /**
     * Створює тип проєкту з назвою
     *
     * @param name назва типу
     */
```

```
public Type(String name) {
    this.name = name;
}

/**
 * Повертає ідентифікатор типу
 *
 * @return ідентифікатор типу
 */
public Long getId() {
    return id;
}

/**
 * Встановлює ідентифікатор типу
 *
 * @param id ідентифікатор типу
 */
public void setId(Long id) {
    this.id = id;
}

/**
 * Повертає назву типу
 *
 * @return назва типу
 */
public String getName() {
    return name;
}

/**
 * Встановлює назву типу
 *
 * @param name назва типу
 */
public void setName(String name) {
    this.name = name;
}

/**
 * Повертає рядкове представлення типу проєкту
 *
 * @return назва типу
 */
@Override
public String toString() {
    return name;
}

/**
 * Порівнює цей об'єкт з іншим на рівність
 *
```

```
     * @param o об'єкт для порівняння
     * @return true, якщо об'єкти рівні
     */
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        Type type = (Type) o;

        if (!Objects.equals(id, type.id)) return false;
        return Objects.equals(name, type.name);
    }

    /**
     * Обчислює хеш-код для об'єкта
     *
     * @return хеш-код
     */
    @Override
    public int hashCode() {
        int result = id != null ? id.hashCode() : 0;
        result = 31 * result + (name != null ? name.hashCode() : 0);
        return result;
    }
}
```

Текст файлу service/GitHubService.java

```
package ckmetrics.service;

import org.eclipse.jgit.api.Git;
import org.eclipse.jgit.api.errors.GitAPIException;
import org.eclipse.jgit.lib.ObjectId;
import org.eclipse.jgit.lib.Ref;
import org.eclipse.jgit.lib.Repository;
import org.eclipse.jgit.revwalk.RevCommit;
import org.eclipse.jgit.revwalk.RevWalk;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.io.IOException;
import java.nio.file.Path;
import java.time.Instant;
import java.util.ArrayList;
import java.util.List;

/**
 * Виконує операції з GitHub-репозиторіями
 */
public class GitHubService {

    private static final Logger logger =
LoggerFactory.getLogger(GitHubService.class);
```

```
    /**
     * Клонує GitHub-репозиторій з опцією завантаження тегів
     *
     * @param url URL репозиторію для клонування
     * @param directory каталог для клонування репозиторію
     * @param fetchTags чи завантажувати теги, наприклад,
     *                  true для роботи з тегами згідно вказаної дати
     *                  false для роботи з головної гілкою
     * @return екземпляр Git для клонованого репозиторію
     * @throws GitAPIException якщо виникла помилка під час клонування
     */
    public Git cloneRepository(String url, Path directory, boolean
fetchTags) throws GitAPIException {
        if (fetchTags) {
            // Клонування з тегами при виборі режиму останнього тегу
згідно вказаної дати
            return Git.cloneRepository()
                    .setURI(url)
                    .setDirectory(directory.toFile())
                    .setCloneAllBranches(false)
                    .call();
        } else {
            // Кнонування без тегів при виборі режиму головної гіки
            return Git.cloneRepository()
                    .setURI(url)
                    .setDirectory(directory.toFile())
                    .setCloneAllBranches(false)
                    .setNoTags()
                    .call();
        }
    }

    /**
     * Перемикає репозиторій на поточну голову головної гілки
     *
     * @param git екземпляр Git
     * @throws GitAPIException якщо не вдалося виконати checkout
     */
    public void checkoutMainBranchHead(Git git) throws GitAPIException
{
        git.checkout()
                .setName(getMainBranchName(git))
                .call();
    }

    /**
     * Перемикає репозиторій на головну гілку на певну дату
     *
     * @param git екземпляр Git для репозиторію
     * @param date дата для перемикання
     *  @throws GitAPIException якщо виникла помилка під час
перемикання
```

```
     * @throws IOException якщо виникла помилка вводу-виводу
     */
    public void checkoutMainBranchAtDate(Git git, Instant date)
            throws GitAPIException, IOException {
        // Отримання головної гілки
        String mainBranch = getMainBranchName(git);

        // Пошук останнього коміту, що не пізніше вказаної дати
        RevCommit commit = findCommitOnOrBeforeDate(git, mainBranch,
date);

        // Перемикання на знайдений коміт
        git.checkout()
                .setName(commit.getName())
                .call();
    }

    /**
     * Перемикає репозиторій на останній тег на певну дату
     * Якщо тегів немає або всі теги пізніші за вказану дату,
     * Тоді використовується головна гілка на цю дату
     *
     * @param git екземпляр Git для репозиторію
     * @param date дата для перемикання
     *  @throws GitAPIException якщо виникла помилка під час
перемикання
     * @throws IOException якщо виникла помилка вводу-виводу
     */
    public void checkoutLatestTagAtDate(Git git, Instant date)
            throws GitAPIException, IOException {
        Ref tag = findLatestTagOnOrBeforeDate(git, date);
        if (tag == null) {
            logger.warn("No tags found in repository. Main branch at
date will be used.");
            checkoutMainBranchAtDate(git, date);
            return;
        }

        // Перемикання на останній тег
        git.checkout()
                .setName(tag.getName())
                .call();
    }

    /**
     * Повертає назву головної гілки, якою зазвичай виступає master
або main
     *
     * @param git екземпляр Git для репозиторію
     * @return назва головної гілки
     * @throws GitAPIException якщо виникла помилка під час операції
     */
```

```
    private String getMainBranchName(Git git) throws GitAPIException
{
        List<Ref> branches = git.branchList().call();

        // Перевірка наявності гілки main або master
        for (Ref branch : branches) {
            String name = branch.getName();
            if (name.endsWith("/main") || name.endsWith("/master")) {
                return name.substring(name.lastIndexOf('/') + 1);
            }
        }

        // Якщо не знайдено main або master, повертаємо першу гілку
        if (!branches.isEmpty()) {
            String name = branches.getFirst().getName();
            return name.substring(name.lastIndexOf('/') + 1);
        }

        // За замовчуванням це master, якщо не знайдено інших гілок
        return "master";
    }

    /**
     * Шукає останній коміт, який не пізніше вказаної дати
     *
     * @param git екземпляр Git для репозиторію
     * @param branch гілка для пошуку
     * @param date дата для пошуку коміту
     * @return останній коміт, що не пізніше вказаної дати
     * @throws GitAPIException якщо виникла помилка під час операції
     * @throws IOException якщо виникла помилка вводу-виводу
     */
    private RevCommit findCommitOnOrBeforeDate(Git git, String branch,
Instant date)
            throws GitAPIException, IOException {
        Iterable<RevCommit> commits = git.log()
                .add(git.getRepository().resolve(branch))
                .call();

        RevCommit earliestCommit = null;
        for (RevCommit commit : commits) {
            long commitTime = commit.getCommitTime() * 1000L;
            earliestCommit = commit;
            if (commitTime <= date.toEpochMilli()) {
                return commit;
            }
        }

        if (earliestCommit == null) {
            throw new IllegalArgumentException("No commits found in
repository");
        }
```

```
        logger.warn("All commits are later than {}. The earliest
commit will be used.", date);
        return earliestCommit;
    }

    /**
     * Повертає останній тег, коміт якого не пізніше за вказану дату
     * Якщо всі теги пізніші за дату, повертається найраніший тег як
fallback
     */
    private Ref findLatestTagOnOrBeforeDate(Git git, Instant date)
            throws GitAPIException, IOException {
        List<Ref> tags = new ArrayList<>(git.tagList().call());
        if (tags.isEmpty()) {
            return null;
        }

        Ref earliestTag = null;
        long earliestTime = Long.MAX_VALUE;
        Ref latestEligibleTag = null;
        long latestEligibleTime = Long.MIN_VALUE;

        for (Ref tag : tags) {
            long                    tagCommitTime                    =
resolveTagCommit(git.getRepository(), tag).getCommitTime() * 1000L;
            if (tagCommitTime < earliestTime) {
                earliestTime = tagCommitTime;
                earliestTag = tag;
            }
            if (tagCommitTime <= date.toEpochMilli() && tagCommitTime
> latestEligibleTime) {
                latestEligibleTime = tagCommitTime;
                latestEligibleTag = tag;
            }
        }

        if (latestEligibleTag == null && earliestTag != null) {
            logger.warn("All tags are later than {}. The earliest tag
will be used.", date);
            return earliestTag;
        }

        return latestEligibleTag;
    }

    /**
     * Резолвить тег до commit-об'єкта, який він позначає
     *
     * @param repository репозиторій Git
     * @param tag тег для резолву
     * @return commit, на який вказує тег
     * @throws IOException якщо не вдалося прочитати об'єкт тега або
коміту
```

```
     */
    private RevCommit resolveTagCommit(Repository repository, Ref tag)
throws IOException {
        try (RevWalk walk = new RevWalk(repository)) {
            return    walk.parseCommit(resolveTagObjectId(repository,
tag));
        }
    }

    /**
     * Визначає фактичний ObjectId тега, враховуючи анотовані та
легковагові теги
     *
     * @param repository репозиторій Git
     * @param tag тег, для якого треба визначити об'єкт
     * @return ObjectId коміту або тега
     * @throws IOException якщо не вдалося визначити ObjectId
     */
    private  ObjectId  resolveTagObjectId(Repository  repository,  Ref
tag) throws IOException {
        Ref peeled = repository.getRefDatabase().peel(tag);
        ObjectId objectId = peeled.getPeeledObjectId();
        if (objectId != null) {
            return objectId;
        }
        if (tag.getObjectId() != null) {
            return tag.getObjectId();
        }
        throw new IOException("Failed to resolve tag object id for "
+ tag.getName());
    }
}
```

Текст файлу service/ck/CkService.java

```
package ckmetrics.service.ck;

import ckmetrics.util.FileUtils;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.List;

/**
 * Координує запуск CK, парсинг CSV, агрегацію та запис XLSX
 */
public class CkService {

    // Шлях до jar-файлу утиліти CK
    private static final Path CK_JAR_PATH = Path.of("lib", "ck-0.7.1-
SNAPSHOT-jar-with-dependencies.jar").toAbsolutePath().normalize();

    private final CkRunner ckRunner;
```

```
    private final CkCsvParser ckCsvParser;
    private final MetricsAggregator metricsAggregator;
    private final ExcelReportWriter excelReportWriter;

    /**
     * Створює CK-сервіс зі стандартними залежностями
     */
    public CkService() {
        this(new CkRunner(), new CkCsvParser(), new
MetricsAggregator());
    }

    /**
     * Створює CK-сервіс із переданими залежностями
     *
     * @param ckRunner компонент запуску CK
     * @param ckCsvParser компонент парсингу CSV-файлів CK
     * @param metricsAggregator компонент агрегації метрик
     */
    public CkService(CkRunner ckRunner, CkCsvParser ckCsvParser,
MetricsAggregator metricsAggregator) {
        this.ckRunner = ckRunner;
        this.ckCsvParser = ckCsvParser;
        this.metricsAggregator = metricsAggregator;
        this.excelReportWriter = new
ExcelReportWriter(metricsAggregator);
    }

    /**
     * Запускає CK для каталогу проєкту та формує або дописує
підсумковий XLSX
     *
     * @param directory каталог проєкту з вихідними .java файлами
     * @param xlsxFile файл підсумкового звіту XLSX буде
створений/дописаний
     * @param excludedFolders перелік рядків/підрядків шляхів, які
слід виключити з аналізу
     * @param projectDisplayName зрозумілий ідентифікатор виду
vendor/repo для виводу у XLSX
     * @param commitHash хеш коміту, на якому виконувалась
аналітика
     * @throws IOException у випадку проблем з файловими
операціями
     * @throws InterruptedException якщо процес CK перервано
     */
    public void runCkMetrics(Path directory, Path xlsxFile,
List<String> excludedFolders,
                             String projectDisplayName, String
commitHash)
            throws IOException, InterruptedException {
        // Перевірка наявності jar-файлу утиліти CK
        if (Files.notExists(CK_JAR_PATH)) {
```

```
            throw new IOException("CK JAR not found: " +
CK_JAR_PATH.toAbsolutePath());
        }
        // Тимчасова директорія для збереження утилітою CK файлів
class.csv, method.csv, variable.csv
        Path tempDir = FileUtils.createTempDirectory("ck-temp");
        try {
            ckRunner.run(CK_JAR_PATH, directory, tempDir);

            Path classCsv = tempDir.resolve("class.csv");
            if (Files.exists(classCsv) && Files.size(classCsv) > 0) {
                CkCsvData csvData = ckCsvParser.parse(classCsv);
                MetricsSummary           metricsSummary           =
metricsAggregator.aggregate(csvData, excludedFolders);
                excelReportWriter.writeProjectMetrics(xlsxFile,
projectDisplayName, commitHash, metricsSummary);
            } else {
                excelReportWriter.writeEmptyProject(xlsxFile,
projectDisplayName, commitHash);
            }
        } finally {
            FileUtils.deleteDirectory(tempDir);
        }
    }
}
```

Текст файлу ui/CheckoutPanel.java

```
package ckmetrics.ui;

import ckmetrics.dao.BaseDAO.DataAccessException;
import ckmetrics.dao.ExcludedFolderDAO;
import ckmetrics.dao.ProjectDAO;
import ckmetrics.dao.SettingDAO;
import ckmetrics.dao.TypeDAO;
import ckmetrics.model.ExcludedFolder;
import ckmetrics.model.Project;
import ckmetrics.model.Type;
import ckmetrics.service.checkout.CheckoutProgress;
import ckmetrics.service.checkout.CheckoutRequest;
import ckmetrics.service.checkout.CheckoutResult;
import ckmetrics.service.checkout.CheckoutService;
import ckmetrics.ui.checkout.CheckoutOptionsPanel;
import ckmetrics.ui.checkout.CheckoutProgressDialog;
import ckmetrics.ui.checkout.CheckoutProjectsTablePanel;
import ckmetrics.ui.checkout.CheckoutSettingsManager;
import ckmetrics.ui.layout.UiSpacing;
import net.miginfocom.swing.MigLayout;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import javax.swing.*;
import java.io.File;
```

```
import java.nio.file.Path;
import java.time.Instant;
import java.util.List;

/**
 * Керує checkout GitHub-проєктів і збором метрик
 * Координує UI-компоненти checkout і викликає окремий checkout-сервіс
 */
public class CheckoutPanel extends JPanel {

    private static final Logger logger = LoggerFactory.getLogger(CheckoutPanel.class);

    private final ProjectDAO projectDAO;
    private final TypeDAO typeDAO;
    private final ExcludedFolderDAO excludedFolderDAO;
    private final CheckoutService checkoutService;
    private final CheckoutSettingsManager settingsManager;

    private CheckoutProjectsTablePanel projectsTablePanel;
    private CheckoutOptionsPanel optionsPanel;

    private boolean isLoadingSettings;

    /**
     * Створює нову панель checkout
     */
    public CheckoutPanel() {
        this.projectDAO = new ProjectDAO();
        this.typeDAO = new TypeDAO();
        this.excludedFolderDAO = new ExcludedFolderDAO();
        this.checkoutService = new CheckoutService();
        this.settingsManager = new CheckoutSettingsManager(new SettingDAO());

        initialize();
    }

    /**
     * Ініціалізує UI та завантажує дані
     */
    private void initialize() {
        isLoadingSettings = true;
        try {
            initComponents();
            loadTypes();
            loadExcludedFolders();
        } finally {
            isLoadingSettings = false;
        }
        loadSettings();
    }
```

```
    /**
     * Ініціалізує компоненти панелі
     */
    private void initComponents() {
        setLayout(new MigLayout(
                "insets " + UiSpacing.DIALOG + ", fill, wrap 1, gapy
" + UiSpacing.UNRELATED,
                "[grow,fill]",
                "[grow,fill][]"
        ));

        projectsTablePanel = new CheckoutProjectsTablePanel();
        add(projectsTablePanel, "grow, push");

        optionsPanel = new CheckoutOptionsPanel();
        optionsPanel.setSettingsChangedHandler(this::saveSettings);

optionsPanel.setTypeSelectionChangedHandler(this::loadProjectsByType
);

optionsPanel.setBrowseDirectoryHandler(this::browseDirectory);
        optionsPanel.setRunCheckoutHandler(this::checkoutProjects);
        add(optionsPanel, "growx, pushx");
    }

    /**
     * Завантажує типи з бази даних
     */
    public void loadTypes() {
        try {
            List<Type> types = typeDAO.getAll();
            optionsPanel.setTypes(types);
        } catch (DataAccessException e) {
            showErrorDialog("Failed    to    load    types:    "    +
e.getMessage());
        }
    }

    /**
     * Завантажує проєкти за обраним типом
     */
    public void loadProjectsByType() {
        Type type = optionsPanel.getSelectedType();
        if (type == null) {
            projectsTablePanel.clearProjects();
            return;
        }

        try {
            List<Project> projects = projectDAO.getByType(type);
            projectsTablePanel.setProjects(projects);
        } catch (DataAccessException e) {
```

```
            showErrorDialog("Failed   to   load   projects:   "   +
e.getMessage());
        }
    }

    /**
     * Завантажує виключені каталоги
     */
    public void loadExcludedFolders() {
        try {
            List<ExcludedFolder>              folders              =
excludedFolderDAO.getAll();
            optionsPanel.setExcludedFolders(folders);
        } catch (DataAccessException e) {
            showErrorDialog("Failed  to  load  excluded  folders:  "  +
e.getMessage());
        }
    }

    /**
     * Відкриває діалог вибору каталогу
     */
    private void browseDirectory() {
        JFileChooser fileChooser = new JFileChooser();

fileChooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
        fileChooser.setDialogTitle("Select directory for checkout");

        int result = fileChooser.showOpenDialog(this);
        if (result == JFileChooser.APPROVE_OPTION) {
            File selectedFile = fileChooser.getSelectedFile();

optionsPanel.setDirectory(selectedFile.getAbsolutePath());
            saveSettings();
        }
    }

    /**
     * Запускає checkout усіх проєктів обраного типу
     */
    private void checkoutProjects() {
        try {
            CheckoutRequest request = buildCheckoutRequest();
            saveSettings();
            runCheckoutWorker(request);
        } catch (IllegalArgumentException e) {
            showErrorDialog(e.getMessage());
        } catch (RuntimeException e) {
            logger.error("Failed to start checkout.", e);
            showErrorDialog("Checkout failed: " + e.getMessage());
        }
    }
```

```
    /**
     * Формує request-об'єкт із поточних даних форми
     *
     * @return параметри checkout
     */
    private CheckoutRequest buildCheckoutRequest() {
        Type selectedType = optionsPanel.getSelectedType();
        if (selectedType == null) {
            throw new IllegalArgumentException("Please select a
project type.");
        }

        String directory = optionsPanel.getDirectory();
        if (directory.isEmpty()) {
            throw new IllegalArgumentException("Please select a
directory.");
        }

        Instant checkoutDate =
optionsPanel.getSelectedCheckoutMode().usesDate()
                ? optionsPanel.getCheckoutDate(true).toInstant()
                : null;

        return new CheckoutRequest(
                selectedType,
                Path.of(directory),
                optionsPanel.getSelectedCheckoutMode(),
                checkoutDate,
                optionsPanel.isDeleteAfterAnalysisSelected(),
                optionsPanel.getSelectedExcludedFolderNames()
        );
    }

    /**
     * Запускає checkout у фоновому SwingWorker
     *
     * @param request параметри checkout
     */
    private void runCheckoutWorker(CheckoutRequest request) {
        CheckoutProgressDialog progressDialog = new
CheckoutProgressDialog(this);

        SwingWorker<CheckoutResult, CheckoutProgress> worker = new
SwingWorker<>() {
            /**
             * Виконує checkout у фоновому потоці SwingWorker
             *
             * @return результат виконання checkout
             * @throws Exception якщо під час обробки виникла помилка
             */
            @Override
            protected CheckoutResult doInBackground() throws
Exception {
```

```
                return      checkoutService.checkoutProjects(request,
this::publishProgress);
            }

            /**
             * Публікує одну подію прогресу у EDT через механізм
SwingWorker
             *
             * @param progress поточний стан прогресу
             */
            private void publishProgress(CheckoutProgress progress) {
                publish(progress);
            }

            /**
             * Оновлює діалог прогресу останнім отриманим значенням
             *
             * @param chunks список пакетів прогресу від SwingWorker
             */
            @Override
            protected void process(List<CheckoutProgress> chunks) {
                if (!chunks.isEmpty()) {
                    progressDialog.updateProgress(chunks.getLast());
                }
            }

            /**
             * Завершує роботу прогрес-діалогу та показує результат
користувачу
             */
            @Override
            protected void done() {
                progressDialog.dispose();
                try {
                    CheckoutResult result = get();
                    if (result.hasErrors()) {
                        showWarningDialog("Checkout  completed  with
errors:\n" + String.join("\n", result.errors()));
                    } else {
                        showInfoDialog(
                                "All    projects    were    processed
successfully.\nMetrics summary saved to:\n"
                                        +
result.metricsFile().toAbsolutePath()
                        );
                    }
                } catch (Exception e) {
                    logger.error("Checkout failed.", e);
                    String   message   =   e.getCause()   !=   null   ?
e.getCause().getMessage() : e.getMessage();
                    showErrorDialog("Checkout failed: " + message);
                }
            }
```

```
        };

        worker.execute();
        progressDialog.setVisible(true);
    }

    /**
     * Завантажує налаштування з бази даних
     */
    private void loadSettings() {
        isLoadingSettings = true;
        try {
            settingsManager.loadSettings(optionsPanel);
            loadProjectsByType();
        } finally {
            isLoadingSettings = false;
        }
    }

    /**
     * Зберігає налаштування checkout у базу даних
     */
    private void saveSettings() {
        if (!isLoadingSettings) {
            settingsManager.saveSettings(optionsPanel);
        }
    }
    /**
     * Оновлює панель checkout
     */
    public void refresh() {
        isLoadingSettings = true;
        try {
            loadTypes();
            loadExcludedFolders();
        } finally {
            isLoadingSettings = false;
        }
        loadSettings();
    }
    /**
     * Показує інформаційне діалогове вікно
     *
     * @param message текст повідомлення
     */
    private void showInfoDialog(String message) {
        JOptionPane.showMessageDialog(this,    message,    "Success",
JOptionPane.INFORMATION_MESSAGE);
    }
    /**
     * Показує діалогове вікно попередження
     *
     * @param message текст повідомлення
```

```
     */
    private void showWarningDialog(String message) {
        JOptionPane.showMessageDialog(this,    message,    "Warning",
JOptionPane.WARNING_MESSAGE);
    }
    /**
     * Показує діалогове вікно помилки
     *
     * @param message текст повідомлення
     */
    private void showErrorDialog(String message) {
        JOptionPane.showMessageDialog(this,     message,     "Error",
JOptionPane.ERROR_MESSAGE);
    }
}
```

Текст файлу ui/ExcludedFoldersPanel.java

```
package ckmetrics.ui;

import ckmetrics.dao.BaseDAO.DataAccessException;
import ckmetrics.dao.ExcludedFolderDAO;
import ckmetrics.model.ExcludedFolder;
import ckmetrics.ui.layout.UiFactory;
import ckmetrics.util.validation.ExcludedFolderNameValidator;
import ckmetrics.util.validation.ValidationResult;

import javax.swing.*;
import java.util.List;

/**
 * Керує виключеними каталогами у CRUD-панелі
 */
public class ExcludedFoldersPanel extends BaseCrudPanel {

    private JTextField folderNameField;
    private    final    ExcludedFolderDAO    folderDAO    =    new
ExcludedFolderDAO();
    private final ExcludedFolderNameValidator folderNameValidator =
new ExcludedFolderNameValidator();

    /**
     * Створює нову панель виключених каталогів
     */
    public ExcludedFoldersPanel() {
        loadData();
    }

    /**
     * Повертає назви колонок таблиці виключених каталогів
     *
     * @return масив назв колонок
     */
```

```
    @Override
    protected String[] getColumnNames() {
        return new String[]{"ID", "Name"};
    }

    /**
     * Створює форму вводу назви виключеного каталогу
     *
     * @return панель форми
     */
    @Override
    protected JPanel createInputPanel() {
        JPanel inputPanel = UiFactory.createCrudFormPanel();
        folderNameField = new JTextField();
        UiFactory.addCrudFormRow(inputPanel, "Excluded folder name:",
folderNameField);
        return inputPanel;
    }

    /**
     * Заповнює поле вводу назви значенням вибраного рядка
     *
     * @param modelRow індекс рядка в моделі таблиці
     */
    @Override
    protected void onTableRowSelected(int modelRow) {
        String folderName = (String) tableModel.getValueAt(modelRow,
1);
        folderNameField.setText(folderName);
    }

    /**
     * Завантажує виключені каталоги з БД у таблицю
     */
    @Override
    protected void loadData() {
        tableModel.setRowCount(0);
        try {
            List<ExcludedFolder> folders = folderDAO.getAll();
            for (ExcludedFolder folder : folders) {
                Object[]        row        =        {folder.getId(),
folder.getFolderName()};
                tableModel.addRow(row);
            }
        } catch (DataAccessException e) {
            showError("Failed   to   load   excluded   folders:   "   +
e.getMessage());
        }
    }

    /**
     * Очищує поле вводу назви каталогу
     */
```

```
    @Override
    protected void clearFields() {
        folderNameField.setText("");
    }

    /**
     * Додає новий виключений каталог після валідації і перевірки дубля
     */
    @Override
    protected void addEntity() {
        String folderName = folderNameField.getText().trim();

        ValidationResult validationResult =
folderNameValidator.validate(folderName);
        if (!validationResult.valid()) {
            showError(validationResult.message());
            return;
        }

        try {
            ExcludedFolder existingFolder =
folderDAO.findByName(folderName);
            if (existingFolder != null) {
                showError("Folder already exists.");
                return;
            }

            folderDAO.saveOrUpdate(new ExcludedFolder(folderName));
            clearFields();
            reloadTableData();
            showSuccess("Excluded folder added successfully.");
        } catch (DataAccessException e) {
            showError("Failed to save excluded folder: " +
e.getMessage());
        }
    }

    /**
     * Редагує вибраний виключений каталог після перевірки змін і
дублювання
     */
    @Override
    protected void editEntity() {
        Long id = getSelectedId();

        if (id == null) {
            showError("Please select an excluded folder.");
            return;
        }

        String folderName = folderNameField.getText().trim();
```

```
        ValidationResult validationResult =
folderNameValidator.validate(folderName);
        if (shouldBlockEditWithName(folderName, validationResult)) {
            return;
        }

        try {
            ExcludedFolder folder = folderDAO.getById(id);
            if (folder == null) {
                showError("Excluded folder not found.");
                return;
            }

            ExcludedFolder existingFolder =
folderDAO.findByName(folderName);
            if (existingFolder != null &&
!existingFolder.getId().equals(id)) {
                showError("Folder already exists.");
                return;
            }

            folder.setFolderName(folderName);
            folderDAO.saveOrUpdate(folder);
            clearFields();
            table.clearSelection();
            reloadTableData();
            showSuccess("Excluded folder updated successfully.");
        } catch (DataAccessException e) {
            showError("Failed to save excluded folder: " +
e.getMessage());
        }
    }

    /**
     * Видаляє вибраний виключений каталог після підтвердження
     */
    @Override
    protected void deleteEntity() {
        Long id = getSelectedId();

        if (id == null) {
            showError("Please select an excluded folder.");
            return;
        }

        try {
            ExcludedFolder folder = folderDAO.getById(id);
            if (folder == null) {
                showError("Excluded folder not found.");
                return;
            }
```

```
            if (showConfirmation("Are you sure you want to delete this
excluded folder?")) {
                boolean deleted = folderDAO.delete(folder);
                if (deleted) {
                    clearFields();
                    reloadTableData();
                    showSuccess("Excluded       folder       deleted
successfully.");
                } else {
                    showError("Failed to delete excluded folder.");
                }
            }
        } catch (DataAccessException e) {
            showError("Failed   to   delete   excluded   folder:   "   +
e.getMessage());
        }
    }
}
```

Текст файлу ui/MainFrame.java

```
package ckmetrics.ui;

import ckmetrics.util.HibernateUtil;

import javax.swing.*;
import java.awt.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

/**
 * Відображає головне вікно застосунку
 */
public class MainFrame extends JFrame {

    private JTabbedPane tabbedPane;
    private TypesPanel typesPanel;
    private ProjectsPanel projectsPanel;
    private ExcludedFoldersPanel excludedFoldersPanel;
    private CheckoutPanel checkoutPanel;

    /**
     * Створює нове головне вікно
     */
    public MainFrame() {
        setTitle("Software  for  automating  GitHub  project  metrics
collection in Java with CK utility");
        setSize(900, 700);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);

        // Додавання слухача вікна для вимкнення Hibernate при виході
        addWindowListener(new WindowAdapter() {
            /**
```

```
             * Коректно завершує Hibernate при закритті вікна
             *
             * @param e подія закриття вікна
             */
            @Override
            public void windowClosing(WindowEvent e) {
                HibernateUtil.shutdown();
            }
        });

        initComponents();
    }

    /**
     * Ініціалізує компоненти вікна
     */
    private void initComponents() {
        // Створення панелі вкладок
        tabbedPane = new JTabbedPane();

        // Створення панелей
        typesPanel = new TypesPanel();
        projectsPanel = new ProjectsPanel();
        excludedFoldersPanel = new ExcludedFoldersPanel();
        checkoutPanel = new CheckoutPanel();

        // Додавання вкладок
        tabbedPane.addTab("Types", typesPanel);
        tabbedPane.addTab("Projects", projectsPanel);
        tabbedPane.addTab("Excluded folders", excludedFoldersPanel);
        tabbedPane.addTab("Checkout", checkoutPanel);

        // Додавання слухача зміни вкладок для оновлення даних
        tabbedPane.addChangeListener(e -> {
            int selectedIndex = tabbedPane.getSelectedIndex();
            if (selectedIndex == 0) {
                // Обрано панель типів
                typesPanel.refresh();
            } else if (selectedIndex == 1) {
                // Обрано панель проєктів
                projectsPanel.refresh();
            } else if (selectedIndex == 2) {
                // Обрано панель виключених каталогів
                excludedFoldersPanel.refresh();
            } else if (selectedIndex == 3) {
                // Обрано панель checkout
                checkoutPanel.refresh();
            }
        });
        // Додавання панелі вкладок до вікна
        getContentPane().add(tabbedPane, BorderLayout.CENTER);
    }
}
```

Текст файлу ui/ProjectsPanel.java

```
package ckmetrics.ui;

import ckmetrics.dao.BaseDAO.DataAccessException;
import ckmetrics.dao.ProjectDAO;
import ckmetrics.dao.TypeDAO;
import ckmetrics.model.Project;
import ckmetrics.model.Type;
import ckmetrics.ui.layout.UiFactory;
import ckmetrics.util.validation.ProjectUrlValidator;
import ckmetrics.util.validation.ValidationResult;

import javax.swing.*;
import java.util.List;

/**
 * Керує GitHub-проєктами у CRUD-панелі
 */
public class ProjectsPanel extends BaseCrudPanel {

    private JTextField urlField;
    private JComboBox<Type> typeComboBox;
    private final ProjectDAO projectDAO;
    private final TypeDAO typeDAO;
    private final ProjectUrlValidator projectUrlValidator;

    /**
     * Створює нову панель проєктів
     */
    public ProjectsPanel() {
        this(new      ProjectDAO(),      new      TypeDAO(),      new
ProjectUrlValidator());
    }

    /**
     * Створює нову панель проєктів із переданими залежностями
     *
     * @param projectDAO DAO для роботи з проєктами
     * @param typeDAO DAO для роботи з типами
     * @param projectUrlValidator валідатор URL проєкту
     */
    ProjectsPanel(ProjectDAO     projectDAO,     TypeDAO     typeDAO,
ProjectUrlValidator projectUrlValidator) {
        this.projectDAO = projectDAO;
        this.typeDAO = typeDAO;
        this.projectUrlValidator = projectUrlValidator;
        loadTypes();
        loadData();
    }

    /**
     * Повертає назви колонок таблиці проєктів
     *
```

```
     * @return масив назв колонок
     */
    @Override
    protected String[] getColumnNames() {
        return new String[]{"ID", "URL", "Type"};
    }

    /**
     * Створює форму вводу URL і типу проєкту
     *
     * @return панель форми
     */
    @Override
    protected JPanel createInputPanel() {
        JPanel inputPanel = UiFactory.createCrudFormPanel();
        urlField = new JTextField();
        typeComboBox = new JComboBox<>();
        UiFactory.addCrudFormRow(inputPanel,     "Project     URL:",
urlField);
        UiFactory.addCrudFormRow(inputPanel,     "Project     Type:",
typeComboBox);
        return inputPanel;
    }

    /**
     * Заповнює поля форми даними вибраного рядка таблиці
     *
     * @param modelRow індекс рядка у моделі таблиці
     */
    @Override
    protected void onTableRowSelected(int modelRow) {
        String url = (String) tableModel.getValueAt(modelRow, 1);
        String typeName = (String) tableModel.getValueAt(modelRow, 2);
        urlField.setText(url);

        // Вибір відповідного типу в комбінованому списку
        for (int i = 0; i < typeComboBox.getItemCount(); i++) {
            Type type = typeComboBox.getItemAt(i);
            if (type != null && type.getName().equals(typeName)) {
                typeComboBox.setSelectedIndex(i);
                break;
            }
        }
    }

    /**
     * Завантажує список проєктів у таблицю
     */
    @Override
    protected void loadData() {
        tableModel.setRowCount(0);
        try {
            List<Project> projects = projectDAO.getAll();
```

```
            for (Project project : projects) {
                Object[] row = {
                        project.getId(),
                        project.getUrl(),
                        project.getType()       !=       null       ?
project.getType().getName() : ""
                };
                tableModel.addRow(row);
            }
        } catch (DataAccessException e) {
            showError("Failed to load projects: " + e.getMessage());
        }
    }

    /**
     * Очищує поля форми створення/редагування
     */
    @Override
    protected void clearFields() {
        urlField.setText("");
        typeComboBox.setSelectedIndex(-1);
    }

    /**
     * Завантажує типи з бази даних
     * Після завантаження поле типу за замовчуванням порожнє
     */
    public void loadTypes() {
        typeComboBox.removeAllItems();
        try {
            List<Type> types = typeDAO.getAll();
            for (Type type : types) {
                typeComboBox.addItem(type);
            }
        } catch (DataAccessException e) {
            showError("Failed to load types: " + e.getMessage());
        }
        typeComboBox.setSelectedIndex(-1);
    }

    /**
     * Оновлює дані проєктів
     */
    @Override
    public void refresh() {
        loadTypes();
        loadData();
        clearFields();
        table.clearSelection();
    }

    /**
     * Додає новий проєкт після валідації URL та перевірки дубля
```

```
     */
    @Override
    protected void addEntity() {
        String url = urlField.getText().trim();
        Type type = (Type) typeComboBox.getSelectedItem();

        ValidationResult validationResult =
projectUrlValidator.validate(url);
        if (!validationResult.valid()) {
            showError(validationResult.message());
            return;
        }

        try {
            Project existingProject = projectDAO.findByUrl(url);
            if (existingProject != null) {
                showError("Project already exists.");
                return;
            }

            projectDAO.saveOrUpdate(new Project(url, type));
            clearFields();
            reloadTableData();
            showSuccess("Project added successfully.");
        } catch (DataAccessException e) {
            showError("Failed to save project: " + e.getMessage());
        }
    }

    /**
     * Оновлює вибраний проєкт після валідації і перевірки на
унікальність URL
     */
    @Override
    protected void editEntity() {
        Long id = getSelectedId();

        if (id == null) {
            showError("Please select a project.");
            return;
        }

        String url = urlField.getText().trim();
        Type type = (Type) typeComboBox.getSelectedItem();

        ValidationResult validationResult =
projectUrlValidator.validate(url);
        if (!validationResult.valid()) {
            showError(validationResult.message());
            return;
        }

        // Перевірка, чи дані не були змінені
```

```
        int selectedViewRow = table.getSelectedRow();
        if (selectedViewRow != -1) {
            int                         modelRow                         =
table.convertRowIndexToModel(selectedViewRow);
            String          currentUrl          =          (String)
tableModel.getValueAt(modelRow, 1);
            String         currentTypeName         =         (String)
tableModel.getValueAt(modelRow, 2);
            String newTypeName = (type != null) ? type.getName() : "";
            boolean urlUnchanged = url.equals(currentUrl);
            boolean                   typeUnchanged                    =
newTypeName.equals(currentTypeName != null ? currentTypeName : "");
            if (urlUnchanged && typeUnchanged) {
                showUnchangedWarning();
                return;
            }
        }

        try {
            Project project = projectDAO.getById(id);
            if (project == null) {
                showError("Project not found.");
                return;
            }

            Project existingProject = projectDAO.findByUrl(url);
            if        (existingProject        !=        null        &&
!existingProject.getId().equals(id)) {
                showError("Project URL already exists.");
                return;
            }

            project.setUrl(url);
            project.setType(type);
            projectDAO.saveOrUpdate(project);
            clearFields();
            table.clearSelection();
            reloadTableData();
            showSuccess("Project updated successfully.");
        } catch (DataAccessException e) {
            showError("Failed to save project: " + e.getMessage());
        }
    }

    /**
     * Видаляє вибраний проєкт після підтвердження користувача
     */
    @Override
    protected void deleteEntity() {
        Long id = getSelectedId();

        if (id == null) {
            showError("Please select a project.");
```

```
            return;
        }

        try {
            Project project = projectDAO.getById(id);
            if (project == null) {
                showError("Project not found.");
                return;
            }

            if (showConfirmation("Are you sure you want to delete this
project?")) {
                boolean deleted = projectDAO.delete(project);
                if (deleted) {
                    clearFields();
                    reloadTableData();
                    showSuccess("Project deleted successfully.");
                } else {
                    showError("Failed to delete project.");
                }
            }
        } catch (DataAccessException e) {
            showError("Failed to delete project: " + e.getMessage());
        }
    }
}
```

Текст файлу ui/TypesPanel.java

```
package ckmetrics.ui;

import ckmetrics.dao.BaseDAO.DataAccessException;
import ckmetrics.dao.TypeDAO;
import ckmetrics.model.Type;
import ckmetrics.ui.layout.UiFactory;
import ckmetrics.util.validation.TypeNameValidator;
import ckmetrics.util.validation.ValidationResult;

import javax.swing.*;
import java.util.List;
/**
 * Керує типами проєктів у CRUD-панелі
 */
public class TypesPanel extends BaseCrudPanel {

    private JTextField nameField;
    private final TypeDAO typeDAO = new TypeDAO();
    private   final   TypeNameValidator   typeNameValidator   =   new
TypeNameValidator();
    /**
     * Створює нову панель типів
     */
    public TypesPanel() {
        loadData();
```

```
    }
    /**
     * Повертає назви колонок таблиці типів
     *
     * @return масив назв колонок
     */
    @Override
    protected String[] getColumnNames() {
        return new String[]{"ID", "Name"};
    }

    /**
     * Створює форму вводу для назви типу
     *
     * @return панель форми
     */
    @Override
    protected JPanel createInputPanel() {
        JPanel inputPanel = UiFactory.createCrudFormPanel();
        nameField = new JTextField();
        UiFactory.addCrudFormRow(inputPanel,      "Type      name:",
nameField);
        return inputPanel;
    }
    /**
     * Заповнює поле назви даними вибраного рядка
     *
     * @param modelRow індекс рядка в моделі таблиці
     */
    @Override
    protected void onTableRowSelected(int modelRow) {
        String name = (String) tableModel.getValueAt(modelRow, 1);
        nameField.setText(name);
    }
    /**
     * Завантажує всі типи з БД у таблицю
     */
    @Override
    protected void loadData() {
        tableModel.setRowCount(0);
        try {
            List<Type> types = typeDAO.getAll();
            for (Type type : types) {
                Object[] row = {type.getId(), type.getName()};
                tableModel.addRow(row);
            }
        } catch (DataAccessException e) {
            showError("Failed to load types: " + e.getMessage());
        }
    }
    /**
     * Очищує поле вводу назви типу
     */
```

```
    @Override
    protected void clearFields() {
        nameField.setText("");
    }
    /**
     * Додає новий тип після валідації та перевірки унікальності
     */
    @Override
    protected void addEntity() {
        String name = nameField.getText().trim();
        ValidationResult validationResult =
typeNameValidator.validate(name);
        if (!validationResult.valid()) {
            showError(validationResult.message());
            return;
        }

        try {
            Type existingType = typeDAO.findByName(name);
            if (existingType != null) {
                showError("Type already exists.");
                return;
            }

            typeDAO.saveOrUpdate(new Type(name));
            clearFields();
            reloadTableData();
            showSuccess("Type added successfully.");
        } catch (DataAccessException e) {
            showError("Failed to save type: " + e.getMessage());
        }
    }
    /**
     * Редагує вибраний тип після валідації, перевірки змін і
дублювання
     */
    @Override
    protected void editEntity() {
        Long id = getSelectedId();

        if (id == null) {
            showError("Please select a type.");
            return;
        }

        String name = nameField.getText().trim();
        ValidationResult validationResult =
typeNameValidator.validate(name);
        if (shouldBlockEditWithName(name, validationResult)) {
            return;
        }

        try {
```

```
            Type type = typeDAO.getById(id);
            if (type == null) {
                showError("Type not found.");
                return;
            }

            Type existingType = typeDAO.findByName(name);
            if (existingType != null &&
!existingType.getId().equals(id)) {
                showError("A type with this name already exists.");
                return;
            }

            type.setName(name);
            typeDAO.saveOrUpdate(type);
            clearFields();
            table.clearSelection();
            reloadTableData();
            showSuccess("Type updated successfully.");
        } catch (DataAccessException e) {
            showError("Failed to save type: " + e.getMessage());
        }
    }
    /**
     * Видаляє вибраний тип після підтвердження користувача
     */
    @Override
    protected void deleteEntity() {
        Long id = getSelectedId();

        if (id == null) {
            showError("Please select a type.");
            return;
        }

        try {
            Type type = typeDAO.getById(id);
            if (type == null) {
                showError("Type not found.");
                return;
            }

            if (showConfirmation("Are you sure you want to delete this
type?")) {
                boolean deleted = typeDAO.delete(type);
                if (deleted) {
                    clearFields();
                    reloadTableData();
                    showSuccess("Type deleted successfully.");
                } else {
                    showError("Failed to delete type. It may be used
by existing projects.");
                }
```

```
            }
        } catch (DataAccessException e) {
            showError("Failed to delete type: " + e.getMessage());
        }
    }
}
```

Текст файлу util/HibernateUtil.java

```
package ckmetrics.util;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
/**
 * Керує SessionFactory Hibernate
 */
public class HibernateUtil {

    private static final SessionFactory sessionFactory =
buildSessionFactory();
    /**
     * Приватний конструктор для запобігання створенню екземплярів
утилітного класу
     */
    private HibernateUtil() {
        throw new UnsupportedOperationException("Utility class");
    }
    /**
     * Створює SessionFactory з hibernate.cfg.xml
     *
     * @return екземпляр SessionFactory
     */
    private static SessionFactory buildSessionFactory() {
        try {
            // Створення SessionFactory з hibernate.cfg.xml
            return                                                  new
Configuration().configure().buildSessionFactory();
        } catch (Throwable ex) {
            System.err.println("Failed to create SessionFactory: " +
ex);
            throw new ExceptionInInitializerError(ex);
        }
    }
    /**
     * Повертає екземпляр SessionFactory
     *
     * @return екземпляр SessionFactory
     */
    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
    /**
     * Ініціалізує SessionFactory примусово
     * Цей метод викликається при запуску застосунку для забезпечення
     * раннього виявлення проблем із конфігурацією Hibernate
```

```
     *
     * @throws IllegalStateException якщо SessionFactory не вдалося
ініціалізувати
     */
    public static void initialize() {
        // Примусова ініціалізація класу через перевірку статичного
поля
        if (sessionFactory == null) {
            throw new IllegalStateException("Failed to initialize
SessionFactory");
        }
        // Якщо вже тут, то ініціалізація пройшла успішно
    }
    /**
     * Вимикає SessionFactory та закриває пул з'єднань
     */
    public static void shutdown() {
        // Закриття кешів та пулу з'єднань
        getSessionFactory().close();
    }
}
```

Текст файлу hibernate.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<hibernate-configuration>
    <session-factory>
        <!-- Налаштування підключення до бази даних -->
        <property
name="hibernate.connection.driver_class">org.sqlite.JDBC</property>
        <property
name="hibernate.connection.url">jdbc:sqlite:javack_metrics.db</prope
rty>
        <!-- SQL діалект для SQLite, тобто сумісний з версією Hibernate
7 -->
        <property
name="hibernate.dialect">org.hibernate.community.dialect.SQLiteDiale
ct</property>
        <!-- Виведення всіх SQL запитів у консоль -->
        <property name="hibernate.show_sql">false</property>
        <property name="hibernate.format_sql">true</property>
        <!-- Автоматичне створення/оновлення схеми бази даних -->
        <property name="hibernate.hbm2ddl.auto">update</property>
        <!-- Вимкнення кешу другого рівня -->
        <property
name="hibernate.cache.use_second_level_cache">false</property>
        <!-- Маппінг сутностей -->
        <mapping class="ckmetrics.model.Type"/>
        <mapping class="ckmetrics.model.Project"/>
        <mapping class="ckmetrics.model.ExcludedFolder"/>
        <mapping class="ckmetrics.model.Setting"/>
    </session-factory>
</hibernate-configuration>
```

ДОДАТОК В – ІНСТРУКЦІЯ КОРИСТУВАЧА ПЗ ДЛЯ АВТОМАТИЗАЦІЇ ЗБОРУ МЕТРИК GITHUB ПРОЄКТІВ НА МОВІ JAVA УТИЛІТОЮ CK

Першим що бачить користувач після запуску, програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK, є вкладка зі списком типів проєктів у вигляді таблиці, яка представлена нижче на рисунку В.1.

Подальша взаємодія з даними цієї таблиці здійснюється за допомогою кнопок керування як-от «Add», «Edit», «Delete» та «Clear».

Кнопка «Add» призначена для додавання нового типу проєкту: користувач вводить назву в поле вводу та підтверджує операцію натисканням цієї кнопки.

Кнопка «Edit» використовується для редагування наявного типу проєкту: користувач обирає потрібний запис у таблиці, змінює його назву в полі вводу та підтверджує операцію натисканням цієї кнопки.

Кнопка «Delete» призначена для вилучення вибраного типу проєкту: користувач обирає запис у таблиці, натискає на цю кнопку та підтверджує операцію в діалоговому вікні.

Кнопка «Clear» використовується для очищення поля вводу та зняття поточного вибору в таблиці.

Екранні знімки інтерфейсу користувача представлено на рисунках В.1-В.5.

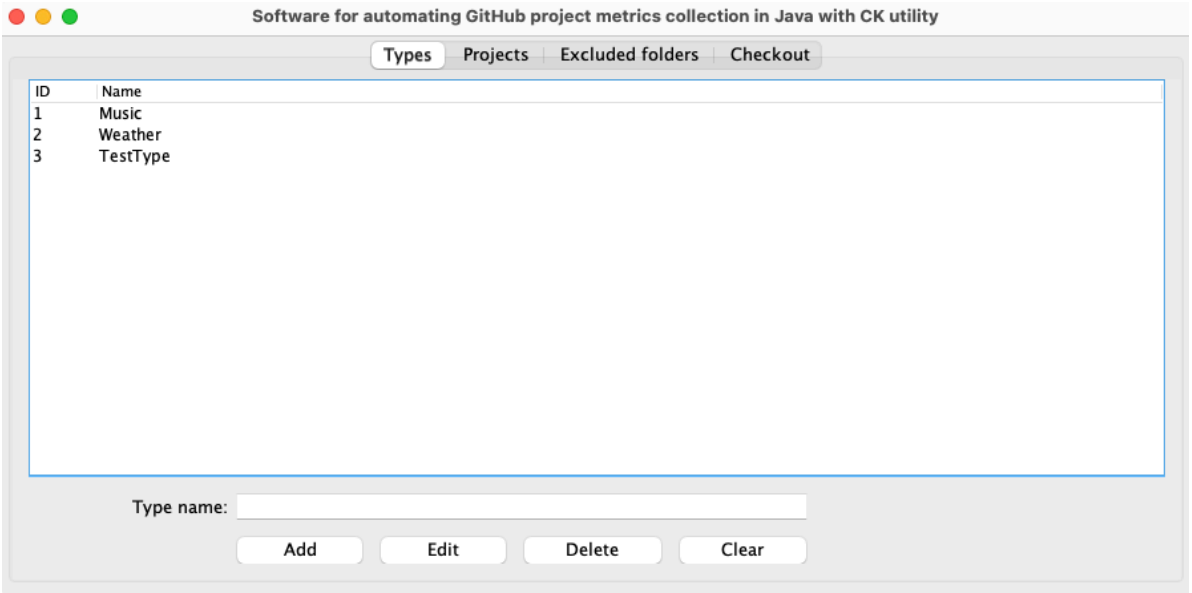


Рисунок В.1 – Екранний знімок вкладки зі списком типів проєктів

На вкладку зі списком проєктів у вигляді таблиці, яка представлена нижче на рисунку В.2, можна перейти через натискання мишею на заголовок «Projects».

Кнопка «Add» призначена для додавання нового проєкту: користувач вводить посилання GitHub-репозиторію в поле вводу, обирає необхідний тип проєкту з випадаючого списку або його відсутність та підтверджує операцію натисканням цієї кнопки.

Кнопка «Edit» використовується для редагування наявного проєкту: користувач обирає потрібний запис у таблиці, змінює посилання GitHub-репозиторію та/або тип проєкту у відповідних полях і підтверджує операцію натисканням цієї кнопки.

Кнопка «Delete» призначена для вилучення вибраного проєкту: користувач обирає запис у таблиці, натискає на цю кнопку та підтверджує операцію в діалоговому вікні.

Кнопка «Clear використовується для очищення поля вводу, поля випадаючого списку та зняття поточного вибору в таблиці.

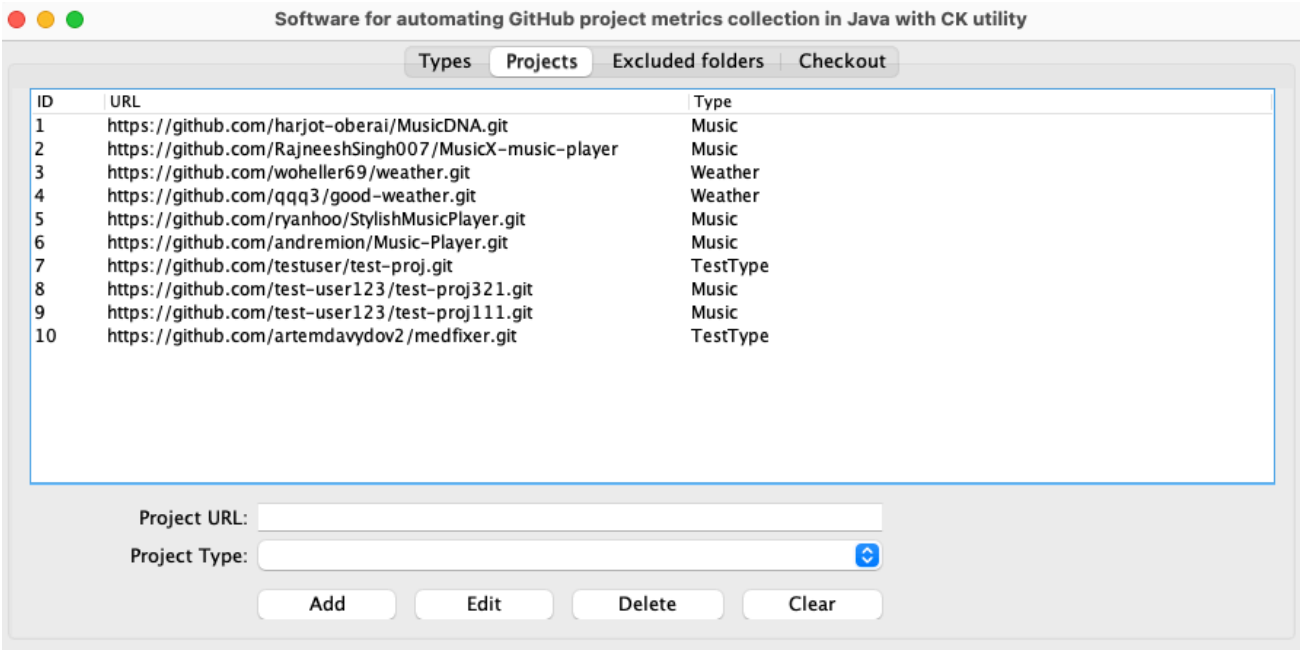


Рисунок В.2 – Екранний знімок вкладки зі списком проєктів

На вкладку зі списком виключених каталогів у вигляді таблиці, яка представлена нижче на рисунку В.3, можна перейти через натискання мишею на заголовок «Excluded folders».

Кнопка «Add» призначена для додавання нового виключеного каталогу: користувач вводить назву в поле вводу та підтверджує операцію натисканням цієї кнопки.

Кнопка «Edit» використовується для редагування наявного виключеного каталогу: користувач обирає потрібний запис у таблиці, змінює його назву в полі вводу та підтверджує операцію натисканням цієї кнопки.

Кнопка «Delete» призначена для вилучення вибраного виключеного каталогу: користувач обирає запис у таблиці, натискає на цю кнопку та підтверджує операцію в діалоговому вікні.

Кнопка «Clear» використовується для очищення поля вводу та зняття поточного вибору в таблиці.

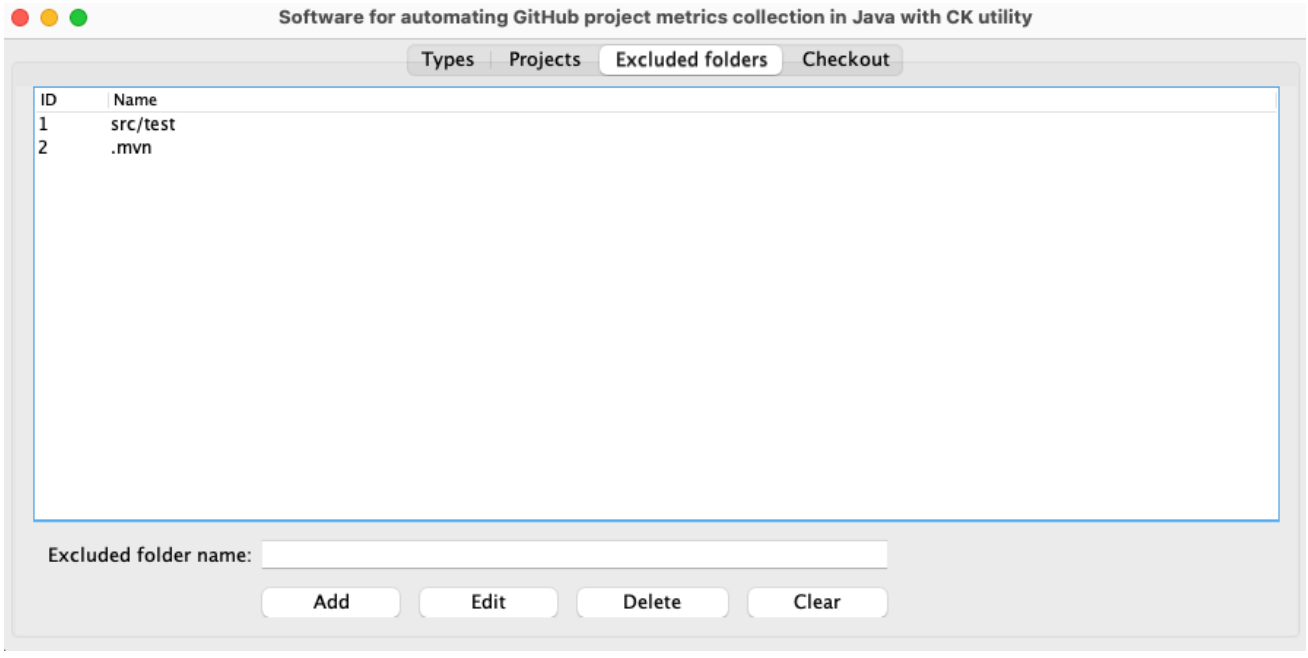


Рисунок В.3 – Екранний знімок вкладки зі списком виключених каталогів

На вкладку зі списком проєктів у вигляді таблиці та різноманітними налаштуваннями для збору метрик, яка представлена нижче на рисунку В.4, можна перейти через натискання мишею на заголовок «Checkout».

Тут користувач може обрати тип проєкту для аналізу, перелік виключених каталогів або їх відсутність, режим checkout, дату для checkout, каталог для збереження результатів, а також визначити те, чи потрібно видаляти клоновані проєкти після завершення аналізу.

Кнопка «Clear» застосовується для очищення обраних виключених каталогів.

Радіокнопка «Main branch» потрібна для встановлення режиму checkout з головної гілки без використання дати.

Радіокнопка «Latest tag» потрібна для встановлення режиму checkout за останнім тегом згідно вказаної дати та активації поля з датою.

Кнопка «Calendar» використовується для відкриття селектору у вигляді календаря, щоб обрати потрібну дату.

Чекбокс «Delete cloned projects» застосовується для вилучення клонованих проєктів з файлової системи після завершення процесу збору метрик.

Кнопка «Run Checkout and Collect Metrics» використовується для checkout проєктів згідно обраних налаштувань, аналізу CK та формування файлу формату XLSX під назвою «ck-metrics-summary» у вибраному каталозі операційної системи.

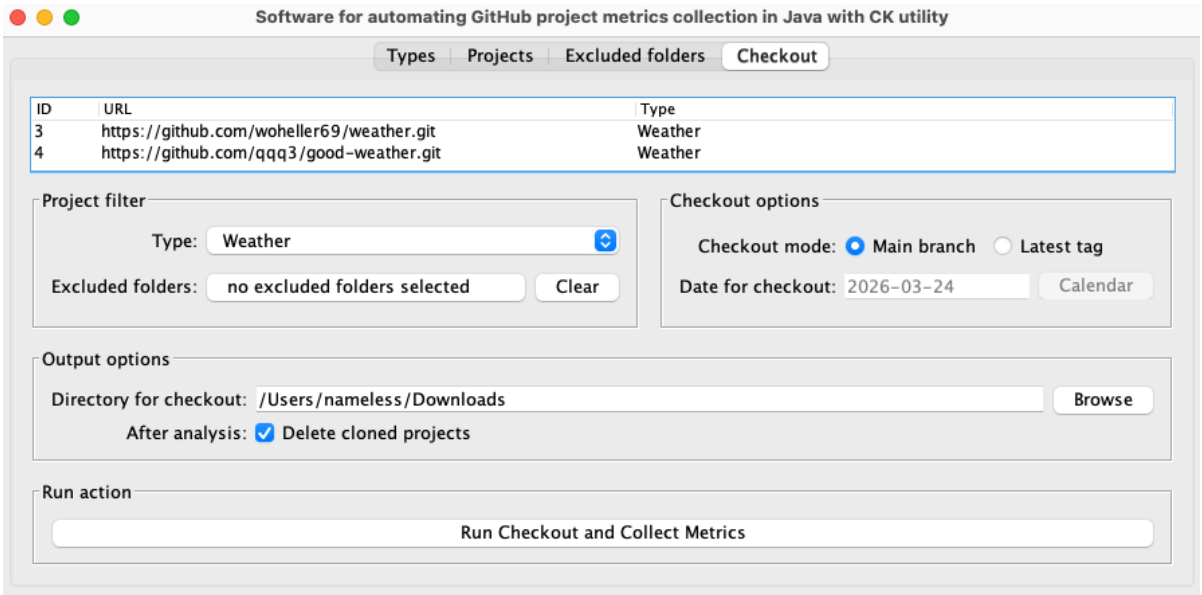


Рисунок В.4 – Екранний знімок вкладки «Checkout»

В результаті, якщо формування метрик GitHub проєктів згідно обраного типу проєкту й інших різних налаштувань виконається успішно, тоді користувач отримає відповідне системне повідомлення, яке представлено нижче на рисунку В.5, щодо коректної обробки всіх клонованих проєктів утилітою CK із назвою результуючого файлу формату XLSX, де збережено як агреговані метрики, так і метрики по кожному класу, та його абсолютним шляхом в операційній системі.

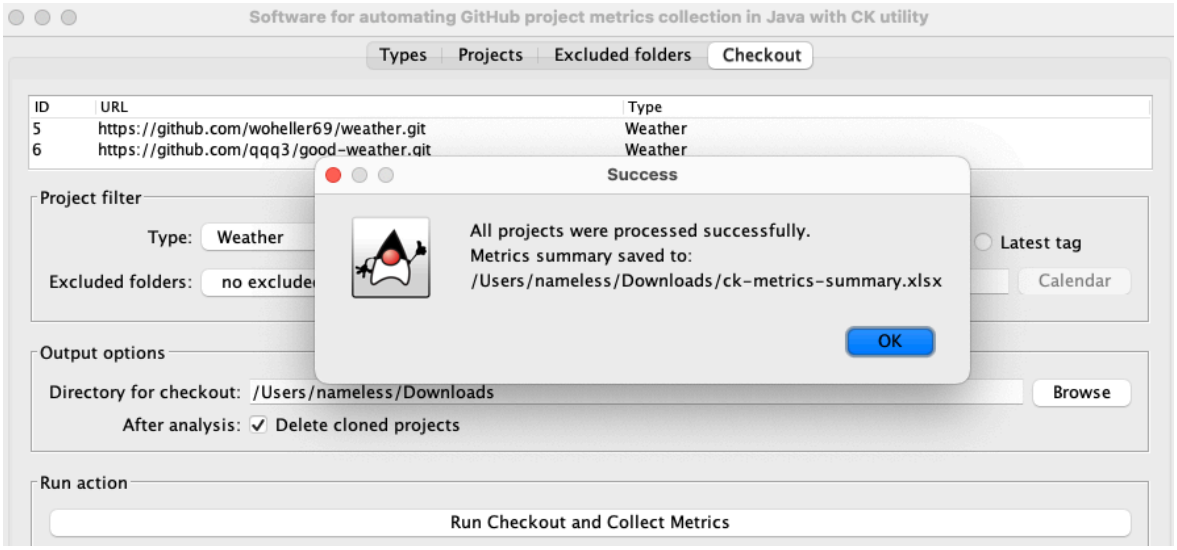


Рисунок В.5 – Екранний знімок результату щодо успішного збору метрик

Файл з метриками формату XLSX згідно обраного типу проєкту представлено нижче на рисунку В.6, де на першому листі міститься інформація про агреговані метрики, а на другому інформація про метрики по кожному класу.

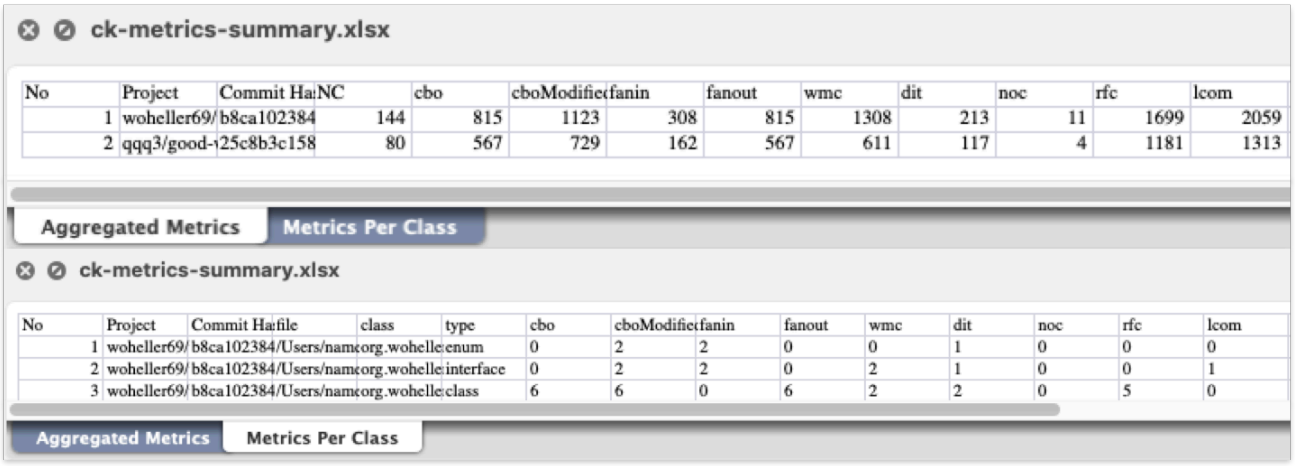


Рисунок В.6 – Вміст результуючого файлу з метриками

ДОДАТОК Г – ОПИС ПЗ ДЛЯ АВТОМАТИЗАЦІЇ ЗБОРУ МЕТРИК GITHUB ПРОЄКТІВ НА МОВІ JAVA УТИЛІТОЮ CK

Вступ

Назва застосунку це програмне забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK.

Позначення застосунку це «Software for automating GitHub project metrics collection in Java with CK utility».

1 Функціональне призначення

1.1 Класи розв’язуваних завдань та призначення програми

Технічної сутністю розробленого ПЗ є автоматизація процесу зі збору метрик відповідно до коду проєктів на мові Java з репозиторіїв GitHub через застосування утиліти CK. Наш застосунок вирішує наступні користувацькі задачі:

- Вести перелік проєктів, що відповідають GitHub-репозиторіям, та класифікувати їх за типами;
- Задавати виключені каталоги для фільтрації вмісту репозиторіїв під час аналізу;
- Клонувати репозиторії з GitHub;
- Виконувати checkout потрібного стану репозиторію;
- Запускати утиліту CK для отримання метрик Java-коду;
- Формувати вихідний файл із результатами аналізу у форматі XLSX.

Ця програма може бути використана в IT-компаніях для моніторингу якості Java-проєктів з відкритим програмним кодом на GitHub, в академічних установах для досліджень метрик коду або індивідуальними розробниками для швидкого аналізу репозиторіїв.

1.2 Функціональні обмеження на застосування

Дане ПЗ підтримує аналіз виключно Java-коду, бо утиліта CK, яка використовується в ньому для збору метрик, підтримує тільки цю мову програмування.

При використанні застосунку варто мати стабільний та швидкісний доступ до інтернету для його стабільної роботи під час клонування проєктів з GitHub репозиторіїв, але при цьому не можливо клонувати та обробляти приватні репозиторії з GitHub або ті, які не існують там взагалі.

2 Опис логічної структури

2.1 Опис структури

Програмне забезпечення побудоване на основі об’єктно-орієнтованого підходу з використанням багатошарової архітектури та елементів шаблону MVC, де функції представлення реалізуються засобами Swing, а частина керуючої та прикладної логіки зосереджена переважно в окремих сервісних класах.

Основні модулі програмного забезпечення такі як:

Модуль доступу до даних реалізований через DAO-класи як-от ProjectDAO, TypeDAO, ExcludedFolderDAO, SettingDAO, що забезпечують виконання CRUD-операцій і взаємодію з таблицями локальної бази даних SQLite.

Модуль роботи з GitHub-репозиторіями реалізований через клас GitHubService, який відповідає за клонування репозиторіїв, визначення головної гілки, отримання тегів і перемикання на потрібний стан репозиторію.

Модуль checkout і керування процесом збору метрик реалізований через клас CheckoutService, який координує отримання списку проєктів, підготовку локальних каталогів, запуск checkout, збір помилок і формування підсумкового результату виконання.

Модуль аналізу метрик реалізований через класи CkService та пов’язані з ним компоненти, наприклад CkCsvParser, що відповідають за запуск утиліти CK,

зчитування згенерованих CSV-результатів та підготовку даних для подальшого звітування.

Модуль графічного інтерфейсу реалізований засобами javax.swing і включає основні панелі такі як TypesPanel, ProjectsPanel, ExcludedFoldersPanel, CheckoutPanel, а також допоміжні інтерфейсні компоненти для введення параметрів, перегляду списків та запуску операцій.

Модуль моделей даних представлений JPA-сутностями як-от Project, Type, ExcludedFolder, Setting, які відображають структуру предметної області та відповідають таблицям локальної бази даних.

Конфігураційні елементи включають в себе два файли, а саме файл hibernate.cfg.xml для виконання конфігурації взаємодії з базою даних SQLite і файл pom.xml для керування залежностями та збіркою проєкту.

2.2 Мови програмування

Для розробки ПЗ було використано мову програмування Java. Вона застосовується для реалізації графічного інтерфейсу, бізнес-логіки, роботи з GitHub-репозиторіями, запуску утиліти CK, обробки результатів аналізу та формування підсумкового звіту.

Додатково у застосунку використовується SQL при роботі з базою даних SQLite, де зберігаються типи проєктів, проєкти, виключені каталоги та налаштування для подальшої генерації метрик.

Виклики утиліти CK та взаємодія з файловою системою також виконуються засобами мови Java, що дозволяє реалізувати всі основні функції застосунку в межах єдиного програмного середовища.

2.3 Вхідні та вихідні дані

- Для додавання інформації про тип проєкту:

Вхідні дані це назва типу проєкту.

Вихідні дані це новий запис про тип проєкту у таблиці БД «Типи» та системне повідомлення про успішне додавання.

- Для корегування інформації про тип проєкту:

Вхідні дані це обраний рядок з таблиці «Типи» і заповнені нові дані щодо його назви.

Вихідні дані це оновлений запис про тип проєкту у таблиці БД «Типи» зі зміненою інформацією та системне повідомлення про успішне корегування.

- Для вилучення інформації про тип проєкту:

Вхідні дані це обраний рядок з таблиці «Типи» і підтвердження дії в діалоговому вікні.

Вихідні дані це вилучений запис про тип проєкту у таблиці БД «Типи» та системне повідомлення про успішне вилучення, якщо даний тип проєкту ще не належить жодному проєкту.

- Для відображення інформації про типи проєктів:

Вхідні дані це відкрита вкладка «Типи».

Вихідні дані це відображення таблиці зі списком типів проєктів та заповненими даними.

- Для додавання інформації про проєкт:

Вхідні дані це посилання проєкту та вибір типу проєкту зі списку або його відсутність.

Вихідні дані це новий запис про проєкт у таблицях БД «Проєкти» і «Типи проєктів» та системне повідомлення про успішне додавання.

- Для корегування інформації про проєкт:

Вхідні дані це обраний рядок з таблиці «Проєкти» і заповнені нові дані щодо його посилання та типу проєкту або його відсутності.

Вихідні дані це оновлений запис про проєкт у таблицях БД «Проєкти» і «Типи проєктів» та системне повідомлення про успішне корегування.

- Для вилучення інформації про проєкт:

Вхідні дані це обраний рядок з таблиці «Проєкти» і підтвердження дії в діалоговому вікні.

Вихідні дані це вилучений запис про проєкт у таблицях БД «Проєкти» і «Типи проєктів» та системне повідомлення про успішне вилучення.

- Для відображення інформації про проєкти:

Вхідні дані це відкрита вкладка «Проєкти».

Вихідні дані це відображення таблиці зі списком проєктів та заповненими даними.

- Для додавання інформації про виключений каталог:

Вхідні дані це назва виключеного каталогу.

Вихідні дані це новий запис про виключений каталог у таблиці БД «Виключені каталоги» та системне повідомлення про успішне додавання.

- Для корегування інформації про виключений каталог:

Вхідні дані це обраний рядок з таблиці «Виключені каталоги» і заповнені нові дані щодо його назви.

Вихідні дані це оновлений запис про виключений каталог у таблиці БД «Виключені каталоги» зі зміненою інформацією та системне повідомлення про успішне корегування.

- Для вилучення інформації про виключений каталог:

Вхідні дані це обраний рядок з таблиці «Виключені каталоги» і підтвердження дії в діалоговому вікні.

Вихідні дані це вилучений запис про виключений каталог у таблиці БД «Виключені каталоги» та системне повідомлення про успішне вилучення.

- Для відображення інформації про виключені каталоги:

Вхідні дані це відкрита вкладка «Виключені каталоги».

Вихідні дані це відображення таблиці зі списком виключених каталогів та заповненими даними.

- Для формування метрик:

Вхідні дані це обраний тип проєкту, з яким пов’язано хоча б один проєкт, задані необхідні виключені каталоги або їх відсутність, визначена гілка коду, вказаний шлях до директорії для збереження вихідного файлу з агрегованими даними.

- Вихідні дані це згенерований файл метрик формату XLSX, що містить у перших трьох стовпцях No проєкту, назву і вендора Project та його Commit Hash, а

потім стовпці із такими метриками як NC, cbo, cboModified, fanin, fanout, wmc, dit, noc, rfc, lcom, totalMethodsQty, staticMethodsQty, publicMethodsQty, privateMethodsQty, protectedMethodsQty, defaultMethodsQty, visibleMethodsQty, abstractMethodsQty, finalMethodsQty, synchronizedMethodsQty, totalFieldsQty, staticFieldsQty, publicFieldsQty, privateFieldsQty, protectedFieldsQty, defaultFieldsQty, finalFieldsQty, synchronizedFieldsQty, nosi, loc, returnQty, loopQty, comparisonsQty, tryCatchQty, parenthesizedExpsQty, stringLiteralsQty, numbersQty, assignmentsQty, mathOperationsQty, variablesQty, maxNestedBlocksQty, anonymousClassesQty, innerClassesQty, lambdasQty, uniqueWordsQty, logStatementsQty, у вказаній директорії.

3 Склад і функції

3.1 Склад програмного забезпечення

Після збірки ПЗ постачається у вигляді одного виконуваного JAR-файлу з основним кодом і всіма необхідними залежностями для запуску застосунку, а утиліта CK використовується як окремий зовнішній JAR-файл.

3.2 Функції програмного забезпечення

- Додавання, корегування, вилучення про тип проєкту та відображення інформації про типи проєктів, тобто наявність дій CRUD для даної сутності.

- Додавання, корегування, вилучення інформації про проєкт та відображення інформації про проєкти, тобто наявність дій CRUD для даної сутності.

- Додавання, корегування, вилучення інформації про виключений каталог та відображення інформації про виключені каталоги, тобто наявність дій CRUD для даної сутності.

- Формування метрик.

4 Умови застосування

4.1 Вимоги до складу та параметрів технічних засобів

Операційна система: macOS, Linux та Windows.

Процесор: 4 ядра або більше із частотою щонайменше 3 ГГц.

Оперативна пам’ять: мінімум 4 ГБ.

Жорсткий диск: не менше 1 ГБ вільного простору для тимчасового збереження клонованих репозиторіїв і формування підсумкових звітів.

Інтернет: зі швидкістю прийнамні 5 Мбіт/с для стабільної роботи.

Монітор: з роздільною здатністю не менше 1920×1080 для зручної роботи з графічним інтерфейсом програмного забезпечення.

Клавіатура та миша: стандартні пристрої введення, що необхідні для повноцінної взаємодії з desktop-застосунком.

4.2 Вимоги до програмної сумісності

Для функціонування ПЗ необхідна наявність середовища виконання Java Runtime Environment чи Java Development Kit з версією 21.

Також ПЗ повинно бути сумісним із засобами локального зберігання даних, бібліотеками доступу до репозиторіїв та компонентами, необхідними для формування вихідних звітів.

4.3 Спосіб виклику програми

Запуск ПЗ здійснюється шляхом виконання основного JAR-файлу застосунку засобами платформи Java, а саме за допомогою команди «java -jar» або через графічне середовище операційної системи за умови належного налаштування середовища виконання.

Вхідною точкою програми є клас Main, у якому виконується ініціалізація графічного інтерфейсу, підключення до локальної бази даних і запуск основних компонентів застосунку, необхідних для роботи з проєктами, параметрами checkout і збором метрик.

ДОДАТОК Д – ПРОГРАМА ТА МЕТОДИКА ПЗ ДЛЯ АВТОМАТИЗАЦІЇ ЗБОРУ МЕТРИК GITHUB ПРОЄКТІВ НА МОВІ JAVA УТИЛІТОЮ CK

1 Об’єкт випробувань

Об’єкт випробувань це програмне забезпечення для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK.

2 Мета випробувань

Основна мета випробувань це процес перевірки щодо відповідності розробленого ПЗ згідно вимог у технічному завданні, а також коректності реалізації всіх наявних функції та стабільності його роботи.

3 Вимоги до програми

Під час випробувань варто перевірити те, що розроблене ПЗ повноцінно забезпечує та реалізує такі функції як:

- Додавання/корегування/вилучення інформації про тип проєкту та відображення інформації про типи проєктів.
- Додавання/корегування/вилучення інформації про проєкт та відображення інформації про проєкти.
- Додавання/корегування/вилучення інформації про виключений каталог та відображення інформації про виключені каталоги.
- Формування метрик.

4 Вимоги до програмної документації

До складу програмної документації включені такі частини як технічне завдання для розробки ПЗ, текст ПЗ, текст із керівництвом користувача для даного ПЗ, текст опису ПЗ, а також власне саме ПЗ та методику його випробувань.

5 Засоби та порядок випробувань

5.1 Засоби випробувань

Склад технічних засобів для використання у випробуваннях наступний:

- ПК із 4-х ядерним процесором та його мінімальною частотою в 3 ГГц.
- Наявність швидкісного доступу до інтернету в даному ПК на рівні 5Мбіт/с.

Склад програмних засобів для використання у випробуваннях наступний:

- Операційна система macOS на версії щонайменше Sequoia.
- Утиліта CK в якості файлу формату jar версії 0.7.1-SNAPSHOT.

5.2 Порядок проведення випробувань

- Перевірка комплексності програмної документації ПЗ, яке призначено для автоматизації збору метрик GitHub проєктів на мові Java утилітою CK.
- Перевірка функцій додавання/корегування/вилучення інформації про тип проєкту та відображення інформації про типи проєктів.
- Перевірка функцій додавання/корегування/вилучення інформації про проєкт та відображення інформації про проєкти.
- Перевірка функцій додавання/корегування/вилучення інформації про виключений каталог та відображення інформації про виключені каталоги.
- Перевірка функції формування метрик.

6 Методи випробувань

- Випробування функції додавання інформації про тип проєкту.

У запущеному ПЗ треба спершу перейти на вкладку «Типи», а вже потім ввести унікальну назву нового типу проєкту у текстове поле для даних під списком та підтвердити виконання операції через натиснення кнопки «Add».

Очікуваний результат: Новий тип проєкту додається до списку у вкладці «Типи» та зберігається в таблиці «Types» бази даних.

- Випробування функції корегування інформації про тип проєкту.

У запущеному ПЗ треба перейти на вкладку «Типи», обрати потрібний тип проєкту зі списку, змінити його назву у текстовому полі для даних та підтвердити виконання операції через натиснення кнопки «Edit».

Очікуваний результат: Назва обраного типу проєкту оновлюється у списку вкладки «Типи», а відповідний запис змінюється в таблиці «Types» бази даних.

- Випробування функції вилучення інформації про тип проєкту.

У запущеному ПЗ треба перейти на вкладку «Типи», обрати потрібний тип проєкту зі списку, виконати натиснення кнопки «Delete» та підтвердити дію у діалоговому вікні через вибір відповідної кнопки підтвердження.

Очікуваний результат: Якщо обраний тип проєкту не пов’язаний з іншими проєктами, він вилучається зі списку і з таблиці «Types», але якщо цей тип проєкту вже використовується, тоді записи в таблицях «Type» і «Types_Projects» залишаються без змін, а система виводить повідомлення про неможливість вилучення.

- Випробування функції відображення інформації про типи проєктів.

У запущеному ПЗ треба перейти на вкладку «Типи» та виконати перегляд наявних записів.

Очікуваний результат: Система відображає список типів проєктів у вигляді таблиці на основі даних із таблиці «Types», а за відсутності записів показує порожній список.

- Випробування функції додавання інформації про проєкт.

У запущеному ПЗ треба перейти на вкладку «Проєкти», ввести коректну URL-адресу GitHub-репозиторію у текстове поле для даних під списком, обрати тип проєкту з випадаючого списку та підтвердити виконання операції через натиснення кнопки «Add».

Очікуваний результат: Новий проєкт додається до списку у вкладці «Проєкти», запис зберігається в таблиці «Projects», а його зв’язок із типом проєкту фіксується через проміжну таблицю «Types_Projects».

- Випробування функції корегування інформації про проєкт.

У запущеному ПЗ треба перейти на вкладку «Проєкти», обрати проєкт зі списку, змінити його URL-адресу у текстовому полі для даних під списком або тип проєкту з випадаючого списку та підтвердити виконання операції через натиснення кнопки «Edit».

Очікуваний результат: Дані обраного проєкту оновлюються у списку вкладки «Проєкти», відповідний запис змінюється в таблиці «Projects», а за потреби оновлюється і зв’язок у таблиці «Types_Projects».

- Випробування функції вилучення інформації про проєкт.

У запущеному ПЗ треба перейти на вкладку «Проєкти», обрати потрібний проєкт зі списку, виконати натиснення кнопки «Delete» та підтвердити дію у діалоговому вікні через вибір відповідної кнопки підтвердження.

Очікуваний результат: Обраний проєкт вилучається зі списку у вкладці «Проєкти», а відповідні записи видаляються з таблиць «Projects» і «Types_Projects» бази даних.

- Випробування функції відображення інформації про проєкти.

У запущеному ПЗ треба перейти на вкладку «Проєкти» та виконати перегляд наявних записів.

Очікуваний результат: Система відображає список проєктів у вигляді таблиці на основі даних із таблиці «Projects», а за відсутності записів показує порожній список.

- Випробування функції додавання інформації про виключений каталог.

У запущеному ПЗ треба перейти на вкладку «Виключені каталоги», ввести назву нового виключеного каталогу у текстове поле для даних під списком та підтвердити виконання операції через натиснення кнопки «Add».

Очікуваний результат: Новий виключений каталог додається до списку у вкладці «Виключені каталоги» та зберігається в таблиці «Excluded_Folders» бази даних.

- Випробування функції корегування інформації про виключений каталог.

У запущеному ПЗ треба перейти на вкладку «Виключені каталоги», обрати потрібний запис зі списку, змінити його назву у текстовому полі для даних під списком та підтвердити виконання операції через натиснення кнопки «Edit».

Очікуваний результат: Назва обраного виключеного каталогу оновлюється у списку вкладки «Виключені каталоги», а відповідний запис змінюється в таблиці «Excluded_Folders» бази даних.

- Випробування функції вилучення інформації про виключений каталог.

У запущеному ПЗ треба перейти на вкладку «Виключені каталоги», обрати потрібний запис зі списку, виконати натиснення кнопки «Delete» та підтвердити дію у діалоговому вікні через вибір відповідної кнопки підтвердження.

Очікуваний результат: Обраний виключений каталог вилучається зі списку у вкладці «Виключені каталоги» та видаляється з таблиці «Excluded_Folders» бази даних.

- Випробування функції відображення інформації про виключені каталоги.

У запущеному ПЗ треба перейти на вкладку «Виключені каталоги» та виконати перегляд наявних записів.

Очікуваний результат: Система відображає список виключених каталогів у вигляді таблиці на основі даних із таблиці «Excluded_Folders», а за відсутності записів показує порожній список.

- Випробування функції формування метрик.

У запущеному ПЗ треба перейти на вкладку «Checkout», обрати потрібний тип проєкту, з яким пов’язано хоча б один проєкт, за потреби задати виключені

каталоги, вибрати гілку або режим checkout за датою, вказати доступну директорію для збереження результатів та підтвердити виконання операції через натиснення кнопки «Run Checkout and Collect Metrics».

Очікуваний результат: На основі даних із таблиць «Types», «Projects», «Types_Projects» і «Excluded_Folders» програма виконує завантаження та аналіз проєктів за допомогою утиліти CK, а потім формує та зберігає у вказаній директорії файл з метриками, який містить як і агреговані метрики, так і метрики по кожному класу.