МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра комп’ютерної інженерії

Чорний Валерій Віталійович

Програмний засіб моніторингу IT-проєктів в реальному часі / Software tool for IT projects monitoring in real time

спеціальність: 123 – Комп’ютерна інженерія

освітньо–професійна програма – Комп’ютерна інженерія

Кваліфікаційна робота

Виконав: студент групи КІ – 42
Чорний Валерій Віталійович

Науковий керівник
к.т.н. Савка Н.Я.

ТЕРНОПІЛЬ – 2025

АНОТАЦІЯ

Чорний В.В. Засіб моніторингу ІТ-проєктів у реальному часі. – Рукопис.

Кваліфікаційна робота на здобуття освітнього ступеня «бакалавр» за спеціальністю 123 «Комп'ютерна інженерія», освітньо-професійна програма. Західноукраїнський національний університет, Тернопіль, 2025.

Робота написана обсягом 66 сторінок і містить 28 рисунків, 5 таблиці, 5 додатки та 36 джерел за переліком посилань. Обсяг графічного матеріалу 2 аркуші формату А3.

Метою кваліфікаційної роботи є розробка веб-застосунку для моніторингу ІТ-проєктів у реальному часі, що виконує функції зберігання, структурування, перегляду та аналізу файлів у межах окремих репозиторіїв.

У розробці використано мову програмування Python і мікрофреймворк Flask, що забезпечує гнучкість та простоту масштабування. Реалізовано функціонал побудови нових репозиторіїв, завантаження файлів різних типів, перегляду їх вмісту, ведення журналу подій і логування взаємодії користувача з системою. Для роботи з базою даних використано ORM-бібліотеку SQLAlchemy. Інтерфейс побудовано з використанням HTML/CSS та Jinja2-шаблонів.

Розроблений застосунок забезпечує централізоване зберігання документів і дозволяє фіксувати історію змін, що робить його зручним для освітнього, командного та аналітичного використання. У роботі представлено поетапний опис реалізації з діаграмами, фрагментами коду та результатами тестування працездатності. Застосунок може слугувати основою для побудови повноцінної системи контролю версій або внутрішньої платформи цифрової взаємодії між учасниками ІТ-проєктів.

Ключові слова: ВЕБ-ЗАСТОСУНОК, FLASK, МОНІТОРИНГ, РЕПОЗИТОРІЙ, ЛОГУВАННЯ, SQLALCHEMY, ТЕСТУВАННЯ

ABSTRACT

Chornuy V.V. A web-based tool for real-time IT project monitoring. – Manuscript.

Qualification work for the Bachelor degree in specialty 123 “Computer Engineering”, educational and professional program. Western Ukrainian National University, Ternopil, 2025.

The work is written in 66 pages and contains 28 figures, 5 tables, 5 appendices and 30 references. The volume of graphic material is 2 sheets of A3 format.

The aim of this thesis is to develop a web application for real-time monitoring of IT projects, supporting storage, structuring, viewing, and analysis of files within individual repositories.

The system is implemented using the Python programming language and the Flask microframework, which provides flexibility and ease of scaling. The application allows the creation of new repositories, uploading various file types, content preview, activity logging, and user interaction tracking. SQLAlchemy ORM is used for database management. The interface is built using HTML/CSS and Jinja2 templates.

The developed application provides centralized document management and tracks changes, making it suitable for educational, collaborative, and analytical purposes. The thesis includes a step-by-step description of implementation, diagrams, code samples, and testing results. The application can serve as a foundation for a full-fledged version control system or an internal collaboration platform for IT project participants.

Keywords: WEB APPLICATION, FLASK, MONITORING, REPOSITORY, LOGGING, SQLALCHEMY, TESTING

ЗМІСТ

ВСТУП

У сучасному цифровому середовищі ефективне управління ІТ-проєктами потребує зручних і гнучких інструментів для зберігання, перегляду та контролю змін у структурі файлів, що використовуються в процесі розробки. З поширенням віддаленої роботи та розподілених команд зростає потреба у веб-застосунках, які забезпечують централізований доступ до матеріалів, фіксацію активностей та надання актуальної інформації про стан виконання задачі у реальному часі.

У нинішньому світі більшість ІТ-проєктів реалізується в умовах динамічного середовища, що вимагає високої прозорості, керованості та синхронізації дій усіх учасників розробки. Розподілені команди, гнучкі методології управління (Scrum, Kanban), швидкі релізи та інтеграція з іншими системами задають потребу у зручних інструментах контролю за виконанням задач. Веб-застосунки для моніторингу ІТ-проєктів стають ключовим елементом, оскільки дозволяють забезпечити єдиний простір для зберігання артефактів проєкту, відстеження змін, організації репозиторіїв та аналізу дій учасників у реальному часі. У той час як глобальні платформи мають великий функціонал, вони часто є недоступними або надлишковими для внутрішніх або академічних проєктів. Власні локальні рішення дають змогу адаптувати функціональність під конкретні потреби, уникати зайвих залежностей і покращувати контроль над середовищем розробки.

Популярні сервіси на кшталт GitHub чи GitLab є потужними платформами з розширеним функціоналом, однак для локального або внутрішнього використання в межах обмежених проєктів вони часто є надлишковими або вимагають складного налаштування. Саме тому виникає потреба у розробці власного веб-застосунку з можливістю зберігати репозиторії, керувати файлами, відслідковувати зміни та події, а також здійснювати моніторинг активностей користувачів.

Метою дослідження є розробка адаптивного та функціонального веб-застосунку для моніторингу ІТ-проєктів, який дозволяє розробити репозиторії, завантажувати й переглядати файли, відображати структуру директорій та вести журнал змін у рамках кожного проєкту.

У межах поставленої мети було сформульовано низку задач:

– проаналізувати особливості існуючих систем;

– провести порівняльний аналіз програмних засобів із подібним функціоналом;

– визначити основні вимоги до додатку;

– спроєктувати архітектуру системи;

– обґрунтувати вибір програмних засобів та інструментів для реалізації проєкту;

– прототип інтерфейсу користувача;

– розробити моделі бази даних та взаємодії компонентів;

– реалізувати програмну частину та провести тестування з метою оцінки працездатності і зручності використання.

– провести тестування функціоналу, стабільності та зручності використання;

– здійснити аналіз результатів тестування та визначити шляхи оптимізації.

Об’єктом дослідження є процеси керування ІТ-проєктами.

Предметом дослідження є алгоритми та програмні засоби розробки програмного модуля для моніторингу ІТ-проєктів.

Практична цінність полягає у розробці веб-додатку для моніторингу ІТ-проєктів у режимі реального часу.

Кваліфікаційну роботу виконано, згідно до поставленої задачі, та враховуючи вимоги, які викладено у [17–18].

Основні результати дослідження опубліковано у тезах доповіді на ІІ Всеукраїнській науково-практичній конференції студентів, аспірантів та молодих вчених «Інтелектуальні комп’ютерні системи та мережі-2025" [16]. Копії публікації можна переглянути у додатку А.

1 СИСТЕМИ МОНІТОРИНГУ ІТ-ПРОЄКТІВ

1.1. Поняття моніторингу ІТ-проєктів

Управління ІТ-проєктами вимагає постійного контролю над усіма фазами реалізації, оскільки сучасний підхід до ведення таких проєктів базується на чіткому плануванні, вимірюванні результатів і гнучкому реагуванні на зміни. Відповідно, моніторинг виступає ключовим інструментом у системі управління, дозволяючи здійснювати безперервне спостереження за динамікою виконання задач, використанням ресурсів та досягненням встановлених цілей. Його роль полягає не лише у фіксації фактичного стану проєкту, а й у формуванні основи для оперативного прийняття рішень.

Моніторинг ІТ-проєкту визначається як інформаційна система, яка забезпечує збір, аналіз і передачу інформації про стан реалізації проєкту кінцевим користувачам. Він охоплює кількісні та якісні показники, що дають змогу оцінити прогрес, своєчасно виявити відхилення від запланованих показників і запобігти ризикам. Відповідно до сучасних підходів, моніторинг включає фінансову, технічну та організаційну складові, що забезпечують комплексну оцінку ефективності кожного етапу життєвого циклу проєкту.

Особливе значення моніторинг має в умовах постійних змін у вимогах замовника, зростаючої складності технічних рішень і високих очікувань щодо якості. Ефективно побудована система моніторингу дозволяє не лише забезпечити прозорість процесу розробки, а й покращити взаємодію між учасниками команди, оптимізувати витрати та знизити ризики невдачі проєкту. Згідно з дослідженням Н. П. Юрчук, така система є сукупністю взаємопов’язаних елементів, що спрямовані на попередження проблем та забезпечення досягнення запланованих результатів [1].

Моніторинг ІТ-проєктів – це невід’ємний процес управління, який дозволяє своєчасно отримувати дані про хід виконання проєкту, визначати ефективність використання ресурсів та забезпечувати досягнення поставлених цілей. Цей процес охоплює постійне спостереження, збір, аналіз і візуалізацію інформації, що

стосується ключових аспектів проєкту – бюджету, термінів, статусу, технічного стану та організаційної складової.

Головною метою моніторингу є забезпечення ефективного контролю над усіма фазами реалізації проєкту шляхом оперативного виявлення відхилень від запланованих показників і прийняття відповідних управлінських рішень. Моніторинг також забезпечує прозорість процесу, дозволяє уникнути помилок і сприяє досягненню високої якості результатів.

Серед основних задач системи моніторингу можна виокремити:

– оцінка загального стану проєкту – включає визначення поточного статусу реалізації, ступеня завершення задач, відповідності запланованим термінам та очікуванням замовника;

– контроль за бюджетом – здійснюється шляхом аналізу витрат і порівняння їх із затвердженими кошторисами;

– аналіз продуктивності команди – визначається за допомогою KPI та індикаторів ефективності окремих учасників і підрозділів;

– управління ризиками – передбачає фіксацію потенційних загроз та відхилень, а також розробку плану реагування;

– прогнозування подальших етапів проєкту – на основі зібраних даних можна оцінити ймовірність дотримання термінів, виявити вузькі місця та оптимізувати процеси;

– формування звітності для керівництва та замовників – дозволяє відображати поточну ситуацію у зручному форматі для прийняття рішень.

У сучасних умовах ефективна система моніторингу повинна бути не лише аналітичним інструментом, але й активним помічником у прийнятті рішень. Саме тому використання спеціалізованих платформ з розширеним візуальним представленням даних, таких як Birdview PSA, стає критично важливим для менеджменту проєктів [2].

На рисунку 1.1 представлено типовий приклад візуального моніторингу IT-проєктів, реалізованого за допомогою спеціалізованої програмної платформи. Графіки демонструють ключові аспекти: розподіл проєктів за типами портфелів,

статус виконання, розподіл по відповідальних особах (Team Lead), стан здоров'я проєкту (Project Health), відповідність графіку (Schedule) та стан бюджету (Budget). Візуалізація дозволяє миттєво виявляти відхилення: наприклад, на графіку Project Health видно, що певна кількість проєктів перебуває в статусі "Action Required", тобто потребують негайного втручання. Така модель моніторингу дозволяє в реальному часі оцінювати стан портфелю проєктів та приймати обґрунтовані управлінські рішення.

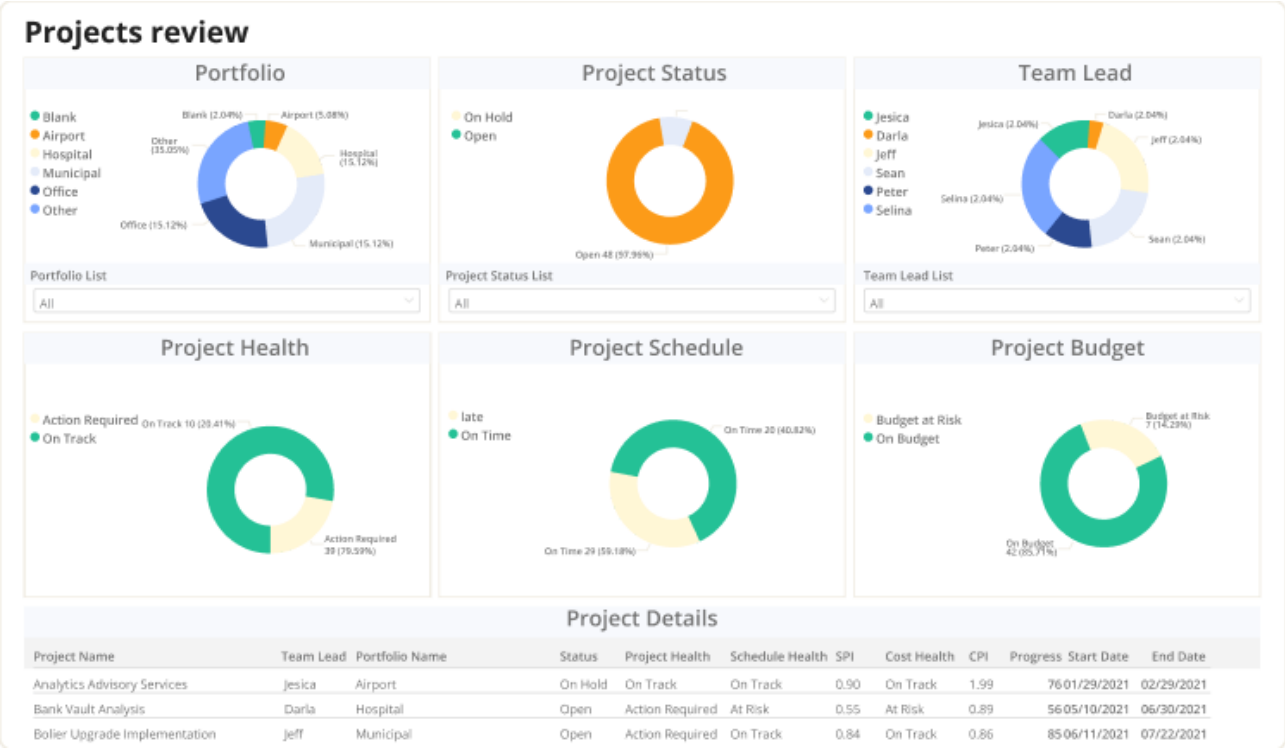

Рисунок 1.1 – Візуалізація стану IT-проєктів за ключовими показниками у системі моніторингу

Моніторинг IT-проєктів є складним багаторівневим процесом, який охоплює весь життєвий цикл проєкту та базується на взаємодії низки функціонально взаємопов'язаних компонентів. Ці компоненти утворюють єдину інформаційну систему, що забезпечує повноцінний контроль за перебігом проєкту, дозволяє виявляти проблемні ділянки, приймати виважені управлінські рішення, формувати звітність і підтримувати інформаційну безпеку. На рисунку 1.2 представлено структурну схему архітектури системи моніторингу IT-проєктів, яка включає основні етапи обробки інформації – від джерел до кінцевого користувача [3].

Рисунок 1.2 – Структурна схема компонентів системи моніторингу ІТ-проєктів

Першим логічним блоком є джерела даних, з яких надходить інформація про функціонування інфраструктури. Сюди належать мережеве обладнання, сервери, різноманітні пристрої технічного обліку, програмні додатки та засоби безпеки. Усі ці джерела є фундаментом для подальшого аналізу, оскільки саме з них надходять первинні технічні сигнали та метрики [3].

Наступним етапом є збір даних. На цьому рівні реалізуються механізми взаємодії з джерелами, визначаються формати та протоколи обміну інформацією, конфігуруються агенти або інтерфейси доступу. Інформація передається до центрального ядра системи моніторингу, яке виконує її попередню обробку, фільтрацію, структурування та зберігання в базі даних. Це ядро є серцем архітектури та забезпечує подальшу аналітику.

Зібрані дані передаються до модуля реагування, який реалізує алгоритми виявлення подій, відхилень або аномалій. Саме тут формуються автоматичні сповіщення, що дозволяють оперативно інформувати відповідальних осіб. Для ефективної роботи системи важливим є модуль безпеки, який реалізує захист від несанкціонованого доступу, забезпечує автентифікацію користувачів, шифрування

та захист переданої інформації. Таким чином, система зберігає цілісність, конфіденційність і доступність даних.

У паралельному напрямку функціонує адміністративний інтерфейс, який є інструментом налаштування всієї системи. За його допомогою відбувається конфігурація модулів, керування доступом користувачів, встановлення порогів сповіщення та побудова звітних шаблонів. Адміністративний інтерфейс пов’язаний із модулем візуалізації та звітності, який відповідає за відображення даних у зрозумілій для людини формі – графіки, таблиці, діаграми, аналітичні панелі. Цей модуль також забезпечує генерацію звітів для різних рівнів управління – від інженера до керівника проєкту.

Фінальним компонентом архітектури є споживачі даних – користувачі або підсистеми, які використовують результати моніторингу для подальшої аналітики, стратегічного планування або оперативного реагування. Інформація, що надходить до них, є кінцевим продуктом багатоступеневого процесу збору, обробки, захисту та візуалізації.

Таким чином, схема на рисунку 1.2 ілюструє повний цикл моніторингу: від отримання сигналів до прийняття рішень. Вона відображає важливість взаємозв’язку всіх компонентів, забезпечує системність і наочність підходу до контролю за реалізацією ІТ-проєктів. Така структурована система дозволяє забезпечити не лише технічну стабільність, а й прозорість та керованість усіх етапів розробки, впровадження та підтримки ІТ-рішень.

1.2. Класифікація та функції систем керування репозиторіями

Системи керування репозиторіями класифікуються залежно від того, де зберігається база даних з версіями файлів, як забезпечується спільна робота з кодом та який рівень незалежності мають користувачі. У контексті управління ІТ-проєктами важливо розуміти принципові відмінності між локальними,

централізованими та розподіленими системами керування версіями, оскільки саме вони формують структуру командної роботи над кодом і визначають можливості співпраці, резервування та контролю змін.

Локальні системи керування версіями є найпростішими. Всі версії змін зберігаються лише на одному комп'ютері користувача. Програма для контролю версій підтримує спеціальну локальну базу даних, де фіксуються знімки стану файлів у кожен момент часу. Користувач працює з файлами, а система зберігає різні версії у внутрішньому сховищі. Основним недоліком є відсутність можливості командної роботи. На рисунку 1.3 показано архітектуру такої системи: локальний комп'ютер має файл і внутрішню базу даних версій, яка зберігає всю історію змін.

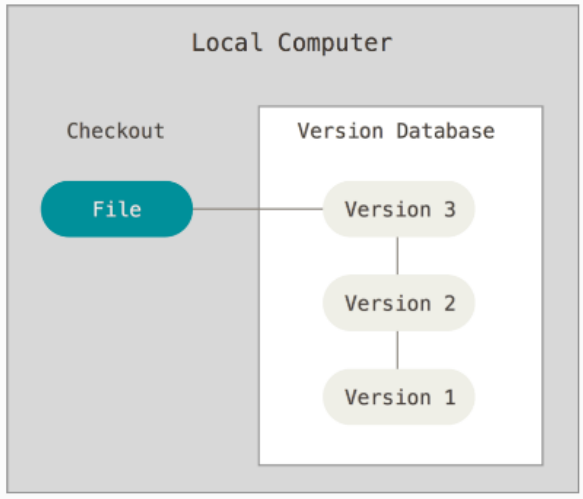

Рисунок 1.3 – Архітектура локальної системи керування версіями

Централізовані системи керування версіями передбачають наявність єдиного сервера, де зберігається основна база даних версій. Кожен клієнт (користувач) підключається до центрального сервера та завантажує останні версії файлів для подальшої роботи.

Усі зміни також надсилаються на сервер, що дозволяє легко координувати спільну діяльність і мати повну історію змін в одному місці. Однак при втраті доступу до сервера, вся система втрачає працездатність. Рисунок 1.4 демонструє взаємодію декількох комп'ютерів із центральним сервером, що зберігає єдину версійну базу.

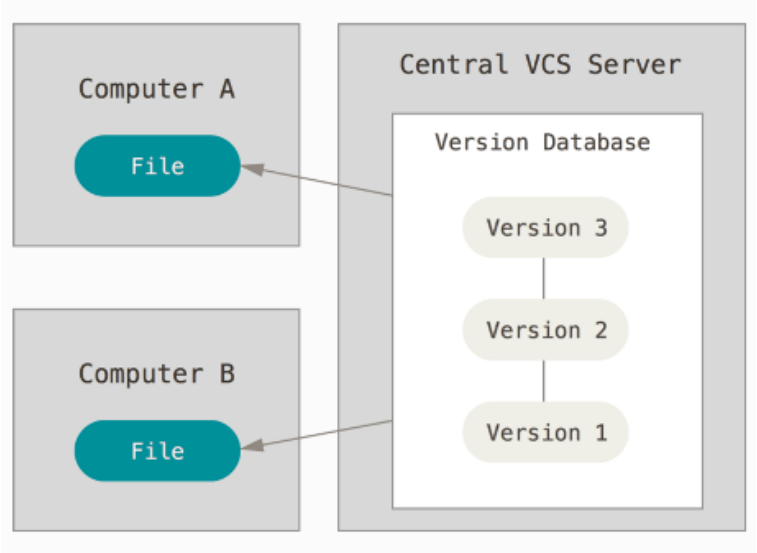

Рисунок 1.4 – Архітектура централізованої системи керування версіями

Розподілені системи, такі як Git, є найсучаснішими. Кожен учасник команди має повну копію репозиторію, включаючи всю історію змін. Це дозволяє працювати автономно, фіксувати локальні коміти, зберігати історію навіть без підключення до мережі. При підключенні до сервера або іншого користувача можна синхронізувати зміни. Такий підхід забезпечує високу стійкість до збоїв та зручність в умовах гнучкої командної роботи. На рисунку 1.5 зображено три комп'ютери з власними версіями бази, які можуть взаємодіяти між собою та з головним сервером.

Таким чином, вибір типу системи керування репозиторіями залежить від специфіки проєкту, вимог до надійності, масштабованості та способу організації командної співпраці. Для сучасних IT-команд найчастіше обирають саме розподілені системи, що поєднують гнучкість, автономність і надійність [4].

Системи керування версіями (Version Control Systems, VCS) є ключовим інструментом у сучасній розробці програмного забезпечення, оскільки дозволяють не лише фіксувати зміни у програмному коді, але й організовувати ефективну роботу в команді, забезпечуючи цілісність, контроль та зворотність усіх змін.

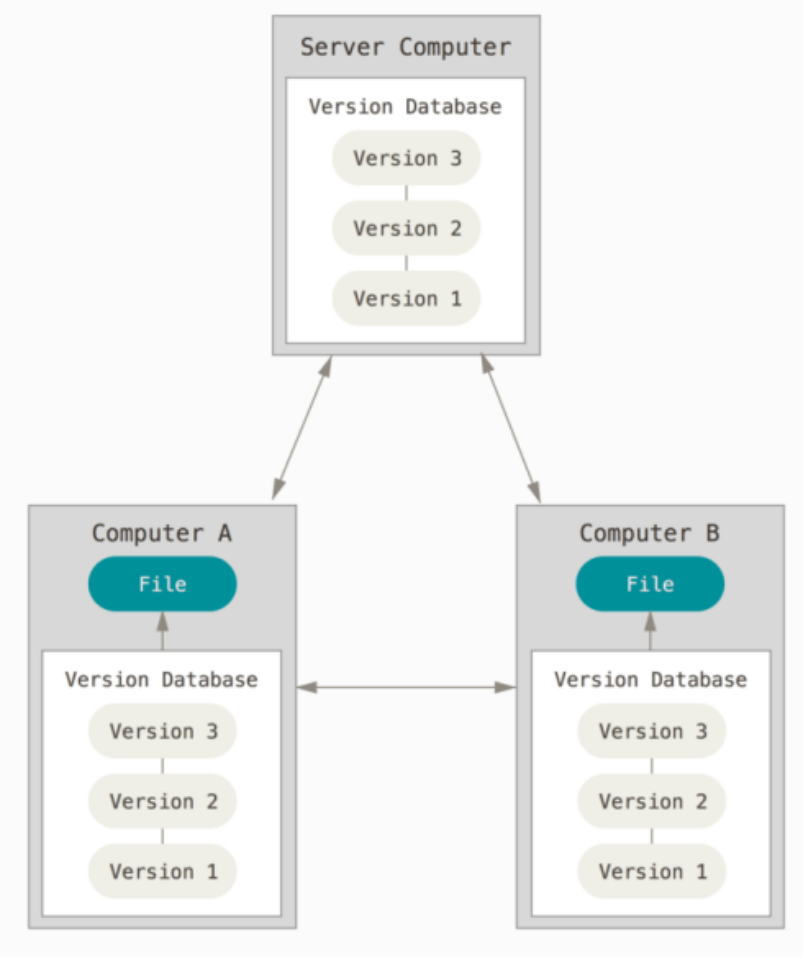

Рисунок 1.5 – Архітектура розподіленої системи керування версіями

Одна з головних функцій таких систем – збереження історії змін. Кожна модифікація коду, файлів чи конфігурацій зберігається як окрема версія, що дозволяє переглянути, порівняти чи повернутися до будь-якого попереднього стану. Це критично важливо при виявленні помилок, впровадженні нових функціональностей або при тестуванні. Кожна версія супроводжується метаданими: автор змін, дата, опис, що забезпечує повну відтворюваність.

Другою важливою функцією є підтримка паралельної роботи кількох розробників. Система дозволяє різним учасникам проєкту працювати над одними і тими ж файлами незалежно, не блокуючи роботу одне одного. Злиття (merge) змін є стандартною операцією, яку підтримує більшість VCS. Системи також розв’язують конфлікти у разі, коли кілька розробників змінюють один і той самий фрагмент коду.

Контроль доступу та безпека змін – ще один важливий аспект. VCS дозволяє обмежити або дозволити доступ до певних частин репозиторію, що критично в проєктах з великою кількістю учасників або при роботі з конфіденційним кодом.

Функції відстеження та відновлення стану коду забезпечують стабільність та надійність: навіть у разі втрати локальної копії, вся історія змін зберігається в репозиторії, і її можна легко відновити. Крім того, VCS дозволяють створювати гілки (branches) – окремі версії проєкту, в яких можна експериментувати, не зачіпаючи основну лінію розробки. Злиття гілок дозволяє повертати зміни в основну гілку лише після повного тестування.

Ще однією важливою функцією є ведення журналу змін (логів), що дає змогу швидко проаналізувати, хто і коли вніс певну зміну, з якою метою та як це вплинуло на загальний стан проєкту. Це підвищує прозорість командної роботи та дисципліну при розробці.

Таким чином, сучасні системи керування версіями забезпечують не лише збереження коду, а й реалізують повноцінну модель співпраці, історії, безпеки та контролю в усьому життєвому циклі розробки [4].

Git – це розподілена система керування версіями, розроблена Лінусом Торвальдсом у 2005 році для керування розробкою ядра Linux. Вона є однією з найпопулярніших систем у світі завдяки своїй ефективності, масштабованості та гнучкості. Ключовою особливістю Git є те, що кожен користувач має повноцінну копію репозиторію, включно з усією історією змін, що забезпечує автономну роботу навіть без підключення до інтернету.

Git відрізняється надзвичайно швидкою роботою з гілками (branches), можливістю легко створювати ізольовані середовища для розробки нових функцій, експериментів або виправлень. Завдяки цьому гілкування та об’єднання (merge) стали невід’ємною частиною сучасного стилю розробки. Крім того, Git використовує механізм контрольних сум (SHA-1), який забезпечує цілісність даних та захист від випадкових або навмисних змін.

Інфраструктура Git активно підтримується та інтегрується з великою кількістю хостинг-сервісів, зокрема GitHub, GitLab і Bitbucket. Це дозволяє не лише зберігати код, але й створювати розширену екосистему – з відслідковуванням

задач, CI/CD, pull-запитами, рев'ю коду тощо. Саме Git став стандартом де-факто для командної розробки у відкритому та комерційному програмному забезпеченні, витіснивши багато інших систем [5].

Рисунок 1.6 – Логотип та ідентифікаційна емблема системи керування версіями Git

Subversion (SVN) – це централізована система керування версіями, яка була розроблена у 2000 році компанією CollabNet як заміна популярній, але застарілій системі CVS. Вона орієнтована на централізовану модель зберігання коду, де головний репозиторій знаходиться на сервері, а всі користувачі отримують доступ до єдиного джерела правди – центральної бази даних версій.

На відміну від розподілених систем, у SVN основна взаємодія відбувається безпосередньо з сервером. Кожна зміна потребує зв'язку з ним: і завантаження нових версій, і відправлення змін. Це дозволяє більш жорстко контролювати процес розробки, обмежувати доступ на рівні файлів або директорій, та централізовано здійснювати аудит і резервне копіювання.

Однією з переваг SVN є простота у використанні, а також відносна зручність при роботі в командах із чіткою ієрархією. Система дозволяє працювати із великими бінарними файлами, надає підтримку «замикання» файлів, щоб уникати конфліктів при паралельній зміні. Завдяки своїй стабільності, SVN до сьогодні використовується в урядових структурах, корпоративному секторі, інженерних та наукових установах.

Попри те, що популярність Subversion поступово знижується через домінування Git, вона залишається актуальною для ряду проєктів, де централізована модель краще відповідає вимогам безпеки, звітності або інтеграції з іншими інструментами [6].

Рисунок 1.7 – Логотип системи керування версіями Subversion (SVN)

Mercurial – це сучасна розподілена система керування версіями, яка з'явилася у 2005 році практично одночасно з Git. Її було розроблено як інструмент для гнучкої та масштабованої командної роботи над програмним забезпеченням із особливим акцентом на простоту використання, швидкодію та зручність. Розробником системи є Метт Макал.

Основна відмінність Mercurial від інших систем полягає в тому, що вона поєднує всі переваги розподіленої моделі з інтуїтивним інтерфейсом команд. Кожен розробник має повноцінну локальну копію репозиторію з усією історією змін, а робота з гілками, комітами, злиттями та відкатами виконується через зрозумілу й передбачувану синтаксичну модель. Усі зміни, які вносяться локально, можна згодом синхронізувати з іншими копіями або з центральним репозиторієм.

Mercurial розроблено на мові програмування Python, що забезпечує його портативність і розширюваність. Система активно підтримувала розширення, що дозволяло адаптувати її під специфічні потреби розробників. За своїм функціональним спектром Mercurial близький до Git, але орієнтований на користувачів, які надають перевагу простоті над гнучкістю.

Попри високу якість реалізації, Mercurial не досягнув такого рівня популярності, як Git. Зокрема, припинення підтримки Mercurial хостингом Bitbucket у 2020 році значно вплинуло на його поширення в індустрії. Тим не менш, ця система продовжує використовуватись у деяких академічних і корпоративних середовищах, де цінується її простота, передбачуваність та стабільність.

Рисунок 1.8 – Логотип системи керування версіями Mercurial

На таблиці 1.1 продемонстровано порівняльний аналіз трьох найбільш поширених систем керування версіями: Git, SVN (Subversion) та Mercurial. Вибір відповідної системи залежить від специфіки проєкту, технічних вимог і досвіду команди. Як видно з наведених характеристик, Git вирізняється високою гнучкістю, швидкістю та повною локальною автономією, що робить його ідеальним вибором для розподіленої командної розробки з великою кількістю паралельних гілок. SVN, натомість, пропонує більш централізовану та передбачувану модель, яка зручна для організацій з чіткою ієрархією та потребою в централізованому контролі доступу. Mercurial виступає компромісним рішенням – поєднуючи переваги розподіленого підходу з простішим інтерфейсом, воно орієнтоване на менш технічно підготовлених користувачів.

Усі три системи мають свої сильні сторони, проте сучасні IT-команди дедалі частіше надають перевагу Git через його потужну екосистему, підтримку з боку хостинг-платформ та активну спільноту розробників.

Таблиця 1.1 – Порівняння систем керування версіями Git, SVN і Mercurial

Характеристика	Git	SVN (Subversion)	Mercurial
Тип системи	Розподілена	Централізована	Розподілена
Рік створення	2005	2000	2005
Автор	Лінус Торвальдс	CollabNet	Метт Макал

Продовження таблиці 1.1

Мова реалізації	C	C	Python
Повна локальна копія історії	Так	Ні	Так
Швидкість операцій	Висока	Середня	Висока
Робота з гілками	Гнучка, швидка	Обмежена	Простіша, ніж у Git
Крива навчання	Складна для новачків	Відносно проста	Помірна, зручна для початківців
Платформи	GitHub, GitLab, Bitbucket та інші	Apache, VisualSVN, Assembla	До 2020 – Bitbucket
Популярність	Дуже висока	Стабільно середня	Обмежена, спад після 2020 року
Сценарії використання	Відкриті проєкти, командна розробка	Корпоративні середовища	Академічні/внутрішні проєкти

Існує низка платформ, які стали стандартом у сфері моніторингу та керування IT-проєктами. Серед них варто виділити Jira, Trello, Asana, Monday.com та Wrike, кожна з яких має власні переваги, обмеження і цільову аудиторію.

Jira вважається потужним інструментом для команд, що працюють за гнучкими методологіями, такими як Scrum чи Kanban. Вона надає широкі можливості для побудови спринтів, створення епік-сценаріїв, налаштування статусів завдань та контролю прогресу, що особливо важливо у великомасштабних проєктах. Її гнучкість та глибока інтеграція з іншими сервісами Atlassian робить її затребуваною в корпоративному середовищі.

Trello, натомість, орієнтований на візуальне представлення процесу за допомогою інтерфейсу у вигляді дошок із картками. Його простота та інтуїтивна зрозумілість приваблюють малі команди, стартапи та індивідуальних виконавців. Хоча Trello не має такого функціонального багатства, як Jira, він забезпечує достатню гнучкість для управління нескладними проєктами.

Asana пропонує баланс між простотою та функціональністю. Вона дозволяє створювати завдання, визначати дедлайни, контролювати завантаженість команди,

формувати звіти про виконання. Її інтерфейс орієнтований на користувача, що забезпечує швидку адаптацію до середовища навіть без спеціальної підготовки.

Monday.com – це ще один яскравий приклад сучасної гнучкої платформи, яка надає велику кількість шаблонів для проєктів різного типу. Вона дозволяє створювати кастомні таблиці, автоматизувати робочі процеси та синхронізувати завдання між різними департаментами. Простота налаштувань і візуальна привабливість роблять цю платформу універсальним рішенням для організацій різного масштабу.

Wrike – це платформа, яка більше орієнтована на великі корпоративні структури з багаторівневою організацією робочих процесів. Вона дозволяє деталізувати проєкти, контролювати ресурси, відстежувати витрати часу, будувати складні залежності між завданнями та інтегруватися з системами бізнес-аналітики. Її сильна сторона – звітність та підтримка ролей, що дозволяє ефективно працювати з великою кількістю користувачів.

1.3. Вимоги до інформаційних систем з моніторинговим функціоналом

Функціональні вимоги до моніторингових систем визначають основні задачі, які система повинна виконувати для забезпечення ефективного управління процесами спостереження, збору, аналізу та реагування на події в IT-інфраструктурі. Такі вимоги базуються на структурній побудові інформаційної системи, що поділяється на функціональну та забезпечувальну частини.

На рисунку 1.9 зображено узагальнену структурну схему інформаційної системи, яка включає функціональні підсистеми, комплекси програм та автоматизовані робочі місця як складові функціональної частини. Забезпечувальна частина, у свою чергу, представлена відповідними забезпечувальними підсистемами. Такий поділ дозволяє чітко ідентифікувати призначення кожного компонента, особливо у системах із моніторинговим функціоналом.

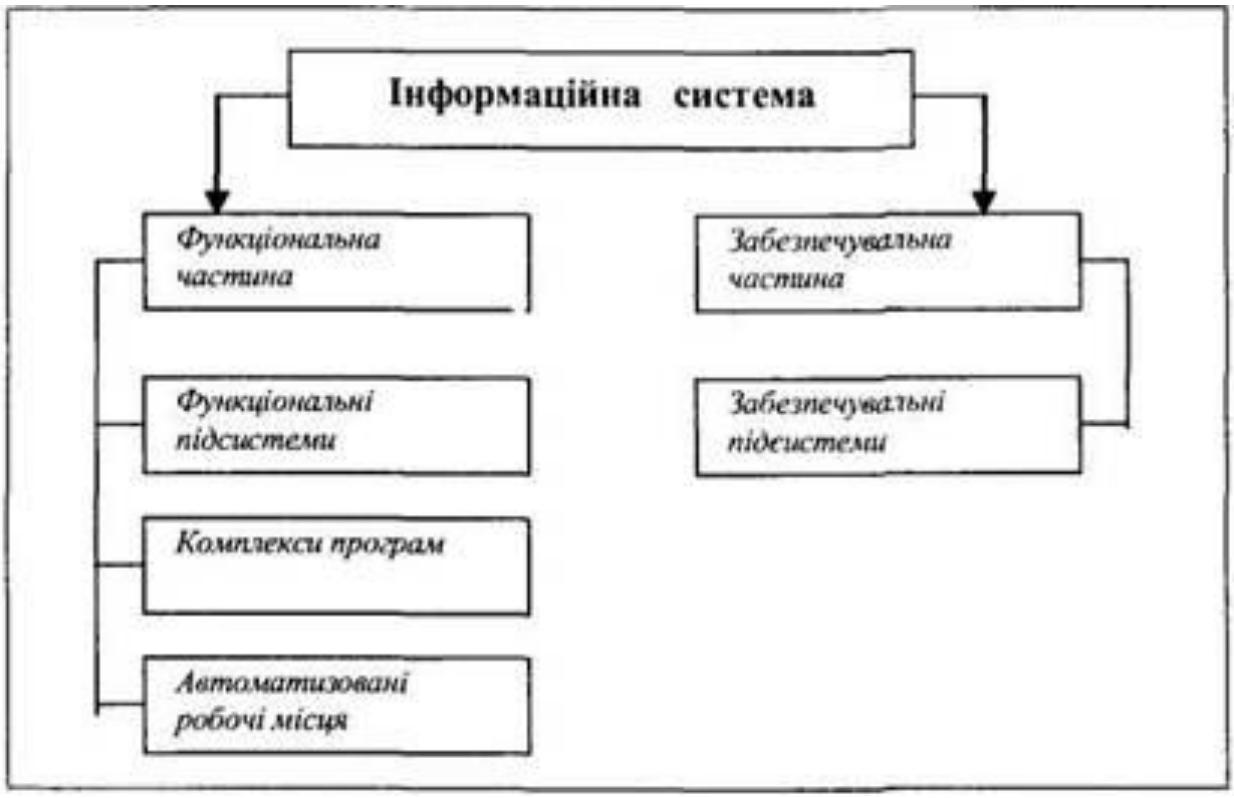

Рисунок 1.9 – Функціональна і забезпечувальна структура інформаційної системи [8]

Основною функцією подібних систем є постійне спостереження за параметрами об’єктів моніторингу – наприклад, серверів, мережевого обладнання, додатків або користувацької активності. Система повинна збирати та зберігати дані про стан об’єктів у реальному часі або з визначеною періодичністю. На основі цих даних виконується аналіз, який дозволяє виявляти відхилення від нормального функціонування.

Крім того, моніторингові системи повинні підтримувати формування звітів, візуалізацію ключових показників (KPI), можливість налаштування сповіщень про події, що вимагають уваги, а також інтеграцію з іншими інформаційними системами підприємства. Однією з важливих вимог є можливість масштабування – тобто розширення кількості об'єктів моніторингу без істотної зміни архітектури.

Надійність, точність, зручність налаштування та адаптивність до конкретних умов експлуатації також відносяться до критичних функціональних вимог, що забезпечують повноцінну роботу моніторингової системи в умовах реального навантаження.

Нефункціональні вимоги до інформаційних систем із моніторинговим функціоналом є не менш важливими, ніж функціональні, оскільки вони визначають якість роботи системи в реальному середовищі, її адаптивність, надійність та ефективність. До основних груп таких вимог належать вимоги до продуктивності, безпеки та масштабованості.

Продуктивність передбачає здатність системи обробляти великі обсяги даних у реальному або наближеному до реального часу. Моніторинг IT-процесів вимагає постійного збору телеметрії, обробки подій, генерації сповіщень та оновлення графіків у панелях адміністратора. Продуктивність системи визначає її здатність реагувати без затримок навіть у критичних навантаженнях, наприклад, при виникненні атаки або збою в інфраструктурі.

Безпека охоплює весь спектр механізмів захисту від зовнішніх та внутрішніх загроз. Моніторингові системи обробляють чутливу інформацію щодо функціонування критичних компонентів IT-інфраструктури, тому потребують реалізації авторизації та автентифікації, шифрування даних при зберіганні й передачі, журналювання дій користувачів та виявлення підозрілої активності. Безпека також передбачає наявність систем резервного копіювання та відновлення.

Масштабованість забезпечує можливість розширення системи без істотної модифікації її архітектури. Це стосується як горизонтального масштабування (додавання вузлів обробки), так і вертикального (збільшення потужностей окремих компонентів). У великих корпоративних мережах або хмарних середовищах моніторинг повинен адаптуватися до динамічних змін кількості пристроїв, сервісів і користувачів.

Для узагальнення ключових характеристик нефункціональних вимог наведено таблицю 1.2.

Таблиця 1.2 – Основні нефункціональні вимоги до моніторингових систем

Категорія	Вимоги
Продуктивність	Висока швидкість обробки подій; мінімальна затримка реакції системи

Продовження таблиці 1.2

Безпека	Шифрування даних; автентифікація; контроль доступу; журналювання
Масштабованість	Підтримка додаткових вузлів; адаптація до зростання об'єктів моніторингу

У сучасних інформаційних системах з моніторинговим функціоналом користувацький інтерфейс (UI) та засоби візуалізації даних відіграють ключову роль у забезпеченні ефективної взаємодії між користувачем та системою. Інтерфейс повинен бути інтуїтивно зрозумілим, адаптивним до потреб користувача та забезпечувати швидкий доступ до необхідної інформації.

Згідно з принципами веб-дизайну інтерфейсу користувача, ефективний UI поєднує в собі приємний зовнішній вигляд та зручність використання. Основні принципи включають [9]:

– знання свого користувача: розуміння потреб і очікувань цільової аудиторії дозволяє інтерфейс, який відповідає їхнім вимогам;

– використання вже відомих моделей: застосування знайомих користувачам елементів інтерфейсу сприяє швидшому освоєнню системи;

– послідовність: однакове розташування та функціональність елементів на різних сторінках системи забезпечує зручність навігації;

– створення візуальної ієрархії: виділення ключових елементів інтерфейсу допомагає користувачам швидко знаходити необхідну інформацію;

– надання фідбеку: система повинна інформувати користувача про результати його дій, що підвищує довіру до системи.

Візуалізація даних є невід'ємною складовою моніторингових систем, оскільки дозволяє ефективно представити великі обсяги інформації у зрозумілому вигляді. Згідно з дослідженнями, візуалізація даних сприяє швидкому виявленню шаблонів, тенденцій та аномалій, що полегшує прийняття обґрунтованих рішень.

Основні вимоги до візуалізації даних включають:

– використання відповідних типів графіків: лінійні графіки, гістограми, карти та бульбашкові діаграми повинні відповідати характеру представлених даних;

– інтерактивність: можливість користувача взаємодіяти з візуалізацією, наприклад, фільтрувати дані або змінювати масштаб, підвищує ефективність аналізу;

– адаптивність: візуалізація повинна коректно відображатися на різних пристроях та екранах;

– доступність: врахування потреб користувачів з обмеженими можливостями, наприклад, забезпечення контрастності кольорів та наявність альтернативного тексту для графічних елементів.

Таким чином, ефективний користувацький інтерфейс та якісна візуалізація даних є критично важливими компонентами моніторингових систем, що забезпечують зручність використання, швидкий доступ до інформації та підтримку прийняття рішень.

1.4. Постановка задачі кваліфікаційної роботи

У сучасному цифровому середовищі управління IT-проєктами вимагає використання ефективних інструментів, які дозволяють не лише планувати задачі, а й забезпечувати їх виконання, контроль прогресу, інтеграцію з іншими сервісами та командну взаємодію. Існує низка платформ, які стали стандартом у сфері моніторингу та керування IT-проєктами. Серед них варто виділити Jira, Trello, Asana, Monday.com та Wrike, кожна з яких має власні переваги, обмеження і цільову аудиторію.

Більшість платформ, попри потужний функціонал, не дозволяють у повній мірі контролювати збереження даних, мають обмежену гнучкість у налаштуванні структури інтерфейсу або не підтримують повноцінне логування подій на рівні окремих користувачів. Це унеможливлює їх повноцінне використання як внутрішньої платформи для контролю процесів у малих проєктах, навчальних лабораторіях або в рамках індивідуальної розробки.

У зв’язку з цим доцільною є розробка власного веб-застосунку для моніторингу IT-проєктів у реальному часі, який забезпечуватиме інтуїтивний інтерфейс, підтримку репозиторіїв, можливість завантаження та перегляду файлів, логування дій користувачів, а також підтримку CLI для адміністрування. Аналіз існуючих рішень показує, що попри наявність розвинених платформ, не завжди вдається знайти універсальне рішення, яке одночасно задовольняє вимоги гнучкості, простоти, продуктивності, безпеки та спеціалізованих функцій. Це відкриває підґрунтя для наступного розділу, в якому буде розглянуто недоліки існуючих систем і обґрунтовано необхідність розробки власного веб-додатку [10].

Незважаючи на широке розповсюдження та функціональність сучасних платформ моніторингу IT-проєктів, зокрема, Jira, Trello, Asana, Monday.com та Wrike, їх використання супроводжується низкою обмежень і недоліків, які можуть ускладнювати ефективне управління проєктами. Однією з основних проблем є складність адаптації нових користувачів до цих систем. Багато з них мають складні інтерфейси та вимагають значного часу на навчання, що може затримувати початок продуктивної роботи нових членів команди. Крім того, деякі системи не підтримують локалізацію, що створює додаткові бар'єри для користувачів, які не володіють англійською мовою.

Ще одним суттєвим недоліком є надлишкова складність і перенасиченість функціоналом. У спробі охопити всі можливі аспекти управління проєктами, деякі платформи стають занадто громіздкими, що ускладнює їх використання для невеликих команд або проєктів з обмеженим обсягом задач. Це може призводити до зниження продуктивності та ефективності роботи.

Крім того, існує проблема відсутності певного функціоналу або його обмежена реалізація. Наприклад, деякі системи не підтримують інтеграцію з іншими інструментами, що використовуються в компанії, або не надають можливості для гнучкого налаштування робочих процесів відповідно до специфіки проєкту. Це обмежує можливості команди адаптувати систему під свої потреби та може призводити до необхідності використання додаткових інструментів, що ускладнює загальну систему управління проєктом.

Також варто зазначити, що деякі платформи мають обмеження щодо масштабованості, що може стати проблемою при розширенні проєкту або збільшенні кількості користувачів. Це може вимагати переходу на інші системи або суттєвого оновлення існуючих, що пов'язано з додатковими витратами та ризиками.

Узагальнюючи вищезазначене, можна зробити висновок, що хоч сучасні платформи моніторингу ІТ-проєктів надають широкий спектр можливостей, їх використання супроводжується рядом обмежень і недоліків, які можуть ускладнювати ефективне управління проєктами. Це підкреслює необхідність ретельного аналізу потреб команди та вибору інструментів, які найбільш повно відповідають специфіці конкретного проєкту [11].

Зважаючи на це, метою кваліфікаційної роботи є розробка веб-застосунку з підтримкою базових функцій моніторингу ІТ-проєктів, який би задовольняв вимоги до зручності, простоти, безпеки, гнучкості, а також забезпечував повний контроль над структурою і логікою обробки даних.

Для досягнення мети необхідно реалізувати наступні задачі:

– дослідити сучасні програмні засоби та середовища для розробки мобільних додатків;

– обґрунтувати вибір технологій, мов програмування та інструментів для реалізації проєкту;

– описати алгоритм проєктування мобільного додатку з урахуванням його функціонального призначення;

– початковий прототип користувацького інтерфейсу для оцінки зручності взаємодії.

– реалізувати архітектуру мобільного додатку згідно з проєктним рішенням;

– розробити інтерфейс користувача відповідно до прототипу;

– забезпечити функціональність основних елементів взаємодії додатку з користувачем;

– провести тестування працездатності програмного продукту та проаналізувати результати перевірки.

Таким чином, розробка індивідуального програмного продукту надає можливість уникнути обмежень, притаманних готовим платформам, та ефективну, безпечну і адаптивну систему управління ІТ-проєктами, яка в повній мірі відповідає специфіці організації.

2. АЛГОРИТМ ПРОЄКТУВАННЯ ВЕБ-ДОДАТКІВ ДЛЯ МОНІТОРИНГУ ІТ-ПРОЄКТІВ

2.1. Вибір архітектури та засобів реалізації

Для реалізації серверної частини веб-додатку було обрано мікрофреймворк Flask, який є легким, гнучким та добре документованим інструментом для розробки веб-застосунків на мові Python. Flask дозволяє швидко розробити RESTful API, обробляти запити, маршрути та управляти сесіями користувачів. Завдяки своїй модульності, Flask надає розробнику можливість самостійно обирати бібліотеки для шаблонізації, роботи з базами даних, автентифікації тощо.

Flask розроблено Арміном Ронахером у 2010 році як експериментальний проєкт на базі Werkzeug і Jinja2. З часом фреймворк здобув широку популярність серед Python-спільноти завдяки своїй простоті, можливості гнучкого налаштування та широкому екосистемному оточенню. Розробник отримує у своє розпорядження мінімалістичний каркас, який можна розширювати лише тими компонентами, які справді потрібні у конкретному проєкті. Це значно зменшує навантаження на систему та полегшує підтримку коду [12].

У межах даного дипломного проєкту Flask використовується для:

– створення маршрутизації між сторінками (маршрути до репозиторіїв, логування, завантаження файлів);

– обробки запитів POST і GET;

– підключення авторизації за допомогою Flask-Login;

– ініціалізації бази даних через SQLAlchemy;

– взаємодії з шаблонами HTML через Jinja2.

Серед переваг, які зробили Flask найбільш придатним для реалізації веб-системи моніторингу ІТ-проєктів:

– легкість старту та низький поріг входу;

– простота інтеграції з іншими бібліотеками;

– підтримка розробки API та інтерактивних форм;

– активна спільнота та велика кількість прикладів використання.

Для забезпечення ефективної взаємодії між серверною частиною веб-додатку та базою даних було обрано бібліотеку SQLAlchemy – потужний інструмент для об’єктно-реляційного відображення (ORM) у Python. Вона надає розробнику можливість працювати з базою даних у вигляді об’єктів, приховуючи низькорівневі SQL-запити, але при цьому не обмежуючи гнучкість і контроль.

SQLAlchemy є однією з найпопулярніших ORM-бібліотек у світі Python-розробки. Вперше представлена Майком Байєрсом у 2006 році, вона швидко завоювала довіру завдяки гнучкому дизайну, підтримці багатьох СУБД та високому рівню абстракції. Основна ідея SQLAlchemy полягає в розділенні ORM-рівня та рівня роботи з SQL-запитами, що дозволяє працювати як з високорівневими моделями, так і писати "сирі" SQL-запити, коли це необхідно [13].

У дипломному проєкті SQLAlchemy виконує такі функції:

– побудова та ініціалізація структури бази даних на основі Python-класів;

– зв’язування таблиць із відповідними моделями (наприклад, User, Repository, File, EventLog);

– керування транзакціями (додавання, оновлення, видалення даних);

– реалізація зв’язків між сутностями (один-до-багатьох, багато-до-багатьох).

Завдяки SQLAlchemy реалізовано можливість легко розширювати схему даних, обробляти взаємодію користувачів із репозиторіями та забезпечувати журналізацію дій через таблицю логів.

Клієнтська частина веб-додатку розроблена з використанням стандартного набору вебтехнологій: HTML, CSS та JavaScript, які відповідають за формування структури, стилів і динамічної поведінки інтерфейсу відповідно.

HTML (HyperText Markup Language) виступає основою всіх сторінок, забезпечуючи семантичну розмітку даних. За його допомогою формуються базові компоненти інтерфейсу – форми реєстрації, кнопки дій, таблиці з файлами, списки репозиторіїв тощо. У структурі проєкту HTML-файли реалізовані як шаблони з підтримкою Jinja2, що дозволяє динамічно передавати змінні з Flask до клієнта [14].

CSS (Cascading Style Sheets) використовується для стилізації сторінок: визначення кольорів, шрифтів, розмірів блоків, а також адаптивності макету. Завдяки каскадній природі стилів та їхньому групуванню проєкт має узгоджене й привабливе оформлення інтерфейсу, а також підтримку темного оформлення для зручності роботи в різних умовах [15].

JavaScript реалізує клієнтську логіку – від обробки форм до взаємодії з елементами сторінки в режимі реального часу. У дипломному проєкті JavaScript використовується для реалізації:

– динамічного оновлення інтерфейсу без перезавантаження сторінки;

– обробки подій (наприклад, підтвердження перед видаленням файлу чи репозиторію);

– інтерактивних елементів, таких як сортування та пагінація в журналі подій;

– розширення можливостей користувача без потреби у встановленні додаткових модулів.

Поєднання HTML, CSS та JavaScript дозволило інтерфейс, який є не лише функціональним, а й інтуїтивно зрозумілим для кінцевого користувача. Важливою особливістю реалізації є використання мінімалістичного та адаптивного дизайну без прив’язки до складних зовнішніх бібліотек, що дозволяє підтримувати швидкість завантаження та зручність користування навіть на мобільних пристроях.

Таким чином, використання стандартного стеку вебтехнологій у поєднанні з серверною логікою на Flask забезпечило повноцінний функціонал, зручний інтерфейс і можливість легкого розширення системи в майбутньому.

2.2. Структура та функціональність веб-додатку

Архітектура веб-застосунку побудована за класичним принципом поділу на серверну логіку (backend), шаблони подання (frontend) та статичні ресурси. Вся

структура збережена у кореневій директорії проєкту й відображена на рисунку 2.1.

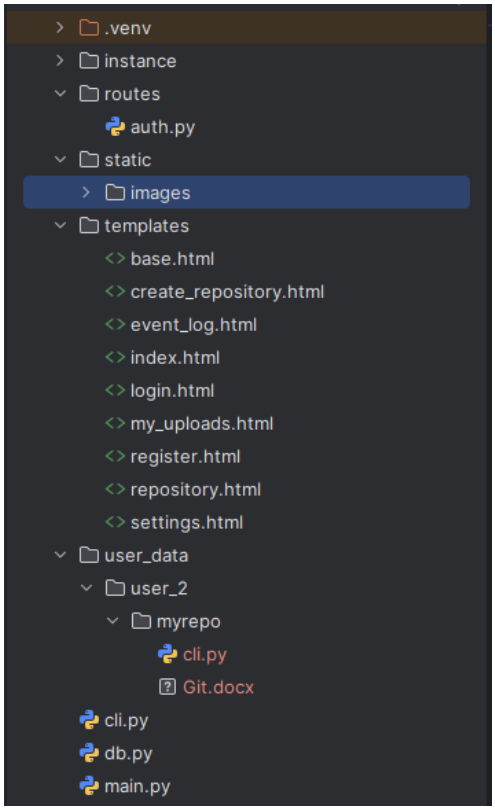

Рисунок 2.1 – Структура директорій і файлів веб-додатку

У корені проєкту розміщено основні модулі:

– main.py – точка входу в додаток із конфігурацією Flask та реєстрацією маршрутів;

– cli.py – реалізує інтерфейс командного рядка (CLI), що дозволяє керувати функціоналом репозиторіїв без вебінтерфейсу;

– db.py – містить моделі бази даних та логіку ініціалізації сховища;

– директорія routes/ включає модуль auth.py, в якому реалізовано маршрути авторизації та реєстрації користувачів.

У майбутньому до цієї папки можуть бути додані інші модулі з маршрутами для окремих функціональних блоків.

Папка templates/ містить HTML-шаблони, які рендеряться сервером за допомогою Jinja2. Основні сторінки включають:

– base.html – базовий шаблон, що містить загальну структуру сторінки, навігаційне меню та стилі;

– index.html – головна сторінка після входу з відображенням усіх репозиторіїв користувача;

– repository.html – перегляд вмісту обраного репозиторію, з можливістю навігації між підкаталогами;

– create_repository.html – форма створення нового репозиторію;

– upload_file.html (опосередковано використовується через repository.html) – завантаження файлів;

– login.html, register.html – сторінки входу й реєстрації;

– event_log.html – відображення журналу дій користувача;

– my_uploads.html – особиста сторінка користувача з усіма завантаженими файлами;

– settings.html – сторінка налаштувань профілю, зокрема для зміни пароля.

Статичні ресурси, такі як стилі, скрипти або зображення, зберігаються у папці static/. У структурі видно підкаталог images/, що містить графічні матеріали для інтерфейсу або логотипи.

Важливим елементом є папка user_data/, у якій фізично зберігаються файли користувачів. Структура реалізована за логікою user_<ID> → назва_репозиторію → файли. Наприклад, user_2/myrepo/cli.py демонструє користувацький файл, завантажений у відповідний репозиторій. Така структура дозволяє легко управляти вмістом, а також забезпечує ізоляцію даних між різними обліковими записами.

Взаємодія з цими папками здійснюється за допомогою функцій у Flask, що автоматично створюють відповідні директорії при реєстрації нового користувача чи створенні репозиторію. Усі дії – створення, завантаження, перегляд, видалення – супроводжуються записом до журналу EventLog, який доступний для перегляду через вебінтерфейс.

На рисунку 2.2 зображено структурну схему веб-додатку, яка демонструє взаємозв’язки між його основними компонентами: клієнтською частиною, серверною логікою, базою даних та зовнішніми сервісами.

Така побудова забезпечує чітке розмежування відповідальностей між модулями, спрощує супровід системи та дозволяє масштабувати окремі її

компоненти за потреби. Схема візуалізує основну архітектурну ідею веб-додатку та дає уявлення про логіку проходження даних у процесі користувацької взаємодії.

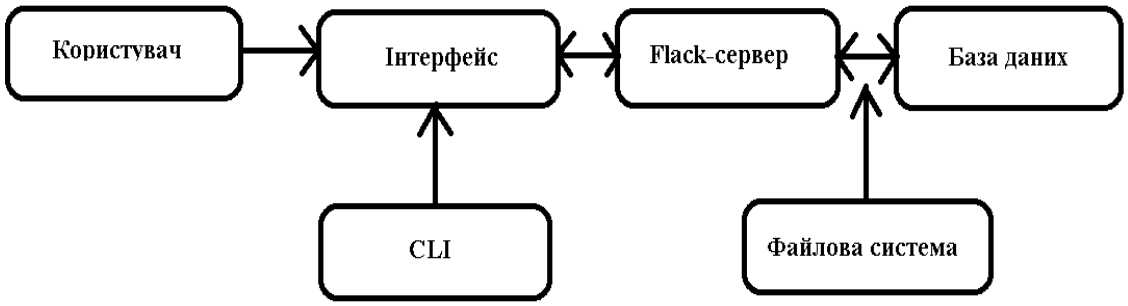

Рисунок 2.2 – Структурна схема веб-додатку

Отже, структура проєкту є логічно організованою, модульною та масштабованою, що дозволяє ефективно розширювати функціональність без порушення основного коду.

Структурну схему моніторингу ІТ-проєктів у реальному часі представлено на КР.КІ.10660340.00.00.000 С1.

2.3. Модель бази даних

У веб-застосунку для моніторингу ІТ-проєктів ключову роль відіграє система управління базами даних, що забезпечує зберігання, доступ і обробку інформації про користувачів, репозиторії, завантажені файли та події. Як засіб взаємодії із базою даних обрано ORM-бібліотеку SQLAlchemy, яка забезпечує зручну абстракцію над SQL-запитами, дозволяючи працювати з даними у вигляді Python-об'єктів.

Структура бази даних спроєктована згідно з реляційною моделлю та складається з чотирьох основних таблиць: users, repositories, files і event_logs. Таблиці між собою зв’язані зовнішніми ключами, що забезпечує цілісність і логічну зв’язність даних. Кожен користувач може мати декілька репозиторіїв,

кожен репозиторій – набір файлів, а всі дії з ними записуються в окремий журнал подій.

На рисунку 2.3 зображено структуру таблиці users, яка відповідає за зберігання облікових даних користувачів. Таблиця містить чотири основні поля:

– id – ціле число, первинний ключ, що автоматично генерується для кожного нового користувача;

– username – рядкове значення до 50 символів, обов'язкове для заповнення, унікальне;

– email – рядок до 120 символів, також є обов'язковим і унікальним значенням;

– password_hash – поле для зберігання хешу пароля, обов'язкове для забезпечення безпеки.

Окрім того, на рівні SQL-запиту реалізовані обмеження унікальності (UNIQUE) для полів username та email, що запобігає дублюванню облікових записів. Первинний ключ (PRIMARY KEY) забезпечує однозначну ідентифікацію кожного запису. Дана таблиця є основою для аутентифікації та авторизації користувачів у системі.

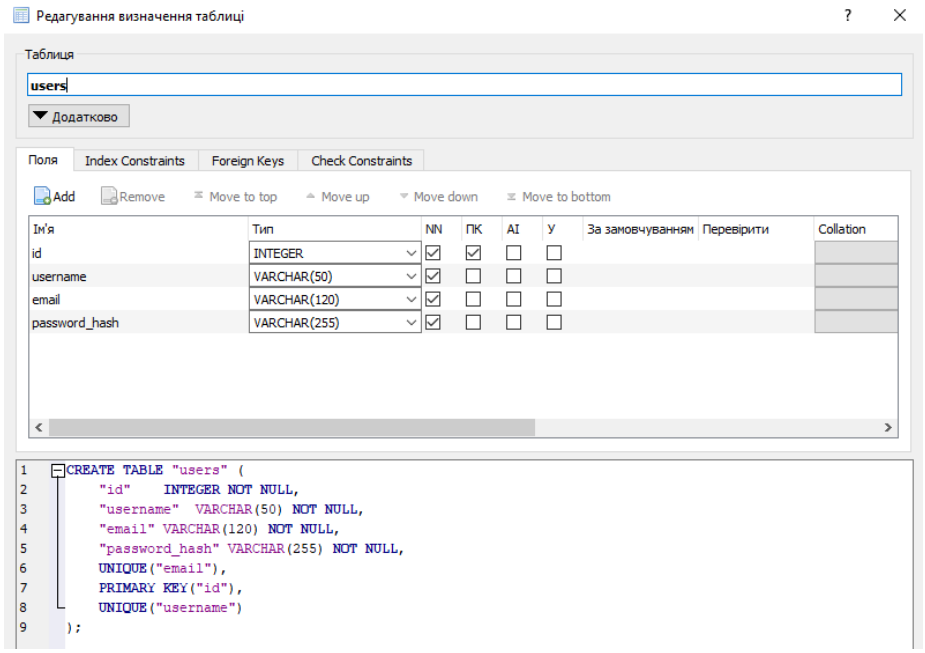

Рисунок 2.3 – Структура таблиці users у базі даних

Таблиця repositories, представлена на рисунку 2.4, відповідає за зберігання інформації про репозиторії, які створюють користувачі в системі. Вона виконує роль центральної сутності для організації файлів у межах окремого облікового запису.

До структури таблиці входять такі поля:

– id – первинний ключ типу INTEGER, що унікально ідентифікує кожен репозиторій;

– name – назва репозиторію (тип VARCHAR(100)), є обов'язковим для заповнення;

– description – текстовий опис, який може містити додаткову інформацію про призначення чи зміст репозиторію;

– created_at – поле з типом DATETIME, що фіксує дату та час створення запису;

– user_id – зовнішній ключ типу INTEGER, який зв'язує репозиторій із конкретним користувачем. Це поле реалізує зв'язок багато-до-одного з таблицею users.

На рівні SQL видно, що user_id посилається на поле id таблиці users, що забезпечує логічну цілісність даних і дає змогу вибірково отримувати репозиторії, які належать певному користувачу.

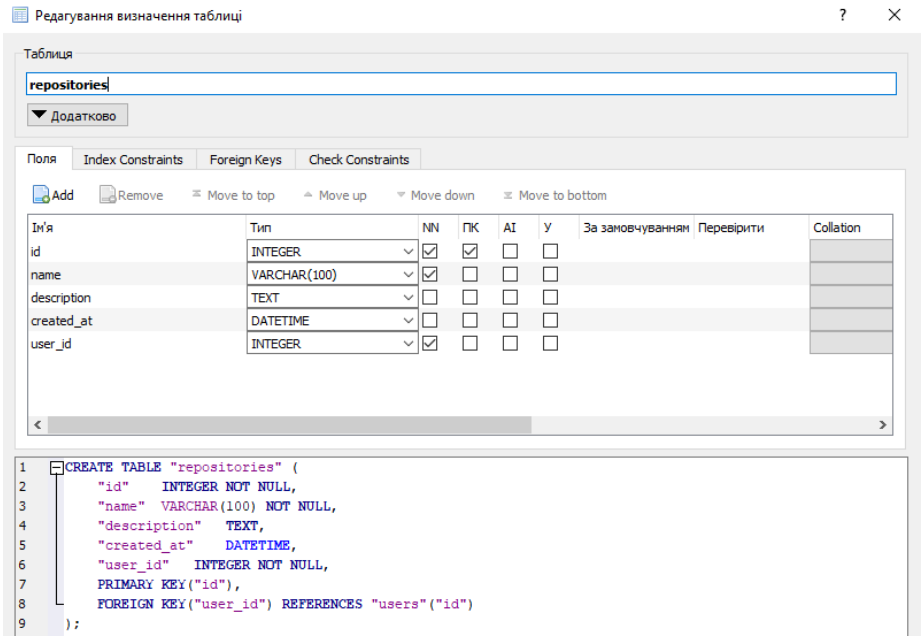

Рисунок 2.4 – Структура таблиці repositories у базі даних

Таблиця files, зображена на рисунку 2.5, виконує роль журналу файлів, завантажених у межах кожного окремого репозиторію. Вона дозволяє зберігати структуровану інформацію про кожен об'єкт, а також підтримує зв'язки як з користувачем, так і з конкретним репозиторієм.

Її структура включає такі поля:

– id – первинний ключ типу INTEGER, який однозначно ідентифікує кожен файл;

– filename – назва файлу (VARCHAR(255)), яку користувач бачить у списку файлів;

– path – шлях до файлу в системі збереження (VARCHAR(500)), який є критично важливим для можливості завантаження та перегляду;

– uploaded_at – дата й час завантаження файлу (тип DATETIME), що дозволяє відстежувати активність;

– repository_id – зовнішній ключ, який вказує на репозиторій, до якого належить файл. Забезпечує зв’язок багато-до-одного з таблицею repositories;

– user_id – зовнішній ключ, який вказує на користувача, що завантажив файл. Забезпечує зв’язок із таблицею users.

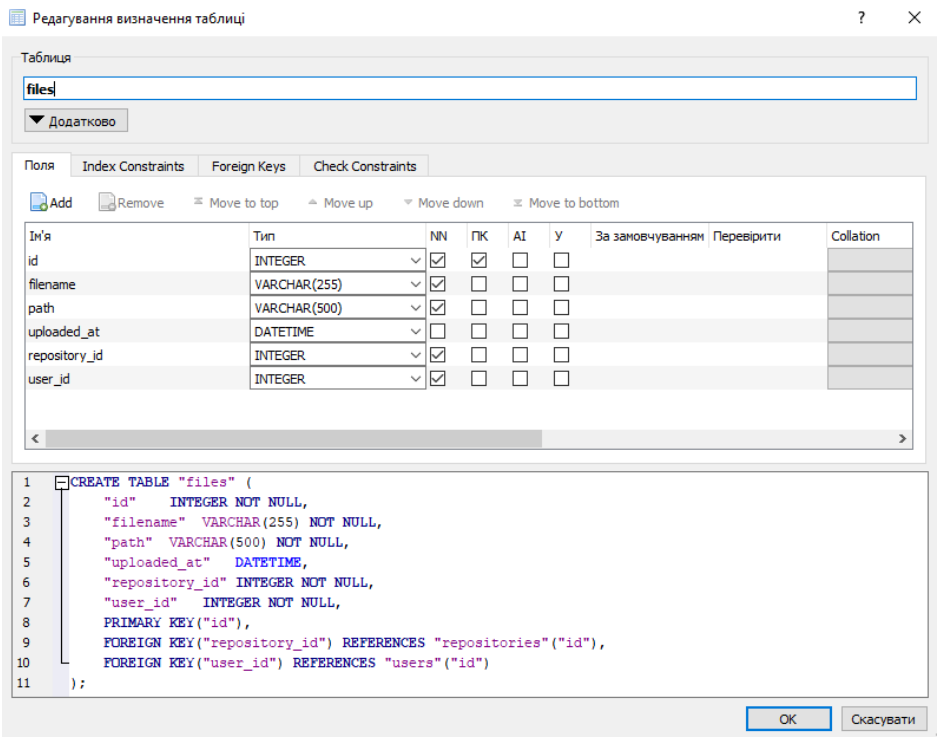

Рисунок 2.5 – Структура таблиці files у базі даних

SQL-запит нижче на рисунку демонструє наявність обох зовнішніх ключів, що гарантує референційну цілісність у відношенні до таблиць repositories та users.

Завдяки таблиці files система може не лише зберігати файли, але й вести детальний облік того, хто, коли і в який репозиторій завантажив конкретний файл. Це також забезпечує можливість реалізації особистих кабінетів із фільтрацією файлів за користувачем.

На рисунку 2.6 представлено структуру таблиці event_logs, яка реалізує функціонал журналювання подій для цілей аудиту, безпеки та зручності аналізу активностей. Це дозволяє адміністраторам або самим користувачам переглядати історію власних дій, таких як створення репозиторіїв, завантаження або видалення файлів тощо.

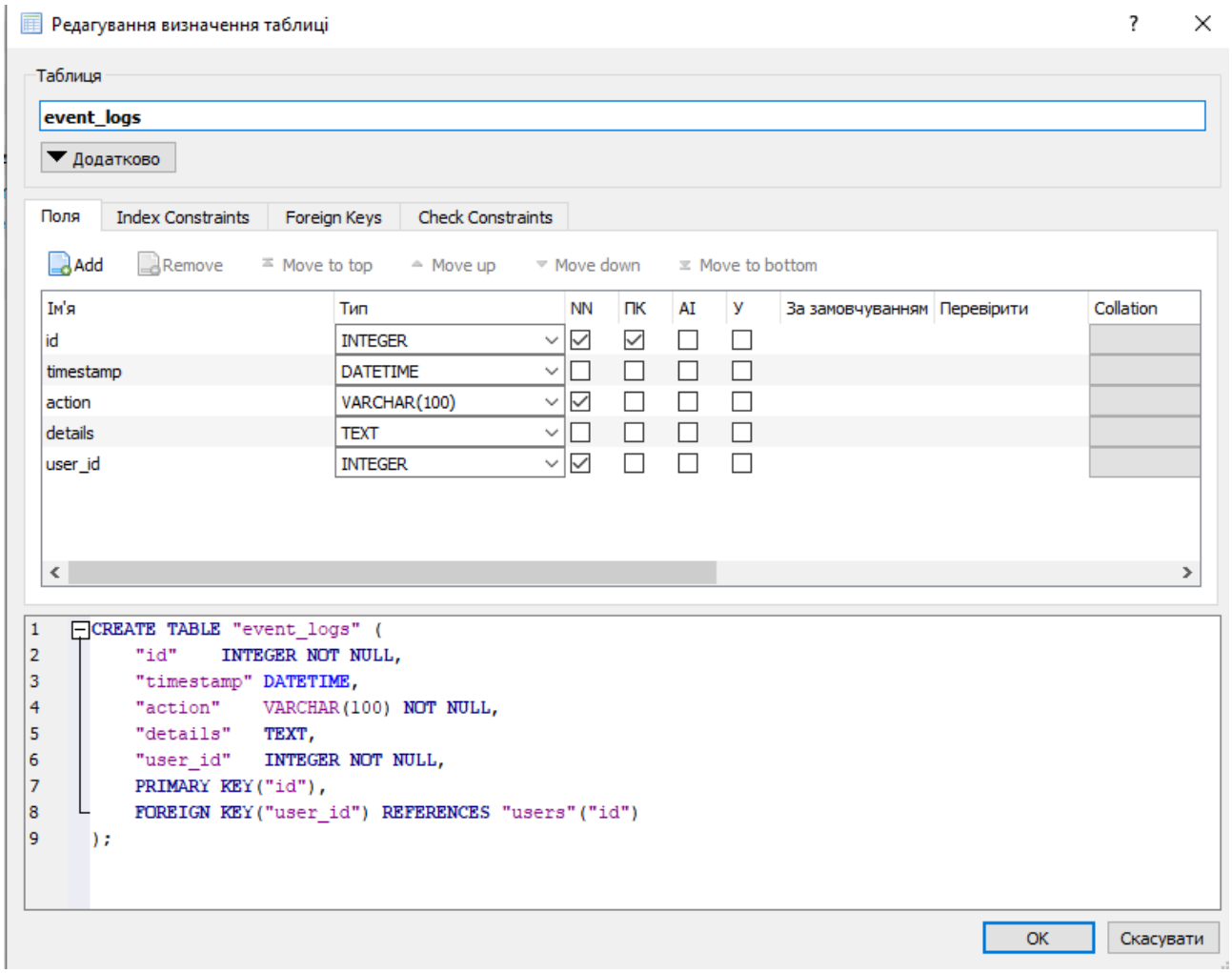

Рисунок 2.6 – Структура таблиці event_logs у базі даних

Таблиця містить такі поля:

– id – первинний ключ типу INTEGER, що однозначно ідентифікує запис події;

– timestamp – мітка часу (DATETIME), яка фіксує момент виникнення дії;

– action – короткий опис типу дії (тип VARCHAR(100)), наприклад: "Створення репозиторію", "Завантаження файлу";

– details – розширений опис події, як-от назва файлу або репозиторію (тип TEXT);

– user_id – зовнішній ключ, що вказує на користувача, який ініціював дію;

– наявність зовнішнього ключа user_id, що посилається на таблицю users, забезпечує цілісність зв'язків між діями та відповідальними за них користувачами.

Завдяки таблиці event_logs веб-додаток отримує повноцінний механізм відстеження та збереження історії змін, що є важливою складовою будь-якої системи з управлінням контентом чи репозиторіями.

Узагальнюючи викладене, можна зробити висновок, що база даних веб-додатку побудована відповідно до принципів нормалізації та забезпечує логічно пов’язану структуру з чітким розмежуванням функціональності. Усі основні об'єкти – користувачі, репозиторії, файли та події – представлені окремими таблицями (users, repositories, files, event_logs) і пов’язані між собою за допомогою зовнішніх ключів.

Таблиця містить такі поля:

– id – первинний ключ типу INTEGER, що однозначно ідентифікує запис події;

– timestamp – мітка часу (DATETIME), яка фіксує момент виникнення дії;

– action – короткий опис типу дії (тип VARCHAR(100)), наприклад: "Створення репозиторію", "Завантаження файлу";

– details – розширений опис події, як-от назва файлу або репозиторію (тип TEXT);

– user_id – зовнішній ключ, що вказує на користувача, який ініціював дію;

– Наявність зовнішнього ключа user_id, що посилається на таблицю users, забезпечує цілісність зв'язків між діями та відповідальними за них користувачами.

Завдяки таблиці event_logs веб-додаток отримує повноцінний механізм відстеження та збереження історії змін, що є важливою складовою будь-якої системи з управлінням контентом чи репозиторіями.

На рисунку 2.7 представлено структуру моделі бази даних, яка використовується для забезпечення роботи веб-додатку. Схема демонструє логічні зв’язки між основними сутностями системи та відображає їх взаємозалежність.

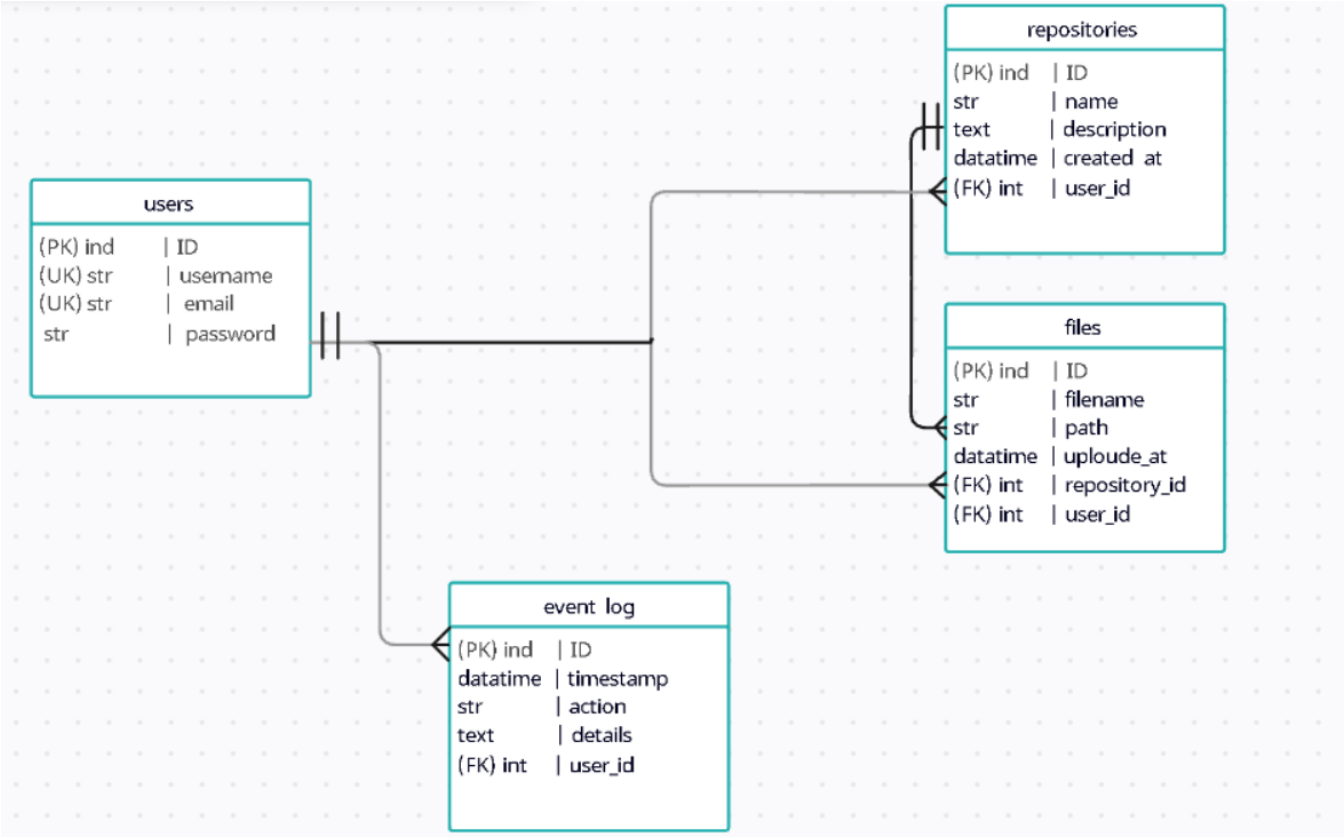

Рисунок 2.7 – Структури моделі бази даних

Узагальнюючи викладене, можна зробити висновок, що база даних веб-додатку побудована відповідно до принципів нормалізації та забезпечує логічно пов’язану структуру з чітким розмежуванням функціональності. Усі основні об'єкти – користувачі, репозиторії, файли та події – представлені окремими таблицями (users, repositories, files, event_logs) і пов’язані між собою за допомогою зовнішніх ключів.

2.4. Система логування подій і взаємодії користувача

Ефективна система логування є важливим елементом будь-якого сучасного веб-додатку, оскільки вона забезпечує контрольованість, підзвітність та можливість ретроспективного аналізу дій користувачів. Саме завдяки логуванню адміністратори можуть своєчасно виявляти помилки, порушення безпеки або несанкціоновану активність. У рамках розробленої системи моніторингу IT-проєктів реалізовано повноцінний механізм фіксації подій, пов’язаних із ключовими діями користувача.

Зокрема, система логування охоплює такі типи дій, як:

- створення, редагування та видалення репозиторіїв;
- завантаження, перегляд або видалення файлів;
- зміну пароля;
- вхід та вихід із системи;
- спроби доступу до захищених розділів без авторизації;
- створення архіву ZIP та інші адміністративні дії.

На рисунку 2.6 представлено фрагмент реалізації функції log_event(), яка відповідає за створення запису про подію в таблиці event_logs. Ця функція приймає два основних параметри: action (назва або тип події) та details (додаткова текстова інформація – назва репозиторію, ім’я файлу тощо). У момент виклику функції автоматично фіксуються також:

- точна дата та час події (timestamp);
- ідентифікатор користувача (user_id), що ініціював дію;
- IP-адреса користувача (опціонально);
- тип клієнта (веб або CLI, якщо реалізовано).

Такий підхід дозволяє не лише відслідковувати дії кожного користувача, але й автоматизувати аналіз подій у разі виникнення інцидентів. У перспективі система логування може бути інтегрована з аналітичними модулями або системами сповіщення, що підвищить рівень проактивного контролю та безпеки.

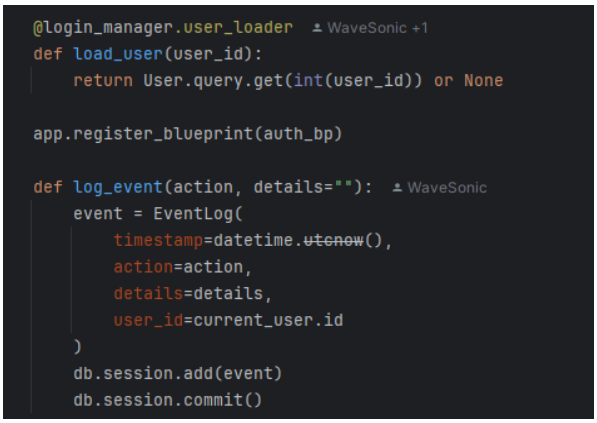

Рисунок 2.6 – Реалізація функції логування подій у веб-додатку

Інтеграція логування відбувається на рівні основних маршрутів Flask-додатку (наприклад, /create_repo, /upload_file, /delete_file, /change_password, /logout) з викликом функції log_event() безпосередньо перед завершенням запиту. Це гарантує, що навіть у разі часткового провалу операції, подія все одно буде зафіксована й доступна для аналізу.

Кожен запис у таблиці event_logs дозволяє ідентифікувати не лише, що саме відбулося, але й ким, коли, де та за яких обставин. Це дає змогу відслідковувати повну історію змін у межах репозиторіїв, а також оцінити навантаження на систему, частоту використання функцій, виявити повторювані помилки, а за потреби – використовувати ці дані як доказову базу при розслідуванні інцидентів.

Журнал подій також може бути розширений у майбутньому для реалізації сповіщень, аналітики або автоматичного реагування на підозрілі дії. Наприклад, при фіксації великої кількості неуспішних входів система зможе автоматично заблокувати обліковий запис або надіслати адміністраторам відповідне повідомлення.

Таким чином, реалізований механізм логування виконує не лише технічну, але й аналітичну функцію, сприяє покращенню якості обслуговування системи, контролю безпеки, та дозволяє будувати прозору й підзвітну систему управління проєктами.

3 РЕАЛІЗАЦІЯ ПРОГРАМНОЇ СИСТЕМИ МОНІТОРИНГУ ІТ-ПРОЄКТІВ

3.1 Реалізація основних маршрутів та логіки бекенду

Побудова серверної частини є фундаментальним етапом розробки будь-якого веб-застосунку, оскільки саме бекенд відповідає за обробку запитів, доступ до даних, автентифікацію користувачів і логіку функціонування системи. У розробленому засобі моніторингу ІТ-проєктів серверну логіку реалізовано на основі мікрофреймворку Flask, який забезпечує мінімалістичне, але достатньо гнучке середовище для побудови REST-сумісних веб-додатків.

Flask надає зручний механізм створення маршрутів, які відповідають за обробку HTTP-запитів клієнта. Кожен маршрут (route) відповідає за певну логічну функцію – наприклад, виведення репозиторіїв, створення нового сховища, завантаження файлу, отримання журналу подій тощо. У системі реалізовано як маршрути з методами GET (для запиту й відображення інформації), так і POST (для надсилання та обробки даних із клієнтської сторони).

Організація серверної логіки базується на використанні Blueprints – механізму, який дозволяє структурувати застосунок за модулями. Це сприяє кращій організації коду, ізоляції відповідальності та спрощенню розширення системи.

Для кожної ключової функціональної області розроблено окремий модуль:

- auth – відповідає за автентифікацію та реєстрацію користувачів;
- repository – забезпечує створення, перегляд і видалення репозиторіїв;
- files – опрацьовує завантаження та збереження файлів;
- events – обробляє перегляд журналу дій.

Комунікація між маршрутом і базою даних реалізована за допомогою ORM SQLAlchemy, що дозволяє працювати з таблицями у вигляді об’єктів. Наприклад, при створенні нового репозиторію формується екземпляр класу Repository, якому присвоюються назва, опис, дата створення й зв’язок з автором. Далі цей об’єкт зберігається у базі даних за допомогою db.session.add() і db.session.commit().

Система збереження даних побудована так, що кожному користувачу відповідає унікальний файловий простір, який ізольовано від інших користувачів. Під час створення репозиторію автоматично створюється відповідна директорія на диску. Таким чином досягається багатокористувацька безпека та впорядкованість структури збережених даних.

Маршрути, які обробляють запити, використовують декоратори Flask – наприклад, @login_required, що перевіряє автентифікацію, або @app.route('/upload/<int:repo_id>/<path:subpath>', methods=['POST']), який відповідає за завантаження файлу у певний каталог. Для захисту імен файлів та запобігання атакам типу Directory Traversal використовується функція secure_filename() з бібліотеки Werkzeug.

Крім безпосередньої обробки запитів, кожна значуща дія в системі (створення репозиторію, завантаження файлу, видалення, зміна налаштувань) логуються у таблицю подій event_logs. Це дозволяє виводити хронологічний журнал активності користувача, що важливо для аудиту та моніторингу роботи системи.

Нижче в таблиці 3.1 наведено узагальнений перелік основних маршрутів веб-застосунку з поясненням їх призначення. Всі вони є частиною загальної логіки системи та відповідають за ключові аспекти взаємодії користувача з платформою.

Таблиця 3.1 – Основні маршрути веб-додатку та їх призначення

Шлях маршруту	Метод	Опис функціональності
/	GET	Перевірка автентифікації та перенаправлення на головну сторінку
/index	GET	Виведення списку репозиторіїв поточного користувача
/create_repository	GET/POST	Створення нового репозиторію з вказаною назвою та описом
/repository/<int:repo_id>/<path>	GET	Перегляд вмісту репозиторію та вкладених папок/файлів
/upload/<int:repo_id>/<path>	POST	Завантаження файлу до певної папки репозиторію

Продовження таблиці 3.1

/download_zip/<int:repo_id>	GET	Архівування всього репозиторію та завантаження ZIP–файлу
/download_file/<int:repo_id>/<filename>	GET	Завантаження конкретного файлу з репозиторію
/delete_file/<int:repo_id>/<filename>	GET	Видалення файлу з файлової системи та запису в базі даних
/delete_repository/<int:repo_id>	POST	Повне видалення репозиторію (файлів та записів)
/event_log	GET	Перегляд журналу дій користувача з підтримкою сортування і пагінації
/my_uploads	GET	Виведення списку всіх завантажених файлів користувача
/settings	GET/POST	Зміна пароля після валідації поточного та підтвердження нового

Побудована система маршрутів виконує роль основного зв’язуючого елемента між користувачем і функціональністю веб-застосунку. Вона реалізує централізовану логіку обробки запитів, що дозволяє не лише передавати дані між клієнтом і сервером, а й ефективно керувати станом користувацьких дій. Кожен маршрут чітко відповідає за певну дію або групу дій, що забезпечує передбачуваність поведінки системи та спрощує її підтримку.

Таким чином, система маршрутів виконує функцію інтерфейсу серверної логіки, який з’єднує зовнішню взаємодію користувача з внутрішніми процесами та операціями над даними. Такий підхід дозволяє забезпечити гнучкість і стабільність роботи навіть у разі підвищеного навантаження або необхідності інтеграції з новими компонентами.

Завдяки використанню захищених запитів, що реалізуються через авторизаційні механізми Flask (зокрема, сесії та декоратори @login_required), кожен маршрут автоматично перевіряє права доступу перед обробкою запиту. Це унеможливлює несанкціонований доступ до критично важливих ресурсів, таких як приватні репозиторії чи конфіденційні файли. Лише автентифіковані користувачі мають змогу виконувати дії, що впливають на структуру або вміст даних.

Окрему роль у побудові бекенду відіграє модульна структура застосунку. Всі маршрути згруповані за функціональними ознаками за допомогою механізму

Blueprint, що дозволяє підтримувати логічну чистоту та масштабованість архітектури. Наприклад, маршрути, пов’язані з автентифікацією, згруповано в одному модулі, тоді як робота з файлами та репозиторіями винесена в окремі компоненти. Це спрощує не лише розробку, а й подальше обслуговування та розширення системи.

Крім цього, застосування ORM-підходу (об’єктно-реляційного відображення) через SQLAlchemy дозволяє оперувати сутностями системи (користувачами, репозиторіями, файлами, подіями) у вигляді Python-об’єктів, не взаємодіючи безпосередньо з SQL-запитами. Це значно підвищує читабельність, безпеку й надійність коду, а також знижує ймовірність помилок на рівні запитів до бази даних. Кожен маршрут, який працює з даними, виконує набір операцій: отримання об’єктів, їх оновлення, додавання або видалення, при цьому підтримується цілісність і узгодженість даних у системі.

Загалом така побудова серверної частини забезпечує високу масштабованість – систему легко доповнювати новими функціональними модулями, змінювати або оптимізовувати наявні маршрути, не порушуючи загальної структури. Безпека гарантується автентифікаційними механізмами та ізоляцією доступу до файлів. Зручність розробки та підтримки досягається завдяки модульності, логічному розділенню відповідальності й абстракції роботи з базою даних. Усі ці фактори роблять серверну логіку розробленого застосунку надійною платформою для подальшого розвитку, впровадження нових функцій і підтримки багатокористувацької взаємодії у реальному часі.

Алгоритмічну схему обробки запитів представлено на КР.КІ.10660340.00.00.000 А1.

3.2. Завантаження, зберігання та перегляд файлів

У веб-додатку для моніторингу ІТ-проєктів однією з найважливіших функціональних підсистем є підсистема керування файлами, яка реалізує механізми

завантаження, збереження, структурованого перегляду та контролю доступу до цифрових артефактів, що належать до проєктів користувачів. Ця підсистема відіграє критично важливу роль, адже дозволяє користувачам працювати з файлами безпосередньо через інтерфейс застосунку, зберігаючи всю необхідну інформацію у контексті створених репозиторіїв.

Механізм завантаження файлів у розробленому веб-застосунку реалізовано через маршрут виду /upload/<int:repo_id>/<subpath>, де repo_id – це унікальний числовий ідентифікатор репозиторію, до якого належить файл, а subpath – відносний шлях у межах структури репозиторію, що вказує на цільову вкладену директорію, куди користувач бажає завантажити файл. Такий підхід дозволяє точно позиціонувати об’єкт у межах ієрархічної файлової структури, яка нагадує класичну файлову систему операційної системи, забезпечуючи зрозумілу логіку зберігання та доступу до файлів.

Коли користувач завантажує файл, веб-додаток спочатку виконує попередню обробку імені файлу за допомогою функції secure_filename() з пакету Werkzeug – стандартного компонента, інтегрованого у Flask. Ця функція забезпечує автоматичну очистку імені файлу від небезпечних символів, пробілів, слешів, зворотних слешів та інших символів, які можуть викликати помилки файлової системи або використовуватись у зловмисних цілях. Завдяки цьому захищено систему від атак типу Directory Traversal, коли хакер може спробувати здійснити доступ до критичних системних файлів шляхом передачі шляху на кшталт ../../etc/passwd.

Після очищення імені та підтвердження цільової директорії система перевіряє, чи існує відповідна структура папок. У разі її відсутності необхідні каталоги створюються автоматично, з використанням методу os.makedirs() з прапором exist_ok=True, що запобігає виникненню помилок при повторному створенні вже існуючих директорій.

Файл зберігається у відповідному місці файлової системи на сервері, а його фізичне розташування прив’язане до користувача, який ініціював завантаження. Така прив’язка реалізується через унікальний ідентифікатор користувача (user ID),

який використовується для створення окремого кореневого каталогу у файловій системі.

Логічне групування за користувачами є надзвичайно важливим для забезпечення приватності, контролю доступу та розділення прав. Завдяки цій структурі, кожен користувач має ізольоване середовище для розміщення своїх проєктів, що унеможливлює випадкове чи навмисне втручання в дані інших користувачів. Крім того, така система значно полегшує виконання адміністративних дій: збереження резервних копій, відновлення окремих репозиторіїв, а також реалізацію механізмів очищення чи міграції даних.

Додатково передбачено можливість створення вкладених директорій усередині репозиторію. Це дозволяє користувачам підтримувати складну, багаторівневу структуру своїх проєктів, поділяючи файли за призначенням (наприклад, /docs, /src, /assets), що покращує навігацію, структурування й обслуговування великих проєктів.

Варто також зазначити, що крім фізичного збереження, кожне завантаження супроводжується записом до бази даних, де фіксується ім’я файлу, шлях до нього, дата завантаження, ідентифікатори користувача та репозиторію. Це дозволяє зберігати повну інформацію про кожен об’єкт у системі, відновлювати його положення, здійснювати пошук, фільтрацію та аналітичний облік.

Такий підхід до організації файлової системи забезпечує фізичну ізоляцію даних, спрощує резервне копіювання, синхронізацію та адміністрування, а також забезпечує масштабованість у разі збільшення кількості користувачів і файлів.

Паралельно з фізичним збереженням об'єкта у файловій системі, у базі даних створюється новий запис у таблиці files, який містить:

- унікальне ім’я файлу;
- відносний шлях до нього;
- дату та час завантаження;
- ID користувача, який здійснив дію;
- ID репозиторію, до якого належить файл.

Ці метадані дозволяють легко ідентифікувати кожен об'єкт, будувати аналітику по завантаженнях, та у разі потреби – швидко знайти потрібні файли за допомогою пошуку або сортування.

Система також реалізує низку захисних механізмів, що включають:

– перевірку дозволених форматів файлів;

– обмеження на максимальний розмір (для запобігання завантаженню надмірно великих або потенційно шкідливих файлів);

– перевірку авторизації користувача перед кожним запитом.

Після завантаження всі файли доступні для перегляду через веб-інтерфейс за допомогою маршруту /repository/<repo_id>/<subpath>. Цей маршрут відповідає за динамічне формування HTML-сторінки, на якій відображається вміст конкретної директорії. Інтерфейс дозволяє користувачеві переглядати вкладені папки, перелік файлів із зазначенням їхніх назв, розмірів та часу завантаження.

Кожен елемент у списку супроводжується інтерактивними кнопками:

– перегляд або завантаження – доступно всім авторизованим користувачам;

– видалення – доступне лише власнику репозиторію.

Це забезпечує контрольовану взаємодію з вмістом проєкту та відповідає принципам багаторівневого доступу до даних.

Особливістю реалізації є також підтримка відображення вмісту як у табличному форматі, так і у вигляді списку, що адаптується до розмірів екрана та дозволяє комфортно користуватися системою з різних пристроїв. Навігація між каталогами реалізована через внутрішні посилання з підтримкою історії переходів, що наближає взаємодію до роботи зі звичайним файловим менеджером.

Кожна дія, пов’язана із завантаженням, переглядом або видаленням файлів, супроводжується автоматичним записом у журнал подій (event_logs). Це дозволяє адміністраторам та самим користувачам відстежувати власну активність, виявляти проблеми або аналізувати внесені зміни у проєкт.

Реалізація підсистеми роботи з файлами у межах цього застосунку відповідає вимогам масштабованості, надійності та безпеки, водночас забезпечуючи зручну взаємодію користувача з репозиторіями та збереженою інформацією. Такий підхід є

гнучким та легко розширюється – наприклад, у майбутньому можливе впровадження попереднього перегляду файлів у браузері, підтримки пошуку за вмістом або інтеграції з хмарними сховищами.

3.3.Інтерфейс користувача

Інтерфейс користувача – це ключовий елемент будь-якого веб-застосунку, що визначає зручність взаємодії користувача із системою, її інтуїтивність та привабливість. У розробленому веб-застосунку «Засіб моніторингу ІТ-проєктів» особлива увага була приділена чистій структурі, адаптивності інтерфейсу та простоті навігації.

Застосунок побудований на основі HTML-шаблонів з використанням фреймворку Bootstrap 5, що забезпечує адаптивний вигляд інтерфейсу на різних пристроях. У шаблоні base.html, який служить основою для всіх сторінок, реалізовано загальну структуру сайту – шапку, бічну панель навігації, динамічний вміст і футер. Шапка сайту містить логотип, назву застосунку та кнопку «Вихід» для авторизованих користувачів. У випадку, якщо користувач неавторизований, виводяться посилання для входу та реєстрації.

Бічне меню навігації дозволяє швидко переходити до основних розділів: «Головна», «Завантаження файлів», «Журнал подій» та «Налаштування». Меню відображається лише для користувачів, що увійшли в систему, що забезпечує простоту інтерфейсу для неавторизованих відвідувачів.

Основний вміст сторінки виводиться в центральній області. Система повідомлень реалізована через Flask-функціонал get_flashed_messages(), що дозволяє інформувати користувача про результат дій (наприклад, успішне створення репозиторію або помилки).

На рисунку 3.1 представлено сторінку «Мої репозиторії». Вона містить таблицю зі списком створених користувачем репозиторіїв, де виводяться назва, опис, кількість файлів, дата створення та кнопка «Видалити». Над таблицею розташована кнопка «+ Новий репозиторій» для створення нового сховища. Всі

дані виводяться динамічно на основі записів у базі даних, пов’язаних із поточним користувачем.

Футер застосунку містить контактну інформацію: електронну пошту служби підтримки та номер телефону, а також назву та розташування організації. Стилізація футера забезпечує єдиний корпоративний стиль, завдяки темному фону та контрастному тексту.

Окрема увага приділена реалізації сторінки «Журнал подій», яка відображає перелік ключових дій, що були виконані користувачем у системі: створення репозиторіїв, завантаження та видалення файлів, зміни пароля тощо. Цей компонент дозволяє користувачу отримати прозору історію взаємодії з системою, що є важливою складовою як з точки зору зручності, так і безпеки.

Сторінка «Завантаження файлів» реалізована з підтримкою перетягування (drag-and-drop) та кнопки вибору файлів. Додатково реалізовано механізм перевірки дозволених форматів файлів і обмеження їх розміру, що забезпечує захист від недопустимого контенту.

На сторінці «Налаштування» користувач має змогу змінити свій пароль, вказавши поточний, новий і підтвердження нового. У разі помилок система інформує про невідповідність або некоректність введених даних, а при успіху – виводиться повідомлення про оновлення. Цей функціонал є невід’ємним елементом системи безпеки й приватності користувача.

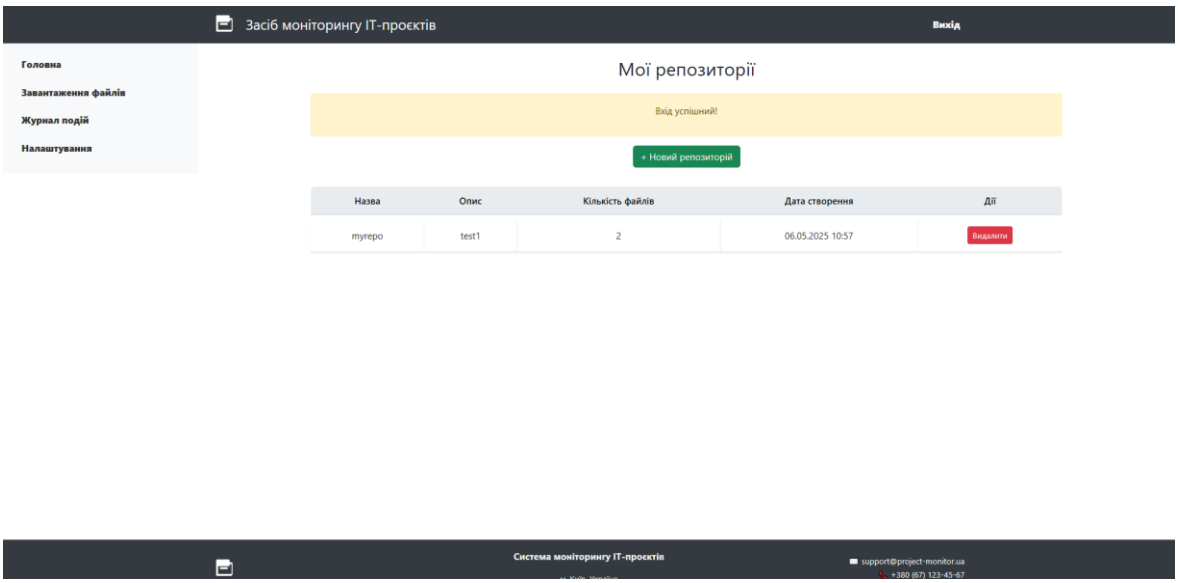

Рисунок 3.1 – Вигляд інтерфейсу користувача з переліком створених репозиторіїв

Загалом, інтерфейс користувача реалізовано з урахуванням принципів юзабіліті, доступності та функціональної завершеності. Це дозволяє ефективно взаємодіяти з функціоналом системи, мінімізуючи час на виконання базових операцій. Застосунок підходить як для досвідчених користувачів, так і для новачків, завдяки інтуїтивній побудові елементів управління та послідовності навігації.

3.4. Аналіз результатів роботи веб-застосунку

Для перевірки надійності та коректної роботи веб-додатку було проведено тестування функціональних модулів на рівні користувацької взаємодії. Метою тестування є впевнитися, що основні сценарії використання системи – такі як реєстрація, авторизація, створення репозиторіїв, завантаження файлів та перегляд подій – реалізовані відповідно до вимог. Кожна функція перевірялась у реальному середовищі запуску з фіксацією як успішних результатів, так і можливих помилок.

Особливу увагу приділено модулям, пов’язаним із введенням даних, перевіркою їх достовірності, взаємодією з базою даних і реакцією інтерфейсу на дії користувача. Усі тести виконувались вручну в браузері із залученням облікових записів тестових користувачів.

На рисунку 3.2 зображено процес реєстрації нового користувача. Користувач вводить логін, електронну адресу, пароль і його підтвердження. Після натискання кнопки «Зареєструватися», система виконує перевірку валідності введених даних, унікальності логіну та email. У результаті успішної перевірки обліковий запис створюється, і користувач автоматично потрапляє в основний інтерфейс системи. Тест підтвердив правильну роботу валідації та взаємодію з базою даних при створенні нового користувача.

Окрім графічного інтерфейсу, система також підтримує взаємодію з користувачем через консоль (CLI-інтерфейс), що є важливою перевагою для розробників та системних адміністраторів. Такий підхід дозволяє швидко

виконувати основні дії без запуску браузера, що підвищує зручність і продуктивність роботи в технічному середовищі.

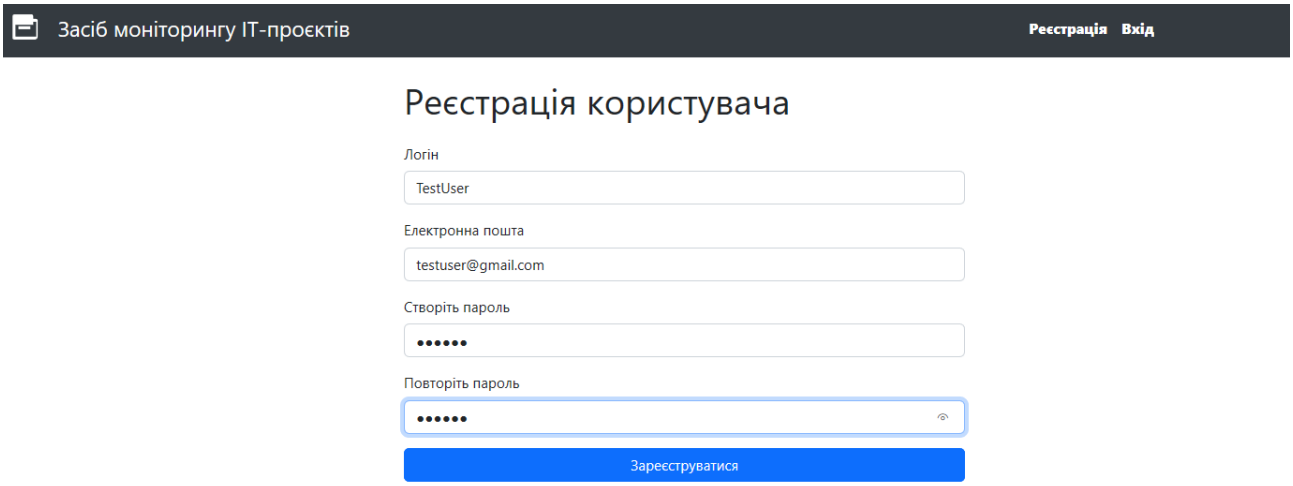

Рисунок 3.2 – Перевірка модуля реєстрації користувача

У межах системи моніторингу IT-проєктів також реалізовано тестову консольну версію інтерфейсу, що дозволяє виконувати основні дії без використання графічного інтерфейсу. Цей інтерфейс запускається через команду python cli.py shell та імітує просту CLI-сесію із підтримкою автентифікації, створення репозиторіїв, завантаження файлів, архівації та перегляду історії подій. Такий інструмент є корисним для системного адміністратора або користувачів, які віддають перевагу роботі з терміналом.

На рисунку 3.3 зображено процес реєстрації нового користувача в CLI-режимі. Після запуску інтерфейсу командного рядка через команду python cli.py shell користувачу пропонується ввести одну з опцій – login або register. У прикладі обрано варіант register, після чого послідовно вводяться логін, email, пароль і його підтвердження. Після валідації система створює обліковий запис і автоматично авторизує користувача. Така можливість підтверджує мультиінтерфейсність додатку та ефективність взаємодії з базою даних навіть поза межами веб-інтерфейсу.

На рисунку 3.4 зображено вивід довідкової інформації щодо підтримуваних команд CLI. Серед них – login та register для входу та реєстрації відповідно, list-

repos для перегляду репозиторіїв, create-repo – для створення нового, upload-file – для завантаження файлу до вказаного репозиторію, download-zip – для завантаження ZIP-архіву, а також view-log – для перегляду журналу подій. Варто зазначити, що ця реалізація є експериментальною та потребує подальшого удосконалення для повноцінної інтеграції з основною веб-частиною системи.

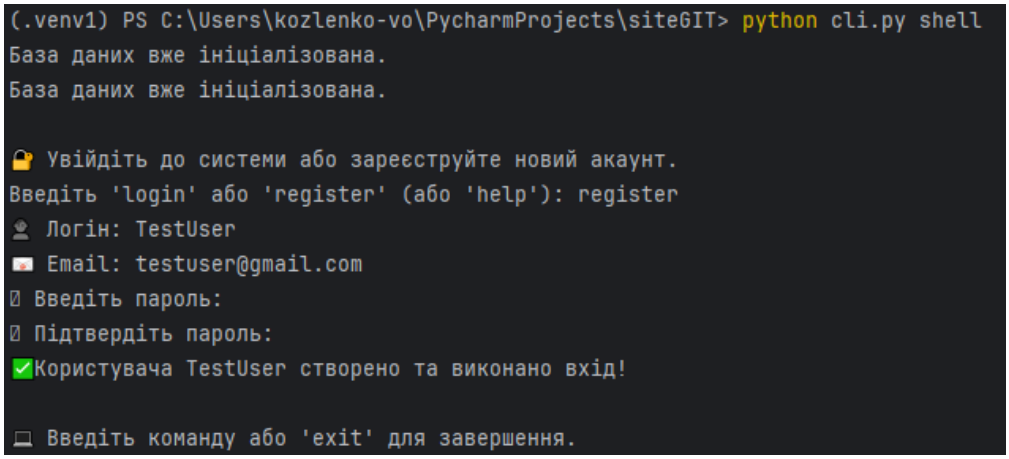

Рисунок 3.3 – Реєстрація користувача через консольний інтерфейс CLI

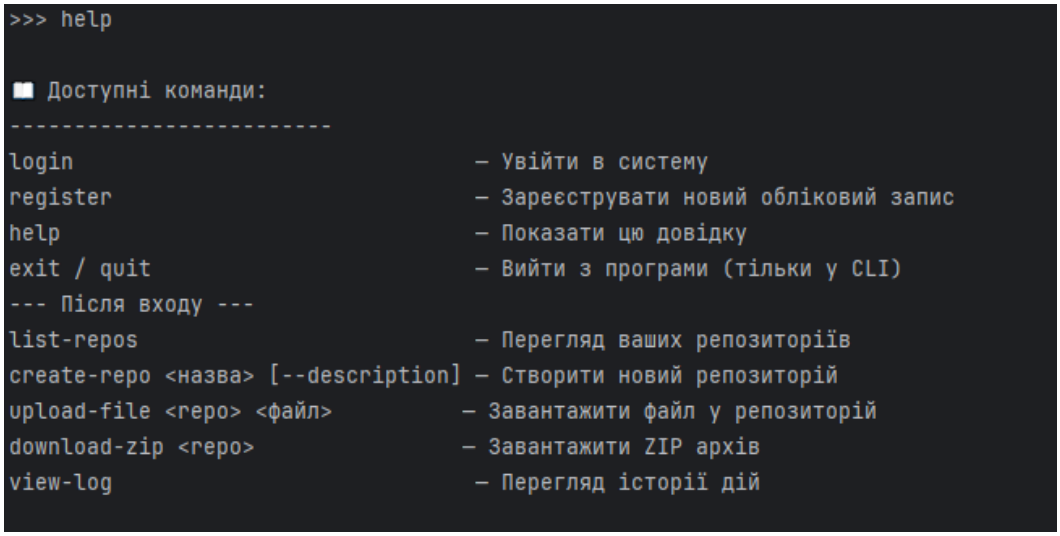

Рисунок 3.4 – Перелік доступних CLI-команд

На рисунку 3.5 зображено інтерфейс створення нового репозиторію у веб-додатку. Користувач має змогу вказати назву репозиторію в текстовому полі, а також за бажанням додати опис – це поле не є обов'язковим для заповнення. Після внесення інформації користувач натискає кнопку «», що ініціює збереження нового

запису в базі даних із відповідним прив’язуванням до користувача. Ця функціональність дозволяє ефективно організовувати структуру проєктів та пов’язаних із ними файлів у межах системи.

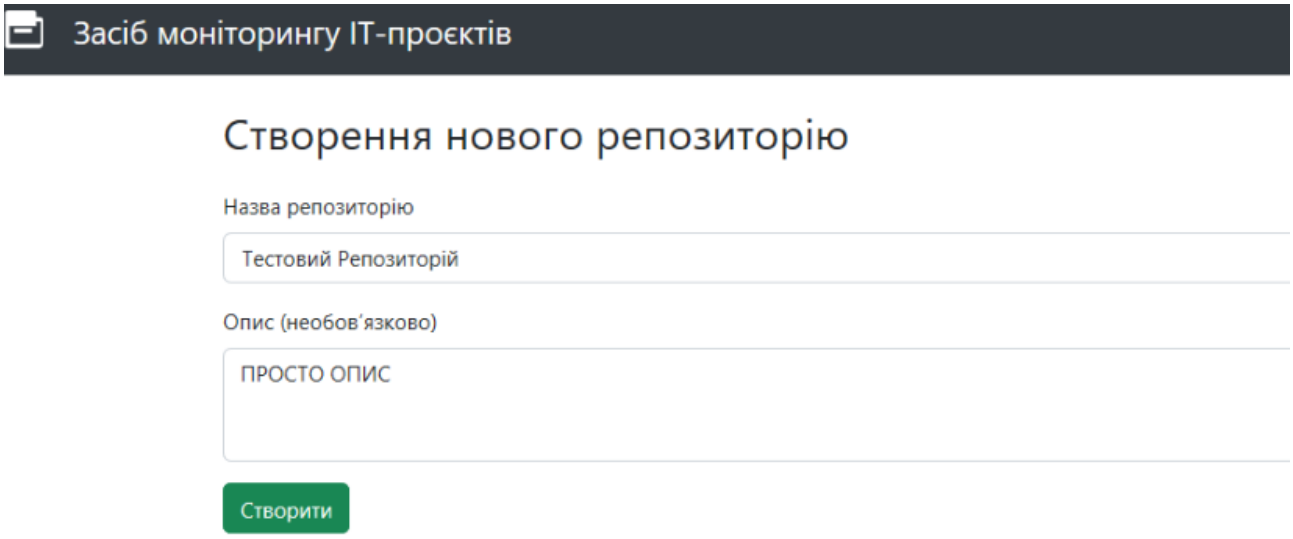

Рисунок 3.5 – Інтерфейс створення нового репозиторію у веб-додатку

На рисунку 3.6 зображено інтерфейс сторінки «Мої репозиторії», що демонструє результат успішного створення нового репозиторію у системі моніторингу IT-проєктів. Після натискання кнопки створення, користувач автоматично перенаправляється на цю сторінку, де відображається підтвердження виконаної дії у вигляді сповіщення «Репозиторій успішно створено!», виділеного жовтим кольором для привернення уваги.

Основна частина інтерфейсу представлена у вигляді таблиці, яка містить назву кожного створеного користувачем репозиторію, його опис, кількість файлів, дату створення та доступну дію – «Видалити». Такий формат дозволяє легко орієнтуватися у списку, зручно керувати проєктами, а також своєчасно виявляти та видаляти застарілі або непотрібні записи. Наявність кнопки «+ Новий репозиторій» додатково заохочує до розширення проєктної діяльності в межах системи. Завдяки структурованому представленню та адаптивному дизайну сторінка є максимально зручною як для початківців, так і для досвідчених користувачів.

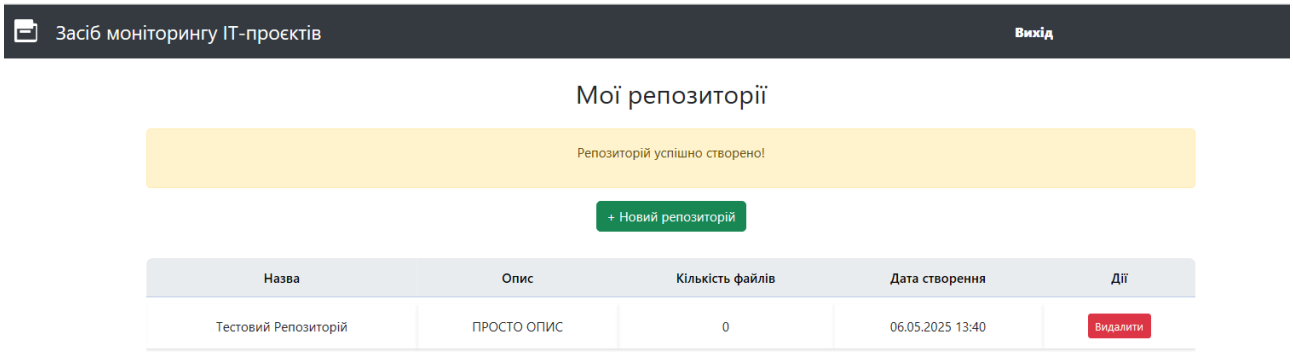

Рисунок 3.6 – Виведення створеного репозиторію у списку користувача

На рисунку 3.7 зображено інтерфейс перегляду вмісту створеного репозиторію під назвою «Тестовий Репозиторій». Цей інтерфейс відкривається після переходу користувача до конкретного репозиторію зі списку. Вгорі сторінки відображено сповіщення про успішне завантаження файлу, що підтверджує правильну роботу механізму додавання об’єктів до системи.

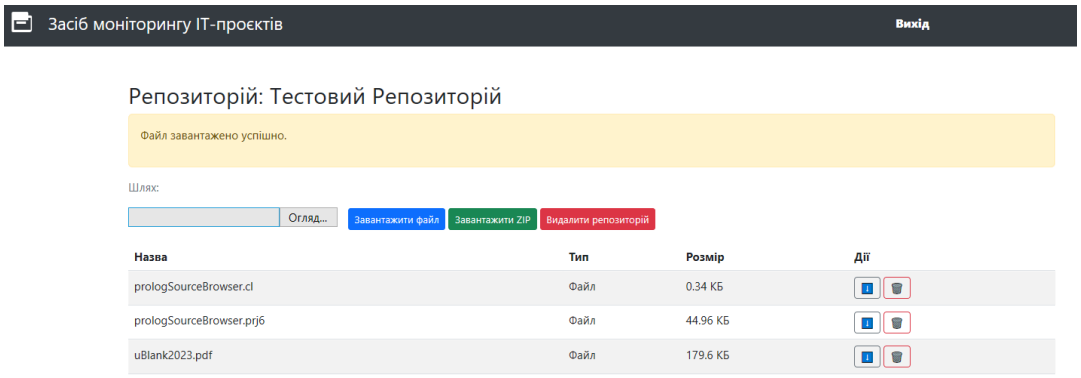

Рисунок 3.7 – Відображення завантажених файлів у межах репозиторію

Нижче розміщено форму для завантаження файлів – із полем введення шляху, кнопкою «Огляд...» для вибору файлу, а також кнопками для завантаження окремого файлу, завантаження ZIP-архіву всього репозиторію або повного його видалення. Під цією панеллю наведено таблицю, що демонструє список файлів, які вже додано до репозиторію. Для кожного файлу зазначено його назву, тип, розмір, а також доступні дії – перегляд (через іконку із синьою кнопкою) і видалення (через іконку смітника). Таким чином, система надає повний контроль над файлами

у межах кожного репозиторію, забезпечуючи зручне управління контентом і гнучку взаємодію з даними.

На рисунку 3.8 зображено процес успішного завантаження ZIP-архіву репозиторію. Користувач після перегляду файлів у межах обраного репозиторію має можливість натиснути кнопку «Завантажити ZIP», що автоматично ініціює формування архіву з усіма файлами. Як видно з нижньої частини знімка, браузер повідомляє про завершення завантаження архіву з назвою Тестовий Репозиторій.zip. Таке тестування підтверджує коректність реалізації функціоналу експорту даних з репозиторію, що є зручним інструментом для резервного копіювання або передавання файлів між системами.

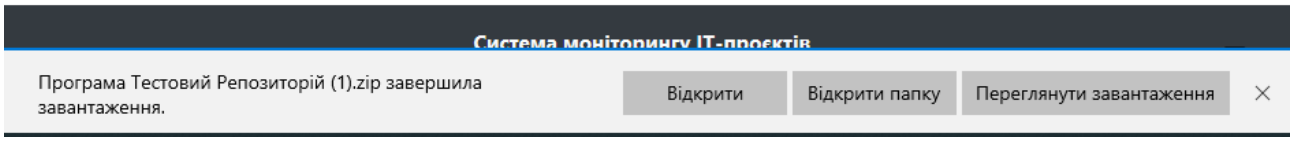

Рисунок 3.8 – Завантаження архіву ZIP із вмістом репозиторію

На рисунку 3.9 зображено інтерфейс журналу подій, який дозволяє відстежувати всі ключові дії, що виконувалися користувачем у системі. Представлена таблиця містить колонки з часом виконання, типом дії та детальною інформацією про неї.

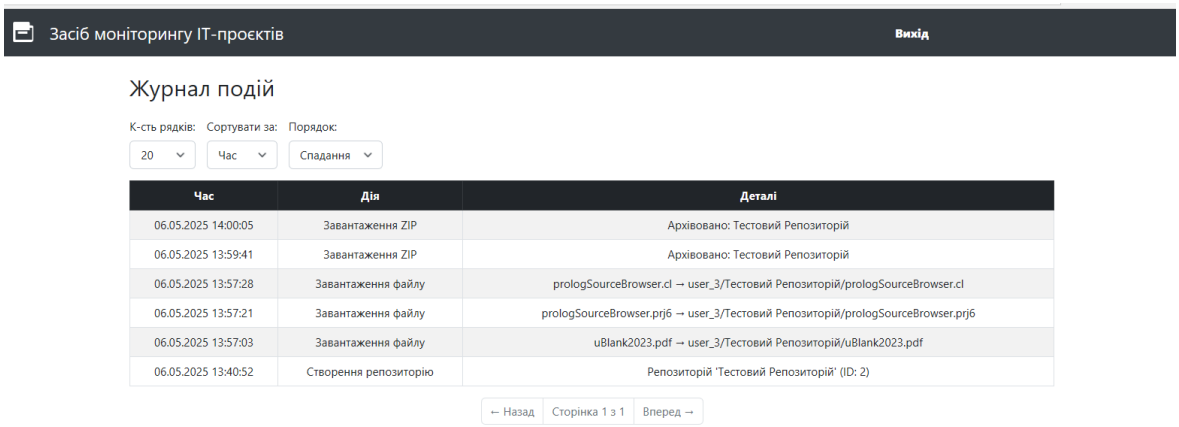

Рисунок 3.9 – Журнал подій із переліком дій користувача в системі

Наприклад, видно, що 06.05.2025 о 13:57:03 було завантажено файл uBlank2023.pdf, а трохи пізніше здійснено архівацію репозиторію. Ця

функціональність є надзвичайно важливою з точки зору аудиту, безпеки та загального контролю за активністю користувачів. Вона дозволяє адміністраторам аналізувати поведінку системи та швидко виявляти потенційно підозрілі чи критичні події.

На рисунку 3.10 зображено сторінку "Мої завантаження", яка надає користувачу зручний доступ до списку всіх раніше завантажених файлів. У таблиці відображається назва файлу, назва репозиторію, шлях до розташування файлу на сервері, розмір та дата завантаження. Крім того, кожен запис містить дві кнопки дій: синю – для завантаження файлу, та червону – для його видалення. Це дозволяє легко управляти власними файлами без потреби шукати їх у репозиторії вручну. Такий підхід сприяє підвищенню зручності користування та покращенню загального досвіду взаємодії з системою.

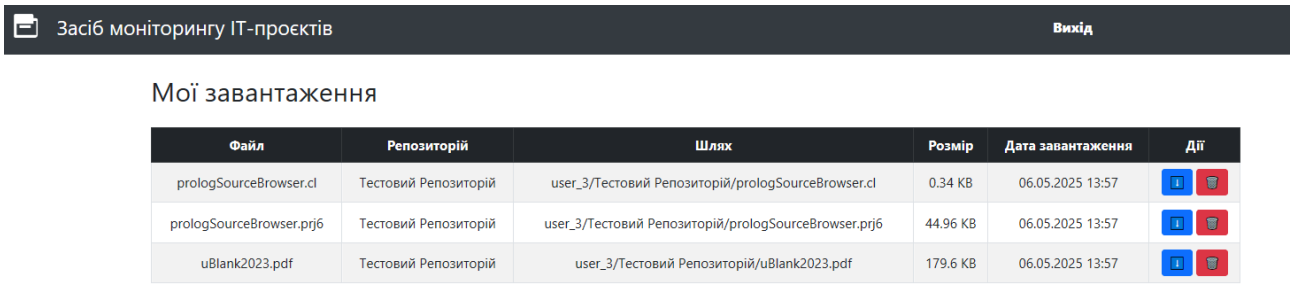

Рисунок 3.10 – Інтерфейс сторінки перегляду власних завантажень користувача

На рисунку 3.11 зображено інтерфейс сторінки "Налаштування профілю", яка дозволяє користувачу змінити пароль. Для цього необхідно ввести поточний пароль, новий пароль і підтвердження нового пароля. Після заповнення всіх полів користувач має натиснути кнопку "Зберегти зміни", що ініціює перевірку правильності введених даних і оновлює облікову інформацію. Цей функціонал є важливим компонентом системи безпеки, оскільки забезпечує можливість регулярної зміни пароля та захисту персонального облікового запису. Крім того, система реалізує валідацію введених даних як на стороні клієнта, так і на стороні сервера, що дозволяє запобігти спробам підбору паролів або введення некоректних значень. У разі успішного оновлення пароля користувач отримує відповідне підтвердження, а всі події фіксуються у журналі безпеки для подальшого аудиту.

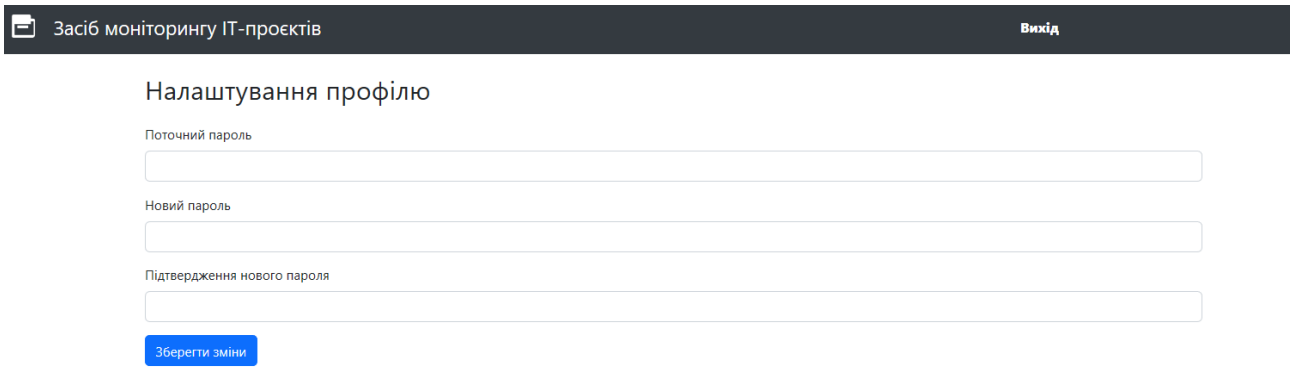

Рисунок 3.11 – Сторінка зміни пароля користувача в налаштуваннях профілю

Крім цього, інтерфейс реалізує валідацію введених даних безпосередньо на стороні клієнта, що дозволяє своєчасно повідомляти користувача про помилки, зокрема, у випадку, якщо новий пароль не відповідає мінімальним вимогам безпеки або підтвердження пароля не збігається з основним полем. У разі успішного оновлення користувач отримує відповідне повідомлення про зміну, що підвищує зручність і прозорість взаємодії з системою. Такий підхід сприяє підвищенню довіри до платформи та відповідає сучасним стандартам захисту облікових записів.

3.5. Тестування роботи веб-застосунку

Проведене тестування підтвердило стабільну та надійну роботу ключових функціональних компонентів розробленого веб-застосунку. Усі базові та додаткові дії, які може виконувати користувач у межах системи, були перевірені – починаючи від первинної реєстрації нового облікового запису до завершення перегляду та аналізу журналу подій, який фіксує активність усіх користувачів. Кожна з протестованих функцій працювала відповідно до попередньо визначених функціональних вимог і очікуваної поведінки.

Користувацький інтерфейс показав стабільну реакцію на взаємодію з боку користувача. Введення даних оброблялось коректно, верифікація проводилась

миттєво, і у випадках введення некоректної інформації система повідомляла про помилки через чіткі, інформативні повідомлення. За наявності коректних даних усі дії виконувалися успішно, включаючи обробку запитів до серверної частини та виведення відповідної інформації на екран користувача.

Особливий акцент було зроблено на аспектах безпеки, зокрема, на перевірці захищеності процесів авторизації. Було підтверджено, що доступ до функціоналу, який потребує автентифікації, можливий лише після успішного входу до системи. Механізми контролю доступу працюють належним чином, забезпечуючи ізоляцію прав користувачів відповідно до їх ролей. Також відзначено коректну взаємодію системи з базою даних у режимі реального часу, з урахуванням операцій читання, запису, оновлення та видалення даних.

У межах функціонального тестування було перевірено кілька сценаріїв, які передбачають створення нових репозиторіїв, їхнє подальше редагування або повне видалення, а також завантаження до системи файлів різних форматів: текстових документів, PDF-файлів, а також проєктних файлів, які можуть бути пов’язані з діяльністю користувача. Також реалізована функція генерації ZIP-архівів, що дає змогу користувачеві експортувати весь вміст обраного репозиторію в один архівований файл. Перевірка показала, що файли зберігаються коректно, доступні для перегляду та повторного завантаження, а також можуть бути безпечно видалені за бажанням користувача.

Важливою перевагою розробленої системи є логічна прив’язка завантажених файлів до конкретного користувача та відповідного репозиторію. Це дозволяє організувати дані у структурованому вигляді та забезпечити контрольований доступ, що є надзвичайно важливим у багатокористувацькому середовищі, де можливі конфлікти через спільне використання даних.

Окремо було протестовано функціонал CLI (Command Line Interface) – інтерфейсу командного рядка, що дає змогу виконувати основні операції в системі без застосування графічного інтерфейсу. Цей режим дозволяє адміністраторам та досвідченим користувачам швидко взаємодіяти із системою. Основні команди виконуються без помилок, однак на цьому етапі розвитку системи зазначений функціонал ще потребує подальшого вдосконалення. Йдеться, зокрема, про

підвищення рівня обробки помилок при введенні некоректних команд, покращення механізмів валідації даних та розширення функціональних можливостей CLI для роботи з більш складними сценаріями.

Таблиця 3.2 – Результати функціонального тестування основних модулів

№	Тестована функція	Очікуваний результат	Фактичний результат	Статус
1	Реєстрація користувача	Користувач створюється та автоматично входить	Користувач успішно зареєстрований та автоматично авторизується.	Успішно
2	Авторизація	Вхід до системи здійснюється при коректних даних	Авторизація успішна, вхід виконано	Успішно
3	Створення репозиторію	Новий репозиторій додається до бази даних	Репозиторій створено та збережено	Успішно
4	Завантаження файлів	Обрані файли прикрілюються до відповідного репозиторію	Завантаження виконано, файли доступні	Успішно
5	Перегляд журналу подій	Виводиться хронологія дій користувача	Журнал подій відображено, дані доступні	Успішно
6	Завантаження ZIP архіву	Формується ZIP-архів із файлами репозиторію	Архів створено та завантажено	Успішно
7	Зміна пароля	Система оновлює пароль користувача після валдації	Новий пароль збережено, підтвердження отримано	Успішно
8	Робота CLI (консоль)	Базові команди (реєстрація, завантаження, перегляд) виконуються коректно	Команди виконуються, виявлено незначні зауваження	Успішно
9	Видалення репозиторію	Репозиторій та пов’язані з ним файли повністю видаляються з системи	Видалення виконано успішно	Успішно
10	Перевірка на неавторизовані дії	Невідомі користувачі перенаправляються на сторінку входу	Захист спрацював, перенаправлення виконано	Успішно

Загалом, за результатами проведеного тестування можна дійти висновку, що веб-застосунок повністю відповідає вимогам, які ставились на етапі проєктування: він є інтерактивною багатофункціональною системою, що дозволяє ефективно

керувати даними, забезпечує безпечну взаємодію з користувачем та має потенціал для подальшого масштабування. Функціональність, інтерфейс та захищеність системи дозволяють рекомендувати її для впровадження в умовах як навчального процесу, так і професійного середовища з реальними користувачами.

ВИСНОВКИ

1.Проаналізовано особливості існуючих систем моніторингу IT-проєктів, визначено їхні сильні та слабкі сторони, а також проведено порівняльний аналіз функціоналу різних варіантів. На основі цього сформовано перелік вимог до застосунку з урахуванням таких параметрів, як зручність використання, безпека, підтримка репозиторіїв та багатокористувацька взаємодія.

2. Розроблено архітектуру веб-застосунку, що включає серверну частину, клієнтський інтерфейс, систему зберігання даних та механізми взаємодії між компонентами. Архітектура передбачає модульність і можливість масштабування.

3. Обґрунтовано вибір інструментів та засобів розробки: для серверної частини використано фреймворк Flask та ORM-бібліотеку SQLAlchemy, що забезпечило швидку розробку та зручну роботу з базою даних. Для побудови інтерфейсу обрано HTML, CSS і Bootstrap, а для взаємодії – REST API.

4. Розроблено прототип користувацького інтерфейсу – побудовано макети сторінок, реалізовано навігацію та основні компоненти, що зробило систему зручною у використанні.

5. Розроблено структуру бази даних та моделі взаємодії між компонентами, що забезпечило збереження інформації про користувачів, проєкти, репозиторії, дії користувачів й дозволяє ефективно керувати процесами в системі.

6. Проведено тестування працездатності та зручності системи, що підтвердило стабільність роботи, зручність інтерфейсу та відповідність функціоналу поставленим вимогам.

7. Здійснено аналіз результатів і визначено можливості для вдосконалення – впровадження командної взаємодії, підтримка сповіщень, розширення прав доступу, інтеграція з хмарними сервісами, підключення зовнішніх API тощо.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Юрчук Н. П. Система моніторингу в управлінні IT-проєктами // Економіка та держава. 2018. № 4. С. 240–245: веб-сайт. URL: http://www.economy.nayka.com.ua/pdf/4_2018/58.pdf (дата звернення 20.01.2025 р.).
2. Birdview PSA. Why Project Monitoring Software is the Best Solution for CEOs to Track Progress and Project Health: веб-сайт. URL: https://birdviewpsa.com/blog/why-pm-software-is-the-best-solution-for-ceos-to-monitor-project-health-and-progress/ (дата звернення 18.04.2025 р.).
3. Components of a Network Monitoring System / ManageEngine: веб-сайт. URL: https://www.manageengine.com/network-monitoring/network-monitoring-components.html (дата звернення 22.02.2025 р.).
4. Пупена А. С. Системи керування версіями. Лекція 7: веб-сайт. URL: https://pupenasan.github.io/ProgIngContrSystems/Лекц/7_git.html (дата звернення 11.03.2025 р.).
5. Brent Laster. Professional Git. John Wiley & Sons, 2017. 480 с.
6. Ben Collins-Sussman, Brian W. Fitzpatrick, C. Michael Pilato. Version Control with Subversion. O'Reilly Media, 2nd Edition, 2021: веб-сайт. URL: https://svnbook.red-bean.com (дата звернення 28.03.2025 р.).
7. Joel Rosdahl. Practical Mercurial. Packt Publishing, 2020. ISBN: 9781784391435
8. Основи проектного менеджменту: веб-сайт. URL: https://buklib.net/books/24045/ (дата звернення 08.02.2025 р.).
9. Принципи дизайну користувацького інтерфейсу: веб-сайт. URL: https://stfalcon.com/uk/blog/post/user-interface-web-design-principles (дата звернення 17.03.2025 р.).
10. Огляд та аналіз систем управління проєктами // Наукові праці Львівської політехніки: веб-сайт. URL: https://science.lpnu.ua/sites/default/files/journal-paper/2024/apr/34358/39.pdf (дата звернення 02.02.2025 р.).

11. Сучасні підходи до оцінки ризиків інформаційних технологій. Управління ризиками IT електронний ресурс / Active Audit Agency: веб-сайт. URL: https://ppt-online.org/172211 (дата звернення 12.02.2025 р.).

12. Antonio Mele. Python Web Development with Flask. Packt Publishing, 2022. ISBN: 9781803238955

13. Rick Copeland. Essential SQLAlchemy: Mapping Python to Databases, 2nd Edition. O'Reilly Media, 2020. ISBN: 9781492047815

14. Bruce Lawson, Remy Sharp. Introducing HTML5, 2nd Edition. New Riders, 2020.ISBN: 9780321965516

15. Eric A. Meyer, Estelle Weyl. CSS: The Definitive Guide, 5th Edition. O'Reilly Media, 2023.ISBN: 9781492054103

16. Чорний В.В., Твердун Я.О. Технології розробки систем моніторингу проєктів та обміну даними в реальному часі // Матеріали II Всеукраїнської науково-практичної конференції «Інтелектуальні комп'ютерні системи та мережі», 20 травня 2025 р. Тернопіль: ЗУНУ, 2025. С. 82–83.

17. Методичні рекомендації до виконання кваліфікаційної роботи з освіт–нього ступеня "Бакалавр" спеціальності 123 «Комп'ютерна інженерія» галузі знань 12 Інформаційні технології / О.М. Березький, Л.О.Дубчак, Г.М. Мельник, Ю.М. Ба–тько, О.Й. Піцун / Під ред. Л.О.Дубчак. Тернопіль: ЗУНУ, 2024. 54 с.

18. Методичні вказівки до оформлення курсових, звітів про проходження практики, випускних кваліфікаційних робіт для студентів спеціальності «Комп'ютерна інженерія» / І.В. Гураль, Л.О.Дубчак / під ред. О.М. Березького. Те–рнопіль: ТНЕУ, 2019. 34 с.

19. Котляревський І. Д. Управління проєктами в інформаційних системах: навч. посіб. Львів: Новий світ 2000, 2018. 224 с.

20. Гринчук Д. В. Сучасні інформаційні технології в управлінні проєктами: монографія Київ: Центр учбової літератури, 2020. 312 с.

21. Жежуха В. Є. Управління проєктами: підручник. Львів: Магнолія 2006, 2019. 384 с.

22. Іващенко О. М. Інформаційні системи і технології в управлінні ор-ганізацією: навч. посіб. Київ: Центр учбової літератури, 2020. 352 с.

23. Бобров Є. А. Управління IT-проєктами: навч. посіб. Харків: ФОП Панов А. М., 2021. 194 с.

24. Калашніков О. О. Інформаційні системи і технології в управлінні: підручник Київ: Каравела, 2022. 336 с.

25. Савченко І. В. Інформаційні технології управління проєктами: навч. посіб. Київ : КНЕУ, 2018. 244 с.

26. Ляшенко В. І. Інформаційні системи і технології в економіці: навч. посіб. Київ : Центр учбової літератури, 2019. 300 с.

27. Баканов С. І. Проєктний менеджмент у сфері IT: навч. посіб. Одеса: ОНЕУ, 2017. 268 с.

28. Кравченко Р. С. Управління проєктами: навч. посіб. Київ: Центр учбової літератури, 2020. 344 с.

29. Бородкіна І. Л., Бородкін Г. О. Інженерія програмного забезпечення: навч. посіб. Київ: Центр учбової літератури, 2021. 204 с.

30. Pressman R. S., Maxim B. R. Software Engineering: A Practitioner's Approach. 9th ed. McGraw-Hill, 2019. 960 p.

31. Schwalbe K. Information Technology Project Management: 9th ed. Cengage Learning, 2018. 672 p

32. The Standish Group. CHAOS Report 2020: веб-сайт. URL: https://www.standishgroup.com/sample_research_files/CHAOSReport2020.pdf (дата звернення 27.04.2025 р.).

33. Atlassian. What is Jira used for: веб-сайт. URL: https://www.atlassian.com/software/jira/guides (дата звернення 17.01.2025 р.).

34. Trello. Getting Started Guide: веб-сайт. URL: https://trello.com/guide (дата звернення 21.03.2025 р.).

35. Asana. Guide to Project Management: веб-сайт. URL: https://asana.com/guide/help/projects (дата звернення 13.02.2025 р.).

36. GitLab. Project Planning with GitLab: веб-сайт. URL: https://docs.gitlab.com/ee/user/project/ (дата звернення 06.02.2025 р.).