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.

«___» _____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти «бакалавр»

на тему: Розробка програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics

Виконав: студентка групи 4151з

_____ Литвинова В.Ю.

(підпис, ПІБ)

Керівник роботи:

Доцент, к.т.н., доцент

(науковий ступень вчене звання)

_____ Пухалевич А.В.

(підпис, ПІБ)

Миколаїв – 2025 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)
«_08_» ___04___ 2025 р.

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

Студенту Литвиновій Валерії Юріївні
(Прізвище, ім'я, по батькові)

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

Керівник роботи: Пухалевич Андрій Володимирович

Затверджені наказом ректора №153 від 08.04.2025 р.

2. Термін подання роботи: 02.06.2025 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 08.04.2025 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	31.03.2025	ПП*
2.	Аналіз практичної задачі інженерії ПЗ	07.04.2025	ПП*
3.	Аналіз вимог до ПЗ	14.04.2025	ПП*
4.	Розробка архітектури ПЗ	21.04.2025	
5.	Розробка проєкту ПЗ	05.05.2025	
6.	Реалізація та тестування ПЗ	12.05.2025	
7.	Підготовка розділу Результати розробки ПЗ	16.05.2025	
8.	Підготовка розділу з охорони праці	19.05.2025	ПП*
9.	Підготовка розділу ВИСНОВКИ	23.05.2025	
10.	Оформлення списку використаних джерел та додатків	26.05.2025	
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.)	02.06.2025	не пізніше, ніж за 14 діб до захисту (згідно п.4.1 зазначеного Порядку
12.	Підготовка презентації та доповіді	06.06.2025	
13.	Попередній захист роботи на засіданні кафедри ПЗАС	09.06.2025	
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	16.06.2025	

* - за результатами переддипломної практики (ПП), яка була з 03.02.2025 до 23.03.2025 р.

Студент _____ Литвинова В.Ю.
(підпис) (ПІБ)

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

АНОТАЦІЯ

Кваліфікаційна робота присвячена вирішенню практичної задачі інженерії програмного забезпечення, що характеризується комплексністю та невизначеністю умов, а саме розробці програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics.

Було проаналізовано практичну задачу інженерії програмного забезпечення з розробки програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics, розглянуто існуючі аналоги та виконано постановку задачі на розробку програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics. Здійснено аналіз вимог до програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics. Розроблено проєкт програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics. Проведено тестування та випробування програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics.

Кваліфікаційна робота викладена на 107 сторінках друкованого тексту та містить 13 рисунків, 6 таблиць, 5 додатків та список використаних джерел з 10 найменувань.

ABSTRACT

The qualification work is dedicated to solving a practical software engineering problem characterized by complexity and uncertainty, specifically the development of software for automating the collection of GitHub project metrics in PHP using the PhpMetrics utility.

The practical software engineering problem of developing software for automating the collection of GitHub project metrics in PHP using the PhpMetrics utility was analyzed, existing analogs were reviewed, and the problem statement for developing software for automating the collection of GitHub project metrics in PHP using the PhpMetrics utility was formulated. The requirements for the software for automating the collection of GitHub project metrics in PHP using the PhpMetrics utility were analyzed. The software design for automating the collection of GitHub project metrics in PHP using the PhpMetrics utility was developed. Testing and trials of the software for automating the collection of GitHub project metrics in PHP using the PhpMetrics utility were conducted.

The qualification work is presented on 107 pages of printed text and includes 13 figures, 6 tables, 5 appendices, and a list of references with 10 sources.

ЗМІСТ

ВСТУП

Дана кваліфікаційна робота присвячена розв’язанню практичної задачі інженерії програмного забезпечення, що характеризується комплексністю та невизначеністю умов, а саме розробці програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics. Збір метрик програмних проєктів є процесом аналізу структури коду, складності, зв’язаності класів та інших характеристик. Крім того, метрики коду використовуються при побудові моделей для оцінювання якості програмного забезпечення [1].

Наразі, збір метрик GitHub проєктів на мові PHP доводиться виконувати в ручному режимі, що є дуже повільним процесом, так як користувач має спочатку отримати код проєкту, використовуючи утиліту Git чи закачавши і розпакувавши Zip архів, після чого має запустити утиліту PhpMetrics і скопіювати значення необхідних метрик в окремий файл. Крім того, для отримання середніх значень метрик на клас, доводиться проводити окремі розрахунки для кожного проєкту, що вимагає ще більше часу. Автоматизація цих дій дозволить значно прискорити збір метрик GitHub проєктів на мові PHP.

Метою роботи є розробка програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics.

Завдання:

1) провести аналіз практичної задачі інженерії програмного забезпечення з розробки програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics;

2) розглянути існуюче програмне забезпечення для автоматизації збору метрик програмних проєктів.

3) виконати постановку задачі на розробку програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics;

4) провести аналіз вимог до програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics;

5) вибрати архітектуру програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics;

6) розробити проєкт програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics;

7) провести тестування та випробування програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics.

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

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

1.1 Аналіз практичної задачі інженерії програмного забезпечення з розробки програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics

Розробка програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics є практичною задачею інженерії програмного забезпечення, що характеризується комплексністю та невизначеністю умов. Комплексність та невизначеність реалізації цього програмного забезпечення зумовлено необхідністю управління роботою утилітами Git і PhpMetrics для отримання значень метрик програмних проєктів розташованих на GitHub, а також необхідністю побудови інтуїтивно зрозумілого інтерфейсу користувача. Запуск і обробка результатів роботи вказаних утиліт вимагає врахування можливості збоїв через проблеми з мережею чи через аварійне завершення роботи утиліт.

Наразі збір метрик для PHP-проєктів, розміщених на GitHub, виконується переважно в ручному режимі. Цей процес включає кілька етапів:

1) отримання вихідного коду проєкту (за допомогою утиліти Git або шляхом завантаження та розпакування Zip-архіву;

2) запуск утиліти PhpMetrics для аналізу коду;

3) ручна обробка результатів роботи утиліти PhpMetrics.

Для отримання вихідного коду проєкту використовується утиліта Git, або ж завантажується та розпаковується Zip-архів. Використання утиліти Git вимагає знання параметрів командного рядка, або встановлення графічного інтерфейсу користувача для даної утиліти (в тому числі необхідно вміти використовувати додаткові параметри, щоб завантажити лише останню

версію проєкту, не завантажуючи історію версій, яка може займати сотні мегабайт місця на диску). Завантаження Zip-архіву вимагає розпаковування завантаженого файлу, а потім його видалення.

Запуск утиліти PhpMetrics також вимагає знання параметрів командного рядка, оскільки за замовчуванням утиліта виводить результати в текстовому форматі, округлюючи усереднені значення метрик до двох знаків, що не є достатньою точністю в деяких випадках. Тому за допомогою параметрів командного рядка треба генерувати звіт у форматі csv, який міститиме повні дані.

Необхідність ручної обробки результатів роботи утиліти PhpMetrics пояснюється тим, що звіт у форматі csv містить значення метрик для кожного класу проєкту окремо, а для подальшого використання необхідно мати суму, або ж усереднені значення метрик (залежить від метрики). Ці розрахунки доводиться виконувати вручну для кожного проєкту окремо. Потім необхідно скопіювати отримані метрики у окремий файл. Такий підхід є трудомістким, повільним і схильним до людських помилок, що знижує ефективність аналізу та ускладнює повторення процесу для великої кількості проєктів. Такий підхід є непрактичним для професійного використання, що підкреслює необхідність автоматизації процесу збору метрик.

Розробка програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics дозволить автоматизувати рутинні операції, пов’язані зі збором метрик, зменшити вплив людського фактора, підвищити швидкість збору метрик коду без помилок, виникнення яких можливе при ручній обробці результатів.

1.2 Аналіз існуючого програмного забезпечення для автоматизації збору метрик програмних проєктів

Спеціалізованим інструментом для аналізу PHP-коду є утиліта PhpMetrics [2], яка дозволяє отримати значення основних метрик

програмного забезпечення, таких як кількість класів і методів у проєкті, цикломатична складність, індекс зв'язаності, метрики LCOM, DIT та інші.

PhpMetrics працює шляхом аналізу вихідного коду PHP-проєктів і генерує детальні звіти у форматі HTML, JSON або CSV, що дозволяє користувачам переглядати результати через вебінтерфейс або обробляти їх програмно. Утиліта підтримує інтеграцію з командним рядком, що робить її придатною для автоматизації.

В той самий час, відсутність вбудованої інтеграції з GitHub для автоматичного отримання коду репозиторіїв вимагає виконання додаткових дій користувача, щоб завантажити код з GitHub, а також щоб отримати суму чи середнє значення метрик зі згенерованих детальних звітів. Відсутність інтуїтивно зрозумілого, графічного інтерфейсу робить неможливим використання утиліти PhpMetrics для користувачів, які не вміють користуватись командним рядком.

Утиліта MetricHunter [3] дозволяє збір метрик програмного забезпечення із публічних GitHub репозиторіїв, в тому числі деякі ООП метрики (рисунок 1.1).

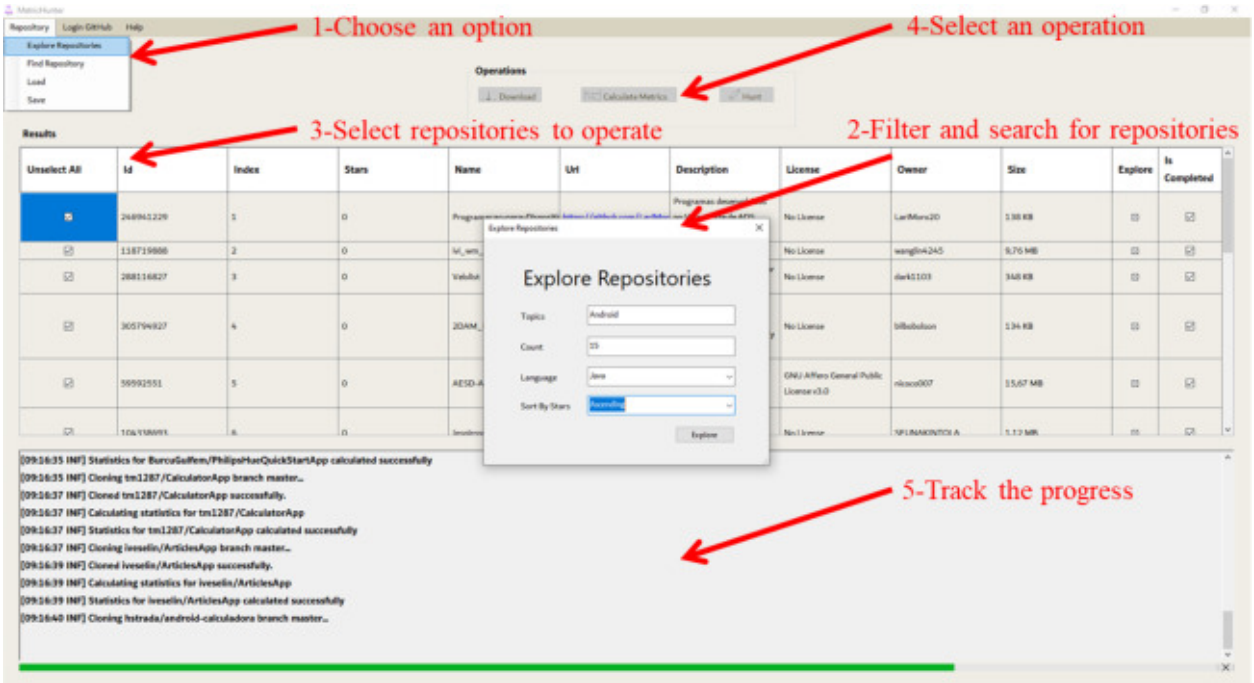


Рисунок 1.1 – Знімок екрану Утиліти MetricHunter [3]

MetricHunter автоматизує процес збору вказаних метрик, виконуючи пошук публічних репозиторіїв за заданими критеріями (наприклад, мова

програмування, кількість зірок), після чого виконуючи завантаження коду, запуск утиліти SourceMonitor для збору метрик, та генерує набори даних у форматі CSV для подальшого аналізу.

MetricHunter підтримує збір метрик для кількох мов програмування, включаючи C++, Java, C#, але не PHP. Відсутність підтримки PHP робить неможливим отримання метрик коду для PHP-проєктів. Крім того, набір метрик утиліти MetricHunter є обмеженим у порівнянні з PhpMetrics (наприклад, відсутні LCOM або DIT). Робота MetricHunter залежна від стабільності GitHub API, що вимагає створення ключа доступу і може викликати збої. Запуск утиліти можливий тільки Windows 11, а на Linux системах MetricHunter працювати не буде.

Проведений аналіз показує, що PhpMetrics є оптимальним вибором для збору метрик коду PHP-проєктів завдяки своїй спеціалізації на мові PHP та підтримці широкого спектру метрик. При цьому PhpMetrics не має повної автоматизації всіх етапів збору метрик (отримання коду, аналіз, збереження та агрегація даних) для PHP-проєктів на GitHub. Для усунення цього недоліку пропонується розробка програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics. Таке рішення дозволить автоматизувати і підвищити швидкість збору метрик коду, а також зробити процес доступним для широкого кола користувачів і зможе працювати як на платформі Windows, так і на платформі Linux.

1.3 Постановка задачі на розробку програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics

Виходячи з виконаного аналізу практичної задачі інженерії програмного забезпечення з розробки програмного забезпечення та аналізу існуючого програмного забезпечення для автоматизації збору метрик програмних проєктів сформулюємо постановку задачі: необхідно розробити програмне забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics.

Вимоги до складу виконуваних функцій

Програмне забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics повинне забезпечити можливість виконання наступних перерахованих функцій:

1) додавання типу проєкту;

2) видалення типу проєкту;

3) перегляд списку типів проєктів;

4) додавання проєкту;

5) видалення проєкту;

6) перегляд списку проєктів;

7) отримання метрик проєктів.

Вимоги до організації вхідних та вихідних даних:

Для додавання типу проєкту: вхідними даними є назва типу проєкту. Вихідними даними є запис у базі даних про новий тип проєкту.

Для видалення типу проєкту: вхідними даними є ідентифікатор або назва типу проєкту. Вихідними даними є оновлена база даних без вказаного типу проєкту.

Для перегляду списку типів проєктів: вхідними даними є запит на отримання списку. Вихідними даними є список типів проєктів у табличному або іншому зрозумілому форматі.

Для додавання проєкту: вхідними даними є URL-адреса GitHub репозиторію та тип проєкту. Вихідними даними є запис у базі даних про новий проєкт.

Для видалення проєкту: вхідними даними є ідентифікатор або URL-адреса проєкту. Вихідними даними є оновлена база даних без вказаного проєкту.

Для перегляду списку проєктів: вхідними даними є запит на отримання списку. Вихідними даними є список проєктів із зазначенням їх URL-адрес та типів у табличному форматі.

Для отримання метрик проєктів: вхідними даними є тип проєкту, дата, станом на яку необхідно отримати код, та шлях до каталогу, в який буде виконано завантаження вихідного проєктів. Вихідними даними є звіт у табличному форматі із проєктами вибраного типу та агрегованими значеннями метрик (сума або середнє) отриманих після запуску утиліти PhpMetrics: кількість рядків коду, кількість логічних рядків коду, кількість рядків коментарів, кількість класів, кількість інтерфейсів, кількість методів, середнє вхідне зчеплення, середнє вихідне зчеплення, середня нестабільність, глибина дерева успадкування, середня цикломатична складність, середня відносна складність системи, середня складність, середня кількість помилок на клас, середня кількість дефектів на клас.

Вимоги до складу та параметрів технічних засобів:

Операційна система: Windows або Linux.

Процесор: Intel Core i3 або еквівалент, 2.5 ГГц, 2 ядра та більше.

Інтернет: стабільне з’єднання зі швидкістю 1 Мбіт/с.

Оперативна пам’ять: 4 ГБ.

Жорсткий диск: мінімум 500 МБ вільного простору для зберігання вихідного коду проєктів і звітів.

Розроблене програмне забезпечення має забезпечити автоматизацію всіх етапів збору метрик, починаючи від завантаження коду до збереження результатів, а також надати зручний графічний інтерфейс для користувачів, які не мають навиків у роботі з командним рядком. Це дозволить використовувати програмне забезпечення для аналізу великої кількості PHP-проєктів на GitHub.

2 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОМАТИЗАЦІЇ ЗБОРУ МЕТРИК GITHUB ПРОЄКТІВ НА МОВІ PHP УТИЛІТОЮ PHPMETRICS

2.1 Аналіз вимог до програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics

Всі дії з програмним забезпеченням будуть виконуватись одним актором – користувачем. Варіанти використання програмного забезпечення, які відповідають постановці задачі на його розробку, зображено на рисунку 2.1.

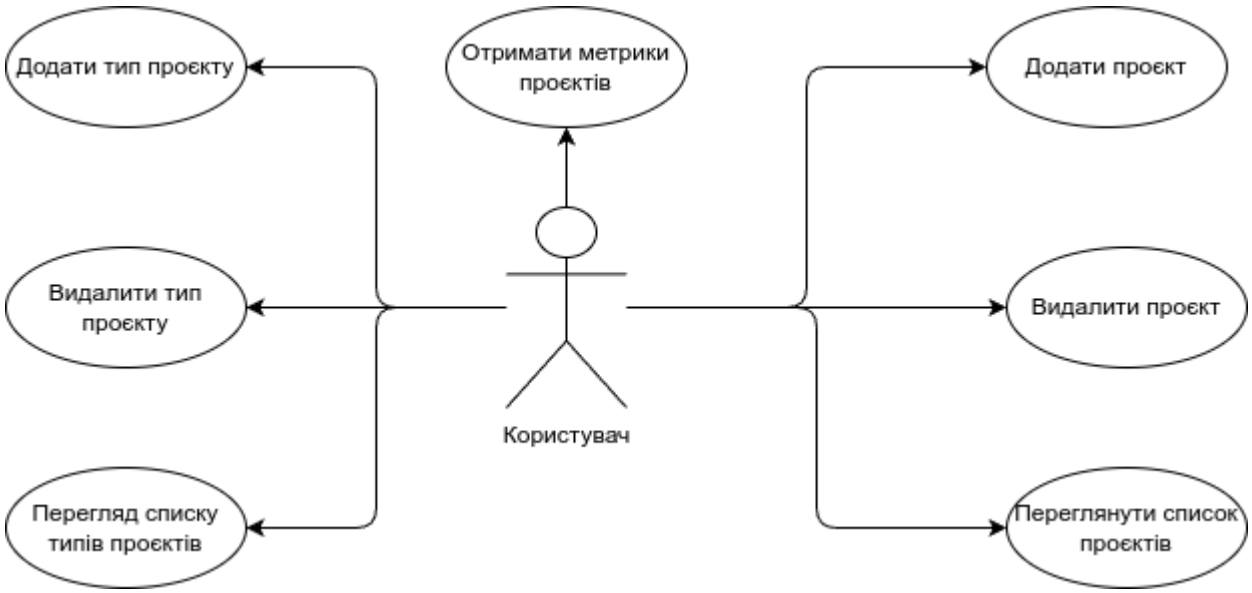


Рисунок 2.1 – Діаграма варіантів використання програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics

Специфікації варіантів використання наведено у таблицях 2.1-2.7.

Таблиця 2.1 – Специфікація варіанту використання «Додати тип проєкту»

Складова	Опис
Назва прецеденту	Додати тип проєкту
Опис прецеденту	Користувач додає новий тип проєкту, щоб класифікувати GitHub проєкти за певними категоріями (наприклад, "Інтернет магазин", "Гра")
Актори	Користувач
Передумови	Назва типу проєкту не існує в базі даних
Базовий потік	Користувач відкриває список типів проєктів. Користувач вводить назву типу проєкту (рядок, до 255 символів). Користувач підтверджує введення натисканням кнопки "Додати".
Альтернативний потік	Якщо назва порожня або перевищує 255 символів, користувачу відображається повідомлення про помилку з проханням ввести коректну назву. Якщо назва вже існує, користувачу відображається повідомлення про дублювання з пропозицією ввести іншу назву.
Післяумови	Новий тип проєкту додано до бази даних із назвою та унікальним ідентифікатором

Таблиця 2.2 – Специфікація варіанту використання «Видалити тип проєкту»

Складова	Опис
Назва прецеденту	Видалити тип проєкту
Опис прецеденту	Користувач видаляє існуючий тип проєкту з бази даних
Актори	Користувач
Передумови	Тип проєкту існує в базі даних
Базовий потік	Користувач відкриває список типів проєктів. Користувач обирає тип проєкту зі списку. Користувач підтверджує видалення натискаючи кнопку "Видалити".
Альтернативний потік	Якщо тип проєкту пов’язаний із проєктами, користувачу відображається повідомлення про неможливість видалення з пропозицією видалити пов’язані проєкти
Післяумови	Тип проєкту видалено з бази даних

Таблиця 2.3 – Специфікація варіанту використання «Додати проєкт»

Складова	Опис
Назва прецеденту	Додати проєкт
Опис прецеденту	Користувач додає новий GitHub проєкт до бази даних для подальшого аналізу метрик
Актори	Користувач
Передумови	URL-адреса є коректною адресою GitHub репозиторію.
Базовий потік	Користувач відкриває список проєктів. Користувач вводить URL-адресу GitHub репозиторію (рядок, наприклад, https://github.com/власник/репозиторій). Користувач обирає тип проєкту зі списку доступних типів. Користувач підтверджує введення натисканням кнопки "Додати".
Альтернативний потік	Якщо URL не валідна, користувачу відображається повідомлення "Некоректна URL-адреса" із проханням ввести коректну адресу. Повернення до кроку 1. Якщо проєкт із такою URL уже існує, користувачу відображається повідомлення про дублювання.
Післяумови	Новий проєкт додано до бази даних із URL, типом та ідентифікатором

Таблиця 2.4 – Специфікація варіанту використання «Видалити проєкт»

Складова	Опис
Назва прецеденту	Видалити проєкт
Опис прецеденту	Користувач видаляє існуючий проєкт із бази даних
Актори	Користувач
Передумови	Проєкт існує в базі даних
Базовий потік	Користувач відкриває список проєктів. Користувач обирає проєкт зі списку. Користувач підтверджує видалення натисканням кнопки "Видалити".
Альтернативний потік	Якщо проєкт не вибрано, користувачу відображається повідомлення "Проєкт не існує".
Післяумови	Проєкт видалено з бази даних

Таблиця 2.5 – Специфікація варіанту використання «Отримати метрики проєктів»

Складова	Опис
Назва прецеденту	Отримати метрики проєктів
Опис прецеденту	Користувач запускає аналіз метрик для проєктів обраного типу за допомогою PhpMetrics і отримує агрегований звіт
Актори	Користувач
Передумови	Для обраного типу є щонайменше один проєкт. Шлях до каталогу для завантаження коду проєктів існує та доступний для запису.
Базовий потік	Користувач обирає тип проєкту. Користувач вводить дату, станом на яку необхідно отримати код (наприклад, 2025-05-20). Користувач вказує шлях до каталогу для завантаження вихідного коду. Користувач підтверджує запит натисканням кнопки "Отримати метрики". Користувачу відображається звіт у табличному форматі.
Альтернативний потік	Якщо для типу немає проєктів, користувачу відображається повідомлення "Проєкти відсутні". Якщо дата не валідна, користувачу відображається повідомлення "Некоректна дата" із проханням ввести правильну дату. Якщо шлях до каталогу не валідний або недоступний, користувачу відображається повідомлення "Недоступний каталог" із проханням вказати інший шлях. Якщо Git Checkout не вдалося виконати, користувачу відображається повідомлення про технічну помилку з пропозицією перевірити налаштування утиліти. Якщо PhpMetrics не вдалося виконати, користувачу відображається повідомлення про технічну помилку з пропозицією перевірити налаштування утиліти.
Післяумови	Користувач отримав звіт із агрегованими метриками.

2.2 Архітектура програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics

Для розробки програмного забезпечення, призначеного для автоматизації збору метрик GitHub проєктів утилітою PhpMetrics, обрано архітектурний шаблон MVC (Model-View-Controller). Цей шаблон забезпечує чіткий розподіл відповідальностей між компонентами системи, сприяючи модульності, зручності підтримки та масштабованості [4].

Шаблон MVC поділяє програмне забезпечення на три основні компоненти:

1) Model (модель): відповідає за управління даними, бізнес-логіку та взаємодію з базою даних.

2) View (вигляд): забезпечує відображення даних користувачу через графічний інтерфейс.

3) Controller (контролер): відповідає за взаємодію між моделлю та виглядом, обробляючи запити користувача та викликаючи відповідні сервіси для отримання або обробки даних.

Діаграма взаємодії архітектури MVC показана на рисунку 2.2.

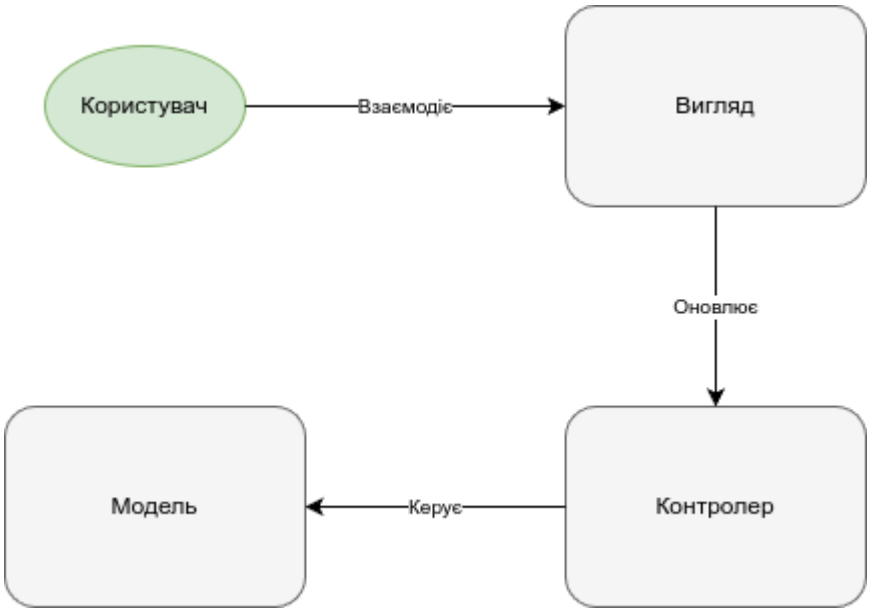


Рисунок 2.2 – Діаграма взаємодії архітектури MVC

Використання MVC дозволяє ізолювати логіку обробки даних від їх представлення, що полегшує тестування та подальший розвиток системи.

Згідно з [5], шаблон MVC є ефективним для створення програмного забезпечення із чітким розподілом функціональності.

2.3 Проєкт програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics

2.3.1 Ескізний проєкт програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics

В ескізному проєкті програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics створено базову структуру, яка визначає основні сценарії взаємодії користувача з програмним забезпеченням, організацію даних та прототип інтерфейсу користувача. Для цього розроблено діаграми діяльності для основних варіантів використання, концептуальну модель для представлення структури даних та прототип інтерфейсу користувача.

Концептуальна модель даних представлена у вигляді ER-діаграми показана на рисунку 2.3.

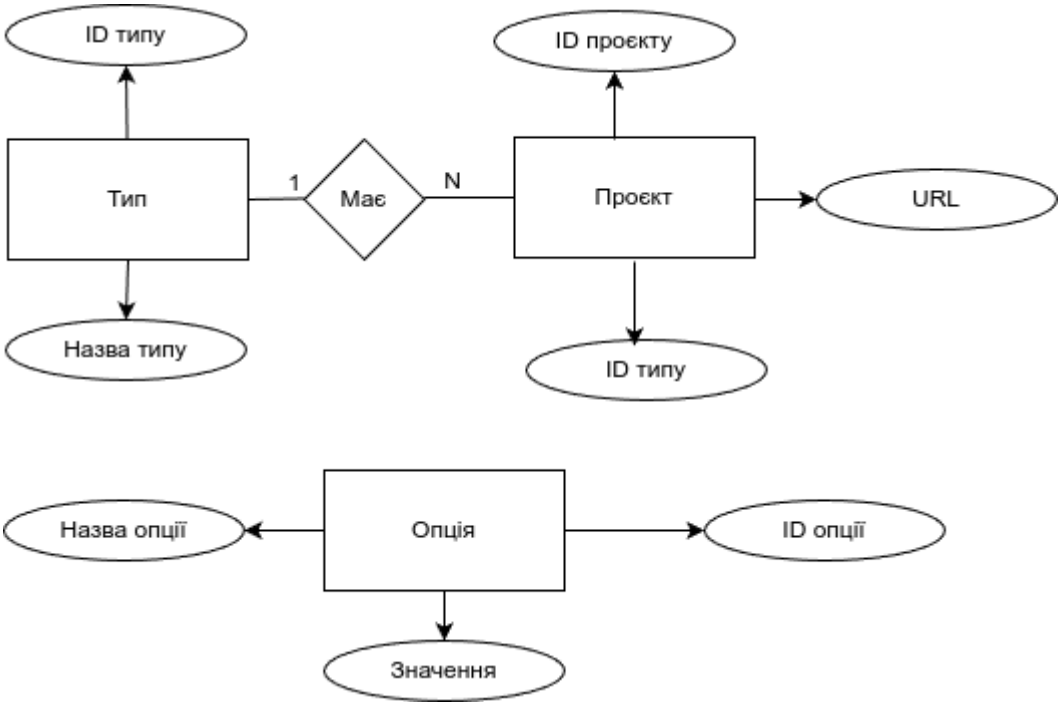


Рисунок 2.3 – Концептуальна модель даних програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics

Діаграми діяльності відображають послідовність дій користувача та програмного забезпечення для виконання функцій, таких як додавання типу проєкту, видалення типу проєкту, додавання проєкту, видалення проєкту та отримання метрик проєктів.

Діаграму діяльності для варіанту використання «Отримати метрики проєктів» показано на рисунку 2.4.

Рисунок 2.4 – Діаграма діяльності для варіанту використання «Отримати метрики проєктів»

Ескізи інтерфейсу користувача показано на рисунках 2.5 – 2.6.

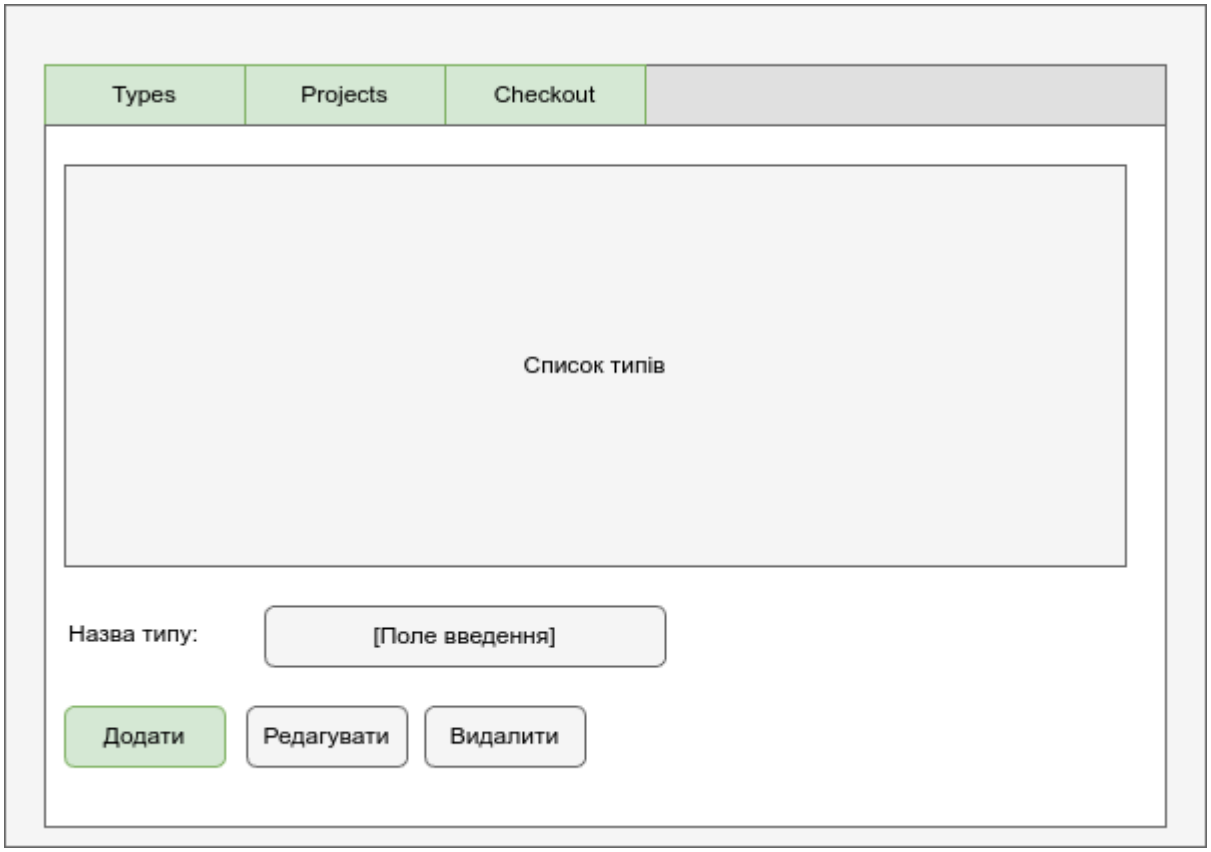


Рисунок 2.5 – Прототип панелі зі списком типів проєктів

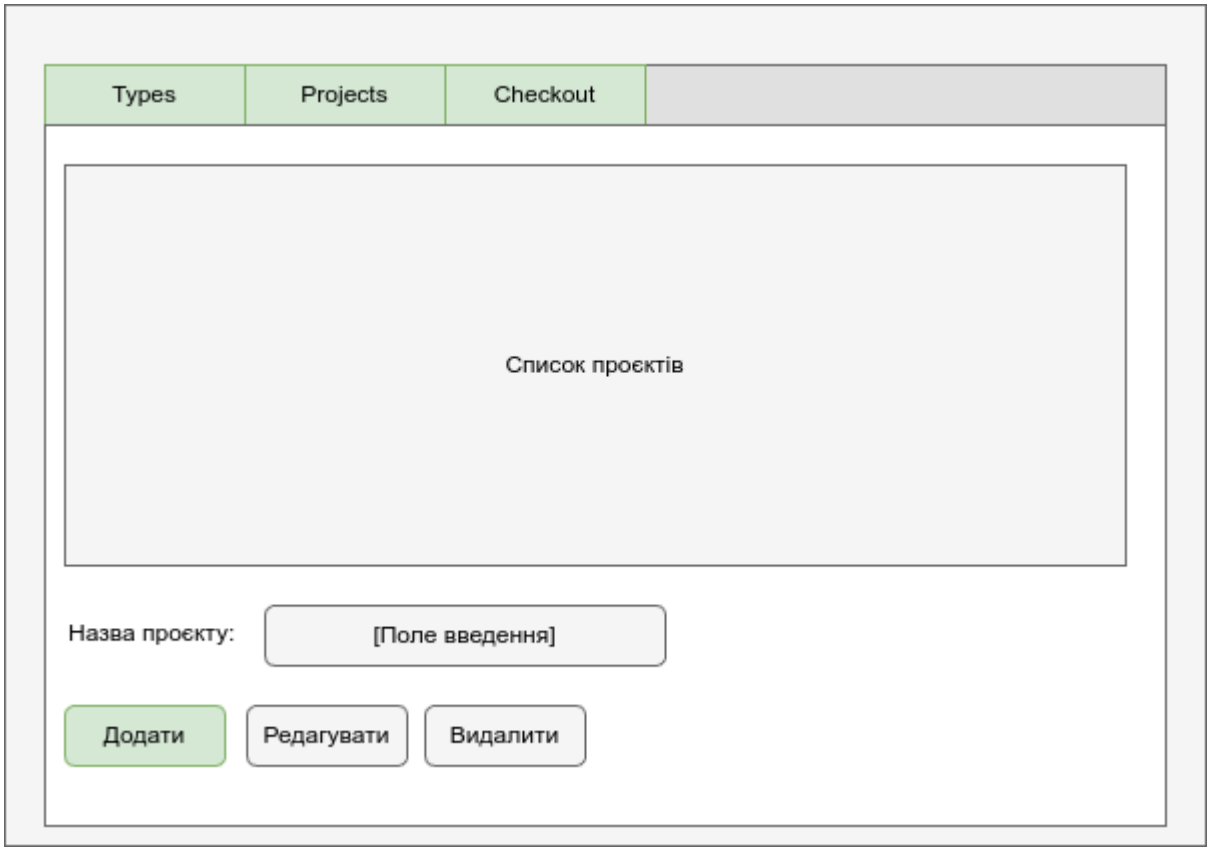


Рисунок 2.6 – Прототип панелі зі списком проєктів

2.3.2 Технічний проєкт програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics

Технічний проєкт програмного забезпечення деталізує статичну та динамічну моделі, а також логічну модель даних, які є основою для реалізації програмного забезпечення. Статична модель представлена UML-діаграмою класів та специфікаціями класів, динамічна модель — UML-діаграмами послідовності для ключових прецедентів.

Діаграма класів програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics показана на рисунку 2.6.

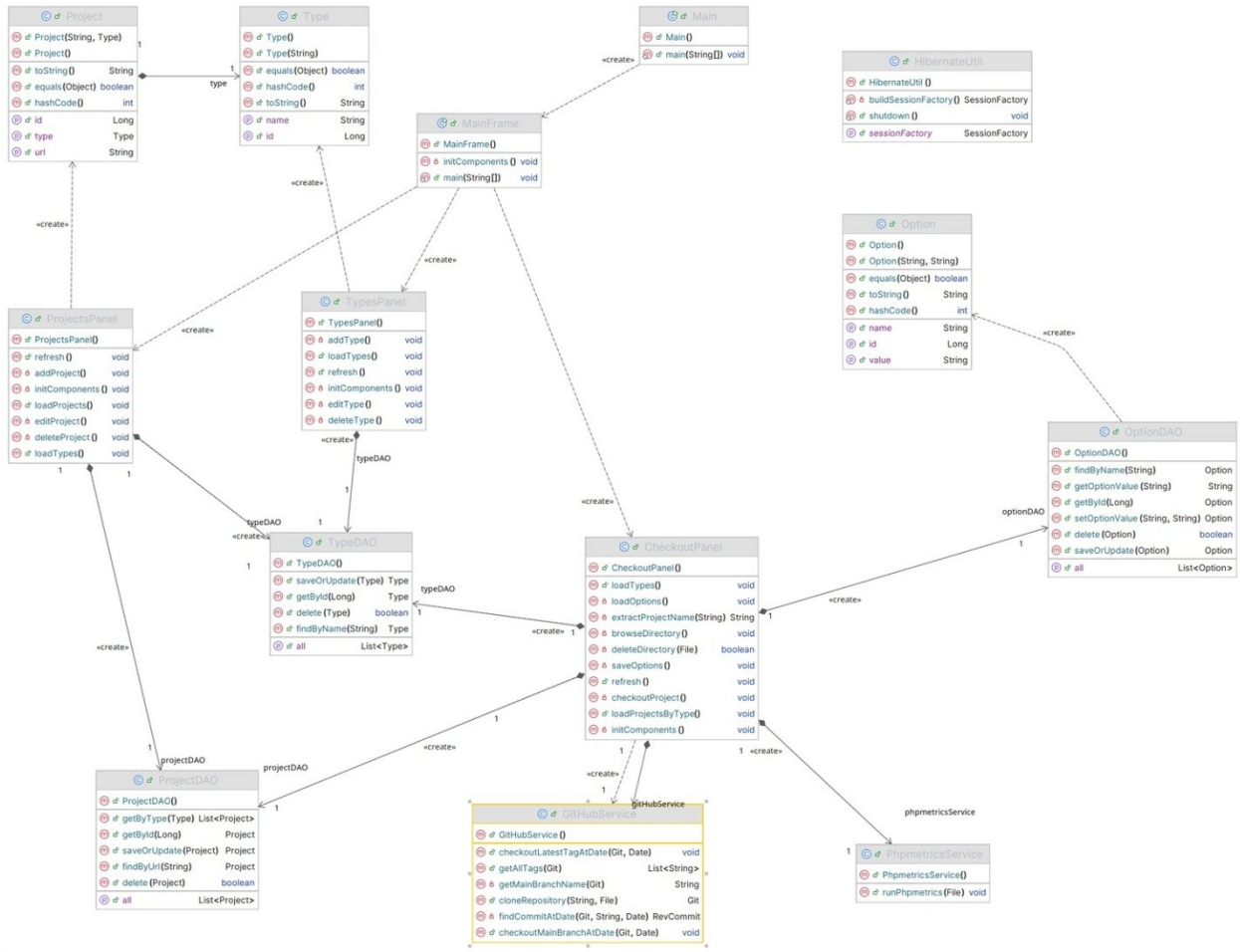


Рисунок 2.6 – Діаграма класів програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics

Специфікація класу Main

Призначення: Точка входу в додаток GitHub Project Manager, ініціалізує Hibernate, налаштовує вигляд інтерфейсу та запускає головне вікно. Клас є простою точкою входу, яка делегує основну функціональність MainFrame та HibernateUtil.

Атрибути: Відсутні (статичний клас).

Методи:

main(args: String[]): void: Запускає додаток

Специфікація класу PhpmetricsService

Призначення: Виконує утиліту PhpMetrics для аналізу вихідного коду проєкту в заданому каталозі та генерує звіти.

Атрибути:

PHPMETRICS_SCRIPT: String (static final) — шлях до скрипта PhpMetrics за замовчуванням.

Методи:

runPhpmetrics(directory: File): void: Запускає PhpMetrics у вказаному каталозі, створюючи звіти.

Специфікація класу GitHubService

Призначення: Надає функціональність для роботи з GitHub-репозиторіями, включаючи клонування, перевірку гілок і тегів за датою, а також отримання списку тегів.

Атрибути: Відсутні.

Методи:

cloneRepository(url: String, directory: File): Git Клонує репозиторій за заданим URL у вказаний каталог з обмеженою глибиною (depth=1) і без тегів. Параметри: url — URL репозиторію, directory — каталог для клонування. Повертає: Об’єкт Git для клонованого репозиторію.

checkoutMainBranchAtDate(git: Git, date: Date): void: Перемикається на коміт основної гілки (main/master), найближчий до заданої дати. Параметри: git — об’єкт Git, date — дата для вибору коміту.

checkoutLatestTagAtDate(git: Git, date: Date): void: Перемикається на найновіший тег, створений до заданої дати. Якщо тегів немає, використовує найстаріший тег або кидає виключення. Параметри: git, date.

getMainBranchName(git: Git): String (private): Визначає назву основної гілки (main/master) або повертає першу знайдену гілку.

findCommitAtDate(git: Git, branch: String, date: Date): RevCommit (private): Знаходить коміт у гілці, найближчий до заданої дати. Повертає: Об’єкт RevCommit.

getAllTags(git: Git): List<string>: Повертає список імен тегів у репозиторії. Повертає: Список назв тегів. Використовує бібліотеку JGit для роботи з Git-репозиторіями.</string>

Специфікація класу Option

Призначення: Модель даних для зберігання налаштувань програми.

Атрибути:

id: Long — унікальний ідентифікатор (генерується автоматично).

name: String — назва налаштування (унікальна, не null).

value: String — значення налаштування. Методи:

Option(): Порожній конструктор, потрібний для Hibernate.

Option(name: String, value: String): Конструктор із параметрами.

getId(): Long, setId(id: Long): void: Гетер і сетер для id.

getName(): String, setName(name: String): void: Гетер і сетер для name.

getValue(): String, setValue(value: String): void: Гетер і сетер для value.

toString(): String: Повертає рядок у форматі name: value.

equals(o: Object): boolean: Порівнює об’єкти за id і name.

hashCode(): int: Генерує хеш-код на основі id і name.

Специфікація класу Project

Призначення: Модель даних для GitHub-проєкту.

Атрибути:

id: Long — унікальний ідентифікатор (генерується автоматично).

url: String — URL репозиторію (унікальний, не null).

type: Type — тип проєкту (зв’язок Many-to-One з таблицею types).

Методи:

Project(): Порожній конструктор для Hibernate.

Project(url: String, type: Type): Конструктор із параметрами.

getId(): Long, setId(id: Long): void: Гетер і сетер для id.

getUrl(): String, setUrl(url: String): void: Гетер і сетер для url.

getType(): Type, setType(type: Type): void: Гетер і сетер для type.

toString(): String: Повертає url.

equals(o: Object): boolean: Порівнює об’єкти за id і url.

hashCode(): int: Генерує хеш-код на основі id і url.

Специфікація класу Type

Призначення: Модель даних для типу проєкту.

Атрибути:

id: Long — унікальний ідентифікатор (генерується автоматично).

name: String — назва типу (унікальна, не null). Методи:

Type(): Порожній конструктор для Hibernate.

Type(name: String): Конструктор із параметром.

getId(): Long, setId(id: Long): void: Гетер і сетер для id.

getName(): String, setName(name: String): void: Гетер і сетер для name.

toString(): String: Повертає name.

equals(o: Object): boolean: Порівнює об’єкти за id і name.

hashCode(): int: Генерує хеш-код на основі id і name.

Специфікація класу MainFrame

Призначення: Головне вікно програми, реалізує графічний інтерфейс із вкладками для управління типами, проєктами та збором метрик.

Атрибути:

tabbedPane: JTabbedPane — панель вкладок для відображення TypesPanel, ProjectsPanel і CheckoutPanel. Методи:

MainFrame(): Конструктор, ініціалізує вікно та викликає initComponents().

initComponents(): void (private): Створює JTabbedPane, додає вкладки (Types, Projects, Checkout) та слухача подій ChangeListener для оновлення даних при перемиканні вкладок.

Специфікація класу TypeDAO

Призначення: Об’єкт доступу до даних (DAO) для роботи з таблицею types через Hibernate.

Атрибути: Відсутні.

Методи:

saveOrUpdate(type: Type): Type: Зберігає або оновлює тип у базі даних. Повертає: Збережений або оновлений об’єкт Type або null при помилці.

getById(id: Long): Type: Отримує тип за ідентифікатором. Повертає: Об’єкт Type або null.

getAll(): List<type>: Повертає список усіх типів. Повертає: Список Type або null при помилці.</type>

delete(type: Type): boolean: Видаляє тип із бази даних.

findByName(name: String): Type: Шукає тип за назвою. Повертає: Об’єкт Type або null. Використовує транзакції Hibernate для забезпечення цілісності даних.

Специфікація класу ProjectDAO

Призначення: DAO для роботи з таблицею projects.

Атрибути: Відсутні.

Методи:

saveOrUpdate(project: Project): Project: Зберігає або оновлює проєкт. Повертає: Збережений або оновлений об’єкт Project або null.

getById(id: Long): Project: Отримує проєкт за ідентифікатором. Повертає: Об’єкт Project або null.

getAll(): List<project>: Повертає список усіх проєктів. Повертає: Список Project або null.

delete(project: Project): boolean: Видаляє проєкт.

findByUrl(url: String): Project: Шукає проєкт за URL. Повертає: Об’єкт Project або null.

getByType(type: Type): List<project>: Повертає список проєктів заданого типу. Повертає: Список Project або null.

Специфікація класу OptionDAO

Призначення: DAO для роботи з таблицею options

Атрибути: Відсутні.

Методи:

saveOrUpdate(option: Option): Option: Зберігає або оновлює налаштування. Повертає: Збережений або оновлений об’єкт Option або null.

getById(id: Long): Option: Отримує налаштування за ідентифікатором. Повертає: Об’єкт Option або null.

getAll(): List<option>: Повертає список усіх налаштувань. Повертає: Список Option або null.

delete(option: Option): boolean: Видаляє налаштування.

findByName(name: String): Option: Шукає налаштування за назвою. Повертає: Об’єкт Option або null.

getOptionValue(name: String): String: Отримує значення налаштування за назвою. Повертає: Значення Option або null.

setOptionValue(name: String, value: String): Option: Встановлює значення налаштування, створюючи нове, якщо не існує. Повертає: Оновлений або створений об’єкт Option.

Специфікація класу HibernateUtil

Призначення: Утиліта для управління SessionFactory Hibernate, забезпечує ініціалізацію та закриття сесій.

Атрибути:

sessionFactory: SessionFactory (static final) — фабрика сесій Hibernate. Методи:

buildSessionFactory(): SessionFactory (private static): Створює SessionFactory на основі hibernate.cfg.xml. Повертає: Об’єкт SessionFactory.

getSessionFactory(): SessionFactory (static): Повертає sessionFactory.

shutdown(): void (static): Закриває sessionFactory, очищаючи кеші та пул з’єднань.

Логічна модель даних для програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics показана на рисунку 2.7.

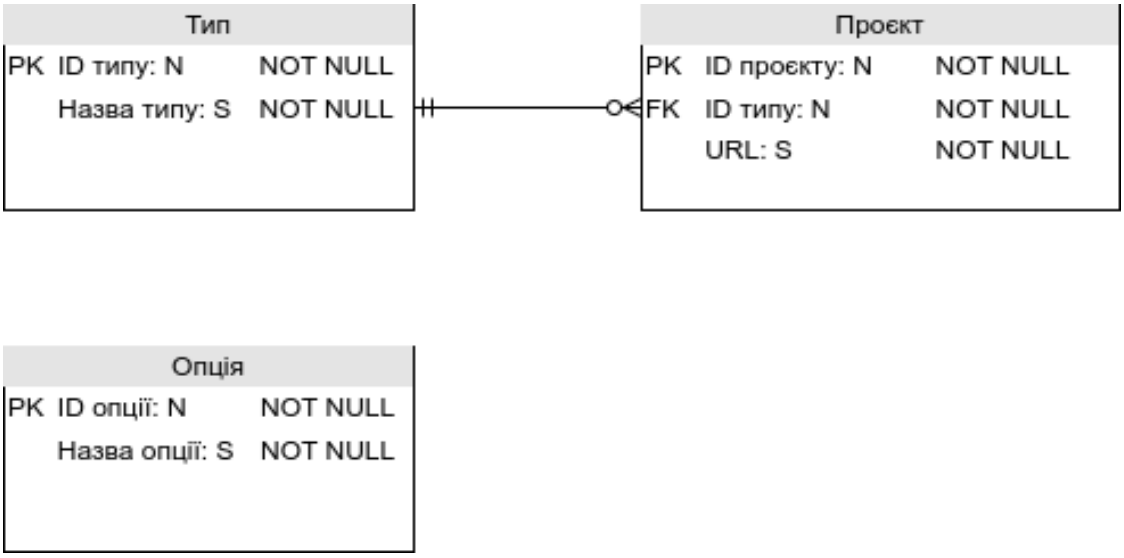


Рисунок 2.7 – Логічна модель даних програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics

2.3.3 Робочий проєкт програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics

Обґрунтування вибору засобів розробки програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics

Вибір засобів розробки було зроблено з врахуванням того, що програмне забезпечення має бути настільним, а також працювати в операційних системах Linux або Windows.

Мова програмування Java була обрана через її кросплатформність, що дозволяє запускати програму на операційних системах Windows і Linux без модифікації коду [6]. Java має розвинену екосистему бібліотек, включаючи бібліотеки для роботи з Git, базами даних і графічними інтерфейсами. Крім того, Java забезпечує надійність завдяки строгій типізації та механізмам обробки винятків, що є важливим для стабільної роботи програми при взаємодії з утилітами Git і PhpMetrics.

Система керування базами даних (СКБД) SQLite була обрана як легковагова вбудована СКБД, яка не потребує окремого серверного процесу, що спрощує розгортання програми [7]. SQLite підтримує SQL-запити, що достатньо для реалізації логічної моделі даних, яка включає таблиці для типів проєктів, проєктів і налаштувань. SQLite є кросплатформною, не потребує складного адміністрування і забезпечує достатню продуктивність для зберігання та обробки даних про GitHub проєкти.

Бібліотеку Hibernate було використано як ORM (Object-Relational Mapping) інструмент для спрощення взаємодії між об’єктами Java і базою даних SQLite. Hibernate забезпечує автоматичне зіставлення об’єктів класів (Type, Project, Option) з таблицями бази даних, спрощує операції CRUD (Create, Read, Update, Delete) і забезпечує управління транзакціями, що підвищує надійність роботи з даними. Використання Hibernate дозволяє зменшити обсяг коду для роботи з базою даних і підвищує його читабельність [8].

Для створення графічного інтерфейсу користувача використано бібліотеку Swing, яка забезпечує створення кросплатформного графічного інтерфейсу з компонентами, такими як вкладки, таблиці, кнопки та текстові поля, що відповідає вимогам до інтуїтивно зрозумілого інтерфейсу. Swing дозволяє створювати легковагові компоненти, які не залежать від операційної системи, що відповідає вимогам кросплатформності [9].

Для роботи з GitHub репозиторіями використано бібліотеку JGit, яка є чистою Java-реалізацією Git. JGit дозволяє клонувати репозиторії, отримувати список тегів, перемикатися на потрібні коміти або гілки за датою без необхідності виклику зовнішньої утиліти Git через командний рядок. Це підвищує надійність програми, зменшує залежність від зовнішнього програмного забезпечення та спрощує обробку помилок, пов'язаних із мережею або доступом до репозиторіїв.

Розробка фізичної моделі даних програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics

Фізична модель даних для СКБД SQLite (рисунок 2.8) розроблена на основі відповідної логічної моделі

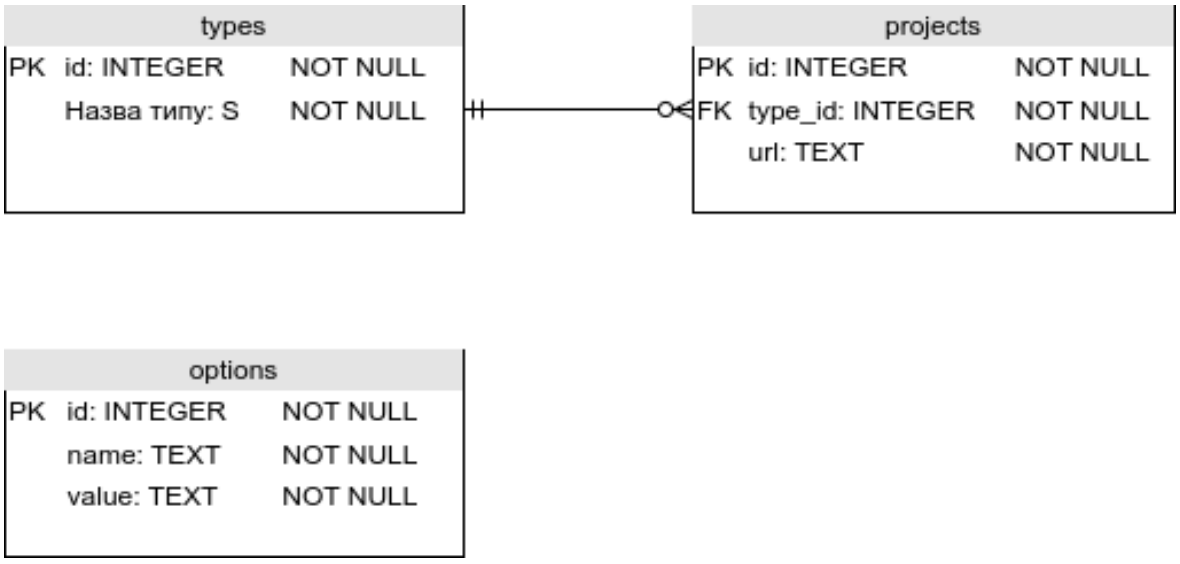


Рисунок 2.8 – Фізична модель даних програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics

Фізична модель включає три основні таблиці: types, projects і options, які відповідають об’єктам даних для типів проєктів, GitHub проєктів і налаштувань програми відповідно. SQL-скрипт, який створює таблиці та їх зв’язки реалізовано у файлі schema.sql (файл наведено в додатку Б).

Реалізація програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics

Далі наведено фрагмент програми для асинхронного завантаження та обробки GitHub проєктів.

Фрагмент коду файлу CheckoutPanel.java:

```
int progress = 0;
StringBuilder errorMessages = new StringBuilder();

for (Project project : projects) {
    String url = project.getUrl();
    String projectName = extractProjectName(url);

    // Update progress
    final int currentProgress = progress;
    SwingUtilities.invokeLater(() -> {
        progressLabel.setText("Processing project: " +
projectName);
        progressBar.setValue(currentProgress);
        progressBar.setString(currentProgress + " of " +
projects.size());
    });

    // Create subdirectory for the project
    File projectDir = new File(baseDir, projectName);

    // Delete directory if it exists
    if (projectDir.exists()) {
        SwingUtilities.invokeLater(() ->
progressLabel.setText("Deleting existing directory: " +
projectName));
        deleteDirectory(projectDir);
    }

    // Create directory
    projectDir.mkdirs();

    try {
        // Clone repository
        SwingUtilities.invokeLater(() ->
progressLabel.setText("Cloning repository: " + projectName));
        org.eclipse.jgit.api.Git git =
gitHubService.cloneRepository(url, projectDir);
```

```
            // Checkout branch or tag
            if (isMainBranch) {
                SwingUtilities.invokeLater(() ->
progressLabel.setText("Checking out main branch at date for: " +
projectName));
                gitHubService.checkoutMainBranchAtDate(git, date);
            } else {
                SwingUtilities.invokeLater(() ->
progressLabel.setText("Checking out latest tag at date for: " +
projectName));
                gitHubService.checkoutLatestTagAtDate(git, date);
            }

            // Run phpmetrics in the checked out project directory
            SwingUtilities.invokeLater(() ->
progressLabel.setText("Running phpmetrics for: " +
projectName));
            phpmetricsService.runPhpmetrics(projectDir,
summaryCsvFile, progress + 1, url);
        } catch (Exception e) {
            errorMessages.append("Error processing
").append(projectName).append(":
").append(e.getMessage()).append("\n");
            e.printStackTrace();
        }

        progress++;
    }

    // Update final progress
    SwingUtilities.invokeLater(() -> {
        progressBar.setValue(projects.size());
        progressBar.setString(projects.size() + " of " +
projects.size());
        progressLabel.setText("Checkout completed");
    });

    // Close progress dialog
    final String errors = errorMessages.toString();
    SwingUtilities.invokeLater(() -> {
        progressDialog.dispose();
        if (errors.isEmpty()) {
            JOptionPane.showMessageDialog(this, "All projects
checked out successfully. Metrics summary saved to " +
summaryCsvFile.getAbsolutePath(), "Success",
JOptionPane.INFORMATION_MESSAGE);
        } else {
            JOptionPane.showMessageDialog(this, "Checkout completed
with errors:\n" + errors, "Warning",
JOptionPane.WARNING_MESSAGE);
        }
    });
```

В цьому фрагменті коду, використовуючи оператор циклу *for*, відбувається проходження по всіх проєктах. Для кожного проєкту відбувається завантаження коду у окремий каталог, після чого запускається PhpMetrics для отримання метрик. При цьому оновлюється індикатор прогресу, а користувачу відображається статус завдання. Якщо в процесі роботи виникне якась помилка, то користувачу буде виведено відповідне повідомлення. Якщо ж всі проєкти буде оброблено без помилок, то користувачу буде виведено повідомлення про успішне завершення роботи.

Тестування програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics

Тестування було проведено методом чорної скриньки, при якому код програного забезпечення не доступний тестувальнику. Для кожної функції було визначено правильні та неправильні класи еквівалентності.

Результати тестування підтвердили, що програмне забезпечення відповідає всім функціональним вимогам, а також коректно обробляє помилки, пов’язані з мережею або некоректними вхідними даними.

Випробування програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics

Випробування програмного забезпечення виконувалось відповідно до програми та методики випробувань (додаток Д). В результаті проведення випробувань кожної функції програмного було отримано очікувані результати. Таким чином, підтверджено працездатність розробленого програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics і його відповідність вимогам технічного завдання.

3 РЕЗУЛЬТАТИ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОМАТИЗАЦІЇ ЗБОРУ МЕТРИК GITHUB ПРОЄКТІВ НА МОВІ PHP УТИЛІТОЮ PHPMETRICS

Результатом виконання кваліфікаційної роботи стало розв’язання практичної задачі інженерії програмного забезпечення, що характеризується комплексністю та невизначеністю умов, а саме розроблено програмне забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics.

Для цього було проведено аналіз цієї задачі, аналіз існуючого програмного забезпечення для автоматизації збору метрик програмних проєктів та виконано постановку задачі на розробку програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics. Після цього було розроблено програмне забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics, для чого виконано аналіз вимог до програмного забезпечення, вибрано архітектуру та створено проєкт програмного забезпечення.

Програмне забезпечення було реалізоване на мові Java, після чого воно було протестовано та випробувано згідно програми та методики випробувань.

Розмір програмного забезпечення складає 1647 рядків коду. Розмір скомпільованого JAR файлу складає 29,0 МБ.

4 ОХОРОНА ПРАЦІ

Під час робіт пов’язаних із тривалим використанням комп’ютерної техніки має приділятися особлива увага забезпеченню безпеки та створенню комфортних умов праці. Охорона праці спрямована на запобігання професійним ризикам, зниження рівня травматизму та підтримання здоров’я працівників. Забезпечення належних умов праці сприяє підвищенню продуктивності, зниженню втоми та створенню сприятливого робочого середовища [10].

Організація робочого місця

Робоче місце працівника, який виконує завдання за комп’ютером, має бути організовано відповідно до санітарних, гігієнічних і ергономічних норм, щоб мінімізувати фізичне навантаження та забезпечити комфорт. Основні вимоги до робочого місця включають:

Ергономічне обладнання: використання регульованих крісел із підтримкою попереку та спини, столів із можливістю налаштування висоти, а також моніторів, розташованих на відстані 50–70 см від очей і на їхньому рівні. Клавіатура та миша повинні розміщуватися так, щоб руки перебували в природному положенні, а лікті утворювали кут 90°.

Освітлення: забезпечення достатнього природного або штучного освітлення (300–500 люкс) для зниження зорового напруження. Світло має бути рівномірним, без відблисків на екрані монітора. Рекомендується використовувати настільні лампи з регульованою яскравістю для локального освітлення робочої зони.

Вентиляція та мікроклімат: підтримання оптимальної температури в приміщенні (18–22°C) і вологості (40–60%) для забезпечення комфортних умов. Регулярне провітрювання або використання систем кондиціонування сприяє підтриманню свіжого повітря. Уникнення протягів і різких перепадів температури також є важливим аспектом.

Шумовий фон: рівень шуму в робочому приміщенні повинен бути мінімальним, щоб не відволікати працівника та не створювати додаткового стресу. Для цього рекомендується використовувати шумопоглинаючі матеріали або гарнітури з шумозаглушенням.

Організація простору: робоче місце має бути вільним від зайвих предметів, а кабелі й дроти повинні бути акуратно закріплені, щоб уникнути випадкового травмування або пошкодження обладнання.

Безпека при роботі з комп'ютерною технікою

Тривала робота за комп'ютером може призводити до професійних ризиків, таких як зорове напруження, захворювання опорно-рухового апарату (наприклад, синдром зап'ястного каналу) або порушення постави. Для їх мінімізації необхідно дотримуватися таких заходів:

Дотримання режиму праці та відпочинку: рекомендуються короткі перерви тривалістю 5–10 хвилин кожні 45–60 хвилин роботи. Під час перерв працівники повинні виконувати вправи для очей (наприклад, фокусування погляду на далеких і близьких об'єктах) і легку фізичну розминку для розслаблення м'язів шиї, плечей і спини.

Захист зору: використання моніторів із технологією зменшення синього світла, налаштування яскравості та контрастності екрана відповідно до умов освітлення.

Електробезпека: комп'ютерне обладнання має бути справним і відповідати стандартам безпеки. Необхідно регулярно перевіряти стан кабелів, розеток і заземлення, а також уникати використання пошкоджених пристроїв. Для захисту від перепадів напруги слід застосовувати стабілізатори або джерела безперебійного живлення.

Правильна постава: працівники повинні сидіти прямо, не сутулитися, з опорою на спинку крісла. Ноги мають стояти на підлозі або на спеціальній підставці, щоб уникнути порушення кровообігу.

Профілактика травм: уникнення тривалого статичного положення рук під час роботи з клавіатурою або мишею. Рекомендується використовувати ергономічні клавіатури та миші, а також виконувати вправи для зап’ясть.

Навчання з охорони праці

Перед початком роботи за комп’ютером усі працівники повинні пройти інструктаж із охорони праці, який охоплює такі аспекти:

Ознайомлення з правилами безпечної експлуатації комп’ютерної техніки, включаючи правильне підключення обладнання та дії в разі несправностей.

Вивчення ергономічних принципів організації робочого місця, зокрема правильного розташування монітора, клавіатури та миші.

Інформування про методи профілактики професійних захворювань, таких як зорове напруження, болі в спині або синдром зап’ястного каналу.

Навчання технікам виконання вправ для очей і тіла під час перерв.

Ознайомлення з діями в надзвичайних ситуаціях, таких як коротке замикання, задимлення або евакуація з приміщення.

Крім первинного інструктажу, рекомендується проводити періодичні тренінги та перевірки знань із охорони праці, щоб забезпечити дотримання всіх норм і стандартів. Працівники також повинні мати доступ до інформаційних матеріалів, таких як пам’ятки або плакати, що нагадують про правила безпечної роботи за комп’ютером.

Дотримання всіх зазначених заходів із охорони праці сприяє створенню безпечного, комфортного та продуктивного робочого середовища для працівників, які працюють за комп’ютером.

ВИСНОВКИ

В кваліфікаційній роботі розв’язано практичну задачу інженерії програмного забезпечення, що характеризується комплексністю та невизначеністю умов: розроблено програмне забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics.

Для цього було вирішено всі поставленні завдання:

1) проведено аналіз практичної задачі інженерії програмного забезпечення з розробки програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics;

2) розглянуто існуюче програмне забезпечення для автоматизації збору метрик програмних проєктів.

3) виконано постановку задачі на розробку програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics;

4) проведено аналіз вимог до програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics;

5) вибрано архітектуру програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics;

6) розроблено проєкт програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics;

7) проведено тестування та випробування програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics.

Результати випробувань показали повну працездатність реалізованого програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Prykhodko S. Evaluating Quality of Software Systems by the Confidence and Prediction Intervals of Regressions for RFC, CBO and WMC Metrics. WSEAS Transactions on Systems. 2024. Vol. 23. P. 322–330. DOI: 10.37394/23202.2024.23.36.

2. PhpMetrics : офіційний репозиторій. URL: https://github.com/phpmetrics/PhpMetrics (дата звернення: 11.03.2025).

3. Özçevik Y., Altay O. MetricHunter: A software metric dataset generator utilizing SourceMonitor upon public GitHub repositories. SoftwareX. 2023. Vol. 23. 101499. DOI: 10.1016/j.softx.2023.101499.

4. Фрімен Е., Робсон Е., Бейтс Б., Сієрра К. Head First. Патерни проєктування : пер. з англ. Харків : Фабула, 2020. 672 с.

5. Мартін Р. Чиста архітектура. Мистецтво розробки програмного забезпечення : пер. з англ. Київ : Фабула, 2018. 416 с.

6. Васильєв О. Програмування мовою Java. Тернопіль : Навчальна книга – Богдан, 2021. 696 с.

7. Kreibich J. Using SQLite. O'Reilly Media, Inc, 2010. 503 p.

8. Hibernate. URL: https://uk.wikipedia.org/wiki/Hibernate (дата звернення: 11.05.2025).

9. Swing (Java). URL: https://uk.wikipedia.org/wiki/Swing_(Java) (дата звернення: 11.05.2025).

10. Про охорону праці: Закон України від 21.11.2002 р. № 2694-XII. Дата оновлення: 04.04.2025. https://zakon.rada.gov.ua/laws/main/2694-12.

ДОДАТОК А – ТЕХНІЧНЕ ЗАВДАННЯ НА РОЗРОБКУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОМАТИЗАЦІЇ ЗБОРУ МЕТРИК GITHUB ПРОЄКТІВ НА МОВІ PHP УТИЛІТОЮ PHPMETRICS

1 Вступ

1.1 Назва програмного забезпечення

Програмне забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics.

1.2 Сфера застосування

Програмне забезпечення призначене для використання в галузі інженерії програмного забезпечення.

2 Підстава для розробки програмного забезпечення

Розробка програмного забезпечення виконується на підставі завдання на кваліфікаційну роботу, виданого кафедрою ПЗАС НУК імені адмірала Макарова.

3 Призначення розробки

3.1 Функціональне призначення

Функціональним призначенням програмного забезпечення є автоматизація процесу збору метрик коду PHP-проєктів, розміщених на GitHub, шляхом інтеграції з утилітами Git і PhpMetrics, обробки отриманих даних та надання зручного інтерфейсу для управління проєктами і перегляду результатів збору метрик.

3.2 Експлуатаційне призначення

Програмне забезпечення 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

Програмне забезпечення повинне забезпечити виконання наступних функцій:

1) додавання типу проєкту;

2) видалення типу проєкту;

3) перегляд списку типів проєктів;

4) додавання проєкту;

5) видалення проєкту;

6) перегляд списку проєктів;

7) отримання метрик проєктів.

4.1.2 Вимоги до організації вхідних та вихідних даних

Для додавання типу проєкту: вхідними даними є назва типу проєкту. Вихідними даними є запис у базі даних про новий тип проєкту.

Для видалення типу проєкту: вхідними даними є ідентифікатор або назва типу проєкту. Вихідними даними є оновлена база даних без вказаного типу проєкту.

Для перегляду списку типів проєктів: вхідними даними є запит на отримання списку. Вихідними даними є список типів проєктів у табличному або іншому зрозумілому форматі.

Для додавання проєкту: вхідними даними є URL-адреса GitHub репозиторію та тип проєкту. Вихідними даними є запис у базі даних про новий проєкт.

Для видалення проєкту: вхідними даними є ідентифікатор або URL-адреса проєкту. Вихідними даними є оновлена база даних без вказаного проєкту.

Для перегляду списку проєктів: вхідними даними є запит на отримання списку. Вихідними даними є список проєктів із зазначенням їх URL-адрес та типів у табличному форматі.

Для отримання метрик проєктів: вхідними даними є тип проєкту, дата, станом на яку необхідно отримати код, та шлях до каталогу, в який буде виконано завантаження вихідного проєктів. Вихідними даними є звіт у табличному форматі із проєктами вибраного типу та агрегованими значеннями метрик (сума або середнє) отриманих після запуску утиліти PhpMetrics: кількість рядків коду, кількість логічних рядків коду, кількість рядків коментарів, кількість класів, кількість інтерфейсів, кількість методів, середнє вхідне зчеплення, середнє вихідне зчеплення, середня нестабільність, глибина дерева успадкування, середня цикломатична складність, середня відносна складність системи, середня складність, середня кількість помилок на клас, середня кількість дефектів на клас.

4.2 Вимоги до надійності

Програмне забезпечення повинне забезпечувати стабільну роботу при обробці великої кількості запитів, коректно обробляти помилки, пов’язані з мережевими збоями, некоректними URL-адресами репозиторіїв або аварійним завершенням роботи утиліт Git і PhpMetrics.

4.3 Умови експлуатації

4.3.1 Кліматичні умови експлуатації

Кліматичні умови експлуатації відповідають кліматичним умовам експлуатації технічних засобів.

4.3.2 Вимоги до чисельності та кваліфікації користувачів

Програмне забезпечення розраховане на одного або кількох користувачів з базовими навичками роботи з комп’ютером. Спеціальних вимог до кваліфікації немає, оскільки інтерфейс має бути інтуїтивно зрозумілим.

4.4 Вимоги до складу і параметрів технічних засобів

Процесор: Intel Core i3 або еквівалент, 2.5 ГГц, 2 ядра та більше.

Оперативна пам’ять: 4 ГБ.

Жорсткий диск: мінімум 500 МБ вільного простору.

Операційна система: Linux або Windows.

Інтернет: стабільне з’єднання зі швидкістю 1 Мбіт/с.

4.5 Вимоги до інформаційної та програмної сумісності

Операційна система: Linux (Ubuntu 20.04 або новіша), Windows 7 або новіша.

JRE.

Утиліта Git: версія 2.30 або новіша.

PhpMetrics: версія 2.8 або новіша.

Мова програмування: PHP версія 8.1 або новіша.

Система керування базами даних: SQLite (версія 3.35 або новіша).

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 повинен включати в себе:

- технічне завдання;
- текст програми;
- керівництво користувача;
- опис програми;
- програму і методику випробувань.

6 Техніко-економічні показники

У кваліфікаційній роботі техніко-економічні показники не розраховуються.

7 Стадії та етапи розробки

Етапи розробки програмного забезпечення наведено в таблиці А.1.

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

Номер	Назва етапів роботи	Терміни виконання
1.	Аналіз практичної задачі інженерії ПЗ	07.04.2025
2.	Аналіз вимог до ПЗ	14.04.2025
3.	Розробка архітектури ПЗ	21.04.2025
4.	Розробка проєкту ПЗ	05.05.2025
5.	Реалізація та тестування ПЗ	12.05.2025

Зазначені етапи розробки програмного забезпечення відповідають календарному плану кваліфікаційної роботи.

8 Порядок контролю і прийомки

Контроль здійснюється керівником кваліфікаційної роботи на кожному етапі розробки програмного забезпечення. Приймання здійснюється за програмою та методикою випробувань. Випробування проводиться в зазначений термін.

ДОДАТОК Б – ТЕКСТ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОМАТИЗАЦІЇ ЗБОРУ МЕТРИК GITHUB ПРОЄКТІВ НА МОВІ PHP УТИЛІТОЮ PHPMETRICS

Текст файлу Main.java

```
package com.github.manager;

import com.github.manager.ui.MainFrame;
import com.github.manager.util.HibernateUtil;

import javax.swing.*;

/**
 * Main class for the GitHub Project Manager application.
 */
public class Main {

    /**
     * Main method to start the application.
     *
     * @param args Command line arguments
     */
    public static void main(String[] args) {
        // Initialize Hibernate
        HibernateUtil.getSessionFactory();

        // Set look and feel to system look and feel
        try {

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

        // Create and show the main frame
        SwingUtilities.invokeLater(() -> {
            MainFrame frame = new MainFrame();
            frame.setVisible(true);
        });
    }
}
```

Текст файлу PhpmetricsService.java

```
package com.github.manager.service;

import java.io.*;
```

```
import java.util.HashMap;
import java.util.Map;

import com.google.gson.Gson;
import com.google.gson.JsonElement;
import com.google.gson.JsonObject;

public class PhpmetricsService {

    private static final String PHPMETRICS_SCRIPT =
"/usr/local/bin/phpmetrics";

    /**
     * Run phpmetrics in the checked out project directory and process
metrics to update the summary CSV.
     *
     * @param directory       The directory of the checked out project
     * @param summaryCsvFile  The CSV file to store aggregated metrics
     * @param projectNumber   The project number for the CSV
     * @param projectUrl      The URL of the project
     * @throws IOException        If an I/O error occurs
     * @throws InterruptedException If the process is interrupted
     */
    public void runPhpmetrics(File directory, File summaryCsvFile, int
projectNumber, String projectUrl) throws IOException,
InterruptedException {
        // Run phpmetrics
        final File jsonReport = new File(directory, "phpmetrics-
report.json");
        ProcessBuilder processBuilder = new ProcessBuilder(
                PHPMETRICS_SCRIPT,
                ".",
                "--report-json=" + jsonReport.getAbsolutePath(),
                "--report-csv=" + new File(directory, "phpmetrics-
report.csv").getAbsolutePath()
        );
        processBuilder.directory(directory);
        Process process = processBuilder.start();

        new Thread(() -> {
            try (BufferedReader reader = new BufferedReader(new
InputStreamReader(process.getInputStream()))) {
                String line;
                while ((line = reader.readLine()) != null) {
                    System.out.println(line);
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }).start();

        // Passthrough stderr
        new Thread(() -> {
```

```
            try (BufferedReader reader = new BufferedReader(new
InputStreamReader(process.getErrorStream()))) {
                String line;
                while ((line = reader.readLine()) != null) {
                    System.err.println(line);
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }).start();

        int exitCode = process.waitFor();
        if (exitCode != 0) {
            throw new RuntimeException("phpmetrics command failed with
code " + exitCode);
        }

        // Process the JSON report and update the summary CSV
        processMetrics(jsonReport, summaryCsvFile, projectNumber,
directory.getName(), projectUrl);
    }

    /**
     * Process the phpmetrics-report.json file and append aggregated
metrics to the summary CSV.
     *
     * @param jsonFile        The JSON report file
     * @param summaryCsvFile  The CSV file to append metrics
     * @param projectNumber   The project number
     * @param projectName     The project name
     * @param projectUrl      The project URL
     */
    private void processMetrics(File jsonFile, File summaryCsvFile,
int projectNumber, String projectName, String projectUrl) throws
IOException {
        Gson gson = new Gson();
        JsonObject jsonObject;
        try (FileReader reader = new FileReader(jsonFile)) {
            jsonObject = gson.fromJson(reader, JsonObject.class);
        }

        // Initialize metrics aggregation
        Map<String, Double> metricsSum = new HashMap<>();
        String[] sumMetrics = {
            "length", "cloc", "loc", "lloc",
            "nbMethods", "nbMethodsIncludingGettersSetters",
"nbMethodsPrivate", "nbMethodsPublic",
            "nbMethodsGetter", "nbMethodsSetters"
        };
        String[] avgMetrics = {
            "afferentCoupling", "efferentCoupling", "couplingSum"
        };
        for (String key : sumMetrics) {
```

```
            metricsSum.put(key, 0.0);
        }
        for (String key : avgMetrics) {
            metricsSum.put(key, 0.0);
        }
        int numberOfClasses = 0;

        // Extract project-level DIT
        double dit = 0.0;
        if (jsonObject.has("tree") &&
jsonObject.getAsJsonObject("tree").has("depthOfInheritanceTree")) {
            try {
                dit =
jsonObject.getAsJsonObject("tree").get("depthOfInheritanceTree").getAs
Double();
            } catch (UnsupportedOperationException e) {
                // Skip non-numeric DIT
            }
        }

        for (Map.Entry<String, JsonElement> entry :
jsonObject.entrySet()) {
            JsonObject classData = entry.getValue().getAsJsonObject();
            if (classData.has("_type") &&

(classData.get("_type").getAsString().equals("Hal\\Metric\\ClassMetric
"))) {
                numberOfClasses++;
                for (String key : sumMetrics) {
                    if (classData.has(key)) {
                        try {
                            double value =
classData.get(key).getAsDouble();
                            metricsSum.put(key, metricsSum.get(key) +
value);
                        } catch (UnsupportedOperationException e) {
                            // Skip non-numeric values
                        }
                    }
                }
                for (String key : avgMetrics) {
                    try {
                        double value = 0;
                        if (key.equals("couplingSum")) {
                            double afferent =
classData.has("afferentCoupling") ?
classData.get("afferentCoupling").getAsDouble() : 0.0;
                            double efferent =
classData.has("efferentCoupling") ?
classData.get("efferentCoupling").getAsDouble() : 0.0;
                            value = afferent + efferent;
                            metricsSum.put(key, metricsSum.get(key) +
value);
```

```
                        } else if (classData.has(key)) {
                            value = classData.get(key).getAsDouble();
                        }

                        metricsSum.put(key, metricsSum.get(key) +
value);
                    } catch (UnsupportedOperationException e) {
                        // Skip non-numeric values
                    }
                }
            }
        }

        // Prepare CSV row
        StringBuilder csvRow = new StringBuilder();
        csvRow.append(projectNumber).append(",")
              .append(projectName).append(",")
              .append(projectUrl).append(",")
              .append(numberOfClasses);
        for (String key : sumMetrics) {
            double value = metricsSum.get(key);
            csvRow.append(",").append(value);
        }
        for (String key : avgMetrics) {
            csvRow.append(",").append(numberOfClasses > 0 ?
metricsSum.get(key) / numberOfClasses : "");
        }
        csvRow.append(",").append(numberOfClasses > 0 ?
metricsSum.get("nbMethods") / numberOfClasses : "");
        csvRow.append(",").append(dit);
        csvRow.append("\n");

        // Write to CSV file
        boolean fileExists = summaryCsvFile.exists();
        try (FileWriter writer = new FileWriter(summaryCsvFile, true))
{
            if (!fileExists) {
                // Write header if file is new
                StringBuilder header = new
StringBuilder("Number,Name,URL,numberOfClasses");
                for (String key : sumMetrics) {
                    header.append(",").append(key);
                }
                for (String key : avgMetrics) {
                    header.append(",").append(key);
                }
                header.append(",nbMethods,dit\n");
                writer.write(header.toString());
            }
            writer.write(csvRow.toString());
        }
    }
}
```

Текст файлу GitHubService.java

```
package com.github.manager.service;

import org.eclipse.jgit.api.Git;
import org.eclipse.jgit.api.errors.GitAPIException;
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.eclipse.jgit.transport.TagOpt;

import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import java.util.stream.Collectors;

/**
 * Service class for GitHub operations.
 */
public class GitHubService {

    /**
     * Clone a GitHub repository.
     *
     * @param url The URL of the repository to clone
     * @param directory The directory to clone the repository to
     * @return The Git instance for the cloned repository
     * @throws GitAPIException If an error occurs during cloning
     */
    public Git cloneRepository(String url, File directory) throws
GitAPIException {
        return Git.cloneRepository()
                .setURI(url)
                .setDirectory(directory)
                .setCloneAllBranches(false)
                .setDepth(1)
                .setNoTags()
                .call();
    }

    /**
     * Checkout the main branch of a repository at a specific date.
     *
     * @param git The Git instance for the repository
     * @param date The date to checkout
     * @throws GitAPIException If an error occurs during checkout
     * @throws IOException If an I/O error occurs
     */
```

```
    public void checkoutMainBranchAtDate(Git git, Date date) throws
GitAPIException, IOException {
        Repository repository = git.getRepository();

        // Get the main branch (usually 'master' or 'main')
        String mainBranch = getMainBranchName(git);

        // Find the commit closest to the specified date
        RevCommit commit = findCommitAtDate(git, mainBranch, date);

        // Checkout the commit
        git.checkout()
                .setName(commit.getName())
                .call();
    }

    /**
     * Checkout the latest tag at a specific date.
     *
     * @param git The Git instance for the repository
     * @param date The date to checkout
     * @throws GitAPIException If an error occurs during checkout
     * @throws IOException If an I/O error occurs
     */
    public void checkoutLatestTagAtDate(Git git, Date date) throws
GitAPIException, IOException {
        // Get all tags
        List<Ref> tags = git.tagList().call();

        // Filter tags by date
        List<Ref> tagsBeforeDate = new ArrayList<>();
        for (Ref tag : tags) {
            RevWalk walk = new RevWalk(git.getRepository());
            RevCommit commit = walk.parseCommit(tag.getObjectId());
            if (commit.getCommitTime() * 1000L <= date.getTime()) {
                tagsBeforeDate.add(tag);
            }
            walk.dispose();
        }

        if (tagsBeforeDate.isEmpty()) {
            // If no tags are found before the specified date, but
there are tags in the repository,
            // use the earliest tag (the one closest to the specified
date)
            if (!tags.isEmpty()) {
                // Sort tags by date (oldest first)
                tags.sort((t1, t2) -> {
                    try {
                        RevWalk walk = new
RevWalk(git.getRepository());
                        RevCommit c1 =
walk.parseCommit(t1.getObjectId());
```

```
                        RevCommit c2 =
walk.parseCommit(t2.getObjectId());
                        walk.dispose();
                        return Long.compare(c1.getCommitTime(),
c2.getCommitTime());
                    } catch (IOException e) {
                        return 0;
                    }
                });

                // Use the earliest tag
                tagsBeforeDate.add(tags.get(0));
            } else {
                throw new IllegalArgumentException("No tags found
before the specified date");
            }
        }

        // Sort tags by date (newest first)
        tagsBeforeDate.sort((t1, t2) -> {
            try {
                RevWalk walk = new RevWalk(git.getRepository());
                RevCommit c1 = walk.parseCommit(t1.getObjectId());
                RevCommit c2 = walk.parseCommit(t2.getObjectId());
                walk.dispose();
                return Long.compare(c2.getCommitTime(),
c1.getCommitTime());
            } catch (IOException e) {
                return 0;
            }
        });

        // Checkout the latest tag
        String tagName = tagsBeforeDate.get(0).getName();
        git.checkout()
                .setName(tagName)
                .call();
    }

    /**
     * Get the name of the main branch (usually 'master' or 'main').
     *
     * @param git The Git instance for the repository
     * @return The name of the main branch
     * @throws GitAPIException If an error occurs during the operation
     */
    private String getMainBranchName(Git git) throws GitAPIException {
        List<Ref> branches = git.branchList().call();

        // Check for 'main' or 'master' branch
        for (Ref branch : branches) {
            String name = branch.getName();
            if (name.endsWith("/main") || name.endsWith("/master")) {
```

```
                return name.substring(name.lastIndexOf('/') + 1);
            }
        }

        // If no main or master branch is found, return the first
branch
        if (!branches.isEmpty()) {
            String name = branches.get(0).getName();
            return name.substring(name.lastIndexOf('/') + 1);
        }

        // Default to 'master' if no branches are found
        return "master";
    }

    /**
     * Find the commit closest to the specified date.
     *
     * @param git The Git instance for the repository
     * @param branch The branch to search in
     * @param date The date to find a commit for
     * @return The commit closest to the specified date
     * @throws GitAPIException If an error occurs during the operation
     * @throws IOException If an I/O error occurs
     */
    private RevCommit findCommitAtDate(Git git, String branch, Date
date) throws GitAPIException, IOException {
        // Get all commits
        Iterable<RevCommit> commits =
git.log().add(git.getRepository().resolve(branch)).call();

        // Find the commit closest to the specified date
        RevCommit closestCommit = null;
        long closestDiff = Long.MAX_VALUE;

        for (RevCommit commit : commits) {
            long commitTime = commit.getCommitTime() * 1000L;
            long diff = Math.abs(date.getTime() - commitTime);

            if (diff < closestDiff) {
                closestDiff = diff;
                closestCommit = commit;
            }

            // If we found a commit before the specified date, we can
stop
            if (commitTime <= date.getTime()) {
                break;
            }
        }

        if (closestCommit == null) {
```

```
            throw new IllegalArgumentException("No commits found in
the repository");
        }

        return closestCommit;
    }

    /**
     * Get all tags in a repository.
     *
     * @param git The Git instance for the repository
     * @return A list of tag names
     * @throws GitAPIException If an error occurs during the operation
     */
    public List<String> getAllTags(Git git) throws GitAPIException {
        List<Ref> tags = git.tagList().call();
        return tags.stream()
                .map(ref ->
ref.getName().substring(ref.getName().lastIndexOf('/') + 1))
                .collect(Collectors.toList());
    }
}
```

Текст файлу model/Option.java

```
package com.github.manager.model;

import javax.persistence.*;

/**
 * Entity class representing an application option.
 */
@Entity
@Table(name = "options")
public class Option {

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

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

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

    // Default constructor required by Hibernate
    public Option() {
    }

    public Option(String name, String value) {
        this.name = name;
```

```
        this.value = value;
    }

    // Getters and setters
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getValue() {
        return value;
    }

    public void setValue(String value) {
        this.value = value;
    }

    @Override
    public String toString() {
        return name + ": " + value;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        Option option = (Option) o;

        if (id != null ? !id.equals(option.id) : option.id != null)
return false;
        return name != null ? name.equals(option.name) : option.name
== null;
    }

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

Текст файлу model/Project.java

```
package com.github.manager.model;

import javax.persistence.*;

/**
 * Entity class representing a GitHub project.
 */
@Entity
@Table(name = "projects")
public class Project {

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

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

    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "type_id")
    private Type type;

    // Default constructor required by Hibernate
    public Project() {
    }

    public Project(String url, Type type) {
        this.url = url;
        this.type = type;
    }

    // Getters and setters
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getUrl() {
        return url;
    }

    public void setUrl(String url) {
        this.url = url;
    }

    public Type getType() {
```

```
        return type;
    }

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

    @Override
    public String toString() {
        return url;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        Project project = (Project) o;

        if (id != null ? !id.equals(project.id) : project.id != null)
return false;
        return url != null ? url.equals(project.url) : project.url ==
null;
    }

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

Текст файлу model/Type.java

```
package com.github.manager.model;

import javax.persistence.*;

/**
 * Entity class representing a project type.
 */
@Entity
@Table(name = "types")
public class Type {

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

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

```
    // Default constructor required by Hibernate
    public Type() {
    }

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

    // Getters and setters
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    @Override
    public String toString() {
        return name;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        Type type = (Type) o;

        if (id != null ? !id.equals(type.id) : type.id != null) return
false;
        return name != null ? name.equals(type.name) : type.name ==
null;
    }

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

Текст файлу ui/TypesPanel.java

```
package com.github.manager.ui;

import com.github.manager.dao.TypeDAO;
import com.github.manager.model.Type;

import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.util.List;

/**
 * Panel for managing project types.
 */
public class TypesPanel extends JPanel {

    private JTable typesTable;
    private DefaultTableModel tableModel;
    private JTextField nameField;
    private TypeDAO typeDAO;

    /**
     * Create a new types panel.
     */
    public TypesPanel() {
        typeDAO = new TypeDAO();
        setLayout(new BorderLayout());

        initComponents();
        loadTypes();
    }

    /**
     * Initialize the components of the panel.
     */
    private void initComponents() {
        // Create table
        String[] columnNames = {"ID", "Name"};
        tableModel = new DefaultTableModel(columnNames, 0) {
            @Override
            public boolean isCellEditable(int row, int column) {
                return false;
            }
        };
        typesTable = new JTable(tableModel);

typesTable.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);

        // Add table to scroll pane
        JScrollPane scrollPane = new JScrollPane(typesTable);
        add(scrollPane, BorderLayout.CENTER);
```

```
        // Create form panel
        JPanel formPanel = new JPanel(new BorderLayout());

        // Create input panel
        JPanel inputPanel = new JPanel(new
FlowLayout(FlowLayout.LEFT));
        inputPanel.add(new JLabel("Name:"));
        nameField = new JTextField(20);
        inputPanel.add(nameField);

        // Create button panel
        JPanel buttonPanel = new JPanel(new
FlowLayout(FlowLayout.LEFT));
        JButton addButton = new JButton("Add");
        JButton editButton = new JButton("Edit");
        JButton deleteButton = new JButton("Delete");

        buttonPanel.add(addButton);
        buttonPanel.add(editButton);
        buttonPanel.add(deleteButton);

        // Add action listeners
        addButton.addActionListener(e -> addType());
        editButton.addActionListener(e -> editType());
        deleteButton.addActionListener(e -> deleteType());

        // Add panels to form panel
        formPanel.add(inputPanel, BorderLayout.NORTH);
        formPanel.add(buttonPanel, BorderLayout.SOUTH);

        // Add form panel to main panel
        add(formPanel, BorderLayout.SOUTH);
    }

    /**
     * Load types from the database.
     */
    public void loadTypes() {
        // Clear table
        tableModel.setRowCount(0);

        // Get all types
        List<Type> types = typeDAO.getAll();

        // Add types to table
        if (types != null) {
            for (Type type : types) {
                Object[] row = {type.getId(), type.getName()};
                tableModel.addRow(row);
            }
        }
    }
```

```
    /**
     * Add a new type.
     */
    private void addType() {
        String name = nameField.getText().trim();

        if (name.isEmpty()) {
            JOptionPane.showMessageDialog(this, "Please enter a name",
"Error", JOptionPane.ERROR_MESSAGE);
            return;
        }

        // Check if type already exists
        Type existingType = typeDAO.findByName(name);
        if (existingType != null) {
            JOptionPane.showMessageDialog(this, "Type already exists",
"Error", JOptionPane.ERROR_MESSAGE);
            return;
        }

        // Create new type
        Type type = new Type(name);

        // Save type
        Type savedType = typeDAO.saveOrUpdate(type);

        if (savedType != null) {
            // Clear input field
            nameField.setText("");

            // Reload types
            loadTypes();
        } else {
            JOptionPane.showMessageDialog(this, "Failed to save type",
"Error", JOptionPane.ERROR_MESSAGE);
        }
    }

    /**
     * Edit the selected type.
     */
    private void editType() {
        int selectedRow = typesTable.getSelectedRow();

        if (selectedRow == -1) {
            JOptionPane.showMessageDialog(this, "Please select a
type", "Error", JOptionPane.ERROR_MESSAGE);
            return;
        }

        Long id = (Long) tableModel.getValueAt(selectedRow, 0);
        String name = nameField.getText().trim();
```

```
        if (name.isEmpty()) {
            JOptionPane.showMessageDialog(this, "Please enter a name",
"Error", JOptionPane.ERROR_MESSAGE);
            return;
        }

        // Get type
        Type type = typeDAO.getById(id);

        if (type == null) {
            JOptionPane.showMessageDialog(this, "Type not found",
"Error", JOptionPane.ERROR_MESSAGE);
            return;
        }

        // Check if name already exists
        Type existingType = typeDAO.findByName(name);
        if (existingType != null && !existingType.getId().equals(id))
{
            JOptionPane.showMessageDialog(this, "Type already exists",
"Error", JOptionPane.ERROR_MESSAGE);
            return;
        }

        // Update type
        type.setName(name);

        // Save type
        Type savedType = typeDAO.saveOrUpdate(type);

        if (savedType != null) {
            // Clear input field
            nameField.setText("");

            // Reload types
            loadTypes();
        } else {
            JOptionPane.showMessageDialog(this, "Failed to save type",
"Error", JOptionPane.ERROR_MESSAGE);
        }
    }

    /**
     * Delete the selected type.
     */
    private void deleteType() {
        int selectedRow = typesTable.getSelectedRow();

        if (selectedRow == -1) {
            JOptionPane.showMessageDialog(this, "Please select a
type", "Error", JOptionPane.ERROR_MESSAGE);
            return;
        }
```

```
        Long id = (Long) tableModel.getValueAt(selectedRow, 0);

        // Get type
        Type type = typeDAO.getById(id);

        if (type == null) {
            JOptionPane.showMessageDialog(this, "Type not found",
"Error", JOptionPane.ERROR_MESSAGE);
            return;
        }

        // Confirm deletion
        int result = JOptionPane.showConfirmDialog(this, "Are you sure
you want to delete this type?", "Confirm", JOptionPane.YES_NO_OPTION);

        if (result == JOptionPane.YES_OPTION) {
            // Delete type
            boolean deleted = typeDAO.delete(type);

            if (deleted) {
                // Reload types
                loadTypes();
            } else {
                JOptionPane.showMessageDialog(this, "Failed to delete
type", "Error", JOptionPane.ERROR_MESSAGE);
            }
        }
    }

    /**
     * Refresh the types data.
     * This method is called when the panel is displayed.
     */
    public void refresh() {
        loadTypes();
    }
}
```

Текст файлу ui/CheckoutPanel.java

```
package com.github.manager.ui;

import com.github.manager.dao.OptionDAO;
import com.github.manager.dao.ProjectDAO;
import com.github.manager.dao.TypeDAO;
import com.github.manager.model.Project;
import com.github.manager.model.Type;
import com.github.manager.service.GitHubService;
import com.github.manager.service.PhpmetricsService;

import javax.swing.*;
```

```
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.io.File;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.List;

/**
 * Panel for checking out GitHub projects.
 */
public class CheckoutPanel extends JPanel {

    private JTable projectsTable;
    private DefaultTableModel tableModel;
    private JComboBox<Type> typeComboBox;
    private JTextField directoryField;
    private JButton browseButton;
    private JRadioButton mainBranchRadio;
    private JRadioButton tagRadio;
    private JSpinner dateSpinner;
    private ProjectDAO projectDAO;
    private TypeDAO typeDAO;
    private OptionDAO optionDAO;
    private GitHubService gitHubService;
    private PhpmetricsService phpmetricsService;

    // Option names
    private static final String OPTION_LAST_DIRECTORY =
"checkout.last_directory";
    private static final String OPTION_USE_MAIN_BRANCH =
"checkout.use_main_branch";
    private static final String OPTION_CHECKOUT_DATE =
"checkout.date";
    private static final String OPTION_SELECTED_TYPE =
"checkout.selected_type";

    /**
     * Create a new checkout panel.
     */
    public CheckoutPanel() {
        projectDAO = new ProjectDAO();
        typeDAO = new TypeDAO();
        optionDAO = new OptionDAO();
        gitHubService = new GitHubService();
        phpmetricsService = new PhpmetricsService();
        setLayout(new BorderLayout());

        initComponents();
        loadTypes();
        loadOptions();
    }

    /**
```

```
     * Initialize the components of the panel.
     */
    private void initComponents() {
        // Create table
        String[] columnNames = {"ID", "URL", "Type"};
        tableModel = new DefaultTableModel(columnNames, 0) {
            @Override
            public boolean isCellEditable(int row, int column) {
                return false;
            }
        };
        projectsTable = new JTable(tableModel);

projectsTable.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);

        // Add table to scroll pane
        JScrollPane scrollPane = new JScrollPane(projectsTable);
        add(scrollPane, BorderLayout.CENTER);

        // Create form panel
        JPanel formPanel = new JPanel(new BorderLayout());

        // Create filter panel
        JPanel filterPanel = new JPanel(new
FlowLayout(FlowLayout.LEFT));
        filterPanel.add(new JLabel("Type:"));
        typeComboBox = new JComboBox<>();
        typeComboBox.addActionListener(e -> {
            loadProjectsByType();
            saveOptions();
        });
        filterPanel.add(typeComboBox);

        // Create checkout options panel
        JPanel checkoutPanel = new JPanel(new GridLayout(3, 2, 5, 5));

        // Directory selection
        checkoutPanel.add(new JLabel("Checkout Directory:"));
        JPanel directoryPanel = new JPanel(new BorderLayout());
        directoryField = new JTextField();
        directoryPanel.add(directoryField, BorderLayout.CENTER);
        browseButton = new JButton("Browse");
        browseButton.addActionListener(e -> browseDirectory());
        directoryPanel.add(browseButton, BorderLayout.EAST);
        checkoutPanel.add(directoryPanel);

        // Branch/Tag selection
        checkoutPanel.add(new JLabel("Checkout:"));
        JPanel radioPanel = new JPanel(new
FlowLayout(FlowLayout.LEFT));
        mainBranchRadio = new JRadioButton("Main Branch", true);
        tagRadio = new JRadioButton("Latest Tag");
        ButtonGroup radioGroup = new ButtonGroup();
```

```
        radioGroup.add(mainBranchRadio);
        radioGroup.add(tagRadio);
        radioPanel.add(mainBranchRadio);
        radioPanel.add(tagRadio);
        checkoutPanel.add(radioPanel);

        // Date selection
        checkoutPanel.add(new JLabel("Date:"));
        SpinnerDateModel dateModel = new SpinnerDateModel(new Date(),
null, null, java.util.Calendar.DAY_OF_MONTH);
        dateSpinner = new JSpinner(dateModel);
        JSpinner.DateEditor dateEditor = new
JSpinner.DateEditor(dateSpinner, "yyyy-MM-dd");
        dateSpinner.setEditor(dateEditor);
        checkoutPanel.add(dateSpinner);

        // Create button panel
        JPanel buttonPanel = new JPanel(new
FlowLayout(FlowLayout.LEFT));
        JButton checkoutButton = new JButton("Checkout");

        buttonPanel.add(checkoutButton);

        // Add action listeners
        checkoutButton.addActionListener(e -> checkoutProject());

        // Add panels to form panel
        formPanel.add(filterPanel, BorderLayout.NORTH);
        formPanel.add(checkoutPanel, BorderLayout.CENTER);
        formPanel.add(buttonPanel, BorderLayout.SOUTH);

        // Add form panel to main panel
        add(formPanel, BorderLayout.SOUTH);
    }

    /**
     * Load types from the database.
     */
    public void loadTypes() {
        // Clear combo box
        typeComboBox.removeAllItems();

        // Get all types
        List<Type> types = typeDAO.getAll();

        // Add types to combo box
        if (types != null) {
            for (Type type : types) {
                typeComboBox.addItem(type);
            }
        }
    }
```

```
    /**
     * Load projects by the selected type.
     */
    public void loadProjectsByType() {
        // Clear table
        tableModel.setRowCount(0);

        // Get selected type
        Type type = (Type) typeComboBox.getSelectedItem();

        if (type != null) {
            // Get projects by type
            List<Project> projects = projectDAO.getByType(type);

            // Add projects to table
            if (projects != null) {
                for (Project project : projects) {
                    Object[] row = {
                        project.getId(),
                        project.getUrl(),
                        project.getType() != null ?
project.getType().getName() : ""
                    };
                    tableModel.addRow(row);
                }
            }
        }
    }

    /**
     * Browse for a directory.
     */
    private void browseDirectory() {
        JFileChooser fileChooser = new JFileChooser();

fileChooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
        fileChooser.setDialogTitle("Select Checkout Directory");

        int result = fileChooser.showOpenDialog(this);

        if (result == JFileChooser.APPROVE_OPTION) {
            File selectedFile = fileChooser.getSelectedFile();
            directoryField.setText(selectedFile.getAbsolutePath());

            // Save the selected directory
            saveOptions();
        }
    }

    /**
     * Checkout all projects of the selected type.
     */
    private void checkoutProject() {
```

```
        Type selectedType = (Type) typeComboBox.getSelectedItem();

        if (selectedType == null) {
            JOptionPane.showMessageDialog(this, "Please select a
project type", "Error", JOptionPane.ERROR_MESSAGE);
            return;
        }

        String directory = directoryField.getText().trim();
        Date date = (Date) dateSpinner.getValue();
        boolean isMainBranch = mainBranchRadio.isSelected();

        if (directory.isEmpty()) {
            JOptionPane.showMessageDialog(this, "Please select a
directory", "Error", JOptionPane.ERROR_MESSAGE);
            return;
        }

        // Save the checkout settings
        saveOptions();

        // Create directory if it doesn't exist
        File baseDir = new File(directory);
        if (!baseDir.exists()) {
            baseDir.mkdirs();
        }

        // Get all projects of the selected type
        List<Project> projects = projectDAO.getByType(selectedType);

        if (projects == null || projects.isEmpty()) {
            JOptionPane.showMessageDialog(this, "No projects found for
the selected type", "Error", JOptionPane.ERROR_MESSAGE);
            return;
        }

        // Show progress dialog
        JDialog progressDialog = new JDialog((Frame)
SwingUtilities.getWindowAncestor(this), "Checkout Progress", true);
        progressDialog.setLayout(new BorderLayout());
        progressDialog.setSize(400, 150);
        progressDialog.setLocationRelativeTo(this);

        JLabel progressLabel = new JLabel("Preparing to checkout
projects...");
        progressLabel.setHorizontalAlignment(JLabel.CENTER);
        JProgressBar progressBar = new JProgressBar(0,
projects.size());
        progressBar.setStringPainted(true);

        JPanel progressPanel = new JPanel(new BorderLayout(5, 5));
        progressPanel.setBorder(BorderFactory.createEmptyBorder(10,
10, 10, 10));
```

```
        progressPanel.add(progressLabel, BorderLayout.NORTH);
        progressPanel.add(progressBar, BorderLayout.CENTER);

        progressDialog.add(progressPanel, BorderLayout.CENTER);

        // Start checkout in a separate thread
        new Thread(() -> {
            try {
                File summaryCsvFile = new File(baseDir, "project-
metrics-summary.csv");
                if (summaryCsvFile.exists()) {
                    progressLabel.setText("Deleting previous report "
+ progressLabel.getName());
                    summaryCsvFile.delete();
                }

                int progress = 0;
                StringBuilder errorMessages = new StringBuilder();

                for (Project project : projects) {
                    String url = project.getUrl();
                    String projectName = extractProjectName(url);

                    // Update progress
                    final int currentProgress = progress;
                    SwingUtilities.invokeLater(() -> {
                        progressLabel.setText("Processing project: " +
projectName);
                        progressBar.setValue(currentProgress);
                        progressBar.setString(currentProgress + " of "
+ projects.size());
                    });

                    // Create subdirectory for the project
                    File projectDir = new File(baseDir, projectName);

                    // Delete directory if it exists
                    if (projectDir.exists()) {
                        SwingUtilities.invokeLater(() ->
progressLabel.setText("Deleting existing directory: " + projectName));
                        deleteDirectory(projectDir);
                    }

                    // Create directory
                    projectDir.mkdirs();

                    try {
                        // Clone repository
                        SwingUtilities.invokeLater(() ->
progressLabel.setText("Cloning repository: " + projectName));
                        org.eclipse.jgit.api.Git git =
gitHubService.cloneRepository(url, projectDir);
```

```
                        // Checkout branch or tag
                        if (isMainBranch) {
                            SwingUtilities.invokeLater(() ->
progressLabel.setText("Checking out main branch at date for: " +
projectName));

gitHubService.checkoutMainBranchAtDate(git, date);
                        } else {
                            SwingUtilities.invokeLater(() ->
progressLabel.setText("Checking out latest tag at date for: " +
projectName));
                            gitHubService.checkoutLatestTagAtDate(git,
date);
                        }

                        // Run phpmetrics in the checked out project
directory and process metrics
                        SwingUtilities.invokeLater(() ->
progressLabel.setText("Running phpmetrics for: " + projectName));
                        phpmetricsService.runPhpmetrics(projectDir,
summaryCsvFile, progress + 1, url);
                    } catch (Exception e) {
                        errorMessages.append("Error processing
").append(projectName).append(":
").append(e.getMessage()).append("\n");
                        e.printStackTrace();
                    }

                    progress++;
                }

                // Update final progress
                SwingUtilities.invokeLater(() -> {
                    progressBar.setValue(projects.size());
                    progressBar.setString(projects.size() + " of " +
projects.size());
                    progressLabel.setText("Checkout completed");
                });

                // Close progress dialog
                final String errors = errorMessages.toString();
                SwingUtilities.invokeLater(() -> {
                    progressDialog.dispose();
                    if (errors.isEmpty()) {
                        JOptionPane.showMessageDialog(this, "All
projects checked out successfully. Metrics summary saved to " +
summaryCsvFile.getAbsolutePath(), "Success",
JOptionPane.INFORMATION_MESSAGE);
                    } else {
                        JOptionPane.showMessageDialog(this, "Checkout
completed with errors:\n" + errors, "Warning",
JOptionPane.WARNING_MESSAGE);
                    }
```

```
                });
            } catch (Exception e) {
                // Close progress dialog and show error
                SwingUtilities.invokeLater(() -> {
                    progressDialog.dispose();
                    JOptionPane.showMessageDialog(this, "Checkout
failed: " + e.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
                });
                e.printStackTrace();
            }
        }).start();

        // Show progress dialog
        progressDialog.setVisible(true);
    }

    private String extractProjectName(String url) {
        // Remove .git extension if present
        String cleanUrl = url.endsWith(".git") ? url.substring(0,
url.length() - 4) : url;

        // Extract the repository name from the URL
        int lastSlashIndex = cleanUrl.lastIndexOf('/');
        if (lastSlashIndex != -1 && lastSlashIndex < cleanUrl.length()
- 1) {
            return cleanUrl.substring(lastSlashIndex + 1);
        }

        // Fallback: use the full URL as a hash
        return String.valueOf(cleanUrl.hashCode()).replace("-", "");
    }

    /**
     * Recursively delete a directory.
     *
     * @param directory The directory to delete
     * @return true if successful, false otherwise
     */
    private boolean deleteDirectory(File directory) {
        if (directory.exists()) {
            File[] files = directory.listFiles();
            if (files != null) {
                for (File file : files) {
                    if (file.isDirectory()) {
                        deleteDirectory(file);
                    } else {
                        file.delete();
                    }
                }
            }
        }

        return directory.delete();
```

```
    }

    /**
     * Load options from the database.
     */
    private void loadOptions() {
        try {
            // Load last directory
            String lastDirectory =
optionDAO.getOptionValue(OPTION_LAST_DIRECTORY);
            if (lastDirectory != null && !lastDirectory.isEmpty()) {
                directoryField.setText(lastDirectory);
            }

            // Load branch/tag selection
            String useMainBranch =
optionDAO.getOptionValue(OPTION_USE_MAIN_BRANCH);
            if (useMainBranch != null) {
                boolean isMainBranch =
Boolean.parseBoolean(useMainBranch);
                mainBranchRadio.setSelected(isMainBranch);
                tagRadio.setSelected(!isMainBranch);
            }

            // Load date
            String dateStr =
optionDAO.getOptionValue(OPTION_CHECKOUT_DATE);
            if (dateStr != null && !dateStr.isEmpty()) {
                try {
                    long timestamp = Long.parseLong(dateStr);
                    Date date = new Date(timestamp);
                    dateSpinner.setValue(date);
                } catch (NumberFormatException e) {
                    // Ignore invalid date format
                    e.printStackTrace();
                }
            }

            // Load selected type
            String typeIdStr =
optionDAO.getOptionValue(OPTION_SELECTED_TYPE);
            if (typeIdStr != null && !typeIdStr.isEmpty()) {
                try {
                    long typeId = Long.parseLong(typeIdStr);
                    Type type = typeDAO.getById(typeId);
                    if (type != null) {
                        // Find the type in the combo box
                        for (int i = 0; i <
typeComboBox.getItemCount(); i++) {
                            Type item = typeComboBox.getItemAt(i);
                            if (item.getId().equals(typeId)) {
                                typeComboBox.setSelectedIndex(i);
                                break;
```

```
                            }
                        }
                    }
                } catch (NumberFormatException e) {
                    // Ignore invalid type ID format
                    e.printStackTrace();
                }
            }
        } catch (Exception e) {
            // Ignore errors when loading options
            e.printStackTrace();
        }
    }

    /**
     * Save options to the database.
     */
    private void saveOptions() {
        try {
            // Save last directory
            String directory = directoryField.getText().trim();
            if (!directory.isEmpty()) {
                optionDAO.setOptionValue(OPTION_LAST_DIRECTORY,
directory);
            }

            // Save branch/tag selection
            boolean isMainBranch = mainBranchRadio.isSelected();
            optionDAO.setOptionValue(OPTION_USE_MAIN_BRANCH,
String.valueOf(isMainBranch));

            // Save date
            Date date = (Date) dateSpinner.getValue();
            optionDAO.setOptionValue(OPTION_CHECKOUT_DATE,
String.valueOf(date.getTime()));

            // Save selected type
            Type selectedType = (Type) typeComboBox.getSelectedItem();
            if (selectedType != null) {
                optionDAO.setOptionValue(OPTION_SELECTED_TYPE,
String.valueOf(selectedType.getId()));
            }
        } catch (Exception e) {
            // Ignore errors when saving options
            e.printStackTrace();
        }
    }

    /**
     * Refresh the checkout panel data.
     * This method is called when the panel is displayed.
     */
    public void refresh() {
```

```
        loadTypes();
        loadProjectsByType();
    }
}
```

Текст файлу ui/CheckoutPanel.java

```
package com.github.manager.ui;

import com.github.manager.dao.OptionDAO;
import com.github.manager.dao.ProjectDAO;
import com.github.manager.dao.TypeDAO;
import com.github.manager.model.Project;
import com.github.manager.model.Type;
import com.github.manager.service.GitHubService;
import com.github.manager.service.PhpmetricsService;

import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.io.File;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.List;

/**
 * Panel for checking out GitHub projects.
 */
public class CheckoutPanel extends JPanel {

    private JTable projectsTable;
    private DefaultTableModel tableModel;
    private JComboBox<Type> typeComboBox;
    private JTextField directoryField;
    private JButton browseButton;
    private JRadioButton mainBranchRadio;
    private JRadioButton tagRadio;
    private JSpinner dateSpinner;
    private ProjectDAO projectDAO;
    private TypeDAO typeDAO;
    private OptionDAO optionDAO;
    private GitHubService gitHubService;
    private PhpmetricsService phpmetricsService;

    // Option names
    private static final String OPTION_LAST_DIRECTORY =
"checkout.last_directory";
    private static final String OPTION_USE_MAIN_BRANCH =
"checkout.use_main_branch";
    private static final String OPTION_CHECKOUT_DATE =
"checkout.date";
```

```
    private static final String OPTION_SELECTED_TYPE =
"checkout.selected_type";

    /**
     * Create a new checkout panel.
     */
    public CheckoutPanel() {
        projectDAO = new ProjectDAO();
        typeDAO = new TypeDAO();
        optionDAO = new OptionDAO();
        gitHubService = new GitHubService();
        phpmetricsService = new PhpmetricsService();
        setLayout(new BorderLayout());

        initComponents();
        loadTypes();
        loadOptions();
    }

    /**
     * Initialize the components of the panel.
     */
    private void initComponents() {
        // Create table
        String[] columnNames = {"ID", "URL", "Type"};
        tableModel = new DefaultTableModel(columnNames, 0) {
            @Override
            public boolean isCellEditable(int row, int column) {
                return false;
            }
        };
        projectsTable = new JTable(tableModel);

projectsTable.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);

        // Add table to scroll pane
        JScrollPane scrollPane = new JScrollPane(projectsTable);
        add(scrollPane, BorderLayout.CENTER);

        // Create form panel
        JPanel formPanel = new JPanel(new BorderLayout());

        // Create filter panel
        JPanel filterPanel = new JPanel(new
FlowLayout(FlowLayout.LEFT));
        filterPanel.add(new JLabel("Type:"));
        typeComboBox = new JComboBox<>();
        typeComboBox.addActionListener(e -> {
            loadProjectsByType();
            saveOptions();
        });
        filterPanel.add(typeComboBox);
```

```
        // Create checkout options panel
        JPanel checkoutPanel = new JPanel(new GridLayout(3, 2, 5, 5));

        // Directory selection
        checkoutPanel.add(new JLabel("Checkout Directory:"));
        JPanel directoryPanel = new JPanel(new BorderLayout());
        directoryField = new JTextField();
        directoryPanel.add(directoryField, BorderLayout.CENTER);
        browseButton = new JButton("Browse");
        browseButton.addActionListener(e -> browseDirectory());
        directoryPanel.add(browseButton, BorderLayout.EAST);
        checkoutPanel.add(directoryPanel);

        // Branch/Tag selection
        checkoutPanel.add(new JLabel("Checkout:"));
        JPanel radioPanel = new JPanel(new
FlowLayout(FlowLayout.LEFT));
        mainBranchRadio = new JRadioButton("Main Branch", true);
        tagRadio = new JRadioButton("Latest Tag");
        ButtonGroup radioGroup = new ButtonGroup();
        radioGroup.add(mainBranchRadio);
        radioGroup.add(tagRadio);
        radioPanel.add(mainBranchRadio);
        radioPanel.add(tagRadio);
        checkoutPanel.add(radioPanel);

        // Date selection
        checkoutPanel.add(new JLabel("Date:"));
        SpinnerDateModel dateModel = new SpinnerDateModel(new Date(),
null, null, java.util.Calendar.DAY_OF_MONTH);
        dateSpinner = new JSpinner(dateModel);
        JSpinner.DateEditor dateEditor = new
JSpinner.DateEditor(dateSpinner, "yyyy-MM-dd");
        dateSpinner.setEditor(dateEditor);
        checkoutPanel.add(dateSpinner);

        // Create button panel
        JPanel buttonPanel = new JPanel(new
FlowLayout(FlowLayout.LEFT));
        JButton checkoutButton = new JButton("Checkout");

        buttonPanel.add(checkoutButton);

        // Add action listeners
        checkoutButton.addActionListener(e -> checkoutProject());

        // Add panels to form panel
        formPanel.add(filterPanel, BorderLayout.NORTH);
        formPanel.add(checkoutPanel, BorderLayout.CENTER);
        formPanel.add(buttonPanel, BorderLayout.SOUTH);

        // Add form panel to main panel
        add(formPanel, BorderLayout.SOUTH);
```

```
    }

    /**
     * Load types from the database.
     */
    public void loadTypes() {
        // Clear combo box
        typeComboBox.removeAllItems();

        // Get all types
        List<Type> types = typeDAO.getAll();

        // Add types to combo box
        if (types != null) {
            for (Type type : types) {
                typeComboBox.addItem(type);
            }
        }
    }

    /**
     * Load projects by the selected type.
     */
    public void loadProjectsByType() {
        // Clear table
        tableModel.setRowCount(0);

        // Get selected type
        Type type = (Type) typeComboBox.getSelectedItem();

        if (type != null) {
            // Get projects by type
            List<Project> projects = projectDAO.getByType(type);

            // Add projects to table
            if (projects != null) {
                for (Project project : projects) {
                    Object[] row = {
                        project.getId(),
                        project.getUrl(),
                        project.getType() != null ?
project.getType().getName() : ""
                    };
                    tableModel.addRow(row);
                }
            }
        }
    }

    private void browseDirectory() {
        JFileChooser fileChooser = new JFileChooser();

fileChooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
```

```
        fileChooser.setDialogTitle("Select Checkout Directory");

        int result = fileChooser.showOpenDialog(this);

        if (result == JFileChooser.APPROVE_OPTION) {
            File selectedFile = fileChooser.getSelectedFile();
            directoryField.setText(selectedFile.getAbsolutePath());

            // Save the selected directory
            saveOptions();
        }
    }

    /**
     * Checkout all projects of the selected type.
     */
    private void checkoutProject() {
        Type selectedType = (Type) typeComboBox.getSelectedItem();

        if (selectedType == null) {
            JOptionPane.showMessageDialog(this, "Please select a
project type", "Error", JOptionPane.ERROR_MESSAGE);
            return;
        }

        String directory = directoryField.getText().trim();
        Date date = (Date) dateSpinner.getValue();
        boolean isMainBranch = mainBranchRadio.isSelected();

        if (directory.isEmpty()) {
            JOptionPane.showMessageDialog(this, "Please select a
directory", "Error", JOptionPane.ERROR_MESSAGE);
            return;
        }

        // Save the checkout settings
        saveOptions();

        // Create directory if it doesn't exist
        File baseDir = new File(directory);
        if (!baseDir.exists()) {
            baseDir.mkdirs();
        }

        // Get all projects of the selected type
        List<Project> projects = projectDAO.getByType(selectedType);

        if (projects == null || projects.isEmpty()) {
            JOptionPane.showMessageDialog(this, "No projects found for
the selected type", "Error", JOptionPane.ERROR_MESSAGE);
            return;
        }
```

```
        // Show progress dialog
        JDialog progressDialog = new JDialog((Frame)
SwingUtilities.getWindowAncestor(this), "Checkout Progress", true);
        progressDialog.setLayout(new BorderLayout());
        progressDialog.setSize(400, 150);
        progressDialog.setLocationRelativeTo(this);

        JLabel progressLabel = new JLabel("Preparing to checkout
projects...");
        progressLabel.setHorizontalAlignment(JLabel.CENTER);
        JProgressBar progressBar = new JProgressBar(0,
projects.size());
        progressBar.setStringPainted(true);

        JPanel progressPanel = new JPanel(new BorderLayout(5, 5));
        progressPanel.setBorder(BorderFactory.createEmptyBorder(10,
10, 10, 10));
        progressPanel.add(progressLabel, BorderLayout.NORTH);
        progressPanel.add(progressBar, BorderLayout.CENTER);

        progressDialog.add(progressPanel, BorderLayout.CENTER);

        // Start checkout in a separate thread
        new Thread(() -> {
            try {
                int progress = 0;
                StringBuilder errorMessages = new StringBuilder();

                for (Project project : projects) {
                    String url = project.getUrl();
                    String projectName = extractProjectName(url);

                    // Update progress
                    final int currentProgress = progress;
                    SwingUtilities.invokeLater(() -> {
                        progressLabel.setText("Processing project: " +
projectName);
                        progressBar.setValue(currentProgress);
                        progressBar.setString(currentProgress + " of "
+ projects.size());
                    });

                    // Create subdirectory for the project
                    File projectDir = new File(baseDir, projectName);

                    // Delete directory if it exists
                    if (projectDir.exists()) {
                        SwingUtilities.invokeLater(() ->
progressLabel.setText("Deleting existing directory: " + projectName));
                        deleteDirectory(projectDir);
                    }

                    // Create directory
```

```
                    projectDir.mkdirs();

                    try {
                        // Clone repository
                        SwingUtilities.invokeLater(() ->
progressLabel.setText("Cloning repository: " + projectName));
                        org.eclipse.jgit.api.Git git =
gitHubService.cloneRepository(url, projectDir);

                        // Checkout branch or tag
                        if (isMainBranch) {
                            SwingUtilities.invokeLater(() ->
progressLabel.setText("Checking out main branch at date for: " +
projectName));

gitHubService.checkoutMainBranchAtDate(git, date);
                        } else {
                            SwingUtilities.invokeLater(() ->
progressLabel.setText("Checking out latest tag at date for: " +
projectName));
                            gitHubService.checkoutLatestTagAtDate(git,
date);
                        }

                        // Run phpmetrics in the checked out project
directory
                        phpmetricsService.runPhpmetrics(projectDir);
                    } catch (Exception e) {
                        errorMessages.append("Error processing
").append(projectName).append(":
").append(e.getMessage()).append("\n");
                        e.printStackTrace();
                    }

                    progress++;
                }

                // Update final progress
                SwingUtilities.invokeLater(() -> {
                    progressBar.setValue(projects.size());
                    progressBar.setString(projects.size() + " of " +
projects.size());
                    progressLabel.setText("Checkout completed");
                });

                // Close progress dialog
                final String errors = errorMessages.toString();
                SwingUtilities.invokeLater(() -> {
                    progressDialog.dispose();
                    if (errors.isEmpty()) {
                        JOptionPane.showMessageDialog(this, "All
projects checked out successfully", "Success",
JOptionPane.INFORMATION_MESSAGE);
```

```
                    } else {
                        JOptionPane.showMessageDialog(this, "Checkout
completed with errors:\n" + errors, "Warning",
JOptionPane.WARNING_MESSAGE);
                    }
                });
            } catch (Exception e) {
                // Close progress dialog and show error
                SwingUtilities.invokeLater(() -> {
                    progressDialog.dispose();
                    JOptionPane.showMessageDialog(this, "Checkout
failed: " + e.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
                });
                e.printStackTrace();
            }
        }).start();

        // Show progress dialog
        progressDialog.setVisible(true);
    }

    /**
     * Extract project name from GitHub URL.
     *
     * @param url The GitHub URL
     * @return The project name
     */
    private String extractProjectName(String url) {
        // Remove .git extension if present
        String cleanUrl = url.endsWith(".git") ? url.substring(0,
url.length() - 4) : url;

        // Extract the repository name from the URL
        int lastSlashIndex = cleanUrl.lastIndexOf('/');
        if (lastSlashIndex != -1 && lastSlashIndex < cleanUrl.length()
- 1) {
            return cleanUrl.substring(lastSlashIndex + 1);
        }

        // Fallback: use the full URL as a hash
        return String.valueOf(cleanUrl.hashCode()).replace("-", "");
    }

    /**
     * Recursively delete a directory.
     *
     * @param directory The directory to delete
     * @return true if successful, false otherwise
     */
    private boolean deleteDirectory(File directory) {
        if (directory.exists()) {
            File[] files = directory.listFiles();
            if (files != null) {
```

```
                for (File file : files) {
                    if (file.isDirectory()) {
                        deleteDirectory(file);
                    } else {
                        file.delete();
                    }
                }
            }
        }
        return directory.delete();
    }

    /**
     * Load options from the database.
     */
    private void loadOptions() {
        try {
            // Load last directory
            String lastDirectory =
optionDAO.getOptionValue(OPTION_LAST_DIRECTORY);
            if (lastDirectory != null && !lastDirectory.isEmpty()) {
                directoryField.setText(lastDirectory);
            }

            // Load branch/tag selection
            String useMainBranch =
optionDAO.getOptionValue(OPTION_USE_MAIN_BRANCH);
            if (useMainBranch != null) {
                boolean isMainBranch =
Boolean.parseBoolean(useMainBranch);
                mainBranchRadio.setSelected(isMainBranch);
                tagRadio.setSelected(!isMainBranch);
            }

            // Load date
            String dateStr =
optionDAO.getOptionValue(OPTION_CHECKOUT_DATE);
            if (dateStr != null && !dateStr.isEmpty()) {
                try {
                    long timestamp = Long.parseLong(dateStr);
                    Date date = new Date(timestamp);
                    dateSpinner.setValue(date);
                } catch (NumberFormatException e) {
                    // Ignore invalid date format
                    e.printStackTrace();
                }
            }

            // Load selected type
            String typeIdStr =
optionDAO.getOptionValue(OPTION_SELECTED_TYPE);
            if (typeIdStr != null && !typeIdStr.isEmpty()) {
                try {
```

```
                    long typeId = Long.parseLong(typeIdStr);
                    Type type = typeDAO.getById(typeId);
                    if (type != null) {
                        // Find the type in the combo box
                        for (int i = 0; i <
typeComboBox.getItemCount(); i++) {
                            Type item = typeComboBox.getItemAt(i);
                            if (item.getId().equals(typeId)) {
                                typeComboBox.setSelectedIndex(i);
                                break;
                            }
                        }
                    }
                } catch (NumberFormatException e) {
                    // Ignore invalid type ID format
                    e.printStackTrace();
                }
            }
        } catch (Exception e) {
            // Ignore errors when loading options
            e.printStackTrace();
        }
    }

    /**
     * Save options to the database.
     */
    private void saveOptions() {
        try {
            // Save last directory
            String directory = directoryField.getText().trim();
            if (!directory.isEmpty()) {
                optionDAO.setOptionValue(OPTION_LAST_DIRECTORY,
directory);
            }

            // Save branch/tag selection
            boolean isMainBranch = mainBranchRadio.isSelected();
            optionDAO.setOptionValue(OPTION_USE_MAIN_BRANCH,
String.valueOf(isMainBranch));

            // Save date
            Date date = (Date) dateSpinner.getValue();
            optionDAO.setOptionValue(OPTION_CHECKOUT_DATE,
String.valueOf(date.getTime()));

            // Save selected type
            Type selectedType = (Type) typeComboBox.getSelectedItem();
            if (selectedType != null) {
                optionDAO.setOptionValue(OPTION_SELECTED_TYPE,
String.valueOf(selectedType.getId()));
            }
        } catch (Exception e) {
```

```
            // Ignore errors when saving options
            e.printStackTrace();
        }
    }

    /**
     * Refresh the checkout panel data.
     * This method is called when the panel is displayed.
     */
    public void refresh() {
        loadTypes();
        loadProjectsByType();
    }
}
```

Текст файлу ui/MainFrame.java

```
package com.github.manager.ui;

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

import com.github.manager.util.HibernateUtil;

/**
 * Main application frame.
 */
public class MainFrame extends JFrame {

    private JTabbedPane tabbedPane;

    /**
     * Create a new main frame.
     */
    public MainFrame() {
        setTitle("GitHub Project Metrics");
        setSize(800, 600);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);

        // Add window listener to close Hibernate session factory on
exit
        addWindowListener(new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent e) {
                HibernateUtil.shutdown();
            }
        });
```

```
        initComponents();
    }

    private void initComponents() {
        // Create tabbed pane
        tabbedPane = new JTabbedPane();

        // Add tabs
        TypesPanel typesPanel = new TypesPanel();
        ProjectsPanel projectsPanel = new ProjectsPanel();
        CheckoutPanel checkoutPanel = new CheckoutPanel();

        tabbedPane.addTab("Types", typesPanel);
        tabbedPane.addTab("Projects", projectsPanel);
        tabbedPane.addTab("Checkout", checkoutPanel);

        // Add change listener to refresh data when tab is selected
        tabbedPane.addChangeListener(new ChangeListener() {
            @Override
            public void stateChanged(ChangeEvent e) {
                int selectedIndex = tabbedPane.getSelectedIndex();
                if (selectedIndex == 0) {
                    // Types panel selected
                    typesPanel.loadTypes();
                } else if (selectedIndex == 1) {
                    // Projects panel selected
                    projectsPanel.loadProjects();
                    projectsPanel.loadTypes();
                } else if (selectedIndex == 2) {
                    // Checkout panel selected
                    checkoutPanel.loadTypes();
                    checkoutPanel.loadProjectsByType();
                }
            }
        });

        // Add tabbed pane to frame
        getContentPane().add(tabbedPane, BorderLayout.CENTER);
    }

    public static void main(String[] args) {
        // Set look and feel to system look and feel
        try {

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

        // Create and show the main frame
        SwingUtilities.invokeLater(() -> {
            MainFrame frame = new MainFrame();
```

```
            frame.setVisible(true);
        });
    }
}
```

Текст файлу dao/TypeDAO.java

```
package com.github.manager.dao;

import com.github.manager.model.Type;
import com.github.manager.util.HibernateUtil;
import org.hibernate.Session;
import org.hibernate.Transaction;

import java.util.List;

public class TypeDAO {

    public Type saveOrUpdate(Type type) {
        Transaction transaction = null;
        try (Session session =
HibernateUtil.getSessionFactory().openSession()) {
            transaction = session.beginTransaction();
            session.saveOrUpdate(type);
            transaction.commit();
            return type;
        } catch (Exception e) {
            if (transaction != null) {
                transaction.rollback();
            }
            e.printStackTrace();
            return null;
        }
    }

    public Type getById(Long id) {
        try (Session session =
HibernateUtil.getSessionFactory().openSession()) {
            return session.get(Type.class, id);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }

    @SuppressWarnings("unchecked")
    public List<Type> getAll() {
        try (Session session =
HibernateUtil.getSessionFactory().openSession()) {
            return session.createQuery("FROM Type").list();
        } catch (Exception e) {
            e.printStackTrace();
```

```
            return null;
        }
    }

    public boolean delete(Type type) {
        Transaction transaction = null;
        try (Session session =
HibernateUtil.getSessionFactory().openSession()) {
            transaction = session.beginTransaction();
            session.delete(type);
            transaction.commit();
            return true;
        } catch (Exception e) {
            if (transaction != null) {
                transaction.rollback();
            }
            e.printStackTrace();
            return false;
        }
    }

    public Type findByName(String name) {
        try (Session session =
HibernateUtil.getSessionFactory().openSession()) {
            return (Type) session.createQuery("FROM Type WHERE name =
:name")
                    .setParameter("name", name)
                    .uniqueResult();
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }
}
```

Текст файлу dao/ProjectDAO.java

```
package com.github.manager.dao;

import com.github.manager.model.Project;
import com.github.manager.model.Type;
import com.github.manager.util.HibernateUtil;
import org.hibernate.Session;
import org.hibernate.Transaction;

import java.util.List;

/**
 * Data Access Object for Project entity.
 */
public class ProjectDAO {
```

```
      public Project saveOrUpdate(Project project) {
        Transaction transaction = null;
        try (Session session =
HibernateUtil.getSessionFactory().openSession()) {
            transaction = session.beginTransaction();
            session.saveOrUpdate(project);
            transaction.commit();
            return project;
        } catch (Exception e) {
            if (transaction != null) {
                transaction.rollback();
            }
            e.printStackTrace();
            return null;
        }
    }

    public Project getById(Long id) {
        try (Session session =
HibernateUtil.getSessionFactory().openSession()) {
            return session.get(Project.class, id);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }

    @SuppressWarnings("unchecked")
    public List<Project> getAll() {
        try (Session session =
HibernateUtil.getSessionFactory().openSession()) {
            return session.createQuery("FROM Project").list();
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }

    public boolean delete(Project project) {
        Transaction transaction = null;
        try (Session session =
HibernateUtil.getSessionFactory().openSession()) {
            transaction = session.beginTransaction();
            session.delete(project);
            transaction.commit();
            return true;
        } catch (Exception e) {
            if (transaction != null) {
                transaction.rollback();
            }
            e.printStackTrace();
            return false;
        }
```

```
    }

    public Project findByUrl(String url) {
        try (Session session =
HibernateUtil.getSessionFactory().openSession()) {
            return (Project) session.createQuery("FROM Project WHERE
url = :url")
                    .setParameter("url", url)
                    .uniqueResult();
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }

    @SuppressWarnings("unchecked")
    public List<Project> getByType(Type type) {
        try (Session session =
HibernateUtil.getSessionFactory().openSession()) {
            return session.createQuery("FROM Project WHERE type =
:type")
                    .setParameter("type", type)
                    .list();
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }
}
```

Текст файлу dao/OptionDAO.java

```
package com.github.manager.dao;

import com.github.manager.model.Option;
import com.github.manager.util.HibernateUtil;
import org.hibernate.Session;
import org.hibernate.Transaction;

import java.util.List;

/**
 * Data Access Object for Option entity.
 */
public class OptionDAO {

    public Option saveOrUpdate(Option option) {
        Transaction transaction = null;
        try (Session session =
HibernateUtil.getSessionFactory().openSession()) {
            transaction = session.beginTransaction();
            session.saveOrUpdate(option);
```

```
            transaction.commit();
            return option;
        } catch (Exception e) {
            if (transaction != null) {
                transaction.rollback();
            }
            e.printStackTrace();
            return null;
        }
    }

    public Option getById(Long id) {
        try (Session session =
HibernateUtil.getSessionFactory().openSession()) {
            return session.get(Option.class, id);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }

    @SuppressWarnings("unchecked")
    public List<Option> getAll() {
        try (Session session =
HibernateUtil.getSessionFactory().openSession()) {
            return session.createQuery("FROM Option").list();
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }

    public boolean delete(Option option) {
        Transaction transaction = null;
        try (Session session =
HibernateUtil.getSessionFactory().openSession()) {
            transaction = session.beginTransaction();
            session.delete(option);
            transaction.commit();
            return true;
        } catch (Exception e) {
            if (transaction != null) {
                transaction.rollback();
            }
            e.printStackTrace();
            return false;
        }
    }

    public Option findByName(String name) {
        try (Session session =
HibernateUtil.getSessionFactory().openSession()) {
```

```
            return (Option) session.createQuery("FROM Option WHERE
name = :name")
                    .setParameter("name", name)
                    .uniqueResult();
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }

     public String getOptionValue(String name) {
        Option option = findByName(name);
        return option != null ? option.getValue() : null;
    }

    public Option setOptionValue(String name, String value) {
        Option option = findByName(name);
        if (option == null) {
            option = new Option(name, value);
        } else {
            option.setValue(value);
        }
        return saveOrUpdate(option);
    }
}
```

Текст файлу util/HibernateUtil.java

```
package com.github.manager.util;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HibernateUtil {
    private static final SessionFactory sessionFactory =
buildSessionFactory();

    private static SessionFactory buildSessionFactory() {
        try {
            // Create the SessionFactory from hibernate.cfg.xml
            return new
Configuration().configure().buildSessionFactory();
        } catch (Throwable ex) {
            System.err.println("Initial SessionFactory creation
failed: " + ex);
            throw new ExceptionInInitializerError(ex);
        }
    }

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
```

```
    public static void shutdown() {
        // Close caches and connection pools
        getSessionFactory().close();
    }
}
```

Текст файлу schema.sql

```
-- Table for project types
CREATE TABLE types (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL UNIQUE
);

-- Table for GitHub projects
CREATE TABLE projects (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    url TEXT NOT NULL UNIQUE,
    type_id INTEGER,
    FOREIGN KEY (type_id) REFERENCES types(id)
);

-- Table for application options
CREATE TABLE options (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL UNIQUE,
    value TEXT
);
```

ДОДАТОК В – ІНСТРУКЦІЯ КОРИСТУВАЧА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОМАТИЗАЦІЇ ЗБОРУ МЕТРИК GITHUB ПРОЄКТІВ НА МОВІ PHP УТИЛІТОЮ PHPMETRICS

Після запуску програмного забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics спочатку відображається вкладка з списком типів GitHub проєктів (рисунок В.1).

Для того щоб додати новий тип GitHub проєкту, треба ввести назву типу і натиснути кнопку «Add».

Для того щоб змінити назву раніше доданого типу GitHub проєкту, треба натисканням кнопки миші вибрати його зі списку, ввести нову назву і натиснути кнопку «Edit».

Для того щоб видалити раніше доданий тип GitHub проєкту, треба натисканням кнопки миші вибрати його зі списку і натиснути кнопку «Delete».

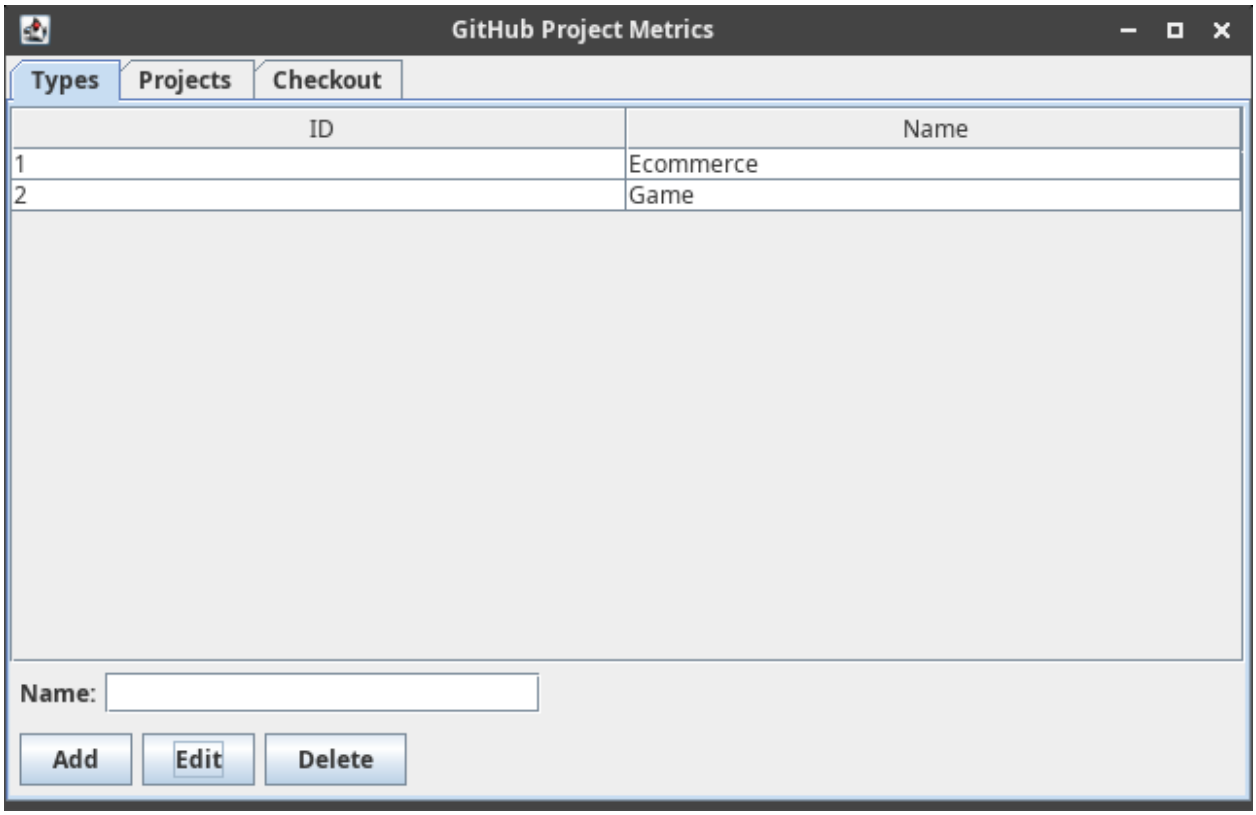


Рисунок В.1 – Знімок екрану вкладки Types

Для того щоб перейти до вкладки зі списком GitHub проєктів (рисунок В.2), треба натиснути кнопкою миші на її заголовок «Projects».

Для того щоб додати новий GitHub проєкт, треба ввести його URL-адресу, вибрати тип до якого належить цей проєкт і натиснути кнопку «Add».

Для того щоб змінити URL або тип раніше доданого GitHub проєкту, треба натисканням кнопки миші вибрати його зі списку, ввести новий URL чи вибрати новий тип проєкту, та натиснути кнопку «Edit».

Для того щоб видалити раніше доданий GitHub проєкт, треба натисканням кнопки миші вибрати його зі списку і натиснути кнопку «Delete».

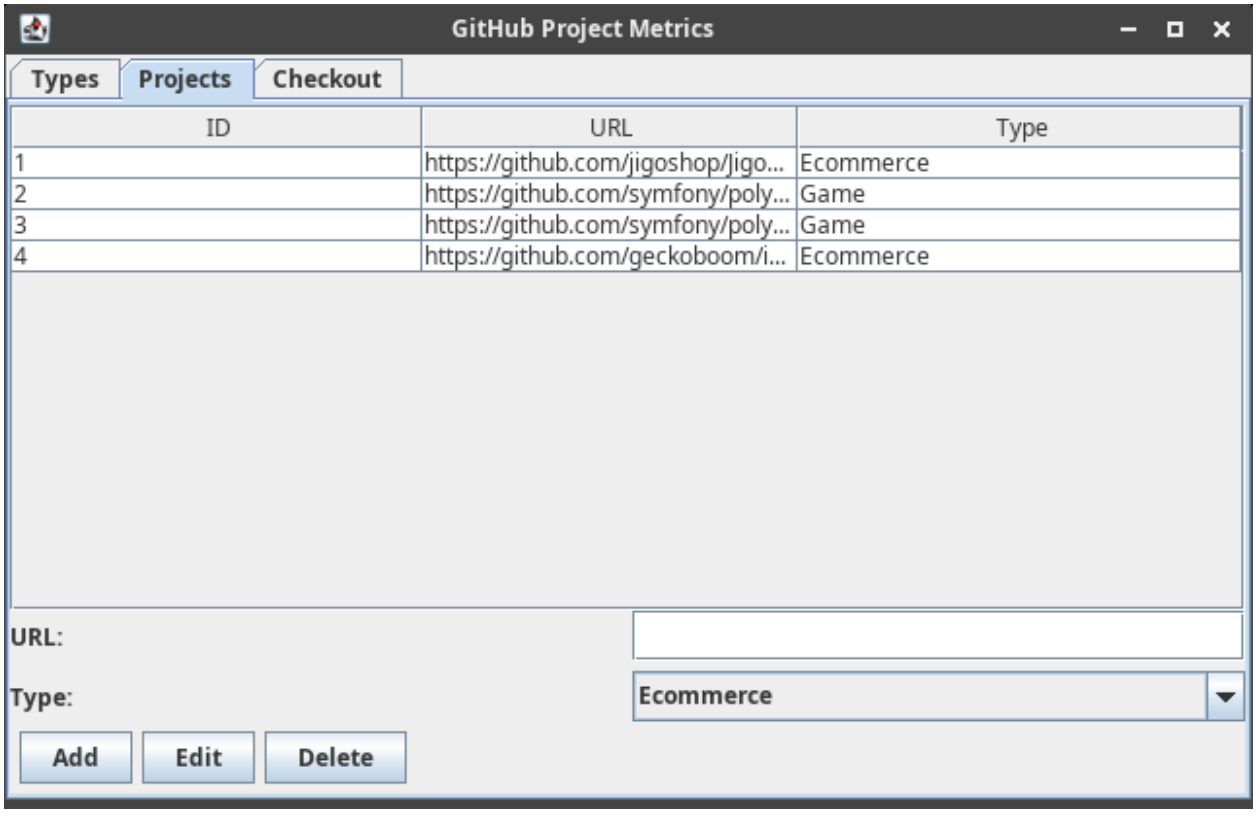


Рисунок В.2 – Знімок екрану вкладки Projects

Для того, щоб здійснити отримання метрик GitHub проєктів, треба спочатку перейти до вкладки «Checkout» (рисунок В.3), натиснувши кнопкою миші на її заголовок.

На цій вкладці необхідно вибрати тип GitHub проєктів, для яких буде потрібно отримати метрики, вибрати каталог для збереження файлів, вибрати вітку і дату коду, після чого натиснути кнопку «Checkout».

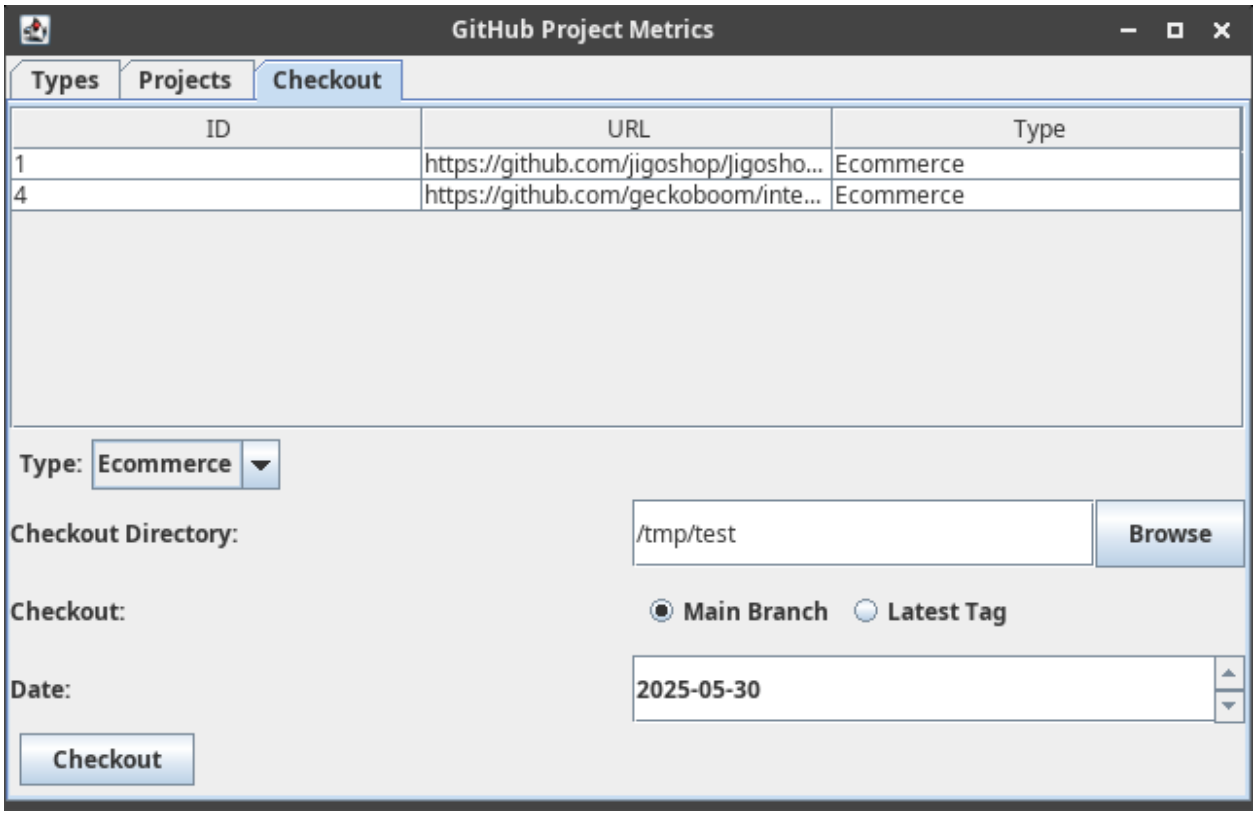


Рисунок В.3 – Знімок екрану вкладки Checkout

В результаті буде отримано метрики GitHub проєктів вибраного типу, про що користувач отримає повідомлення, яке міститиме ім’я файлу зі звітом та шляхом до нього (рисунок В.4).

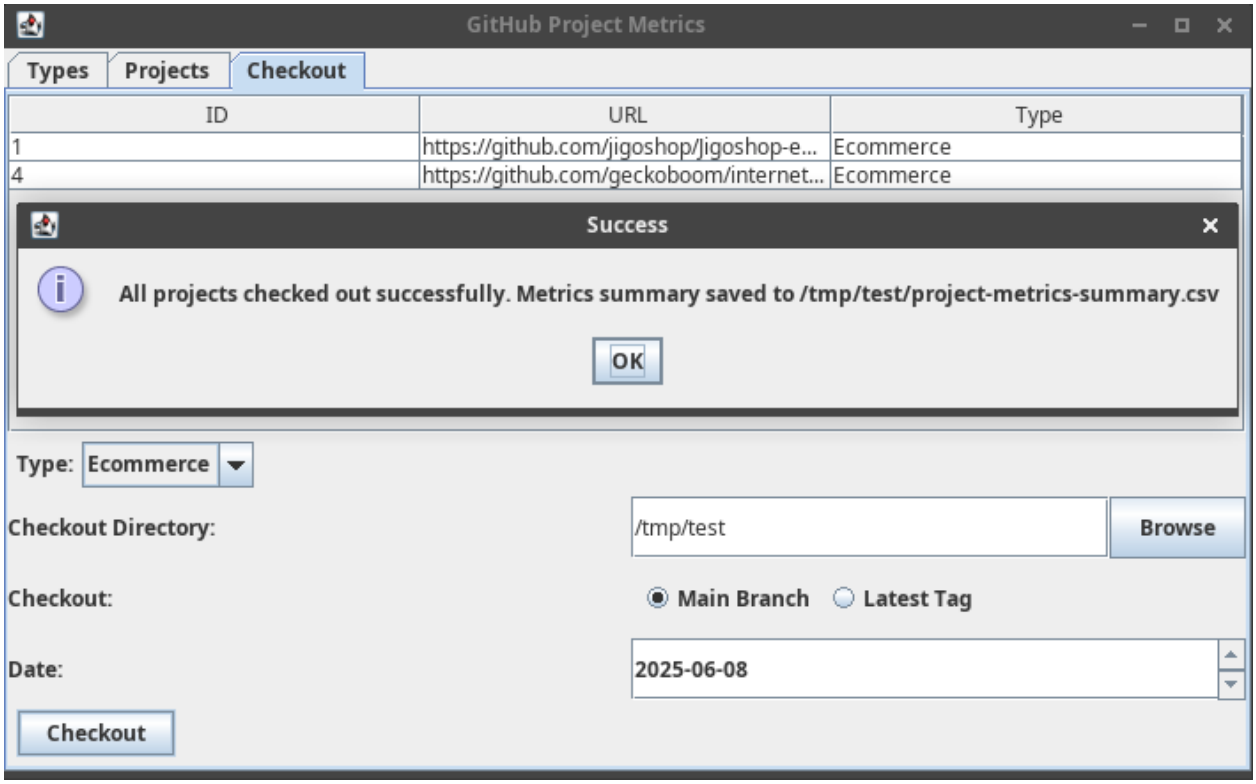
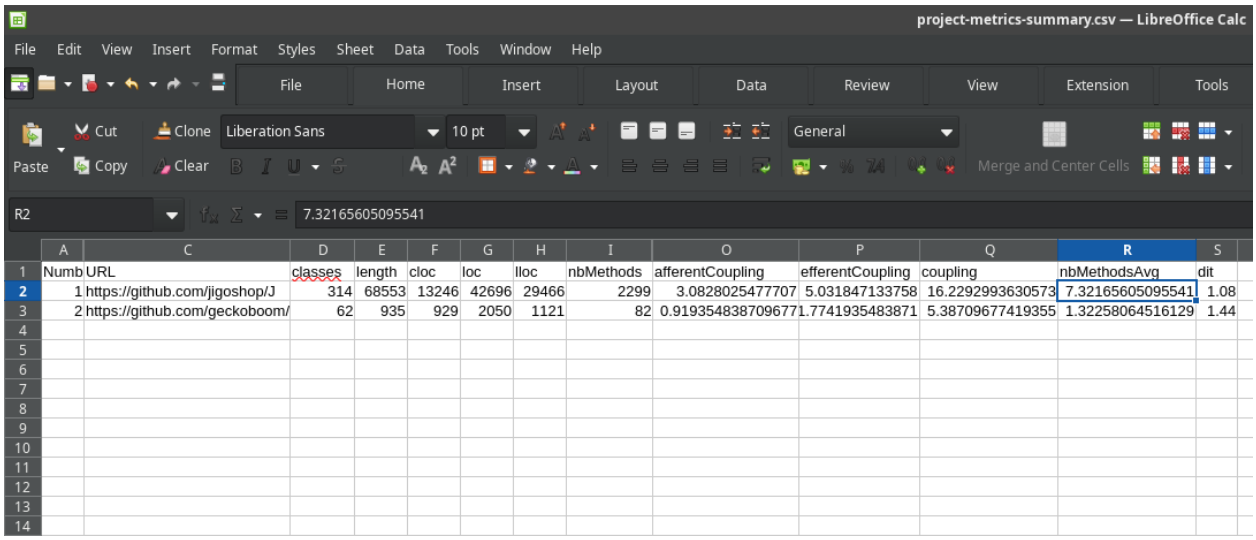


Рисунок В.4 – Повідомленні про успішне завершення збору метрик

Файл зі звітом у форматі CSV (рисунок В.5) буде збережено у каталозі, який використовувався для завантаження файлів.



	A	C	D	E	F	G	H	I	O	P	Q	R	S
1	Numb	URL	classes	length	cloc	loc	lloc	nbMethods	afferentCoupling	efferentCoupling	coupling	nbMethodsAvg	dit
2	1	https://github.com/jigoshop/J	314	68553	13246	42696	29466	2299	3.0828025477707	5.031847133758	16.2292993630573	7.32165605095541	1.08
3	2	https://github.com/geckoboom/	62	935	929	2050	1121	82	0.9193548387096771	1.7741935483871	5.38709677419355	1.32258064516129	1.44
4													
5													
6													
7													
8													
9													
10													
11													
12													
13													
14													

Рисунок В.5 – Файл зі звітом у форматі CSV

Після завершення збору метрик програму можна закрити. Завантажені файли і звіт залишаться, і автоматично видалятись не будуть.

ДОДАТОК Г – ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОМАТИЗАЦІЇ ЗБОРУ МЕТРИК GITHUB ПРОЄКТІВ НА МОВІ PHP УТИЛІТОЮ PHPMETRICS

Вступ

Найменування програми: Програмне забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics.

Позначення програми: GitHub Manager.

1 Функціональне призначення

1.1 Класи розв’язуваних завдань та призначення програми

Призначенням розроблюваного програмного забезпечення є автоматизація збору метрик PHP-коду з GitHub репозиторіїв за допомогою утиліти PhpMetrics. Програма дозволяє користувачам:

1) додавати та класифікувати GitHub репозиторії за типами проєктів;

2) клонувати репозиторії з GitHub;

3) запускати утиліту PhpMetrics для отримання метрик коду та формувати звіти з метриками у зрозумілому табличному форматі. Програма призначена для розробників, аналітиків та дослідників, які потребують аналізу якості PHP-коду в проєктах, розміщених на GitHub.

1.2 Функціональні обмеження на застосування

Програма підтримує аналіз лише PHP-коду, оскільки використовує утиліту PhpMetrics. Для роботи програми потрібен доступ до мережі Інтернет для клонування GitHub репозиторіїв. Програма не може працювати з приватними GitHub репозиторіями.

2 Опис логічної структури

2.1 Опис структури

Програмне забезпечення побудоване на основі об’єктно-орієнтованого програмування (ООП) з використанням шаблону проєктування MVC (Model-View-Controller).

Основні модулі програмного забезпечення:

Модуль управління базою даних для операцій CRUD (Create, Read, Update, Delete) з таблицями у базі даних SQLite.

Модуль роботи з GitHub: Реалізований через бібліотеку org.eclipse.jgit і клас GitHubService, який відповідає за клонування репозиторіїв, отримання тегів і перемикання на потрібні коміти.

Модуль аналізу метрик: Реалізований через клас PhpMetricsService, який викликає утиліту PhpMetrics для аналізу PHP-коду та обробляє отримані звіти.

Модуль графічного інтерфейсу: Реалізований через бібліотеку javax.swing і включає класи для створення вікон, вкладок, таблиць і діалогових вікон.

Конфігураційні файли: Включають schema.sql для створення структури бази даних і конфігураційні параметри для налаштування шляхів до утиліт і каталогів.

2.2 Мови програмування

Для розробки використано мову програмування Java. Додатково використовується SQL для створення та управління структурою бази даних SQLite. Утиліта PhpMetrics викликається через командний рядок, тому для обробки її викликів використовуються системні команди, реалізовані через Java.

2.3 Вхідні та вихідні дані

Для додавання типу проєкту: вхідними даними є назва типу проєкту. Вихідними даними є запис у базі даних про новий тип проєкту.

Для видалення типу проєкту: вхідними даними є ідентифікатор або назва типу проєкту. Вихідними даними є оновлена база даних без вказаного типу проєкту.

Для перегляду списку типів проєктів: вхідними даними є запит на отримання списку. Вихідними даними є список типів проєктів у табличному або іншому зрозумілому форматі.

Для додавання проєкту: вхідними даними є URL-адреса GitHub репозиторію та тип проєкту. Вихідними даними є запис у базі даних про новий проєкт.

Для видалення проєкту: вхідними даними є ідентифікатор або URL-адреса проєкту. Вихідними даними є оновлена база даних без вказаного проєкту.

Для перегляду списку проєктів: вхідними даними є запит на отримання списку. Вихідними даними є список проєктів із зазначенням їх URL-адрес та типів у табличному форматі.

Для отримання метрик проєктів: вхідними даними є тип проєкту, дата, станом на яку необхідно отримати код, та шлях до каталогу, в який буде виконано завантаження вихідного проєктів. Вихідними даними є звіт у табличному форматі із проєктами вибраного типу та агрегованими значеннями метрик (сума або середнє) отриманих після запуску утиліти PhpMetrics: кількість рядків коду, кількість логічних рядків коду, кількість рядків коментарів, кількість класів, кількість інтерфейсів, кількість методів, середнє вхідне зчеплення, середнє вихідне зчеплення, середня нестабільність, глибина дерева успадкування, середня цикломатична складність, середня відносна складність системи, середня складність, середня кількість помилок на клас, середня кількість дефектів на клас.

3 Склад і функції

3.1 Склад програмного забезпечення

Програмне забезпечення складається з одного скомпільованого JAR файлу на програмної документації.

3.2 Функції програмного забезпечення

1) додавання типу проєкту;

2) видалення типу проєкту;

3) перегляд списку типів проєктів;

4) додавання проєкту;

5) видалення проєкту;

6) перегляд списку проєктів;

7) отримання метрик проєктів.

4 Умови застосування

4.1 Вимоги до складу та параметрів технічних засобів

Двоядерний процесор з частотою 2 ГГц (наприклад, Intel Core i3 або еквівалент).

Оперативна пам’ять: 4 ГБ.

Дисковий простір: 50МБ для програми та бази даних, додатково місце для клонованих репозиторіїв (залежить від їх розміру).

Операційна система: Windows 10/11 або Linux (Ubuntu 22.04/24.04 або сумісні дистрибутиви).

Мережевий доступ: Інтернет-з’єднання для клонування репозиторіїв.

Монітор з роздільною здатністю 1024x678 для роботи з графічним інтерфейсом.

4.2 Вимоги до програмної сумісності

Java Runtime Environment (JRE) версії 11 або вище для запуску програми.

4.3 Спосіб виклику програми

Виклик програмного забезпечення відбувається шляхом запуску виконуваного JAR-файлу командою java -jar GitHubManager.jar з командного

рядка, або подвійним кліком у графічному середовищі, якщо JRE налаштовано.

Вхідною точкою в програмне забезпечення є клас Main. У цьому класі визначається ініціалізація графічного інтерфейсу, підключення до бази даних і створення основних сервісів (GitHubService, PhpMetricsService). Метод main викликається та використовується для запуску програми, створення головного вікна (JFrame) і відображення вкладок для роботи з типами проєктів, проєктами та звітами.

ДОДАТОК Д – ПРОГРАМА ТА МЕТОДИКА ВИПРОБУВАНЬ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОМАТИЗАЦІЇ ЗБОРУ МЕТРИК GITHUB ПРОЄКТІВ НА МОВІ PHP УТИЛІТОЮ PHPMETRICS

1 Об’єкт випробувань

Об’єктом випробувань є програмне забезпечення для автоматизації збору метрик GitHub проєктів на мові PHP утилітою PhpMetrics.

2 Мета випробувань

Мета проведення випробувань - це перевірка програмного забезпечення на відповідність вимогам, що наведені в технічному завданні.

3 Вимоги до програми

В процесі проведення випробувань потрібно перевірити, що розроблене програмне забезпечення реалізує наступні функції:

1) додавання типу проєкту;
2) видалення типу проєкту;
3) перегляд списку типів проєктів;
4) додавання проєкту;
5) видалення проєкту;
6) перегляд списку проєктів;
7) отримання метрик проєктів.

4 Вимоги до програмної документації

До програмної документації, що надається на випробування входить:

- технічне завдання на розробку програмного забезпечення;
- текст програмного забезпечення;
- інструкція користувача програмного забезпечення;
- опис програмного забезпечення;
- програма і методика випробувань програмного забезпечення.

5 Засоби та порядок випробувань

5.1 Засоби випробувань

До складу технічних засобів, які мають використовуватися при випробуваннях належать:

- Комп’ютер з процесором Intel Core i3 або еквівалентним, підключений до інтернету зі швидкістю від 1 МБіт.

До складу програмних засобів, які мають використовуватися при випробуваннях належать:

- операційна система Ubuntu 24.04;
- PhpMetrics 3.

5.2 Порядок проведення випробувань

1) перевірка комплектності програмної документації програмного забезпечення, призначеного для автоматизації збору метрик GitHub проєктів утилітою PhpMetrics;
2) перевірка функції додавання типу проєкту;
3) перевірка функції редагування типу проєкту;
4) перевірка функції видалення типу проєкту;
5) перевірка функції додавання проєкту;
6) перевірка функції редагування проєкту;
7) перевірка функції видалення проєкту;
8) перевірка функції отримання метрик проєктів.

6 Методи випробувань

1) випробування функції додавання типу проєкту;

У графічному інтерфейсі відкрити вкладку «Types», ввести унікальну назву типу проєкту (наприклад, «Web Application») у текстове поле та натиснути кнопку "Add".

Очікуваний результат: Новий тип проєкту додається до списку у вкладці «Types» і записується в таблицю types бази даних SQLite.

2) випробування функції редагування типу проєкту;

У вкладці «Types» вибрати зі списку тип проєкту (наприклад, «Web Application») натисканням кнопки миші, ввести нову назву (наприклад, «E-Commerce») у текстове поле та натиснути кнопку «Edit».

Очікуваний результат: Назва обраного типу проєкту оновлюється в списку у вкладці «Types» і відповідний запис змінюється в таблиці types бази даних SQLite.

3) випробування функції видалення типу проєкту;

У вкладці «Types» вибрати тип проєкту (наприклад, «E-Commerce») натисканням кнопки миші, натиснути кнопку «Delete» та підтвердити дію у діалоговому вікні. Очікуваний результат: Обраний тип проєкту видаляється зі списку у вкладці «Types» і відповідний запис видаляється з таблиці types. Якщо до типу прив’язані проєкти, відображається повідомлення про неможливість видалення.

4) випробування функції додавання проєкту;

Перейти до вкладки "Projects", натиснувши на її заголовок, ввести коректну URL-адресу GitHub репозиторію (наприклад, https://github.com/jigoshop/Jigoshop-eCommerce), вибрати тип проєкту зі списку та натиснути кнопку "Add".

Очікуваний результат: Новий проєкт додається до списку у вкладці "Projects" і записується в таблицю projects бази даних SQLite з відповідним type_id.

5) випробування функції редагування проєкту;

У вкладці "Projects" вибрати проєкт зі списку натисканням кнопки миші, змінити URL-адресу (наприклад, на https://github.com/example/new-repo) або обрати новий тип проєкту зі списку, після чого натиснути кнопку "Edit".

Очікуваний результат: Дані проєкту (URL або тип) оновлюються в списку у вкладці "Projects" і відповідний запис змінюється в таблиці projects бази даних SQLite.

6) випробування функції видалення проєкту;

У вкладці "Projects" вибрати проєкт зі списку натисканням кнопки миші, натиснути кнопку "Delete" та підтвердити дію у діалоговому вікні.

Очікуваний результат: Обраний проєкт видаляється зі списку у вкладці "Projects" і відповідний запис видаляється з таблиці projects бази даних SQLite.

7) випробування функції отримання метрик проєктів;

Перейти до вкладки «Checkout», натиснувши на її заголовок, обрати тип проєктів зі списку, вказати каталог для завантаження файлів, вибрати гілку та дату коміту, після чого натиснути кнопку «Checkout».

Очікуваний результат: Програма клонує репозиторії обраного типу, запускає PhpMetrics, генерує звіт у форматі CSV у вказаному каталозі. Звіт міститиме номер за порядком, назву, URL та агреговані значення метрик для кожного проєкту.