Міністерство освіти і науки України
Університет митної справи та фінансів

Факультет інноваційних технологій
Кафедра комп'ютерних наук та інженерії програмного забезпечення

Кваліфікаційна робота магістра

на тему: «Розробка сервісу для аналізу ефективності роботи працівників використовуючи ключові показники ефективності (KPI) на основі даних із Git-репозиторіїв»

Виконала: студентка групи _ІПЗ23-1зм__

Спеціальність 121 Інженерія програмного забезпечення

Решетник Катерина Олександрівна
(прізвище та ініціали)

Керівник _к.т.н., доц. Ульяновська Ю. В.__
(науковий ступінь, вчене звання, прізвище та ініціали)

Рецензент _____
(місце роботи)

(посада)

(науковий ступінь, вчене звання, прізвище та ініціали)

Дніпро – 2025

АНОТАЦІЯ

Решетник К.О. Розробка сервісу для аналізу ефективності роботи працівників використовуючи ключові показники ефективності (KPI) на основі даних із Git-репозиторіїв.

Кваліфікаційна робота виконана на здобуття освітнього ступеня магістра за спеціальністю 121 «Інженерія програмного забезпечення» в Університеті митної справи та фінансів у 2024 році.

Основним завданням роботи було створення системи яка аналізу ефективності роботи розробників на основі даних з системи контролю версій Git. У результаті досліджень та розробок було проект, який надає тімлідам зручний інструмент для оцінки роботи працівників за допомогою ключових показників ефективності (KPI). Система дозволяє додавати як локальні, так і віддалені Git-репозиторії, збирати дані про кількість комітів, частоту змін у коді, середній час між комітами, кількість доданих і видалених рядків коду тощо. Зібрані дані обробляються та візуалізуються у вигляді графіків і таблиць через веб-інтерфейс. Система надає можливість налаштування ваги метрик KPI, щоб тімліди могли адаптувати оцінювання до специфіки своїх команд.

Проєкт спрямований на підвищення прозорості та ефективності управління командами розробників, допомагає виявляти сильні сторони працівників та зони для покращення, а також сприяє оптимізації робочих процесів у команді.

Ключові слова: аналіз продуктивності працівників, сервіс, Git-репозиторії, ключові показники ефективності, управління командами.

ABSTRACTS

Reshetnyk K.O. Development of a service for analyzing employee performance using key performance indicators (KPI) based on data from Git repositories.

The qualification work was completed for a master's degree in the specialty 121 “Software Engineering” at the University of Customs and Finance in 2024.

The main task of the work was to create a system that analyzes the efficiency of developers based on data from the Git version control system.

The result of the research and development was a project that provides team leaders with a convenient tool for evaluating the work of employees using key performance indicators (KPIs). The system allows you to add both local and remote Git repositories, collect data on the number of commits, the frequency of changes in the code, the average time between commits, the number of added and deleted lines of code, etc.

The collected data is processed and visualized in the form of graphs and tables via the web interface. The system allows customizing the weight of KPI metrics so that team leaders can adapt the assessment to the specifics of their teams.

The project aims to increase transparency and efficiency of development team management, helps to identify employees' strengths and areas for improvement, and helps to optimize workflows in the team.

Keywords: employee productivity analysis, service, Git repositories, key performance indicators, team management

ЗМІСТ

ВСТУП

У сучасній індустрії інформаційних технологій, що стрімко розвивається, управлінню ефективності співробітників приділяється все більше уваги. Успіх проекту значною мірою залежить від аналізу роботи команди, виявлення сильних і слабких сторін її членів та оптимізації процесу розробки. У даному контексті ключові показники ефективності (KPI) є важливим інструментом для вимірювання роботи розробників і дозволяють приймати управлінські рішення на основі об'єктивних даних. Аналізуючи ці дані за допомогою KPI, можна створити прозору систему оцінки. Дана система необхідна для підвищення продуктивності, забезпечення якості виконання завдань і раціонального розподілу ресурсів всередині команди.

Актуальність дослідження теми зумовлена потребою в інструменті, який дозволить автоматизувати процес збору, аналізу та інтерпретації даних щодо роботи команди розробників. В умовах дедалі зростаючої конкуренції на ринку IT-продуктів організації потребують ефективних рішень для моніторингу продуктивності працівників, які базуються на прозорих, об'єктивних та кількісно вимірюваних метриках. Аналіз даних із Git-репозиторіїв дозволяє швидко оцінювати прогрес виконання завдань і виявляти проблемні моменти у роботі команди, що сприяє підвищенню ефективності процесів розробки.

Новизна дослідження полягає у використанні даних із системи контролю версій Git для розрахунку KPI, які дозволяють оцінити ефективність роботи окремих співробітників та команди загалом. Запропонований сервіс поєднує автоматизований збір даних із репозиторіїв із подальшим аналізом та візуалізацією метрик продуктивності, що робить процес оцінювання прозорим і зрозумілим.

Мета дослідження створити інструмент, який дозволить підвищити прозорість та ефективність управління продуктивністю працівників шляхом

аналізу ключових показників ефективності (KPI) на основі даних із Git-репозиторіїв.

Методи дослідження: у роботі використано методи об’єктно-орієнтованого програмування для створення системи аналізу даних, методи математичної статистики для обчислення KPI, а також технології веб-розробки для створення інтерфейсу користувача.

Об’єкт дослідження: процес оцінки ефективності роботи працівників у галузі розробки програмного забезпечення.

Предмет дослідження: інформаційна система для збору, обробки, аналізу та візуалізації даних про продуктивність працівників із використанням Git-репозиторіїв.

Практичне значення отриманих результатів: розроблений сервіс дозволяє автоматизувати процес збору та аналізу даних із Git-репозиторіїв для оцінки ефективності працівників. Використання даного сервісу дає змогу:

1. Підвищити прозорість і об'єктивність оцінки внеску кожного співробітника.
2. Оптимізувати управління командою через виявлення проблемних зон і сильних сторін.
3. Підтримувати тімлідів у прийнятті рішень, заснованих на об'єктивних даних, що сприяє поліпшенню якості виконання завдань.
4. Ефективно моніторити прогрес виконання проєктів, що дозволяє організаціям залишатися конкурентоспроможними на ринку інформаційних технологій.

Структура роботи:

Розділ 1 Постановка задачі та теоретичні основи з аналізом предметної області. У цьому розділі буде сформульовано основні цілі та задачі дослідження, виконано огляд сучасних підходів до аналізу ефективності

роботи програмного забезпечення, а також розглянуто теоретичні основи метрик якості та надійності. Буде проведено аналіз предметної області з метою виявлення актуальних проблем і можливостей для їх вирішення.

Розділ 2 Проектування сервісу для аналізу ефективності роботи. В даному розділі буде розроблено архітектуру сервісу для аналізу ефективності роботи програмного забезпечення. Буде виконано моделювання процесу збору, обробки та візуалізації даних, а також спроектовано структуру бази даних для збереження метрик. Крім того, буде обґрунтовано вибір технологій, інструментів і підходів до реалізації системи.

Розділ 3 Реалізація, тестування і розгортання системи. У даному розділі буде представлено процес розробки програмного забезпечення на основі запропонованого проекту. Буде описано реалізацію ключових компонентів сервісу, виконано тестування функціоналу та проведено оцінку якості роботи системи. Завершальною частиною стане розгортання системи, що забезпече підтримку локального та віддаленого використання, інтеграцію з базою даних та API, а також створення документації для користувачів і адміністраторів.

Робота складається з вступу, трьох розділів, висновків до кожного розділу, загального висновку, списку використаної літератури з 17 джерел, 3-х додатків. Обсяг роботи 75 сторінок, 44 рисунків та 1 таблиця.

РОЗДІЛ 1. ПОСТАНОВА ЗАДАЧІ ТА ТЕОРЕТИЧНІ ОСНОВИ З АНАЛІЗОМ ПЕРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постанова задачі та вимоги до програмного забезпечення

Метою даної кваліфікації є розробка сервісу для аналізу роботи співробітників за допомогою ключових показників ефективності (KPI) на основі даних з репозиторію Git.

Сервіс забезпечує автоматичний збір даних про діяльність розробників, розраховує відповідні показники ефективності та візуалізує результати у вигляді графіків і таблиць, що дозволяє керівникам команд оцінювати продуктивність своїх співробітників на основі об’єктивних даних.

Основна задача сервісу полягає у визначення кількості комітів, виконаних кожним співробітником, розрахунку середньої частоти комітів за обраний період, аналізу середньої кількості рядків коду, доданих або видалених комітів, оцінка кількості та типів файлів, які були змінені та візуалізація активності співробітників у часовому розрізі. Базовий веб-інтерфейс, дозволяє керівникам команди переглядати результати аналізу та отримувати загальні звіти по своїй команді.

Можливості сервісу надає розширені функціональні можливості, які забезпечують його універсальність, зручність у використанні та надійність. Програма надає автоматично підключення для збору метаданих завдяки інтеграції з Git-репозиторіями. Дозволяючи отримувати ідентифікатори комітів, інформацію про авторів, дані про змінений код відповідно до кількості доданих/видалених рядків та типи файлів. Дана інтеграція забезпечує отримання максимально точних даних без необхідності ручного введення інформації, що суттєво скорочує час і зменшує ризик помилок.

Всі зібрані данні зберігаються в реляційній базі даних, що забезпечує структурованість і цілісність, можливість виконання складних запитів для

аналізу великих обсягів інформації та масштабованість для роботи з великими репозиторіями.

Автоматизований розрахунок KPI дозволяє виконує розрахунки за допомогою автоматизованих алгоритмів, що забезпечує високу швидкість і точність, гнучкість у виборі показників та можливість адаптації під конкретні вимоги або методики оцінювання, прийняті в різних командах чи організаціях.

Для взаємодії з користувачами надається інтуїтивно зрозумілий веб-інтерфейс, який забезпечує візуалізацію даних у вигляді графіків та таблиць, а також швидкий доступ до результатів аналізу без потреби додаткових знань у програмуванні чи роботі з базами даних

Таким чином, даний сервіс є потужним інструментом, що сприяють ефективному управлінню командою розробників та забезпечує точність при оцінювані роботи.

1.2 Огляд існуючих підходів до аналізу ефективності роботи працівників

Оцінка ефективності роботи працівників здійснюється через різні підходи, кожен з яких має свої особливості та методи вимірювання [1]. Одним із основних методів є метричний аналіз, який орієнтується на кількісні показники такі, як кількість виконаних завдань або частота доданих комітів. Для цього використовуються KPI (ключові показники ефективності), що дає змогу вимірювати досягнення цілей і продуктивність працівників. Показники мають бути чітко визначеними та відповідати принципам SMART (Specific, Measurable, Achievable, Relevant, Time-bound), щоб забезпечити точність оцінки. Водночас важливо, щоб вибір і застосування таких інструментів не призводили до суб’єктивних суджень, тому коректний вибір методів збору і аналізу даних є критичним для об’єктивної оцінки продуктивності.

Поведінковий підхід до оцінки ефективності базується на вивченні взаємодії співробітників у командах і їхньому внеску в загальний результат проекту. Такий підхід дозволяє більш детально оцінити ефективність та виявити зв’язки між індивідуальною продуктивністю та командною динамікою.

Індивідуальні KPI, в свою чергу, включають персоналізовані цілі для кожного працівника, що дозволяє більш точно виміряти його особистий внесок і адаптувати підхід до специфічних умов і завдань проекту. В кінцевому рахунку, оцінка продуктивності може також здійснюватися на рівні команди, що дає змогу оцінити, як працює колектив разом для досягнення спільних цілей і визначити, які процеси потребують покращення для підвищення ефективності.

Таким чином, вибір методу оцінки ефективності роботи працівників залежить від конкретних цілей організації, її технічних можливостей і специфіки діяльності. Усі ці підходи сприяють не лише вимірюванню продуктивності, а й виявленню слабких місць в організаційних процесах, прийняттю управлінських рішень і мотивуванню працівників до досягнення кращих результатів.

1.3 Існуючі системи аналізу продуктивності на основі даних з Git-репозиторіїв

Існує кілька інструментів, які використовують дані з Git для аналізу продуктивності команд і розробників, надаючи різноманітні метрики та аналітику. Розглянепо побулярні іструменти які існіють на ринку

1. *GitAnalytics* - це інструмент для аналізу даних з Git-репозиторіїв, який допомагає розробникам та командам відстежувати ключові показники ефективності (KPI).

Можливості – аналіз частоти коммітів, оцінка активності розробників у проекті, графічне представлення змін в репозиторії в хронологічному порядку, моніторинг середнього часу між коммітами.

Недоліки данаго інструменту пов’язані з обмеженим набором метрик, що використовуються для аналізу (деякі метрики, такі як кількість доданих/видалених рядків коду, не враховуються), також бракує можливості інтеграції з іншими системами для збору додаткових даних, наприклад, тестовими системами та CI/CD.

2. *CodeScene* - це аналітичний інструмент, який поєднує дані Git з метриками коду для виявлення «зон ризику» та оцінки ефективності роботи команди. Виявляє «гарячі точки» коду, включаючи код, що часто змінюється, та потенційні проблеми, аналізує технічний борг і продуктивність команди, інтеграція з системами CI/CD для автоматизованого відстеження, моніторинг впливу скриптів на бізнес-показники.

До недоліків можна віднести, складність при розгортанні для невеликих проектів, висока вартість ліцензії, недоступність невеликим командам і стартапів, занадто багато уваги приділяється технічному боргу, що може бути надмірним для простих проектів.

Порівняння функціональності та недоліків (Таблиця1.1)

Таблиця1.1

Параметр	**GitAnalytics**	**CodeScene**
Збір даних	Основні метрики (коміти, активність)	Глибокий аналіз коду й командної роботи
Графічні інтерфейси	Обмежений функціонал	Інтуїтивно зрозумілі графіки
Інтеграція	Відсутня	З CI/CD
Вартість	Безкоштовний/недорогий	Дорога ліцензія

Придатність для малих проєктів	Висока	Складно налаштувати

Порівняння цих інструментів дає змогу оцінити їхні переваги та недоліки в контексті цілей даної системи. Наприклад, GitAnalytics є доступним і простим у використанні, але має обмеження в наборі метрик. CodeScene, з іншого боку, надає глибокий аналіз коду та командної роботи, але його висока вартість та складність налаштування роблять його менш привабливим для малих команд.

Обґрунтування створення нової системи

Доступність для малого та середнього бізнесу. Існуючі рішення часто або занадто прості, або надмірно складні й дорогі. Створення нової програма може запропонувати доступний інструмент із оптимальним набором функцій, які підходять для невеликих команд і стартапів.

Кастомізація показників**.** Більшість наявних систем обмежуються стандартними метриками, які не завжди відповідають потребам користувачів. В новій системі буде додана можливість обирати показники ефективності (KPI), які будуть релевантними для кожного конкретного проєкту.

Простота у використанні. Сучасних інструментів надто складні у використанні, особливо для команд без технічного досвіду. Цей проект зосереджений на створенні інтуїтивно зрозумілому і зручному інтерфейсі для всіх користувачів.

Орієнтація на продуктивність команди [3]. Дана система не обмежується оцінкою індивідуальної активності розробників, буде враховуватись командна взаємодія та пропонуватись рекомендації, які допоможуть покращити працездатність усього колективу.

Підтримка освітніх і дослідницьких потреб. Оскільки цей сервіс створюється в рамках наукового дослідження та відповідає стандартам спеціальності, він є частиною процесу здобуття освітнього ступеня магістра. Кінцевий результат може привернути увагу освітніх установ, зацікавлених в аналізі ефективності командної роботи та проведенні досліджень у сфері командного програмування.

Таким чином, дана розробка проекту заповнює нішу між занадто простими та надто складними інструментарієм, пропонуючи функціональність, яка адаптується до реальних потреб користувачів.

1.4 Методологія розробки сервісу

Розробка сервісу орієнтована на використання ефективних метрик для оцінки продуктивності працівників і якості роботи в рамках програмних проектів [2]. Процес аналізу цих метрик дозволяє забезпечити об’єктивну оцінку роботи команди, виявити слабкі місця і сприяти прийняттю обґрунтованих управлінських рішень.

Аналіз предметної області починається з дослідження структури Git-репозиторіїв, оскільки саме дані, що зберігаються у репозиторіях, є основними для оцінки ефективності розробників. До основних метрик, що відображають кількісні показники діяльності працівників, належать кількість комітів, середня кількість доданих і видалених рядків коду, час між комітами та кількість змінених файлів. Всі ці метрики дозволяють оцінити обсяг виконаної роботи, частоту змін та ритм роботи розробників.

Вибір архітектури сервісу базується на мікросервісному підході, що забезпечує масштабованість і гнучкість при роботі з великою кількістю даних. Це дає змогу інтегрувати різні сервіси для збору, обробки та зберігання метрик без порушення стабільності системи.

Технологічний стек, що використовуватиметься в розробці, включає Spring Boot для серверної частини, MySQL для зберігання даних, а також HTML/CSS/JS для реалізації фронтенду, що дозволяє створити зручний інтерфейс для взаємодії з користувачем.

Методи розрахунку KPI включають алгоритми для аналізу активності розробників, визначення частоти комітів, а також змін у коді. Для цього будуть використовуватися спеціалізовані сервіси, такі як localGitService і gitHubGitService, які перевіряють існування репозиторіїв і збирають метрики для збереження в базі даних. Дані сервіси, також надають узагальнені дані за ID проекту, включаючи середні значення та активність авторів.

Командний і персональний аналіз надає детальну інформацію про внесок кожного працівника, дозволяючи оцінити результати роботи на рівні індивідуума та команди в цілому.

Завдяки впровадженню зазначених кількісних і якісних показників, створюється система, яка забезпечує детальний аналіз ефективності роботи працівників, сприяє прийняттю ефективних управлінських рішень та підвищує якість розробки програмного забезпечення [17].

1.5 Аналіз методів побудови системи

Для створення системи аналізу ефективності роботи співробітників на основі метрик з Git-репозиторіїв розробники застосовують різноманітні технології та підходи [4]. Ці методи дозволяють автоматизувати процес збору даних, їх обробку та візуалізацію, що, в свою чергу, допомагає тімлідам об'єктивно оцінювати продуктивність команди.

Основні способи побудови системи для збору та аналізу KPI

Інтеграція з Git-репозиторієм є ключовим елементом при створенні даного проекту за для того, щоб збирати дані про активність розробників. Завдяки цій інтеграції система автоматично отримує метадані про коміти, авторів, типи використованих файлів у коді, доданих та видалених рядків та інші важливі показники, які використовуються для розрахунку KPI, що дозволяє тімлідам отримувати точну і своєчасну інформацію без потреби в ручному зборі даних, що значно підвищує ефективність і зручність використання сервісу.

Основні підходи до інтеграції з Git-репозиторієм:

JGit – бібліотека для роботи з Git, написана на Java. Вона надає можливість здійснювати операції з Git-репозиторіями (класти, комітити, отримувати інформацію про репозиторій) без необхідності використовувати зовнішні інструменти Git. Завдяки JGit можна програмно працювати з репозиторіями Git на сервері чи локальному комп'ютері, що дозволяє автоматично отримувати метадані з репозиторіїв.

GitHub API – дозволяє отримати доступ до інформації про репозиторії, коміти, pull-реквести, гілки та інші елементи з онлайн-платформи GitHub. Взаємодія з цими API здійснюється за допомогою HTTP-запитів, де можна отримати наступну інформацію: коміти, автори та їх активність, статистика по репозиторію. Інтеграція з GitHub API [8] дозволяє швидко отримати доступ до репозиторіїв і ефективно збирати необхідні дані для подальшого аналізу KPI.

Збереження даних у базу даних

У даному проекті для забезпечення надійності, ефективності та зручності взаємодії з даними використовуються реляційні бази даних, такі як MySQL.

Для зручності роботи з об'єктами Java та автоматизації взаємодії з базою даних застосовується Hibernate (Java Persistence API - JPA). Це дозволяє:

1. Маппінг об'єктів на таблиці – за допомогою анотацій JPA можна зв'язувати Java-об'єкти з таблицями бази даних.
2. Автоматичне керування зв'язками між таблицями – Hibernate автоматично керує зовнішніми ключами та забезпечує цілісність даних.
3. Підтримка транзакцій – Hibernate дозволяє зручно управляти транзакціями для забезпечення консистентності даних.

Таким чином, використання Hibernate для роботи з базою даних дозволяє знизити складність взаємодії з реляційною базою даних, спростити зберігання та маніпуляції даними, а також полегшити масштабування системи в майбутньому.

Розрахунок KPI для аналізу ефективності роботи співробітників

Для автоматизованого розрахунку KPI (ключових показників ефективності) в системі аналізу діяльності розробників використовуються спеціалізовані алгоритми, що ґрунтуються на метках активності, зібраних з Git-репозиторіїв. Це дозволяє об'єктивно оцінити ефективність роботи співробітників, використовуючи фактичні дані про їхні коміти, зміни в коді, активність і взаємодію з проектами [9].

Основний показник який використовується для розрахунку KPI:

Середня кількість рядків коду, доданих або видалених
Цей показник відображає активність розробника з точки зору обсягів змін у коді. Він може бути розрахований як середнє значення кількості рядків коду, доданих або видалених за певний період (наприклад, за тиждень чи місяць). Алгоритм для розрахунку:

1. Збір всіх комітів, зроблених конкретним розробником, за вказаний період.
2. Підрахунок рядків, доданих і видалених у кожному коміті.
3. Розрахунок середнього значення по цих даних.

Розрахунок KPI в даній системі базується на використанні даних з Git-репозиторіїв для оцінки різних аспектів роботи розробників. Алгоритми, які автоматично обчислюють ці показники, дозволяють тімлідам та керівникам проектів отримувати об'єктивні та деталізовані звіти про ефективність роботи команди, що є важливим для оптимізації процесу розробки, покращення результатів і ухвалення обґрунтованих рішень.

Візуалізація даних через веб-інтерфейс

Веб-інтерфейс дозволяє тімлідам зручно переглядати результати аналізу, налаштовувати часові межі для розрахунку KPI та отримувати звіти. Використовуються сучасні веб-технології (HTML, CSS, JavaScript), а також бібліотеки для візуалізації даних, такий як Chart.js, для створення графіків і таблиць, що чітко відображають результати аналізу.

Такі технології та методи допомагають створити гнучку, масштабовану та ефективну систему для аналізу роботи співробітників, яка дозволяє автоматизувати збору даних і забезпечити точність та об'єктивність в оцінці ефективності роботи команди.

1.6 Висновоки до першого розділу

У першому розділі було сформульовано завдання розробки сервісу для аналізу ефективності роботи команди розробників на основі даних з Git репозиторію. Проведено огляд існуючих підходів до оцінки ефективності, визначено сильні та слабкі сторони сучасного рішення та ключові показники.

Також проаналізовано основні технології інтеграції з Git-репозиторіями, зокрема через локальний аналіз та GitHub API, що дозволяють отримувати точні дані про активність розробників.

Розроблена методологія охоплює аналіз кількісних та якісних показників і враховує архітектуру, функціональність та вимоги до якості сервісу. Таким чином, аналіз предметної області окреслив шляхи вирішення задачі на основі сучасних технологій, яка лягла в основу подальшого проектування та впровадження системи.

РОЗДІЛ 2. ПРОЕКТУВАННЯ СЕРВІЗУ ДЛЯ АНАЛІЗУ ЕФЕКТИВНОСТІ РОБОТИ

2.1 Архітектурний дизайн системи

При реалізації системи було обрано клієнт-серверний підхід, який дозволяє розподіляти обробку даних між серверною та клієнтською частинами [5]. Така архітектура забезпечує високу гнучкість і масштабованість системи, оскільки серверна частина відповідає за обробку та збереження даних, а клієнтська частина — за зручний доступ користувачів до результатів аналізу через веб-інтерфейс. Клієнт-серверна модель сприяє простоті інтеграції зовнішніх сервісів, таких як Git API, а також дає змогу централізовано управляти даними та доступом до них.

Задля ефективної взаємодії між компонентами було обрано REST API, яке дозволяє клієнтській частині здійснювати взаємодію з сервером через HTTP-запити. Це рішення дозволяє обробляти запити від користувачів та передавати дані про ефективність роботи команди в реальному часі.

Система складається з кількох основних компонентів, кожен з яких виконує конкретні функції в процесі збору, обробки та виведення даних:

1. Клієнтська частина (UI) веб-інтерфейс, надає тімліду можливість взаємодіють з системою, вводячи дані про проект, переглядати метрики ефективності роботи команди, аналізувати статистику та отримувати звіти. Всі ці функції доступні через інтуїтивно зрозумілий та зручний інтерфейс.
2. Серверна частина (API) реалізована на основі фреймворка Spring Boot [6] і відповідає за обробку запитів клієнта, взаємодію з базою даних, а також збирання та обробку даних з Git-репозиторіїв. За допомогою API здійснюється доступ до метрик, збережених у базі даних, а також виконуються операції з аналізу даних.

3. База даних зберігаєь метрик використовуються реляційні бази даних. У базі зберігаються інформація про коміти, кількість змін у коді, активність розробників та інші показники, що дозволяють оцінити ефективність роботи команди. Всі дані організовано в таблицях, де є зв'язки між проектами, комітами та розробниками.
4. Зовнішні сервіси забезпечує збір даних з Git-репозиторіїв використовуються зовнішні GitHub API [8] для інтеграції з репозиторіями.

Взаємодія між компонентами системи відбувається за допомогою чітко визначених інтерфейсів та протоколів. Клієнтська частина відправляє запити на сервер через REST API, отримуючи від нього необхідну інформацію у вигляді метрик або звітів. Сервер здійснює обробку запитів, звертається до бази даних для отримання збережених метрик і, в разі необхідності, взаємодіє з зовнішніми сервісами для збору даних з Git-репозиторіїв.

Система забезпечує доступ до даних за допомогою простих запитів: для отримання загальних метрик проекту, а також для отримання персональних даних про активність конкретних розробників. Результати аналізу передаються клієнтській частині, де вони виводяться у вигляді таблиць або графіків, що дозволяють тімліду отримати детальну інформацію про ефективність роботи команди.

Для забезпечення безпеки даних було реалізовано автентифікацію користувачів та захист API за допомогою токенів, що гарантує доступ до метрик лише авторизованим користувачам. Також для підвищення ефективності системи використовується кешування даних, що дозволяє зменшити навантаження на сервер і забезпечує швидший доступ до необхідних метрик.

Цей підхід забезпечує високу ефективність, зручність використання та безпеку при зборі та аналізі метрик ефективності роботи команди, що є основною метою розробленої системи.

2.2 Проектування базових компонентів

Проектування базових компонентів системи включає визначення ключових модулів, які забезпечують її функціонування [6]. Основні компоненти системи поділяються на кілька основних етапів: збір даних, обробка метрик, збереження в базі даних та виведення результатів.

Модуль збору даних – відповідає за отримання інформації з Git-репозиторіїв. Для цього використовуються API Git, через які отримуються дані про коміти, авторів, зміни в коді та інші важливі метрики.

Модуль обробки метрик – займається аналізом отриманих даних, обчисленням різноманітних показників ефективності, таких як середня кількість комітів, зміни в коді, активність авторів тощо. Обробка метрик дозволяє отримати детальну інформацію про роботу команди та кожного розробника окремо.

Модуль збереження в базі даних – відповідає за структурування та збереження отриманих і оброблених даних у базі даних. Для цього використовуються відповідні таблиці та зв'язки між ними, що забезпечують ефективне зберігання та подальшу обробку інформації.

Модуль виведення результатів – надає користувачеві доступ до збережених метрик, відображаючи їх у зручному вигляді через інтерфейс системи. Дані можуть бути представлені у вигляді таблиць, графіків чи інших візуальних елементів, що дозволяє зручно оцінити ефективність роботи команди.

Взаємодія між модулями здійснюється через API та протоколи обміну даними. Використовуються стандартизовані формати, такі як JSON для передачі даних між компонентами. Застосування чітко визначених API дозволяє забезпечити гнучкість системи та спрощує інтеграцію з іншими сервісами в майбутньому.

2.3 Життєвий цикл та побудова моделі даних

Життєвий цикл розробки системи описує етапи, через які проходить проект, починаючи від планування та аналізу до підтримки та вдосконалення. Кожен етап життєвого циклу тісно пов'язаний з наступним і має на меті забезпечити успішну реалізацію продукту. Зокрема, важливу роль на етапах проектування та розробки відіграє побудова моделі даних, яка забезпечує структуру для ефективного зберігання та обробки даних, необхідних для подальшого аналізу і ухвалення рішень на основі отриманих метрик.

Рисунок 2.1 – Циклічна модель життєвого циклу

На (Рис. 2.1), життєвий цикл розробки системи має циклічну природу, де кожен етап тісно взаємопов'язаний з іншими, що дозволяє системі постійно вдосконалюватися і оновлюватися на основі зворотного зв'язку та нових вимог. Рисунок відображає основні етапи життєвого циклу розробки системи, які починаються з планування та аналізу вимог і продовжуються через проектування, розробку, тестування, впровадження та підтримку. Кожен етап циклічно взаємодіє з наступним, забезпечуючи ефективну реалізацію та оновлення продукту. Під час етапу проектування і розробки важливу роль відіграє створення моделі даних для оптимізації обробки і зберігання інформації.

Модель даних для системи аналізу ефективності роботи команди розробників організована на основі ключових метрик, що дозволяють оцінити продуктивність. Схема бази даних розроблена таким чином, щоб забезпечити ефективне зберігання та доступ до даних, необхідних для аналізу. Описані сутності, їх атрибути та зв'язки між ними визначаються з урахуванням бізнес-логіки та вимог до швидкості запитів.

У базі даних визначені кілька основних сутностей

1. Employees (employees): містить інформацію про працівників команди.
2. Project Metrics (project_metrics): зберігає метрики проектів, такі як середня кількість доданих і видалених рядків коду.
3. Commit Metrics (commit_metrics): зберігає інформацію про коміти.

Teams (teams): містить інформацію про команди, їх членів та проекти, над якими працюють.

Зв'язки між сутностями в базі даних організовуються так, щоб зберегти логіку роботи команди та проектів (Рис.2.2)

Employees ↔ Commit Metrics: один працівник (автор) може зробити кілька комітів. Зв'язок типу один до багатьох.

Project Metrics ↔ Commit Metrics: один проект може мати кілька комітів з відповідними метриками. Зв'язок один до багатьох.

Teams ↔ Employees: одна команда складається з кількох членів, і кожен член є працівником. Зв'язок один до багатьох.

Teams ↔ Project Metrics: кожна команда працює над конкретним проектом і має відповідні метрики проекту.

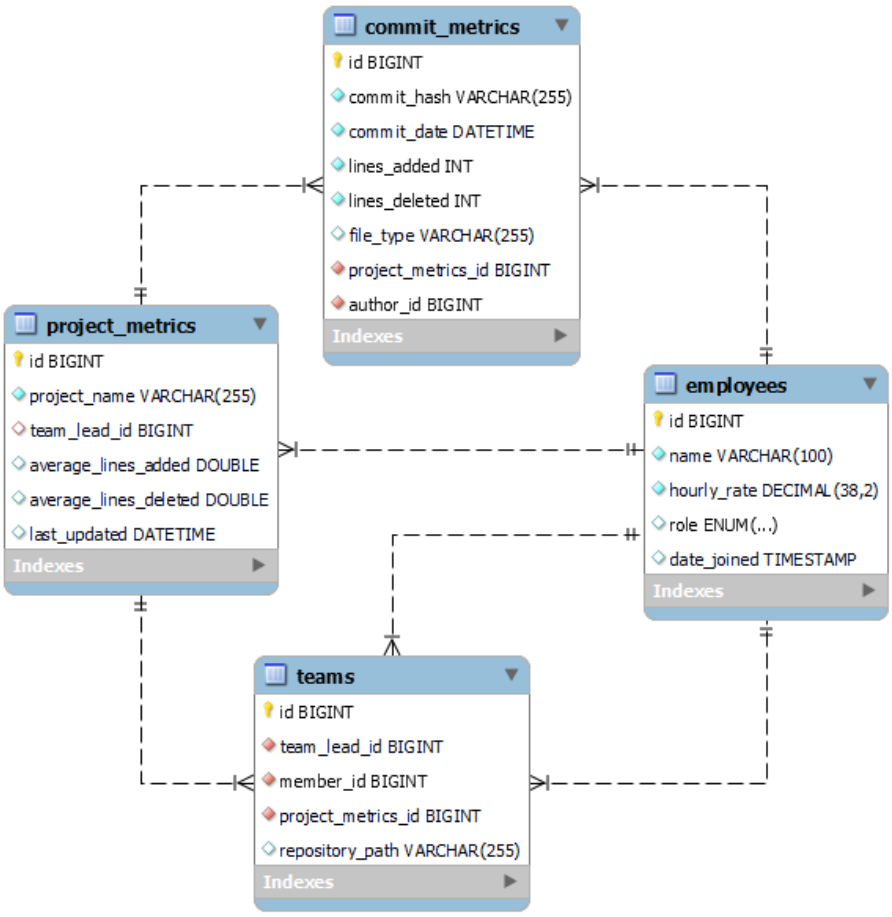


Рисунок 2.2 – ER-діаграма зв'язки між сутностями в базі даних

Розподіл даних за таблицями забезпечить ефективність запитів і зберігання даних:

- employees: зберігає дані про працівників (ім'я, роль, ставка, дата вступу).
- project_metrics: містить метрики проектів (середня кількість змінених рядків).

- commit_metrics: зберігає дані про коміти (хеш, автор, час, метрики проекту).
- teams: описує команди, їх учасників, проекти і репозиторії.

Таким чином, життєвий цикл розробки системи і побудова моделі даних взаємопов'язані: кожен етап життєвого циклу потребує ефективної організації даних, що, в свою чергу, забезпечує стабільну роботу та зручність у використанні системи.

2.4 Інтеграція зовнішніх сервісів та API

Інтеграція з зовнішніми сервісами та API є критично важливою складовою архітектури системи, оскільки вона дозволяє отримувати актуальні дані з Git-репозиторіїв, а також використовувати сторонні інструменти для покращення аналізу коду [14]. Це підвищує точність та ефективність оцінки продуктивності команди розробників, оскільки автоматизує процес збору даних та дозволяє системі працювати з великими обсягами інформації.

Використання Git API для отримання даних із репозиторіїв (Таблиця 2.1)

Одним із основних джерел даних для аналізу ефективності роботи команди є Git-репозиторії. Через Git API система може автоматично отримувати різноманітні дані про коміти, авторів, зміни в коді та інші метрики. Це дозволяє забезпечити точність та своєчасність збору інформації, що необхідна для обчислення показників продуктивності.

Таблиця 2.1

Категорія	**Дані/Функціональність**	**Інструменти/Сервіси**
Коміти	- Хеш коміту - Дата та час коміту - Автор коміту - Змінені файли	Git API (GitHub)

Метрики проектів	- Кількість доданих та видалених рядків - Частота комітів - Середній час між комітами - Статистика по кожному автору	Git API
Інформація про гілки	- Список гілок - Історія змін - Статуси злиття	Git API
Збір даних з репозиторіїв	- Інтеграція з різними Git платформами для отримання даних безпосередньо з репозиторіїв	GitHub API
Реальний час/Актуальність	- Автоматичне оновлення даних в реальному часі на основі комітів, тестів та змін в коді	Git API

Таким чином, інтеграція зовнішніх сервісів та API створює необхідну основу для побудови масштабованої та ефективної системи моніторингу, що дозволяє отримувати точні та своєчасні дані для аналізу продуктивності команди.

2.6 Підготовка до реалізації

Перед початком реалізації системи необхідно визначити основні інструменти та середовище виконання, а також узгодити ключові принципи, що забезпечать ефективну розробку, масштабованість та високу якість системи [17]. Цей етап є важливим для вибору технологій, що найбільше відповідають вимогам проекту, а також для створення чіткої стратегії розвитку та підтримки системи в майбутньому.

Першим кроком було обрано набір технологій, який дозволяє ефективно обробляти великі обсяги даних, забезпечує масштабованість та високу продуктивність, а також підтримує зручність розробки та подальшого супроводу системи.

Технології для бекенд частини (Рис. 2.3)

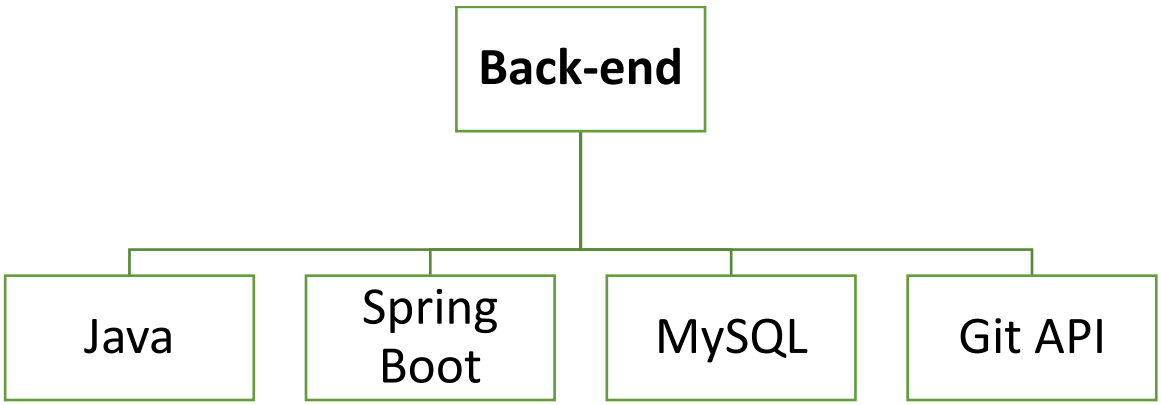


Рисунок 2.3 – Технології для серверної частини

Spring Boot було обрано як основну платформу для розробки внутрішньої частини системи, це популярний фреймворк на основі Java для швидкого створення масштабованих і надійних веб-додатків. Завдяки можливості інтеграції з різними базами даних, API та іншими сервісами, Spring Boot [6] полегшує розробку і тестування, а також забезпечує високу продуктивність і надійність системи.

СУБД для зберігання даних було обрано реляційну базу даних MySQL, яка підходить для обробки великих обсягів структурованих даних і пропонує високу швидкість запитів та зручний інтерфейс для маніпулювання даними за допомогою SQL [7]. Причини, чому було обрано MySQL це через її популярність, стабільність та підтримку у сфері веб-розробки.

Git API забезпечує інтеграцію з репозиторіями та збору даних про коміти та інші метрики, це дозволяє системі взаємодіяти з GitHub та іншими хмарними платформами для доступу до репозиторіїв і автоматичного збору необхідної інформації.

Технології для фронтенд частини (Рис. 2.4)

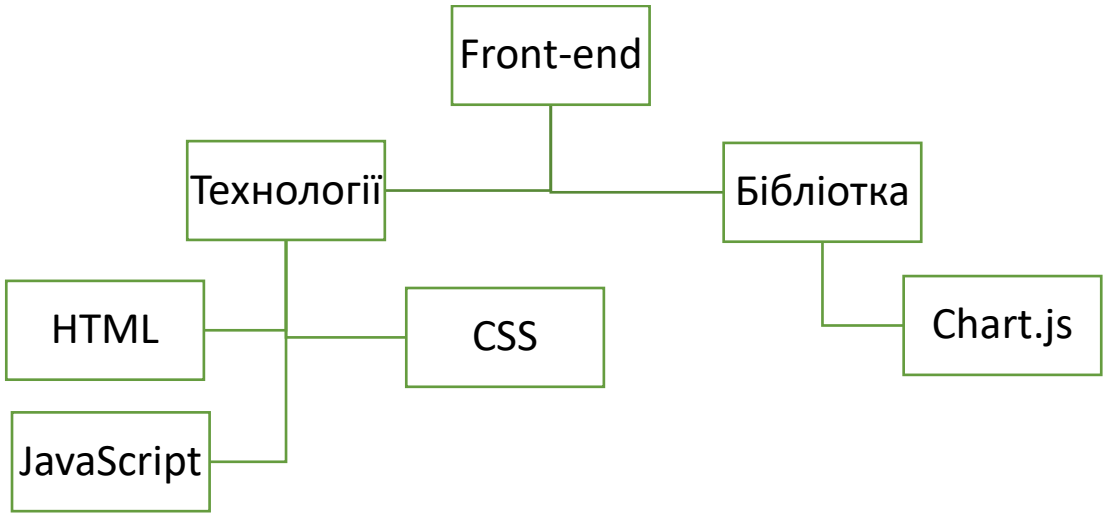


Рисунок 2.4 – Технології для візуального інтерфейсу

JavaScript, HTML, CSS забезпечують простоту у реалізації для фронтенд-частини. Система використовує базові веб-технології JavaScript, HTML та CSS, що дозволяє створити легкий та адаптивний інтерфейс, який буде зручний для користувачів і забезпечить високу швидкість роботи.

Chart.js бібліотека, яка дозволяє візуалізувати метрик та створювати графіки. Вона дозволяє створювати інтерактивні графіки та діаграми, що робить результати аналізу зрозумілими та наочними для користувача.

Під час підготовки до реалізації системи були визначені ключові принципи, що забезпечують масштабованість, надійність і якість програмного продукту. Узгодження цих принципів дозволило створити фундамент для побудови сучасної та ефективної системи.

Масштабованість

Система була спроектована з урахуванням потенційного зростання обсягу даних, таких як кількість репозиторіїв, комітів та користувачів. Архітектура передбачає підтримку горизонтального та вертикального масштабування бази даних, API та інших компонентів [10]. Використання модульного підходу в архітектурі дає змогу легко розширювати функціональність системи без впливу на її продуктивність. Зокрема, обрані технології, такі як Spring Boot та MySQL, мають вбудовані інструменти для оптимізації роботи з великими обсягами даних.

Модульність та розширюваність

Кожен компонент системи був розроблений як окремий модуль із чітко визначеними інтерфейсами для взаємодії. Це дозволяє у майбутньому інтегрувати нові функції чи адаптувати існуючі без ризику порушення роботи інших частин. Наприклад, функціональність аналізу метрик Git реалізована у вигляді окремого модуля, який можна розширити або замінити за необхідності.

Надійність та безпека

Забезпечення безпеки та стійкості до збоїв стало пріоритетом при виборі технологій та проектуванні системи. Для захисту даних передбачено використання протоколів HTTPS, а також механізмів авторизації та аутентифікації користувачів за допомогою Spring Security. Система була протестована на стійкість до високих навантажень, що гарантує безперебійну роботу навіть за умови інтенсивного використання.

Автоматизація та тестування

Для забезпечення високої якості розробки були впроваджені стратегії автоматизованого тестування. Це включає:

Юніт-тестування для перевірки роботи окремих компонентів.

Інтеграційне тестування для оцінки взаємодії між модулями.

Стрес-тестування для перевірки продуктивності під великими навантаженнями.

Документація

Особлива увага приділялася створенню документації. Технічна документація описує структуру системи, принципи роботи ключових компонентів та API для інтеграції з іншими сервісами. Для кінцевих користувачів створено інструкції щодо використання функціоналу системи, що спрощує її освоєння та впровадження.

Стратегії реалізації

Впровадження системи було сплановано поетапно. Початковий етап передбачав реалізацію базової функціональності, такої як збирання метрик із Git-репозиторіїв та їх збереження у базі даних. Подальші етапи включали інтеграцію з зовнішніми сервісами, автоматизацію збору даних та впровадження засобів візуалізації метрик. Кожен етап супроводжувався ретельним тестуванням для забезпечення стабільності та відповідності вимогам.

2.7 Висновок до другого розділу

Даний розділ був присвячений проектуванню сервісу для аналізу ефективності роботи розробників, і в процесі проектування була врахована циклічна модель життєвого циклу розробки системи. Всі етапи проектування, починаючи від планування та аналізу до підтримки та вдосконалення, були тісно пов'язані між собою, що дозволяє забезпечити безперервний розвиток системи і її здатність адаптуватися до нових вимог.

На першому етапі було розроблено архітектурний дизайн системи, що передбачає модульну структуру з чітко визначеними інтерфейсами між компонентами. Такий підхід забезпечує простоту розширення та підтримки системи в майбутньому.

Проектування базових компонентів включало визначення ключових функціональних модулів, які виконуватимуть основні задачі сервісу. Особливу увагу було приділено забезпеченню ефективної взаємодії між компонентами та їхній гнучкості для адаптації до змін у вимогах.

Життєвий цикл системи в рамках цього проекту передбачає постійну оцінку ефективності та внесення змін відповідно до нових вимог і зворотного зв'язку. Усі етапи — від проектування до підтримки та вдосконалення — забезпечують високу якість програмного забезпечення і допомагають постійно вдосконалювати його відповідно до потреб користувачів.

Модель даних була побудована з урахуванням потреб у зборі, зберіганні та обробці великих обсягів інформації. Реляційна база даних MySQL і оптимальні структури даних забезпечують ефективність роботи з репозиторіями, комітами та іншими метриками, що дозволяє проводити детальний аналіз продуктивності.

Інтеграція зовнішніх сервісів та API значно розширює функціональність системи, дозволяючи отримувати дані з Git-репозиторієв за допомогою

існуючих рішень. Використані інструменти та підходи гарантують надійність і швидкість обробки запитів, що є важливими для ефективної роботи системи.

На етапі підготовки до реалізації були визначені ключові принципи, які забезпечать високу якість розробки та масштабованість системи. Тестування, безпека та документація стали важливими аспектами цієї підготовки.

Таким чином, проектування сервісу не лише створило міцну основу для подальшої реалізації, але й визначило циклічний підхід до життєвого циклу системи, що дозволяє постійно удосконалювати і адаптувати систему відповідно до змінюваних вимог та умов. Вибрані підходи, технології та принципи дозволяють розробити ефективний інструмент для аналізу продуктивності, що відповідає сучасним вимогам до якості програмного забезпечення.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ, ТЕСТУВАННЯ І РОЗГОРТАННЯ СИСТЕМИ

3.1 Реалізація модуля збору та обробки даних

Процес збору та обробки даних з Git-репозиторіїв у рамках проекту став важливою складовою системи. Основна задача полягала у створенні механізмів для ефективного отримання інформації про коміти, авторів, змінені файли та інші метрики, які дозволяють аналізувати внесок працівників у проекти.

Алгоритми парсингу даних із Git-репозиторіїв

У ході розробки було реалізовано два окремі підходи для роботи з різними типами репозиторіїв, локальними та віддаленими.

LocalGitService – цей сервіс створено для роботи з локальними Git-репозиторіями, які зберігаються на сервері. За допомогою бібліотеки JGit здійснюється доступ до репозиторію, збирається інформація про коміти, авторів, файли, а також кількість змінених рядків. Наприклад, якщо у локальному репозиторії зберігаються дані про кілька тисяч комітів, сервіс швидко отримує всі необхідні метрики, забезпечуючи продуктивність.

GitHubGitService – взаємодіє з віддаленими репозиторіями на платформі GitHub через GitHub API. Цей сервіс дозволяє отримувати інформацію про коміти, гілки, авторів та інші метрики через стандартні запити до API. Використовуючи бібліотеку GitHub REST API, система може отримувати і обробляти відповіді у форматі JSON, витягаючи ключові дані.

Алгоритм парсингу будується за таким принципом (Рис. 3.1) локальні репозиторії обробляються за допомогою JGit, тоді як віддалені репозиторії — через запити до API GitHub. Отримані дані перевіряються, щоб зібрати лише інформацію, що відповідає заданим умовам. Аналіз змін у файлах

здійснюється, включаючи тип файлів, кількість доданих та видалених рядків. Це дозволяє визначити обсяг і характер роботи, виконаної кожним автором.

Рисунок 3.1 – Алгоритм парсингу даних

Авторизація та безпека

Для взаємодії з GitHub API використовується авторизація за допомогою особистих токенів доступу (Personal Access Tokens), що дозволяє виконувати запити до API, дотримуючись політик безпеки GitHub. Токени зберігаються у зашифрованому вигляді в конфігураційному файлі, що унеможливлює їх компрометацію.

Збереження отриманих даних у базу даних

Для збереження зібраної інформації в базі даних були розроблені відповідні моделі даних та репозиторії. Основними сутностями є:

- CommitMetric
- ProjectMetrics
- Employee
- Team

Сутність CommitMetric відповідає за зберігання метрик, пов'язаних із кожним комітом у проекті. Вона включає в себе основну інформацію про коміти, такі як хеш коміту, автор, дата коміту, кількість доданих і видалених

рядків, а також назву проєкту і тип змінених файлів (Рис. 3.2), ця сутність також має зв'язок із ProjectMetrics, що дозволяє відслідковувати, до якого проєкту належить коміт.

```
3 •  select*from commit_metrics;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	commit_hash	author_id	commit_date	lines_added	lines_deleted	project_name	file_type	project_metrics_id
1	878317c9a4a70d9b63ee649a5d5d274a4188ba91	1	2024-06-04 13:43:35	3	0	gs-gradle	AsciiDoc	1
2	93bb7cee04456b8c731bc6aa204111ed0b95e3c9	2	2021-06-01 23:49:44	44	0	gs-gradle	Unknown	1
3	64ceba6d0f1d691f5bd863c3c32724530e421c78	2	2021-06-01 23:44:05	0	9	gs-gradle	Unknown	1
4	15db19654c6da64c61135a64e2a8d01910e41173	2	2021-04-27 02:36:52	3	3	gs-gradle	AsciiDoc	1
5	cd0ab0e2f1970e124f4f5bcde80f6c5ecb28b944	3	2020-12-14 17:41:28	41	19	gs-gradle	AsciiDoc	1
6	0e5debb0d144cb5b787d30ab78cc866c65695e36	2	2020-11-02 17:43:54	202	16	gs-gradle	Unknown	1
7	aaa279ee5642dba1409830edda3575960ac2cfef	4	2019-11-26 21:08:19	30	30	gs-gradle	Java	1
8	66bf5b5618f3ce5dd13f773d2e2c5b3c746b79a6	4	2019-11-19 17:43:40	18	17	gs-gradle	Unknown	1
9	94db591accb686c3a9352fa340a270eca994911c	4	2019-08-12 17:07:45	1	1	gs-gradle	Unknown	1

Рисунок 3.2 – Таблиця сутності CommitMetric

Сутність ProjectMetrics використовується для зберігання агрегованих метрик проєктів. Вона містить дані, які дозволяють аналізувати продуктивність та динаміку внеску в проект таку, як середню кількість доданих і видалених рядків у комітах (Рис. 3.3).

```
3 •  select*from project_metrics;
```

Result Grid | Filter Rows: | Edit: | Export/Import:

id	project_name	average_lines_added	average_lines_deleted	last_updated
1	gs-gradle	20.232876712328768	12.212328767123287	2024-12-23 18:45:05

Рисунок 3.3 – Таблиця сутності ProjectMetrics

Сутність Employee описує працівників, які вносять зміни до репозиторію. Вона включає ідентифікаційні дані про працівника, його роль, дату приєднання до проекту, а також погодинну оплату (Рис. 3.4).

```
3 •  select*from employees;
```

Result Grid | Filter Rows: | Edit:

id	name	hourly_rate	role	date_joined
1	GitHub	40	TeamLead	2024-12-23
2	Greg L. Turnquist	30	Developer	2024-12-23
3	Jay Bryant	30	Developer	2024-12-23
4	Greg Turnquist	30	Developer	2024-12-23
5	Spring Operator	30	Developer	2024-12-23
6	Marcin Grzejszczak	30	Developer	2024-12-23

Рисунок 3.4 – Таблиця сутності Employee

Сутність Team використовується для зберігання інформації про команди, що працюють над проектами. Вона містить інформацію про лідера команди, її членів, назву проекту і шлях до репозиторію (Рис. 3.5).

```
3 •  select*from teams;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	team_lead_id	member_id	project_name	repository_path	project_metrics_id
1	1	1	gs-gradle	C:\Users\Kathryn\ProjectGit\Git_6\gs-gradle\.git	1
2	1	2	gs-gradle	C:\Users\Kathryn\ProjectGit\Git_6\gs-gradle\.git	1
3	1	3	gs-gradle	C:\Users\Kathryn\ProjectGit\Git_6\gs-gradle\.git	1
4	1	4	gs-gradle	C:\Users\Kathryn\ProjectGit\Git_6\gs-gradle\.git	1
5	1	5	gs-gradle	C:\Users\Kathryn\ProjectGit\Git_6\gs-gradle\.git	1
6	1	6	gs-gradle	C:\Users\Kathryn\ProjectGit\Git_6\gs-gradle\.git	1

Рисунок 3.5 – Таблиця сутності Team

Розроблена структура дозволяє ефективно обробляти дані, створювати звіти та аналізувати продуктивність працівників.

3.2 Розрахунок KPI працівників та визначення лідера проекту

У цьому підрозділі розглядається механізм оцінки ефективності працівників та автоматичного визначення лідера проекту на основі даних з

комітів. Цей процес є важливим для оцінки продуктивності команди та коригування ролей у проекті [13].

Основним критерієм ефективності працівника є його участь у розробці проекту, що вимірюється через кількість доданих рядків коду в комітах. Для цього в проекті використовується метод calculateEmployeeEfficiency, який збирає всі коміти працівника через commitMetricRepository.findByAuthorId, потім визначається кількість рядків, доданих у кожному коміті, і розраховується середнє значення рядків на один коміт, що виступає як показник ефективності працівника (ДОДАТОК А).

Згідно з результатами ефективності, за допомогою методу updateEmployeeHourlyRate коригується ставка працівника. Якщо ефективність перевищує 50%, ставка збільшується на 10 одиниць, а якщо менше 20% — зменшується на 5 одиниць (ДОДАТОК Б).

Автоматизований процес визначення лідера проекту ґрунтується на першому коміті. Якщо лідер ще не призначений, то перший автор коміту отримує роль TeamLead. У методі saveCommitMetrics автоматично призначається лідер, який стає першим працівником, що зробив коміт, якщо ця роль ще не визначена (ДОДАТОК В).

Працівникам призначаються ролі Developer або TeamLead на основі їхньої участі в проекті. Лідер отримує вищу ставку 50$, а решта учасників — стандартну 30$ (Рис. 3.6).

	id	name	hourly_rate	role	date_joined
▶	1	GitHub	50	TeamLead	2024-12-27
	2	Greg L. Turnquist	30	Developer	2024-12-27
	3	Jay Bryant	30	Developer	2024-12-27
	4	Greg Turnquist	30	Developer	2024-12-27
	5	Spring Operator	30	Developer	2024-12-27
	6	Marcin Grzejszczak	30	Developer	2024-12-27
	7	Szymon Szydełko	30	Developer	2024-12-27
	8	Chris Beams	30	Developer	2024-12-27
	9	Beverley Talbott	30	Developer	2024-12-27

Рисунок 3.6 – Визначення ролі та погодинної оплати.

Окрім ефективності, також важливо оцінити внесок кожного працівника в проект. Метод getContribution визначає індивідуальний внесок працівника, враховуючи кількість змінених рядків та комітів.

Ключові показники ефективності (KPI) працівників розраховуються на основі їхніх комітів, зокрема через кількість доданих рядків на коміт. Визначення лідера проекту автоматизоване й базується на першому коміті, що забезпечує об'єктивність у розподілі ролей та відповідальності в команді. Цей підхід дозволяє динамічно оцінювати продуктивність працівників і коригувати їхню ставку, що сприяє кращій мотивації та результативності команди.

3.3 Тестування системи

Метою тестування було перевірити функціональність REST API, забезпечити правильність обробки запитів до серверу та гарантувати, що отримані відповіді відповідають очікуваним результатам [12].

Під час тестування використовувався інструмент Postman, який дозволяє виконувати HTTP-запити до API, аналізувати відповіді та перевіряти коректність роботи серверної частини [16].

Розглянемо функціональні можливості контролера GitController, який відповідає за управління даними до репозиторіїв у системі. Цей контролер реалізує основні операції з репозиторіями, зокрема збереження коміт-метрик для нових репозиторіїв та перевірку їх наявності в базі даних.

Метод *POST /api/git/save-commits* створений для того, щоб забезпечити збереження коміт-метрик для певного локального репозиторію. Його основна функція полягає в тому, щоб приймати шлях до репозиторію та інші необхідні дані через тіло запиту, перевіряти можливість збереження цієї інформації в базі даних і обробляти відповідні сценарії, які можуть виникнути під час цього процесу. При успішному збережені даних для нового репозиторію у відповідь повертається статус *HTTP 201 Created* (Рис.3.7) разом із повідомленням про успіх.

Рисунок 3.7 – Результат успішного збереження проекту

При спробі збергти вже існуючий репозитирій контролер повертає статус *HTTP 409 Conflict* (Рис. 3.8), щоб повідомити, що збереження неможливе через дублювання даних.

Рисунок 3.8 – Результат при повторному збережені проекту

Якщо запит не містить усіх необхідних параметрів (наприклад, шляху до репозиторію або іншої критичної інформації), контролер повертає статус *HTTP 400 Bad Request.* У повідомленні помилки вказується, які саме параметри були відсутні або некоректні.

У разі непередбачених обставин, наприклад, збою у взаємодії з базою даних або помилок у коді обробки запиту, повертається статус *HTTP 500 Internal Server Error.* Це повідомляє про критичну проблему, яка потребує уваги розробника.

Метод *GET /api/git/check/repository/{projectName}* виконує перевірку, чи існує у базі даних інформація про репозиторій із заданою назвою проекту. Цей запит використовується для перевірки наявності перед додаванням нового репозиторію або для отримання підтвердження, що репозиторій вже зареєстрований.

Якщо репозиторій із вказаною назвою проекту знайдено у базі, контролер повертає статус *HTTP 200 OK.*

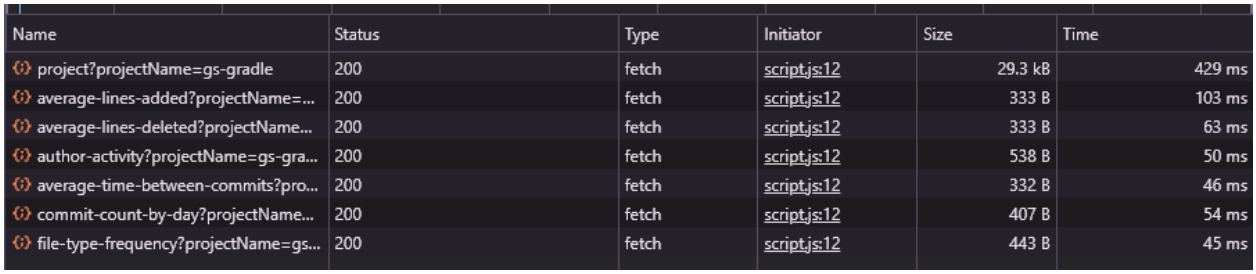
Якщо в базі відсутній запис про репозиторій із зазначеною назвою проекту, контролер повертає статус *HTTP 404 Not Found.*

Ці два методи забезпечують важливі функції для роботи із збереженням і перевіркою репозиторіїв у межах проєкту, дозволяючи ефективно керувати даними та уникати дублювання.

Метод *GET /api/metrics/project*, який використовується для отримання метрик певного проекту, приймає параметр projectName і повертає метрики для вказаного проекту, якщо метрики для цього проекту знайдені, вони будуть передані в тілі відповіді зі статусом *HTTP 200 OK*. Якщо ж метрик для зазначеного проекту не існує, сервер поверне статус *HTTP 404 Not Found*, що означає, що проект не знайдений або не має відповідних метрик у базі даних.

Методи для різноманітної аналітики проекту використовують запити *GET* таких, як середнє число доданих/видалених рядків, активність авторів, підрахунок комітів за днями тижня та частота типів файлів.

Один із основних сценаріїв тестування передбачає успішне повернення даних при правильному запиті з проектом, для якого є збережені метрики. У таких випадках контролер повинен повертати статус 200 (OK) з відповідними даними у форматі JSON (Рис. 3.9). Тести перевіряють, чи коректно повертаються аналітичні результати, наприклад, середнє число рядків, доданих за комітами, або список активних авторів з їхньою кількістю комітів.

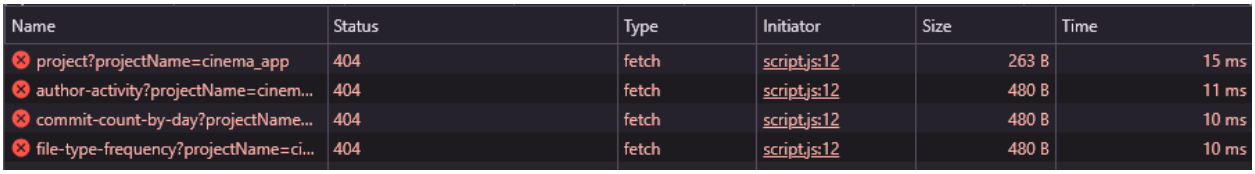


Name	Status	Type	Initiator	Size	Time
project?projectName=gs-gradle	200	fetch	script.js:12	29.3 kB	429 ms
average-lines-added?projectName=...	200	fetch	script.js:12	333 B	103 ms
average-lines-deleted?projectName...	200	fetch	script.js:12	333 B	63 ms
author-activity?projectName=gs-gra...	200	fetch	script.js:12	538 B	50 ms
average-time-between-commits?pro...	200	fetch	script.js:12	332 B	46 ms
commit-count-by-day?projectName...	200	fetch	script.js:12	407 B	54 ms
file-type-frequency?projectName=gs...	200	fetch	script.js:12	443 B	45 ms

Рисунок 3.9 – Результат запитів середнє число доданих/видалених рядків, активність авторів, підрахунок комітів за днями тижня та частота типів файлів

Інший сценарій тестування передбачає перевірку випадку, коли для запитуваного проекту немає збережених даних. У цьому випадку контролер має повертати статус 404 (Not Found), що вказує на відсутність метрик для зазначеного проекту (Рис. 3.10).

Таким чином, тестування включає в себе: перевірку правильності обробки запитів на аналітичні дані, тестування на випадки успішного отримання аналітики з кодом 200, тестування на випадки, коли для проекту не знайдено аналітичних даних статус 404 (Рис. 3.10).



Name	Status	Type	Initiator	Size	Time
project?projectName=cinema_app	404	fetch	script.js:12	263 B	15 ms
author-activity?projectName=cinem...	404	fetch	script.js:12	480 B	11 ms
commit-count-by-day?projectName...	404	fetch	script.js:12	480 B	10 ms
file-type-frequency?projectName=ci...	404	fetch	script.js:12	480 B	10 ms

Рисунок 3.10 – Результат не знайдених даних системою

HTTP запити для взаємодії з інформацією про працівників здійснює операції такі, як отримання інформації, створення, оновлення та обчислення ефективності.

Для отримання всіх працівників, які працюють над проектом використовується метод getEmployeesByProject, приймається ім'я проекту через шлях (URL) /project/{projectName}, якщо працівники для цього проекту знайдені, повертається їх список(Рис. 3.11).

Name	Status	Type	Initiator	Size	Time
gs-gradle	200	fetch	script.js:624	1.5 kB	64 ms

Рисунок 3.11 – отримання всіх працівників за проект

Якщо ні, повертається помилка 404 (Not Found) з порожнім списком (Рис. 3.12).

Рисунок 3.12 – Результат не знайдених працівників за назвою проекта

Для того щоб отримати інформацію про конкретного працівника використовується метод getEmployee, який приймає ім'я працівника як параметр у шляху (URL) /api/employees/{name} та повертає об'єкт працівника за цим ім'ям (Рис. 3.13).

Рисунок 3.13 – Статус для знайденого працівника у проекті

При необхідності долучити нового робітника до проекту використовується метод *POST* з адресою /api/employees, що дозволяє створити нового працівника, передаючи його ім'я в тілі запиту (Рис. 3.14). Після створення працівника встановлюється поточна дата його долучення і повертається повідомлення про успішне створення працівника з його ID, роллю та ставкою.

Рисунок 3.14 – Статус успішного додавання працівника

Якщо лідер команди або відповідальна особа має повноваження змінювати оплату праці працівників, використовується метод *PUT* з запитом */api/employees/{id}/update-rate* для оновлення погодинної ставки в залежності від ефективності працівника, якщо працівник знайдений, його ставка оновлюється, і повертається повідомлення, що містить стару та нову ставку, але якщо ж працівника не знайдено, повертається помилка 404 (Рис. 3.15).

Name	Status	Type	Initiator	Size	Time
update-rate	200	fetch	script.js:719	438 B	87 ms
update-rate	404	fetch	script.js:719	372 B	16 ms

Рисунок 3.15 – Статус оновленої та не оновленої ставки

При тестування методу *GET* з запитом */api/employees/{id}/efficiency*, можна отримати ефективність працівника за його ID (Рис. 3.16). Відповідь містить кількість доданих рядків коду, яка визначається за допомогою методу calculateEmployeeEfficiency.

Name	Status	Type	Initiator	Size	Time
efficiency	200	fetch	script.js:747	392 B	33 ms

Рисунок 3.16 – Статус при отриманні ефективності робітника

Для відстеження внеску працівника в проект реалізовано метод, який дозволяє перевірити цей внесок за допомогою *GET* запиту до шляху */api/employees/{id}/contribution* з параметром projectName (Рис. 3.17). Відповідь містить статистику внеску працівника, зокрема кількість комітів за різними типами файлів, якщо дані не знайдено, повертається повідомлення про відсутність інформації.

Name	Status	Type	Initiator	Size	Time
contribution?projectName=gs-gradle	200	fetch	script.js:778	1.1 kB	59 ms

Рисунок 3.17 – Результат при отриманні внеску працівника у проект

Також, в системі є API-ендпоінти, які дозволяють отримувати інформацію про команди на основі ID лідера команди або назви проекту.

Перший ендпоінт — HTTP GET запит, який прив'язаний до шляху /team-lead/{teamLeadId}. Він дозволяє користувачам отримати всі команди, що очолюються певним лідером команди (Рис. 3.18). Контролер очікує параметр шляху, teamLeadId, для ідентифікації лідера команди. Коли ендпоінт викликається, контролер делегує завдання сервісу teamService для отримання списку команд, пов'язаних з цим лідером. Метод сервісу getTeamsByTeamLeadId повертає список команд, який потім перетворюється на об'єкти TeamDTO за допомогою MetricConverter. DTO-об'єкти повертаються у відповіді у вигляді списку.

Другий ендпоінт також є HTTP GET запитом, але він прив'язаний до шляху /project/{projectName}, і дозволяє користувачам отримати всі команди, які працюють над певним проектом(Рис. 3.18).

Ендпоінт очікує параметр шляху, projectName, для ідентифікації проекту як і в попередньому випадку, контролер викликає метод сервісу getTeamsByProjectName для отримання команд, що працюють над зазначеним проектом. Результати також перетворюються на об'єкти TeamDTO, які потім повертаються у відповіді у вигляді списку.

У обох ендпоінтів перетворення сутностей Team на об'єкти TeamDTO здійснюється за допомогою MetricConverter, що забезпечує повернення лише необхідних даних у відповіді, дотримуючись принципу абстракції даних.

Name	Status	Type	Initiator	Size	Time
gs-gradle	200	fetch	script.js:599	4.1 kB	40 ms
1	200	fetch	script.js:572	4.1 kB	41 ms

Рисунок 3.18 – Результат при отриманні команді за проект, та отримання команди за лідером

Отже, при тестувані у Postman підтвердило коректність роботи API [16]. Ендпоінти працюють згідно з вимогами, забезпечуючи надійний доступ до функціональності системи. Отримані результати дозволяють зробити висновок про відповідність розробленого програмного забезпечення заявленим функціональним вимогам.

3.4 Розробка інтерфейсу користувача

Однією з ключових складових успішної роботи системи є зручний та інтуїтивно зрозумілий інтерфейс користувача, який забезпечує ефективну взаємодію з програмним продуктом. У рамках розробки цього проекту було реалізовано інтерфейс для відображення та аналізу ключових показників ефективності (KPI) працівників. Інтерфейс дозволяє користувачам легко взаємодіяти з системою, вводити необхідні дані та отримувати зрозумілу інформацію.

На етапі планування інтерфейсу було визначено основні функціональні вимоги, які повинні бути реалізовані:

1. Збереження локальних та віддалених репозиторіїв.
2. Відображення даних про працівників, включаючи їх ефективність та внесок у проекти.
3. Створення інтерфейсу для введення даних про працівників та їх KPI.

4. Візуалізація цих даних у вигляді графіків та таблиць для зручного аналізу.

Ці вимоги стали основою для створення структури інтерфейсу та інтерактивних елементів, що дозволяють ефективно працювати з даними.

Технічний вибір інструментів для реалізації інтерфейсу

HTML/CSS/JavaScript — основні технології для створення структури сторінки, стилів та інтерактивних елементів.

Bootstrap — для забезпечення адаптивного дизайну, що дозволяє інтерфейсу автоматично підлаштовуватися під різні розміри екрану.

Chart.js — для побудови інтерактивних графіків, які дозволяють візуалізувати зміни в KPI працівників у вигляді лінійних або стовпчикових діаграм.

JavaScript Fetch API — для асинхронних запитів до серверу, що дає змогу динамічно оновлювати дані без перезавантаження сторінки.

Дані інструменти забезпечили необхідну функціональність та дозволили реалізувати інтерактивний інтерфейс, зручний для користувачів.

В рамках розробки інтерфейсу для управління репозиторіями було реалізовано три розділи.

Перший розділ реалізує збереження комітів, це дозволяє користувачам додавати нові коміти до бази даних, обираючи проект, шлях до репозиторію та тип репозиторію локальний (Рис.3.19) або GitHub (Рис.3.20). Інтерфейс використовує компоненти Bootstrap для забезпечення адаптивного дизайну та інтерактивності

Основні елементи розділу «Зберегти Коміт»:

1. Назва проекту
2. Шлях до репозиторію

3. Тип репозиторію
4. Кнопка "Зберегти Коміти"
5. Індикатор завантаження
6. Виведення відповіді

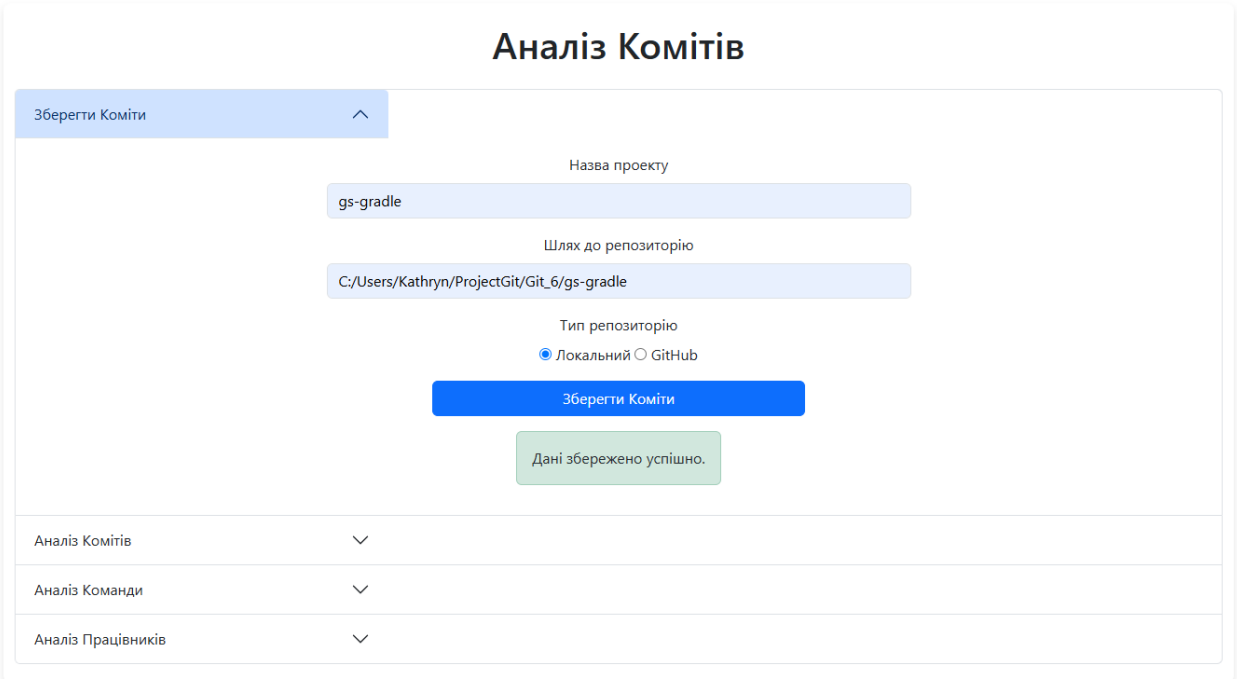


Рисунок 3.19 – Розділ для збереження локальних комітів.

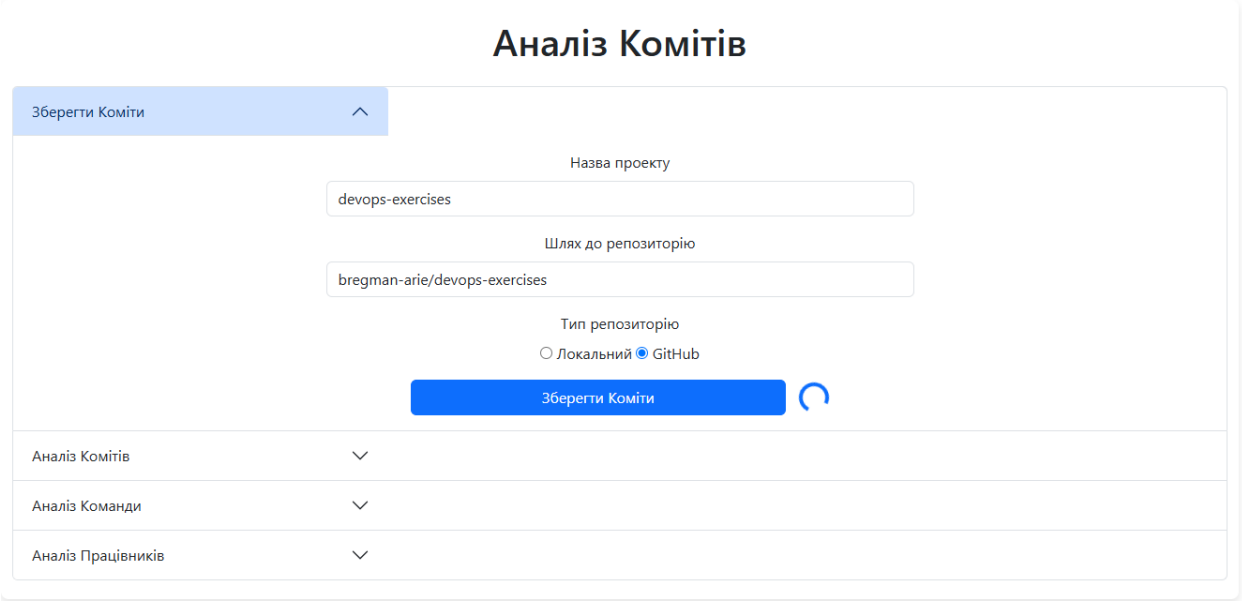


Рисунок 3.20 – Розділ для збереження віддалених комітів.

При повторному збереженні коміту виводиться повідомлення яке інформує користувача, що данні за цим репозиторієм вже були збереженні (Рис. 3.21).

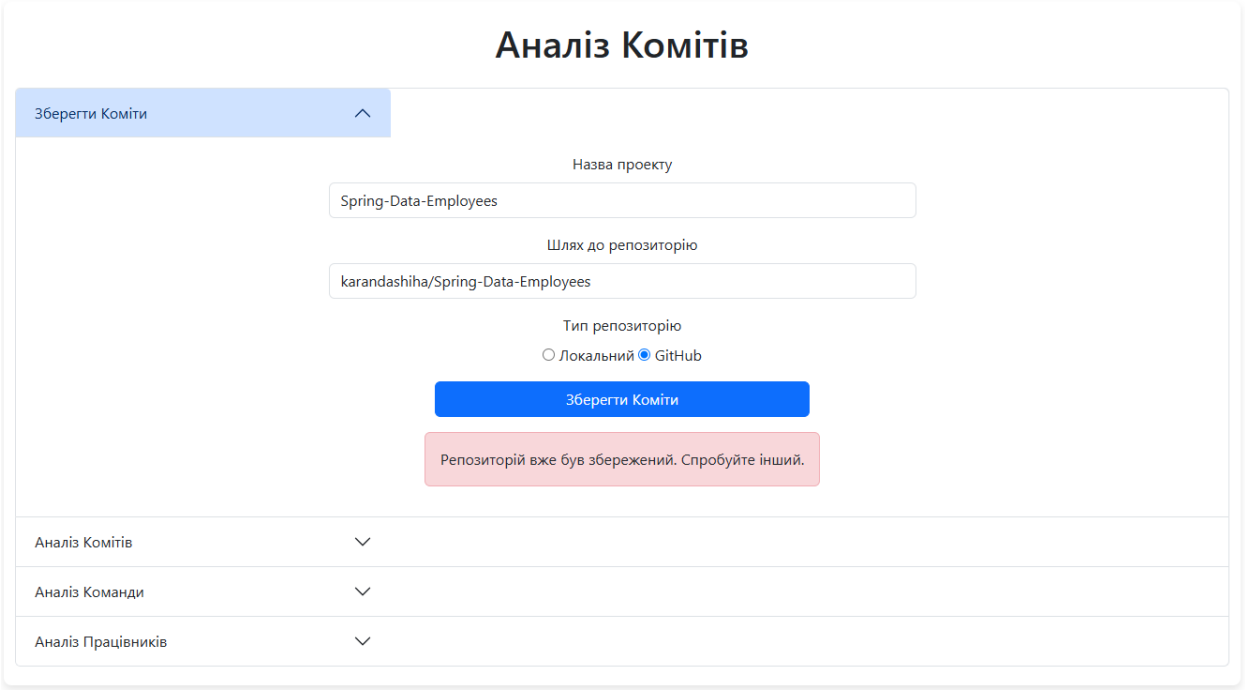


Рисунок 3.21 – Повідомлення про вже збережений репозиторій

Другий розділ надає можливість здійснювати різноманітні запити для аналізу даних комітів у вибраному проекті. Користувач може ввести назву проекту і обрати одну з кількох опцій для отримання метрик, таких як середня кількість доданих чи видалених рядків, активність авторів, розподіл комітів по днях тижня тощо.

Основні елементи розділу «Аналіз Комітів»:

1. Поле для введення назви проекту
2. Кнопки для запитів
3. Відображення результатів у вигляді таблиць та графіків

Розглянемо функціональність кнопок та відображення результатів

Кнопка «Отримати всі метрики» виконує комплексний запит до бази даних і отримує повний набір метрик, пов'язаних із проектом. Набір включає дані про коміти, авторів, кількість доданих і видалених рядків коду, а також

частоту використання різних типів файлів. Результати відображаються у зручній формі: графік, що демонструє динаміку змін, і таблиця, що містить детальну інформацію для подальшого аналізу (Рис. 3.22). Все це дозволяє користувачеві швидко оцінити загальний стан проекту та ключові показники.

Рисунок 3.22 – Функціонал кнопки «Отримати всі метрики»

Кнопка «Середня кількість доданих рядків» дозволяє отримати дані про середню кількість рядків, доданих у кожному коміті, ці дані важливі для оцінки обсягу роботи, виконаної в проекті, і можуть використовуватися для аналізу продуктивності команди. Результати представлені у вигляді таблиці, що містить числові значення, і графіка, який візуалізує ці показники для полегшення сприйняття (Рис. 3.23).

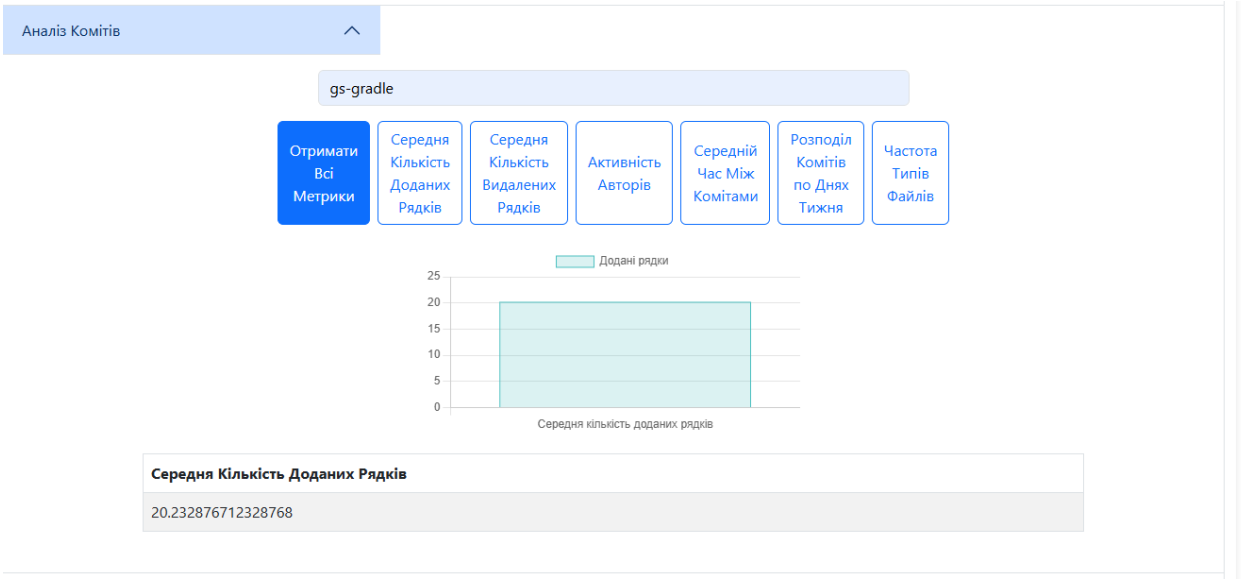


Рисунок 3.23 – Функціонал кнопки «Середня кількість доданих рядків»

Кнопка «Середня кількість видалених рядків» надає статистику про середню кількість рядків коду, видалених у кожному коміті. Ці дані є корисними для аналізу змін у кодовій базі та оцінки процесу рефакторингу або видалення застарілого коду. Як і у попередньому випадку, результати виводяться у формі таблиці для детального перегляду та графіка для візуалізації змін (Рис. 3.24).

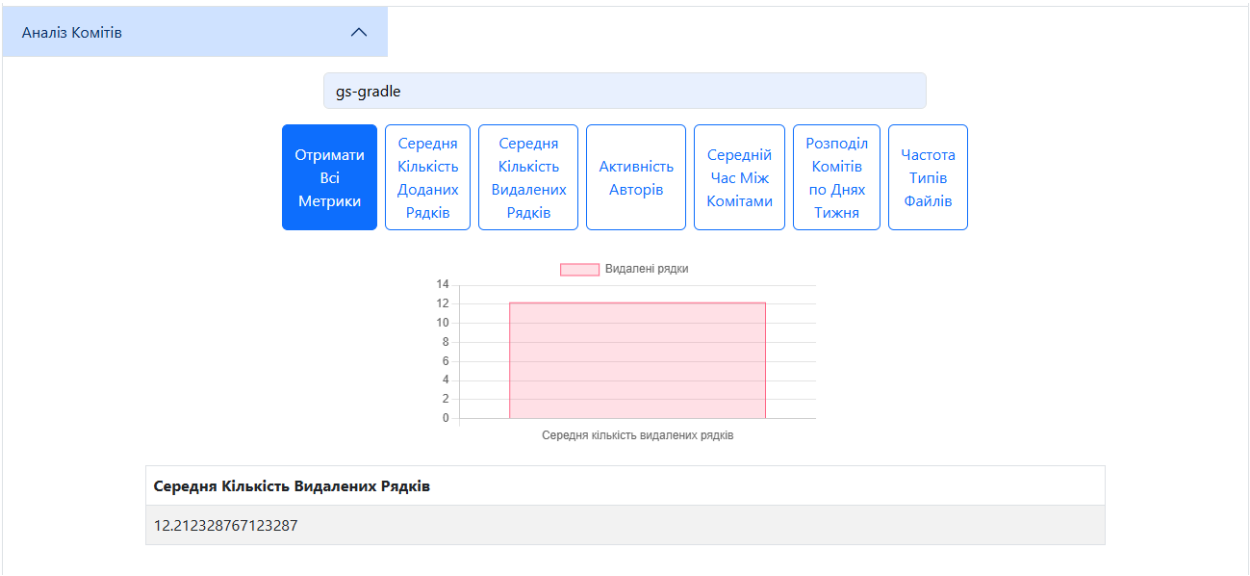


Рисунок 3.24 – Функціонал кнопки «Середня кількість видалених рядків»

Кнопка «Активність авторів» надає детальну інформацію про внесок кожного автора у проект, що дозволяє аналізувати, хто з команди найбільш активний, а також оцінити рівень участі кожного члена команди у розробці. Дані виводяться у вигляді таблиці, яка містить кількість комітів для кожного автора, а також графіка, що ілюструє активність у зрозумілій формі (Рис. 3.25).

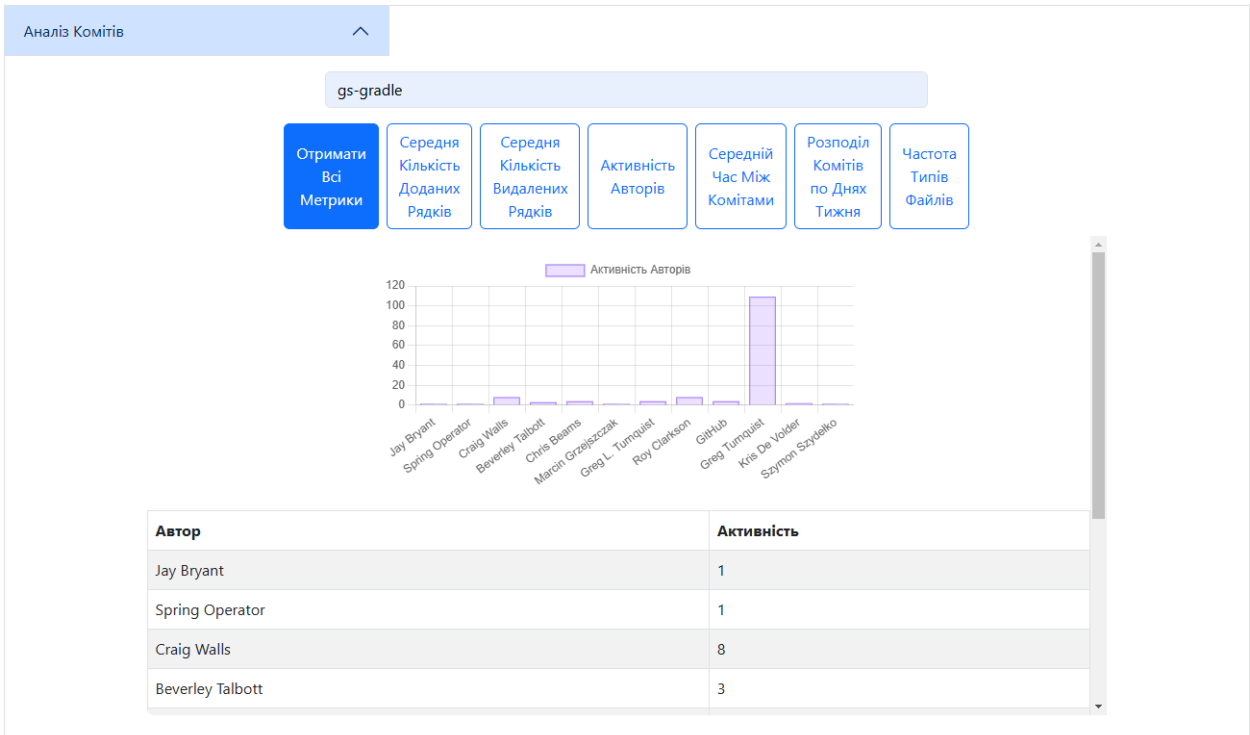


Рисунок 3.25 – Функціонал кнопки «Активність авторів»

Кнопка «Розподіл комітів по днях тижня» аналізує активність команди залежно від дня тижня дозволяючи виявити, в які дні продуктивність команди є найвищою, і оптимізувати робочі процеси відповідно до отриманих даних. Результати представлені у вигляді таблиці із зазначенням кількості комітів для кожного дня тижня та графіка, який показує динаміку активності (Рис. 3.26).

День Тижня	Кількість Комітів
MONDAY	26
SUNDAY	3
TUESDAY	24
THURSDAY	28
SATURDAY	15

Рисунок 3.26 – Функціонал кнопки «Розподіл комітів по днях тижня»

Кнопка «Частота типів файлів» надає інформацію про те, які типи файлів найчастіше використовуються у проекті.

Дана статистика може бути корисною для аналізу структури проекту та оцінки його складності. Результати виводяться у формі таблиці, яка містить перелік типів файлів і кількість їх появ у комітах, а також графіка, що візуалізує ці показники (Рис. 3.27).

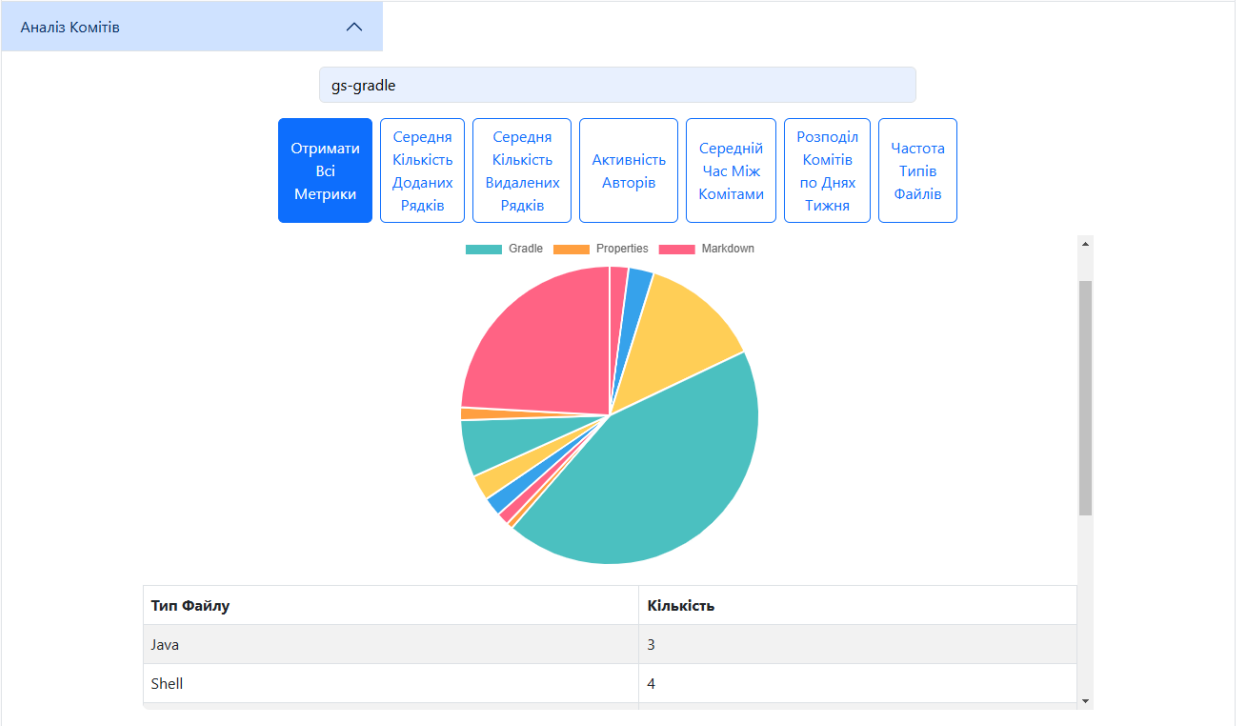


Рисунок 3.27 – Функціонал кнопки «Частота типів файлів»

Таким чином, для кожної метрики створені інтерактивні графіки за допомогою Chart.js та таблиці, що дозволяють користувачеві візуально оцінити динаміку змін після натискання відповідної кнопки.

Третій розділ дозволяє проводити аналіз команди за проектом або лідером.

Основні елементи розділу «Аналіз Команди»:

1. Форма для введення даних
2. Кнопки для запитів
3. Таблиця для відображення результатів

Розглянемо перший запит, за назвою проекту. В поле вводу вводимо назву проекту та отримуємо результат у вигляді таблиці(Рис. 3.28).

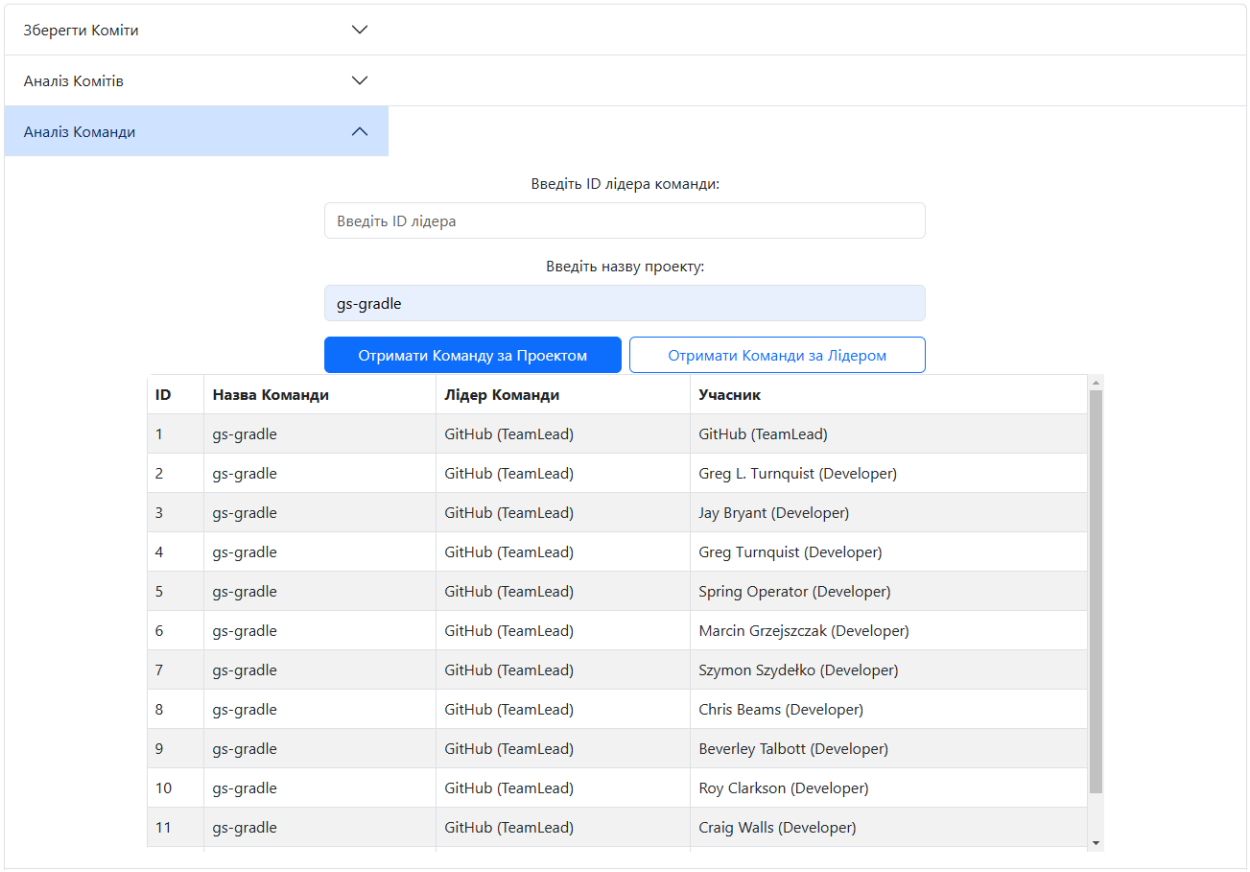


Рисунок 3.28 – Аналіз команди за назвою проекту

Розглянемо другий запит, який виконується за ID лідера команди (Рис. 3.29). У цьому проекті ID лідера команди визначається користувачем, який першим ініціалізував проект та здійснив перший коміт.

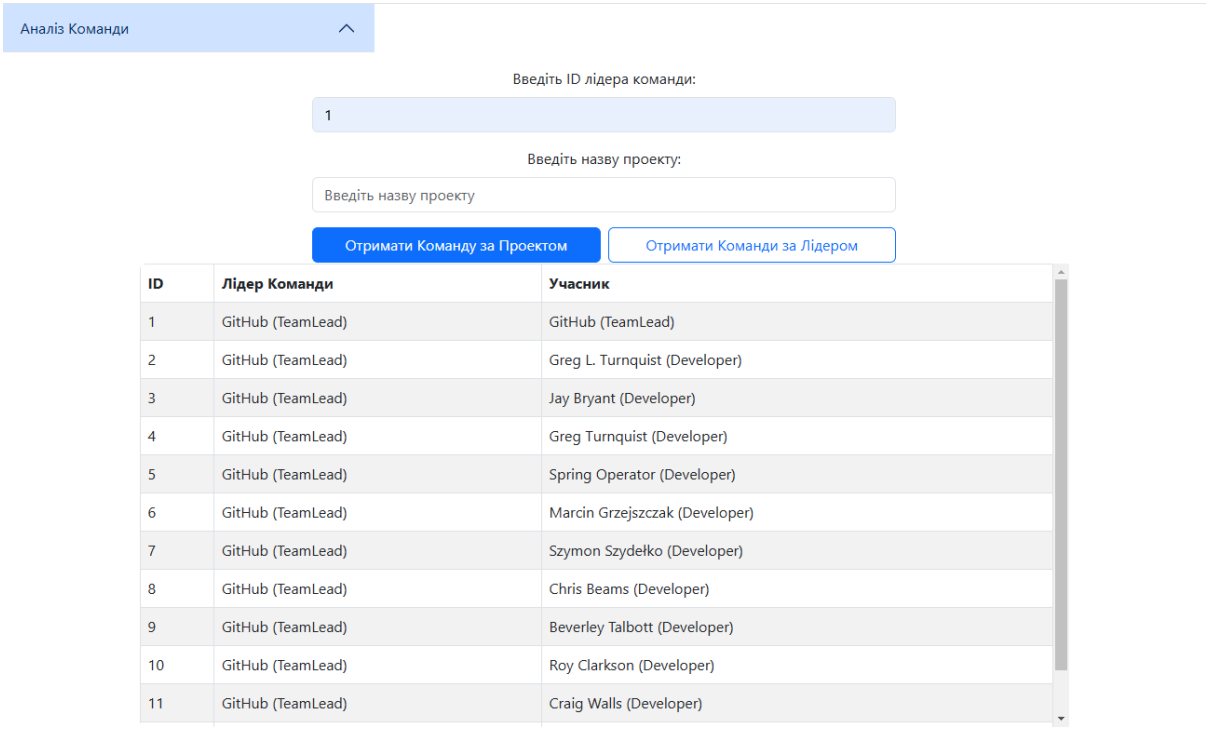


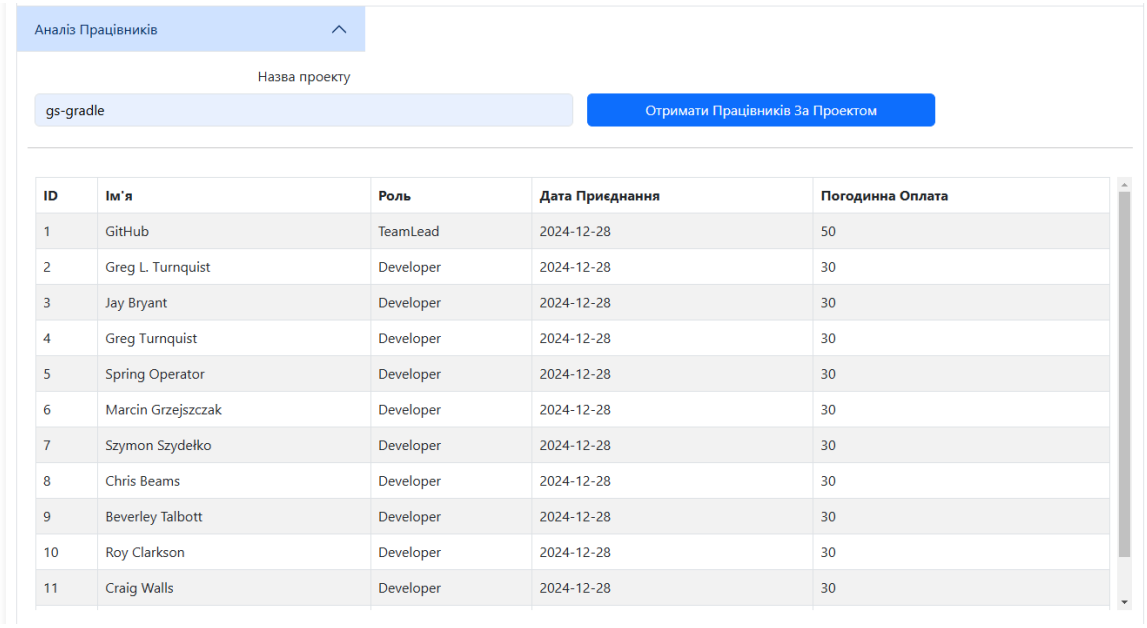
Рисунок 3.29 – Аналіз команди за ID лідера команди

Четвертий розділ дозволяє взаємодіяти з даними працівників через зручну форму та кілька підфункцій.

Основні елементи розділу «Аналіз Працівників»:

1. Введення назви проекту та отримання списку працівників
2. Пошук працівника за іменем
3. Створення нового працівника
4. Оновлення ставки працівника
5. Отримання ефективності працівника
6. Аналіз внеску працівника до проекта

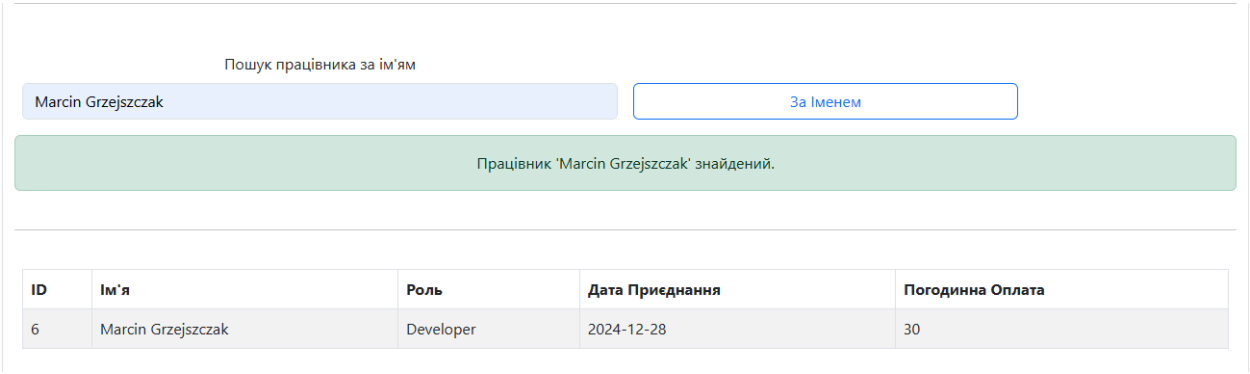
Розглянемо функціонал першої кнопки «Отримати працівників за проектом» (Рис. 3.30). Користувач має ввести назву проекту у відповідному полі та натиснути кнопку, після чого виводяться данні у вигляді таблиці, яка містить інформацію про працівників : ID, ім’я, роль, дата приєднання, погодинна оплата.



ID	Ім'я	Роль	Дата Приєднання	Погодинна Оплата
1	GitHub	TeamLead	2024-12-28	50
2	Greg L. Turnquist	Developer	2024-12-28	30
3	Jay Bryant	Developer	2024-12-28	30
4	Greg Turnquist	Developer	2024-12-28	30
5	Spring Operator	Developer	2024-12-28	30
6	Marcin Grzejszczak	Developer	2024-12-28	30
7	Szymon Szydełko	Developer	2024-12-28	30
8	Chris Beams	Developer	2024-12-28	30
9	Beverley Talbott	Developer	2024-12-28	30
10	Roy Clarkson	Developer	2024-12-28	30
11	Craig Walls	Developer	2024-12-28	30

Рисунок 3.30 – Інформація про працівників проекту

Перейдемо до наступної кнопки «Пошук працівника за ім’ям», яка дозволяє шукати конкретного робітника (Рис. 3.31).



ID	Ім'я	Роль	Дата Приєднання	Погодинна Оплата
6	Marcin Grzejszczak	Developer	2024-12-28	30

Рисунок 3.31 – Пошук працівника за ім’ям

Також, за необхідністю можна додати нового працівника заповнивши поле вводу «Новий працівник» після чого з’являється повідомлення про успішну операцію (Рис. 3.32).

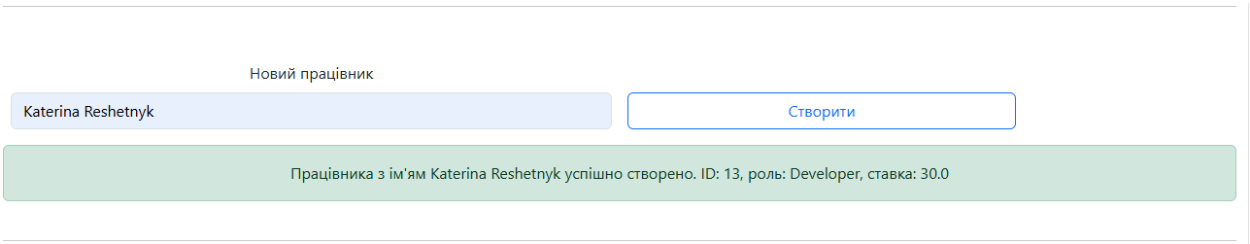


Рисунок 3.32– Створення нового працівника

Також, передбачена можливість оновлювати ставку працівникам аналізуючи їх ефективність роботи, якщо робітник демонструє високу активність, робить багато комітів його ставку можна підвищити для заохочення подальшої ефективної роботи. У випадку з низькою активність ставку можна зменшити, щоб стимулювати працівника покращити продуктивність. Розглянемо на прикладі.

У другому розділі «Аналіз комітів» на графіку «Активність авторів» можна побачити найвищий та найнижчий показник активності. Самим активним працівником є Greg Turnquist (Рис. 3.33) для нього можна збільшити ставку, як мотивацію продовжувати працювати в тому ж руслі (Рис. 3.34).

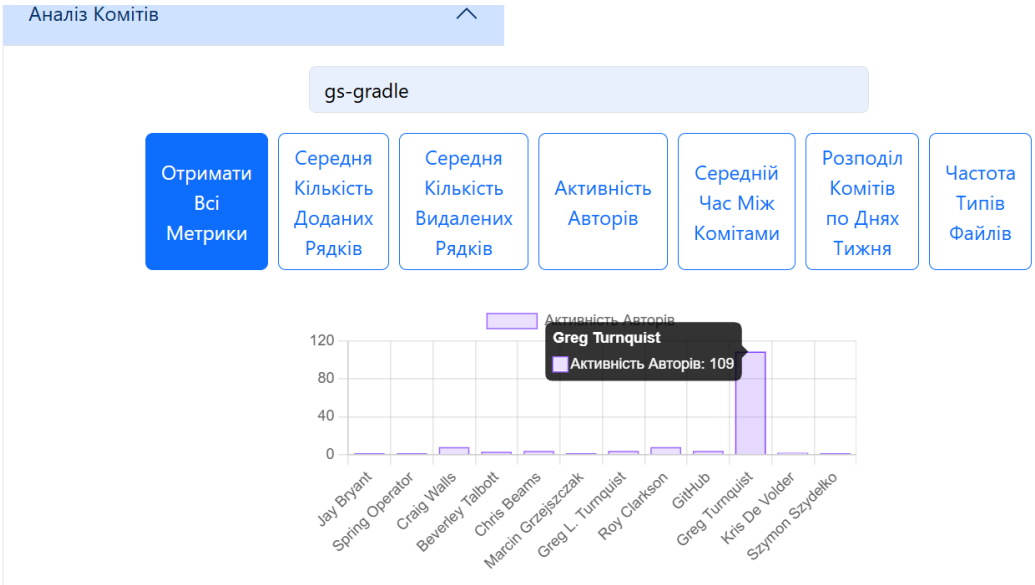


Рисунок 3.33 – Найвищий показник активності

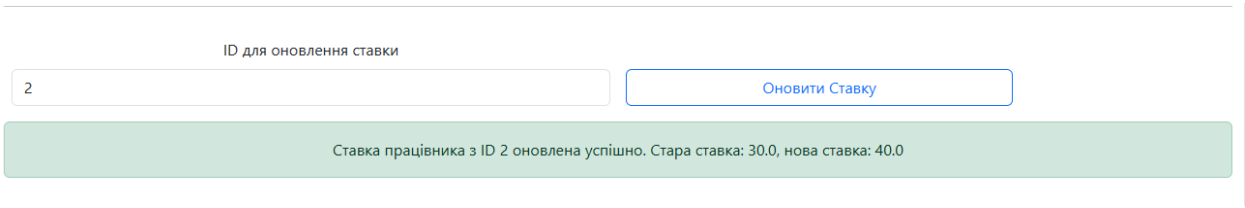


Рисунок 3.34 – Збільшення ставки

Працівник із найнижчими показниками активності — Jay Bryant (Рис. 3.35). Для нього доцільно розглянути можливість зниження ставки, щоб забезпечити баланс між оплатою праці та результативністю (Рис. 3.36). Такий підхід дозволяє підтримувати справедливу систему винагороди: працівники з високою активністю, як Greg Turnquist, отримують додаткові заохочення, а менш продуктивні працівники отримують стимул підвищувати свою ефективність.

Усвідомлення залежності рівня оплати від продуктивності може мотивувати Jay Bryant активніше долучатися до проектів, виконувати свої обов’язки відповідальніше та покращувати результати своєї роботи.

Крім того, працівник із низьким рівнем активності споживає ресурси компанії (час, гроші, обладнання), не забезпечуючи належного внеску у кінцевий результат. Зменшення ставки дозволяє оптимізувати витрати компанії, спрямувати заощаджені кошти на розвиток або стимулювання більш продуктивних працівників і, таким чином, підтримувати ефективність всієї команди.

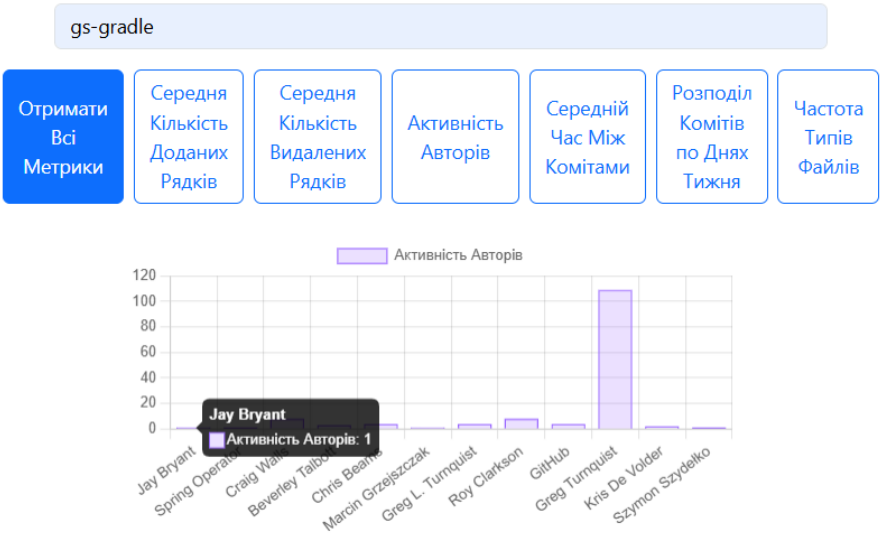


Рисунок 3.35 – Найнижчий показник активності

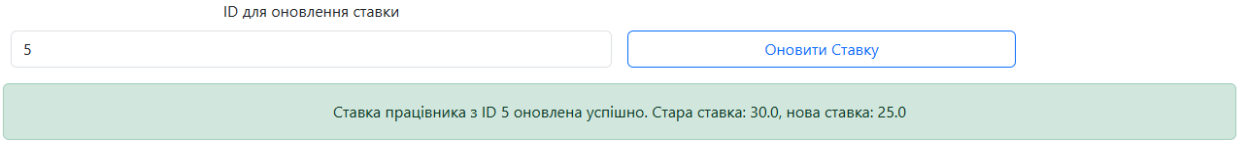


Рисунок 3.36 – Зменшення ставки

Також у системі реалізовано можливість оцінки ефективності працівника за показником кількості доданих рядків коду. Для цього передбачено функціонал кнопки «Ефективність», яка дозволяє отримати швидкий аналіз внеску окремого працівника в рамках проекту.

Після натискання кнопки результат обчислення ефективності відображається у вигляді інформативного повідомлення, що містить дані про кількість доданих рядків (Рис. 3.37) це допомагає керівнику оперативно оцінити продуктивність працівника та прийняти відповідні управлінські рішення, наприклад, щодо підвищення ставки, додаткової мотивації або необхідності вдосконалення навичок.

Такий підхід є важливим інструментом для забезпечення прозорості процесу оцінювання та підтримання високої ефективності роботи команди.

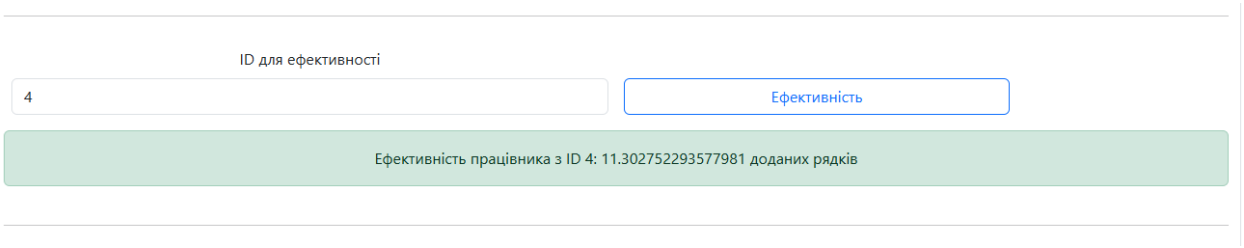


Рисунок 3.37 – Ефективність працівника за кількістю доданих рядків

Відстеження внеску працівників у проект дозволяє детально оцінити, з якими файловими системами вони працюють, а також зрозуміти розподіл обов'язків між учасниками команди (Рис 3.38).

Дана функція забезпечує прозорість у процесі роботи, даючи змогу аналізувати, які саме завдання виконує кожен працівник і як вони взаємодіють із різними частинами системи. Для отримання інформації про внесок працівника необхідно вказати його ID та назву проєкту. На основі цих даних система надасть детальний звіт про типи змін у файлах, кількість зроблених комітів над якою велась робота.

Таким чином, моніторинг внеску працівників не лише підвищує ефективність роботи команди, але й допомагає керівникам приймати обґрунтовані рішення щодо подальшого планування проєкту.

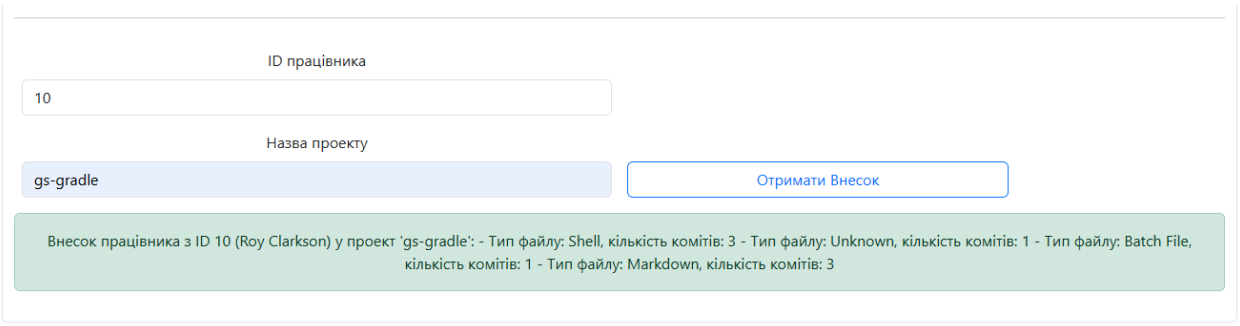


Рисунок 3.38 – Внесок працівника у проект

Запропонований функціонал забезпечує ефективний і зручний спосіб взаємодії з даними завдяки своїй адаптивності та простоті. Організація у вигляді акордеону дозволяє користувачам швидко знайти потрібну інформацію, не перевантажуючи інтерфейс. Динамічне оновлення даних без необхідності перезавантаження сторінки значно покращує користувацький досвід, а інтуїтивно зрозумілий дизайн форм і таблиць сприяє легкому доступу до ключових функцій системи.

Така реалізація підвищує продуктивність роботи з інтерфейсом і забезпечує зручність як для звичайних користувачів, так і для адміністраторів.

3.5 Налаштування та розгортання проекту

В даному проекті було впроваджено систему на базі середовища розробки IDE, використовуючи мову програмування Java та фреймворк Spring Boot. Процес впровадження передбачає налаштування серверного середовища, інтеграцію з базою даних, підключення необхідних залежностей та тестування системи в реальних умовах. Розглянемо основні етапи впровадження [15].

Для того, щоб почати працювати з проектом, спочатку потрібно підготувати середовище розробки.

Встановити Java Development Kit (JDK) 11 або вище, це дозволить компілювати та запускати Java-програми, останню версію JDK можна завантажити з офіційного сайту Oracle JDK.

Для подальшої розробки проекту рекомендовано встановити інтегровану середу розробки IntelliJ IDEA , яку можна завантажити з офіційного сайту IntelliJ IDEA, це потужний інструмент для розробки Java-застосунків, який дозволить зручно працювати з проектом.

Оскільки проект використовує базу даних MySQL для зберігання метрик, потрібно встановити MySQL на комп'ютер. Завантжито можна за посиланням MySQL Community Server. Після встановлення необхідно створити базу даних, яка зберігатиме метрики CREATE DATABASE git_metrics_db; а також створити нового користувача та надати йому права доступу до цієї бази даних.

Коли середовище розробки готове, можна перейти до налаштування самого проекту. Спочатку склонувати проект з Git-репозиторію, для цього використовуйте команду Git:

git clone <URL репозиторію>

Цей крок дозволить отримати актуальну версію проекту для подальшої роботи. Для того, щоб налаштувати проект з нуля або додати нові залежності, можна використати Spring Initializr та встановити такі параметри для проекту (Рис. 3.39):

- Project: Maven Project
- Language: Java
- Spring Boot: остання версія (3.4.1)
- Group: com.example.app
- Artifact: Software-Quality-Monitoring-System-Using-Reliability-Metrics
- Name: Software-Quality-Monitoring-System-Using-Reliability-Metrics
- Description: Project for Git metrics analysis
- Packaging: Jar
- Java: 21

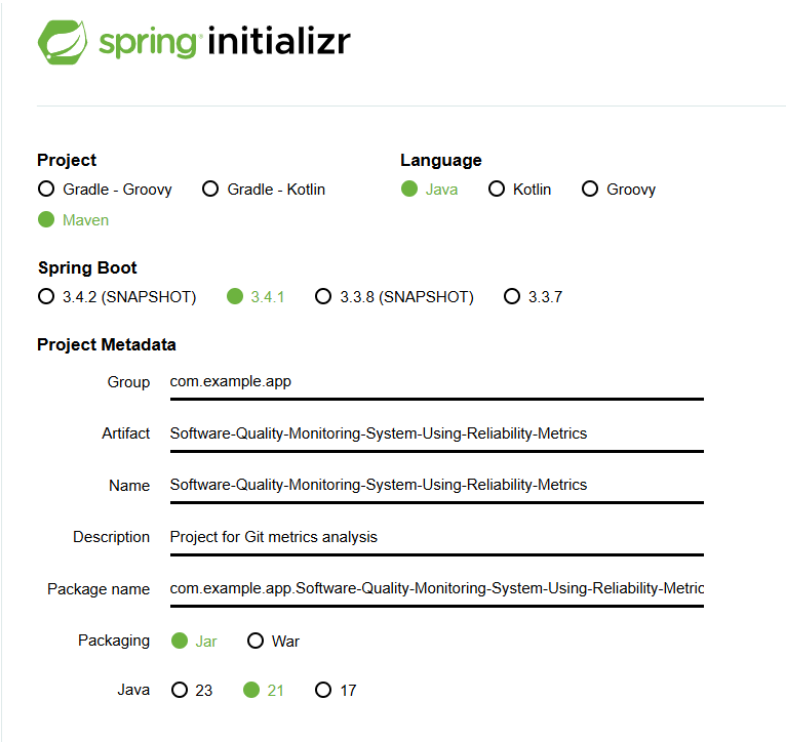


Рисунок 3.39 – Створення проекту за допомогою Spring Initializr

Підключення залежності (Рис. 3.40):

- Spring Web – для підтримки веб-застосунків і REST API
- Spring Data JPA – для інтеграції з базами даних за допомогою JPA.
- MySQL Driver – для з'єднання з MySQL базою даних
- Thymeleaf – для роботи з HTML-шаблонами
- Spring Boot DevTools
- Lombok – для автоматичної генерації коду (геттери, сеттери, конструктори).
- Spring Boot Starter Test
- Spring Boot Starter AOP
- JGit – для роботи з Git-репозиторіями.
- Spring Boot Starter Validation
- GitHub API – для інтеграції з GitHub.

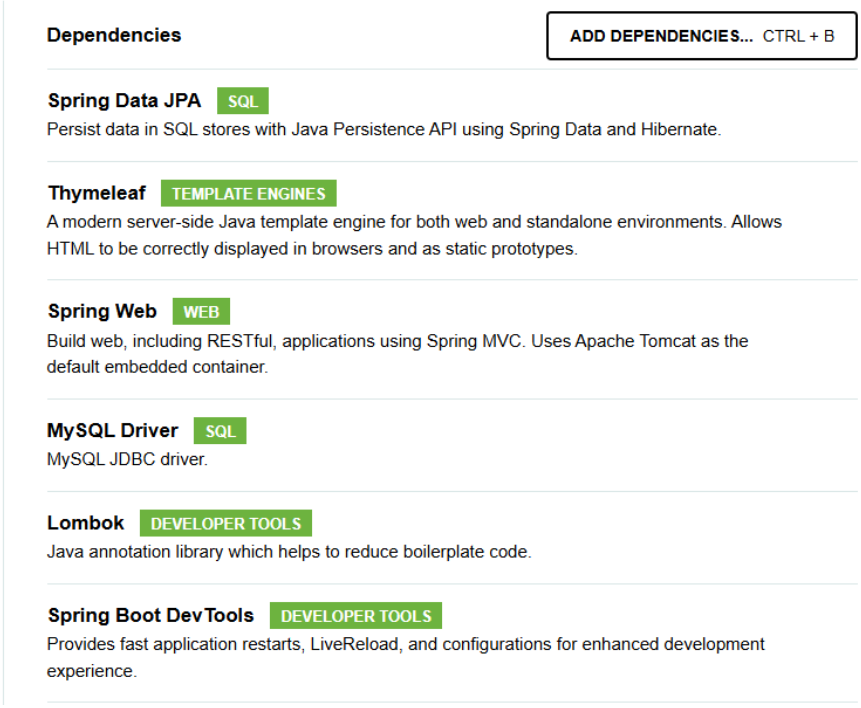


Рисунок 3.40 – Підключені залежності

Після встановлення всіх необхідний залежностей, збережений архів буде готовий для подальшого розгортання проекту.

Також, необхідно перевірити файл pom.xml, щоб були додані всі необхідні залежності.

Для налаштування з'єднання з базою даних треба перейти у файл application.properties, та змінити значення на відповідні для вашого середовища

spring.datasource.url=jdbc:mysql://localhost:3306/git_metrics_db

spring.datasource.username="your_username"

spring.datasource.password="your_password"

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

spring.jpa.properties.hibernate.format_sql=true

spring.profiles.active=development

spring.jpa.properties.hibernate.cache.use_second_level_cache=false

spring.jpa.properties.hibernate.cache.region.factory_class=org.hibernate.cache.jcache.JCacheRegionFactory

```
spring.cache.type=jcache
spring.jpa.open-in-view=false
github.token="your_github_token"
```

Тепер можна здійснити запуск додатка, та перевірити порт під яким запустився проект. Має бути http://localhost:8080.

Після успішного впровадження та перевірки функціональності системи необхідно протестувати REST API [12] за допомогою інструменту Postman, щоб переконатися в коректній обробці запитів. Окрім цього, важливо перевірити базу даних, щоб упевнитися, що метрики Git-репозиторіїв зберігаються правильно й відповідають очікуваним результатам.

3.6 Висновок до третього розділу

У третьому розділі висвітлено всі ключові аспекти, що забезпечують ефективну роботу системи, зокрема її реалізацію, тестування та розгортання. Система включає модуль збору та обробки даних, що дозволяє автоматично отримувати інформацію з Git-репозиторіїв, аналізувати коміти та зберігати важливі метрики у базі даних для подальшого використання. Впроваджено алгоритми розрахунку ключових показників продуктивності (KPI), які дають змогу оцінювати ефективність співробітників і визначати найактивніших учасників команди, що сприяє оптимізації управління ресурсами.

Розробка інтуїтивно зрозумілого та адаптивного інтерфейсу користувача дозволяє динамічно відображати й аналізувати зібрані дані без потреби в перезавантаженні сторінки [11]. Інтерфейс забезпечує зручний доступ до інформації, таблиць і візуалізацій, що сприяє швидкому аналізу метрик і прийняттю рішень. У ході тестування підтверджено працездатність API-

ендпоінтів, коректність збереження даних у базі та взаємодію між модулями системи.

Завершальним етапом стало налаштування середовища та успішне розгортання системи. Реалізація всіх цих компонентів дозволяє досягти високого рівня автоматизації процесів збору, обробки та аналізу даних, що робить систему ефективним інструментом для управління проектами, оцінювання продуктивності команди та покращення загальної якості роботи.

ВИСНОВОК

У процесі виконання кваліфікаційної роботи на здобуття освітнього ступеня магістра за спеціальністю 121 «Інженерія програмного забезпечення» було розроблено сервіс для аналізу ефективності роботи працівників, який базується на ключових показниках ефективності (KPI) та використовує дані з Git-репозиторіїв.

Враховуючи швидкий розвиток цифрових технологій та їх вплив на всі сфери діяльності, створення такого інструменту є важливим кроком у напрямку автоматизації та покращення процесів управління продуктивністю працівників. Розроблена система в рамках цієї роботи, дозволяє автоматично збирати та обробляти дані з репозиторіїв, що дозволяє більш точно вимірювати та оцінювати індивідуальну ефективність членів команди.

Аналіз існуючих підходів до оцінки продуктивності працівників, представлений у першому розділі, дозволив виділити основні методи збору та обробки даних, які є найбільш ефективними для реалізації подібних систем. Вивчення існуючих систем, що працюють з даними з Git-репозиторіїв, дало змогу визначити ключові аспекти, на які потрібно звертати увагу під час розробки нового інструменту. Таким чином, виконана робота створює фундамент для більш точних та адаптивних механізмів оцінки ефективності працівників.

У другому розділі було спроектовано архітектуру системи, яка є основою для її подальшої реалізації. Розробка моделі даних, проектування основних компонентів та їх інтеграція з зовнішніми API забезпечили можливість для зручного та швидкого збору та обробки даних. Окрему увагу було приділено масштабованості системи та її здатності адаптуватися під

різноманітні проекти, що забезпечує гнучкість і ефективність її використання в умовах змінних вимог.

Третій розділ зосереджено на практичній реалізації, тестуванні та розгортанні системи. У межах цього етапу було створено модуль збору даних із Git-репозиторіїв, який забезпечує автоматизований процес отримання інформації про коміти, авторів і зміни у файлах. Даний модуль дозволяє зчитувати дані з локальних та віддалених репозиторіїв, аналізувати їх за визначеними метриками та зберігати результати в базі даних для подальшої обробки.

Алгоритми розрахунку ключових показників ефективності (KPI) були розроблені з урахуванням потреб управління продуктивністю. Вони включають такі показники, як кількість комітів, середній обсяг змін (додані та видалені рядки коду), активність авторів і частота змін у різних типах файлів. Дані алгоритми дозволяють виявляти основні тенденції в роботі команди, визначати лідерів проєкту та ідентифікувати потенційні проблеми в роботі.

Особливу увагу приділено розробці адаптивного інтерфейсу користувача, який забезпечує зручний доступ до аналітичної інформації. Інтерфейс реалізовано у вигляді інтерактивних таблиць і графіків, що дозволяють користувачам легко відслідковувати динаміку змін, аналізувати дані за обраними параметрами. Завдяки інтеграції з бібліотекою Chart.js вдалося реалізувати графічне відображення метрик, таких як активність авторів і частота комітів, що значно спрощує інтерпретацію отриманих даних.

Процес тестування охопив як функціональні, так і нефункціональні аспекти системи. Було перевірено коректність роботи API, точність обробки запитів, відповідність збережених даних очікуваним результатам, а також продуктивність системи за умовою обробки великих обсягів даних. Окремо було протестовано модуль збору даних із різних типів Git-репозиторіїв і

розрахунок KPI для проєктів із різною структурою та обсягами даних. Результати тестування підтвердили стабільність роботи системи, її адаптивність і відповідність вимогам.

Розгортання системи включало налаштування серверного середовища, інтеграцію бази даних та API, а також підготовку документації для користувачів і адміністраторів. Забезпечено підтримку локального та віддаленого використання системи, що дозволяє командам працювати як із централізованими, так і з розподіленими проєктами. Завдяки оптимізації роботи з базою даних і налаштуванню кешування вдалося досягти швидкої обробки запитів і високої продуктивності системи навіть за великого навантаження.

Загалом, розроблений сервіс є потужним інструментом для автоматизації процесу збору та аналізу даних про ефективність роботи працівників. Він дозволяє здійснювати точну оцінку продуктивності на основі реальних даних з Git-репозиторіїв, що сприяє оптимізації управління проектами, поліпшенню координації між членами команди та підвищенню ефективності роботи.

Таким чином, реалізована система не лише забезпечує автоматизацію процесу аналізу ефективності працівників, а й має великий потенціал для подальшого розвитку, включаючи інтеграцію з іншими інструментами управління проектами та покращенням інтерфейсу для зручності користувачів. У результаті система може стати важливим елементом в управлінні командною роботою та аналізі продуктивності в різних сферах діяльності.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Andrew S. Grove «High Output Management » Performance Appraisal: Manager as Judge and Jury, New York 1983р. 181с. https://archive.org/details/dli.ernet.213936/page/181/mode/2up
2. Tom DeMarco, Timothy Lister «Peopleware: Productive Projects and Teams» Teamicide, New York 1999р, 20 параграф,132с. https://orion2020.org/archivo/articulos/00_peopleware.pdf
3. Джез Гамбл, Джин Кім, Ніколь Форсгрен «Accelerate: The Science of Lean Software and DevOps», Measuring and Changing Culture, 2018р https://books.google.com.ua/books?id=Kax-DwAAQBAJ&printsec=frontcover&redir_esc=y#v=onepage&q&f=false
4. Марк Річардс, Ніл Форд, Прамод Садалаж «Software Architecture: The Hard Parts» Надання позачасових порад щодо архітектури програмного забезпечення, 2021р, 3с. https://dl.ebooksworld.ir/books/Software.Architecture.The.Hard.Parts.Neal.Ford.Oreilly.9781492086895.EbooksWorld.ir.pdf
5. Еоін Вудс, Нік Розанський «Software Systems Architecture» Architecture Definition Activities, 2005р, 9с. https://ptgmedia.pearsoncmg.com/images/9780321718334/samplepages/032171833X.pdf
6. Документація Spring boot. (n.d.). https://spring.io/projects/spring-boot
7. Маркус Вінанд «SQL Performance Explained» 2012р. https://pdfcoffee.com/sql-performance-explainedpdf-pdf-free.html
8. Документація GitHub API: https://docs.github.com/en/rest
9. Дослідження провідних компаній з впровадження KPI в IT-сфері https://www.it.ua/knowledge-base/technology-innovation/key-performance-indicators-kpi
10. Fowler, M. (2019, August 1). Software Architecture Guide. Martinfowler.com. https://martinfowler.com/architecture/

11. Fowler, M. (2019, August 1). Agile Software Guide. Martinfowler.com. https://martinfowler.com/agile.html
12. About the REST API – GitHub Docs. (2022, November 28). GitHub Docs. https://docs.github.com/en/rest/about-the-rest-api/about-the-rest-api?apiVersion=2022-11-28
13. Gene Kim, Patrick Debois, John Willis, and Jez Humble «The Phoenix Project: A Novel About IT, DevOps, and Helping Your Business Win», How to Get Great Outcomes by Integrating Operations into the Daily Work of Developmen, 2019p, 95c. http://images.itrevolution.com/documents/DevOps_Handbook_Intro_Part1_Part2.pdf
14. Кент Бек і Мартін Фаулер «Рефакторинг. Поліпшення існуючого коду»,1999р. https://dl.ebooksworld.ir/motoman/Refactoring.Improving.the.Design.of.Existing.Code.2nd.edition.www.EbooksWorld.ir.pdf \
15. Майкл Фетерс «Working Effectively with Legacy Code» 2004р. https://dokumen.pub/working-effectively-with-legacy-code-14th-printingnbsped-0131177052-9780131177055.html
16. Explore APIs on the Postman API Network | Postman Learning Center. (2024, December 18). Postman Learning Center. https://learning.postman.com/docs/collaborating-in-postman/public-api-network/use-the-public-api-network/
17. ДСТУ ISO/IEC 25010:2016 Інженерія систем і програмних засобів. Вимоги до якості систем і програмних засобів та її оцінювання (SQuaRE). Моделі якості системи та програмних засобів (ISO/IEC 25010:2011, IDT). (n.d.). https://online.budstandart.com/ua/catalog/doc-page.html?id_doc=69134

ДОДАТОК А

Метод для обчислення ефективності працівника

```
public double calculateEmployeeEfficiency(Employee employee) {
    List<CommitMetric> commitMetrics =
commitMetricRepository.findByAuthorId(employee.getId());

    int totalLinesAdded = 0;
    for (CommitMetric commit : commitMetrics) {
        totalLinesAdded += commit.getLinesAdded();
    }

    int totalCommits = commitMetrics.size();

    if (totalCommits > 0) {
        return (double) totalLinesAdded / totalCommits;  // Середнє
число рядків на один коміт
    } else {
        return 0;  // Якщо комітів немає, ефективність 0
    }
}
```

ДОДАТОК Б

Оновлення ставок на основі ефективності

```
public void updateEmployeeHourlyRate(Employee employee) {
    // Розрахунок ефективності
    double efficiency = calculateEmployeeEfficiency(employee);

    // Збільшення ставки залежно від ефективності
    if (efficiency > 50) {
        employee.setHourlyRate(employee.getHourlyRate() + 10.0);  //
Збільшення ставки на 10
    } else if (efficiency < 20) {
        employee.setHourlyRate(employee.getHourlyRate() - 5.0);  //
Зменшення ставки на 5
    }

    // Зберігаємо оновлену інформацію
    employeeRepository.save(employee);
}
```

ДОДАТОК В

Призначаємо лідера для проекту, якщо він ще не встановлений

```
if (projectLead.get() == null) {
    projectLead.set(author); // Призначаємо першого автора лідером
    newTeam.setTeamLead(author);
    roleAndRateAssigner.assignRoleAndRate(author, true);
} else {
    newTeam.setTeamLead(projectLead.get()); // Встановлюємо вже
існуючого лідера
    roleAndRateAssigner.assignRoleAndRate(author, false);
}
```