Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Методи управління якістю програмних проєктів на основі стандартів ISO

(тема)

Виконав:
здобувач 2 року навчання,
групи СПм-23-3
Олег СОБОЛЬ
(власне ім'я, прізвище)

Спеціальність
123 «Комп'ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма
Системне програмування
(повна назва освітньої програми)

Керівник: доц. Олег ЗАПОРОЖЕЦЬ
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ЕОМ _____ Андрій КОВАЛЕНКО
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 «Комп'ютерна інженерія»
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системне програмування
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
"_____" _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Соболю Олегу Руслановичу
(прізвище, ім'я, по батькові)

1. Тема роботи Методи управління якістю програмних проєктів на основі стандартів ISO

затверджена наказом по університету від " 21 " квітня 2025 р. № 296 Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 16 червня 2025 р.

3. Вхідні дані до роботи 1) міжнародні стандарти управління якістю; 2) програмні метрики якості та моделі їх класифікації; 3) стек технологій веброзробки (PostegreSQL, Node.js, React); 4) структура та процеси життєвого циклу; 5) методи візуалізації даних.

4. Перелік питань, що потрібно опрацювати у роботі
1) аналіз основних міжнародних стандартів управління якістю;
2) дослідження методів класифікації та вимірювання програмних метрик;
3) обґрунтування вибору технологій і підходів до створення системи моніторингу;
4) проєктування та реалізація дашборду для оцінювання метрик якості;
5) тестування та формування висновків.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій

Слайд-презентація – 18 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз міжнародних стандартів управління	22.04.25-25.04.25	
	якістю програмних проєктів		
2	Обгрунтування методології дослідження та	26.04.25-30.04.25	
	побудови системи моніторингу		
3	Вибір і налаштування засобів для розробки	01.05.25-08.05.25	
4	Розробка дашборду для оцінювання метрик	09.05.25-19.05.25	
5	Тестування розробленої системи та оцінка її	20.05.25-31.05.25	
	ефективності		
6	Оформлення матеріалів кваліфікаційної роботи	01.06.25-05.06.25	
7	Подання кваліфікаційної роботи керівникові та	06.06.25-09.06.25	
	її попередній захист		
8	Подання кваліфікаційної роботи на рецензу-	10.06.25-12.06.25	
	вання		

Дата видачі завдання 21 квітня 2025 р.

Здобувач _____ (підпис)

Керівник роботи _____ (підпис) доц. Олег ЗАПОРОЖЕЦЬ (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 100 с., 32 рис., 2 табл., 1 дод., 27 джерел.

ISO/IEC 25010, ISO/IEC 12207, ISO/IEC 27001, ISO 9001, ЯКІСТЬ, МЕТРИКА, КАТЕГОРІЯ, СТАНДАРТ, МОДЕЛЬ ЯКОСТІ, ЖИТТЄВИЙ ЦИКЛ, АНАЛІТИКА, ВІЗУАЛІЗАЦІЯ, ДАШБОРД, АВТЕНТИФІКАЦІЯ, API, PostgreSQL, NODE.JS, REACT, JWT, ВАЛІДАЦІЯ, ЗВІТ, ОЦІНЮВАННЯ, УПРАВЛІННЯ.

Метою кваліфікаційної роботи є дослідження сучасних методів управління якістю програмних проєктів на основі міжнародних стандартів ISO, а також створення прикладного інструменту для збору, аналізу та візуалізації метрик якості.

У ході виконання кваліфікаційної роботи було проаналізовано структурні підходи до побудови моделей якості програмного забезпечення відповідно до ISO/IEC 25010, вивчено процеси розробки та супроводу ПЗ згідно з ISO/IEC/IEEE 12207, а також вимоги до інформаційної безпеки, передбачені стандартом ISO/IEC 27001. Особливу увагу було приділено практичній реалізації системи моніторингу – інтерактивного дашборду, що дозволяє відображати метрики за категоріями якості, обробляти вхідні дані, формувати звіти, керувати параметрами оцінювання та обмежувати доступ через механізми автентифікації.

ABSTRACT

Master’s thesis: 100 pages, 32 figures, 2 tables, 1 appendix, 27 sources.

ISO/IEC 25010, ISO/IEC 12207, ISO/IEC 27001, ISO 9001, QUALITY, METRICS, CATEGORY, STANDARD, QUALITY MODEL, LIFECYCLE, ANALYTICS, VISUALIZATION, DASHBOARD, AUTHENTICATION, API, PostgreSQL, NODE.JS, REACT, JWT, VALIDATION, REPORTING, ASSESSMENT, MANAGEMENT.

The purpose of the qualification work is to study modern methods of quality management of software projects based on international ISO standards, as well as to create an applied tool for collecting, analyzing and visualizing quality metrics.

During the qualification work, structural approaches to building software quality models in accordance with ISO/IEC 25010 were analyzed, software development and maintenance processes in accordance with ISO/IEC 12207 were studied, as well as information security requirements provided for by the ISO/IEC 27001 standard. Particular attention was paid to the practical implementation of the monitoring system - an interactive dashboard, which allows displaying metrics by quality categories, processing input data, generating reports, managing evaluation parameters and restricting access through authentication mechanisms.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

API – програмний інтерфейс прикладного програмування (англ., Application Programming Interface)

CRUD – операції створення, зчитування, оновлення й видалення даних (англ., Create, Read, Update, Delete)

CSV – формат значень, розділених комами (англ., Comma-Separated Values)

DOM – об’єктна модель документа (англ., Document Object Model)

HTTP – протокол передавання гіпертексту (англ., HyperText Transfer Protocol)

IEC – Міжнародна електротехнічна комісія (англ., International Electrotechnical Commission)

IEEE – Інститут інженерів з електротехніки та електроніки (англ., Institute of Electrical and Electronics Engineers)

ISO – Міжнародна організація зі стандартизації (англ., International Organization for Standardization)

JSON – формат обміну даними JavaScript (англ., JavaScript Object Notation)

JWT – компактний формат безпечного передавання інформації (англ., JSON Web Token)

KPI – ключові показники ефективності (англ., Key Performance Indicators)

PDF – формат електронного документа (англ., Portable Document Format)

QA – забезпечення якості (англ., Quality Assurance)

REST – архітектурний стиль для створення веб-сервісів (англ., Representational State Transfer)

SPA – односторінковий застосунок (англ., Single Page Application)

SQL – мова структурованих запитів (англ., Structured Query Language)

UI – інтерфейс користувача (англ., User Interface)

UX – досвід користувача (англ., User Experience)

XSS – міжсайтове виконання сценаріїв (англ., Cross-Site Scripting)

ВСТУП

У сучасній програмній інженерії забезпечення високої якості програмного забезпечення є одним із ключових пріоритетів. Постійне зростання вимог користувачів, стрімкий розвиток технологій і жорстка конкуренція змушують компанії впроваджувати надійні та системні підходи до управління якістю. У цьому контексті особливу роль відіграють міжнародні стандарти ISO, які надають структуровану основу для формування вимог до якості як продукту, так і процесів його створення.

Запровадження стандартів ISO у сфері управління якістю розпочалося ще у 1980-х роках із серії ISO 9000, яка започаткувала універсальну модель системи управління якістю, застосовну в різних галузях, включно з IT. Згодом з’явилися спеціалізовані стандарти, орієнтовані саме на розробку програмного забезпечення: ISO/IEC/IEEE 12207 регламентує життєвий цикл ПЗ, ISO/IEC 25010 визначає модель якості програмного продукту, а ISO/IEC 27001 фокусується на забезпеченні інформаційної безпеки. Їх використання дозволяє організаціям не лише відповідати зовнішнім вимогам, а й оптимізувати внутрішні процеси, зменшити ризики та підтримувати стабільний рівень якості й безпеки.

Водночас впровадження стандартів ISO на практиці часто пов’язане з певними труднощами. Серед основних викликів – складність трактування формулювань, нестача адаптованих інструментів під конкретні проєкти, а також обмежена автоматизація процесів контролю якості. Це формує потребу в глибшому розумінні змісту стандартів і розробці прикладних рішень, які полегшують їх інтеграцію в повсякденну роботу IT-команд.

Метою цієї роботи є дослідження сучасних підходів до управління якістю програмних проєктів з орієнтацією на міжнародні стандарти ISO та розробка інструментального рішення для їх ефективного впровадження. Основну увагу приділено аналізу таких ключових стандартів, як ISO 9001,

ISO/IEC 25010, ISO/IEC/IEEE 12207 і ISO/IEC 27001, а також оцінці їхнього впливу на якість програмного продукту. Окремий акцент зроблено на практичних аспектах інтеграції зазначених стандартів у процес розробки ПЗ, зокрема через створення інструменту моніторингу ключових метрик якості.

В умовах, коли якість цифрових рішень стає критично важливою для успіху компаній, подібні інструменти можуть суттєво підвищити ефективність IT-команд і забезпечити відповідність високим стандартам у розробці програмного забезпечення.

1 ТЕОРЕТИЧНІ ОСНОВИ УПРАВЛІННЯ ЯКІСТЮ ПРОГРАМНИХ ПРОЄКТІВ

1.1 Поняття якості програмного забезпечення

Якість програмного забезпечення – це комплексна характеристика, яка визначає, наскільки програмний продукт відповідає встановленим вимогам, очікуванням користувачів і прийнятим стандартам. Вона охоплює широкий спектр параметрів, включаючи функціональність, надійність, зручність використання та можливість подальшої підтримки. У сучасному світі, де програмне забезпечення відіграє важливу роль у більшості сфер діяльності, його якість стає одним із ключових чинників успіху на ринку [1].

Оцінювати якість програмного забезпечення можна з різних позицій [2]. Відповідно до стандарта ISO/IEC 25010, вона визначається як «ступінь, у якій система, продукт або компонент задовольняє встановленим вимогам і очікуванням користувачів». Таким чином, якість – це не тільки відсутність дефектів, а й здатність програмного забезпечення ефективно виконувати свої функції, забезпечуючи стабільність роботи та комфортність використання.

Якість програмного забезпечення визначається не лише його технічними характеристиками, а й тим, наскільки він відповідає потребам замовників і кінцевих користувачів [2, 3].

Однак у контексті управління якістю важливо також розглядати процеси, що забезпечують досягнення цієї якості, а не лише її характеристики. Це включає [2, 3]:

– планування якості;

– забезпечення та контроль якості;

– аналіз відхилень і впровадження коригувальних дій;

– застосування стандартів як інструментів управління.

Вимоги до якості можуть бути формалізовані у вигляді специфікацій,

стандартів або регламентуючих документів, які описують, яким критеріям повинен відповідати продукт. Наприклад, вимоги можуть включати [3]:

– функціональні вимоги (що саме повинен робити продукт);

– нефункціональні вимоги (як продукт повинен працювати, наприклад, швидкість, надійність, безпека);

– вимоги до інтерфейсу користувача (зручність, доступність).

Якість програмного забезпечення є критично важливою для успішності ІТ-проєкту, адже безпосередньо впливає на задоволеність користувачів, витрати на підтримку та розвиток, а також репутацію розробника. З цієї причини методи управління якістю, що базуються на стандартах ISO, дедалі частіше впроваджуються в процеси розробки – як у класичних, так і у гнучких середовищах.

1.2 Основні підходи до управління якістю в ІТ-проєктах

Управління якістю в ІТ-проєктах охоплює систематичні дії, спрямовані на планування, забезпечення, контроль та постійне вдосконалення якості програмного продукту [4–6]. Ефективне управління якістю передбачає застосування перевірених методів, інструментів і стандартів, що дозволяють досягти стабільного та прогнозованого рівня якості на всіх етапах життєвого циклу проєкту. Особливої актуальності ці підходи набувають у контексті впровадження міжнародних стандартів, зокрема ISO 9001 та ISO/IEC 25010, які слугують методологічною основою для побудови системи управління якістю (СУЯ) [6, 9].

1.2.1 Тестування програмного забезпечення

Тестування є фундаментальним інструментом забезпечення та контролю якості. Воно дозволяє верифікувати відповідність продукту вимогам, виявляти дефекти на ранніх етапах та мінімізувати ризики їх прояву в експлуатації.

Практика управління якістю передбачає планування тестових сценаріїв, визначення критеріїв приймання та використання показників покриття.

Важливу роль відіграє автоматизоване тестування, що забезпечує швидке зворотне повідомлення в умовах ітеративної розробки. У межах стандарту ISO/IEC 29119 визначено ключові принципи організації процесу тестування, які рекомендується враховувати у межах проєктів [7, 18].

1.2.2 Використання стандартів та методологій

Запровадження міжнародних стандартів та методологій, таких як ISO 9001, ISO/IEC/IEEE 12207, CMMI (Capability Maturity Model Integration) і Agile, допомагає структурувати процеси управління якістю [6, 8, 12]. Вони сприяють впорядкуванню розробки, чітко визначаючи її ключові етапи. Наприклад, методологія Agile базується на ітераційному підході, що дає змогу швидко адаптувати продукт до змін і забезпечувати високу якість на кожному етапі розробки.

1.2.3 Управління ризиками

Важливою складовою контролю якості є управління ризиками. Воно дозволяє передбачати потенційні проблеми ще на початкових етапах і вчасно вживати заходів для їх усунення. Цей процес включає ідентифікацію ризиків, аналіз їхнього впливу на проєкт і розробку стратегій для їх мінімізації. Наприклад, якщо існує ймовірність невідповідності продукту очікуванням замовника, команда може провести додаткові тести або залучити експертів для оцінки [13].

1.2.4 Впровадження процесів безперервного вдосконалення

Концепція безперервного вдосконалення (Continuous Improvement)

передбачає регулярний аналіз процесів розробки та їхню оптимізацію для підвищення якості продукту. У практиці Agile це реалізується через ретроспективи, під час яких команда аналізує результати своєї роботи після кожної ітерації. Також застосовуються методики, такі як Kaizen, що спрямовані на поступове покращення процесів та усунення недоліків.

1.2.5 Використання інструментів автоматизації

Автоматизація процесів є ключовим фактором підвищення якості в управлінні програмними проєктами. Інструменти для автоматизації дозволяють швидше виявляти помилки, спрощують тестування та забезпечують стабільність продукту. Наприклад, системи аналізу коду, такі як SonarQube, допомагають виявляти потенційні проблеми ще на етапі розробки, що значно зменшує витрати на виправлення помилок у майбутньому [18, 19].

1.2.6 Взаємодія з користувачами та зворотний зв’язок

Одним із важливих аспектів забезпечення якості є активна комунікація з користувачами та збір їхніх відгуків. Це дає змогу виявляти проблеми, які могли залишитися непоміченими під час тестування, і вдосконалювати продукт відповідно до реальних потреб. Наприклад, бета-тестування, у межах якого продукт випробовують кінцеві користувачі, допомагає оцінити його зручність і функціональність у реальних умовах експлуатації [18].

1.3 Роль стандартів ISO в управлінні якістю програмних проєктів

Стандарти ISO (Міжнародної організації зі стандартизації) відіграють надзвичайно важливу роль у формуванні системного підходу до управління якістю програмного забезпечення [6]. Вони забезпечують уніфіковану, зрозумілу та перевірену базу знань, на основі якої компанії можуть будувати

власні процеси розробки, контролю, вдосконалення та оцінювання якості ПЗ.

Основна мета впровадження ISO-стандартів – створення цілісної системи управління якістю, що охоплює всі етапи життєвого циклу програмного продукту – від аналізу вимог до супроводу після впровадження. Наприклад, стандарт ISO/IEC/IEEE 12207 описує процеси життєвого циклу ПЗ, а ISO/IEC 25010 визначає критерії оцінки його якості, такі як надійність, ефективність, зручність у використанні та безпечність.

Інтеграція таких стандартів у діяльність організації дозволяє уникнути імпровізацій у проєктному управлінні та сприяє формуванню передбачуваних і повторюваних результатів. Важливо, що ISO-стандарти підтримують гнучкість – вони не нав'язують жорстких моделей, а, навпаки, дозволяють адаптувати підходи до конкретних умов компанії та її технологічного стеку.

Стандарти ISO є основою для побудови ефективних систем управління якістю, адже вони надають чіткі рекомендації щодо організації розробки, контролю якості та впровадження заходів для її покращення. Це дає змогу мінімізувати ризики, знизити витрати на усунення дефектів і забезпечити стабільність програмних рішень.

Одна з головних переваг ISO – їхня універсальність. Вони застосовуються в різних сферах, зокрема в IT, що робить їх незамінним інструментом для компаній, які прагнуть до високих стандартів якості. Крім того, впровадження стандартів ISO сприяє уніфікації процесів між різними організаціями, що особливо важливо у контексті міжнародної співпраці.

Використання стандартів ISO у сфері управління програмними проєктами дає кілька ключових переваг [6]:

– систематизація процесів. Стандарти ISO надають чітку структуру для організації процесів розробки та управління якістю. Це дозволяє уникнути хаосу, забезпечити прозорість та підвищити ефективність роботи команди;

– підвищення довіри клієнтів. Сертифікація за стандартами ISO є свідченням того, що компанія дотримується міжнародних стандартів якості. Це підвищує довіру клієнтів, інвесторів та партнерів, що є важливим фактором

успіху на ринку;

– зниження ризиків. Стандарти ISO допомагають виявляти та усувати потенційні проблеми на ранніх етапах, що знижує ризики невідповідності продукту вимогам та зменшує витрати на виправлення помилок;

– покращення якості продукту. Використання стандартів ISO сприяє впровадженню кращих практик, які дозволяють забезпечити високу якість програмного забезпечення та задоволеність користувачів.

2 ОГЛЯД СТАНДАРТІВ ISO ДЛЯ УПРАВЛІННЯ ЯКІСТЮ ПРОГРАМНИХ ПРОЄКТІВ

2.1 ISO 9001:2015 – Вимоги до системи управління якістю

ISO 9001 є одним із найвідоміших міжнародних стандартів, який визначає вимоги до систем управління якістю (СУЯ). Він застосовується до будь-яких організацій, незалежно від їх розміру чи галузі, і спрямований на забезпечення стабільної якості продуктів та послуг [8, 21].

ISO 9001 базується на семи ключових принципах, які формують основу для ефективного управління якістю [8, 21]:

– орієнтація на клієнта – підкреслює важливість розуміння потреб клієнтів та забезпечення їхнього задоволення;

– лідерство – вимагає від керівництва активної ролі у створенні єдиної стратегії та цілей, спрямованих на якість;

– залучення працівників – акцентує на важливості мотивації та участі всіх членів команди у досягненні спільних цілей;

– процесний підхід – передбачає розгляд усіх діяльностей як взаємопов’язаних процесів, що дозволяє оптимізувати ресурси та покращувати результативність;

– постійне вдосконалення – вимагає від організацій регулярного аналізу своїх процесів та впровадження змін для підвищення ефективності;

– прийняття рішень на основі фактів – підкреслює важливість використання даних та аналітики для обґрунтованого управління;

– управління взаємовідносинами – спрямований на побудову довгострокових партнерських відносин із постачальниками та іншими зацікавленими сторонами.

Ці принципи формують основу для створення системи управління якістю, яка забезпечує стабільність, прозорість та відповідність вимогам клієнтів.

2.1.1 Цикл PDCA у контексті ISO 9001

Одним із ключових інструментів, який ISO 9001 пропонує для реалізації цих принципів, є метод PDCA (Plan-Do-Check-Act) [8, 21]. Ця циклічна методика застосовується до будь-якого процесу в організації та включає чотири етапи (рисунок 2.1):

1) Plan (Планування): визначення цілей, процесів та ресурсів, необхідних для досягнення бажаних результатів;
2) Do (Виконання): впровадження запланованих процесів та дій;
3) Check (Перевірка): моніторинг та аналіз результатів для оцінки їх відповідності поставленим цілям;
4) Act (Дія): впровадження коригувальних дій для покращення процесів та досягнення кращих результатів.

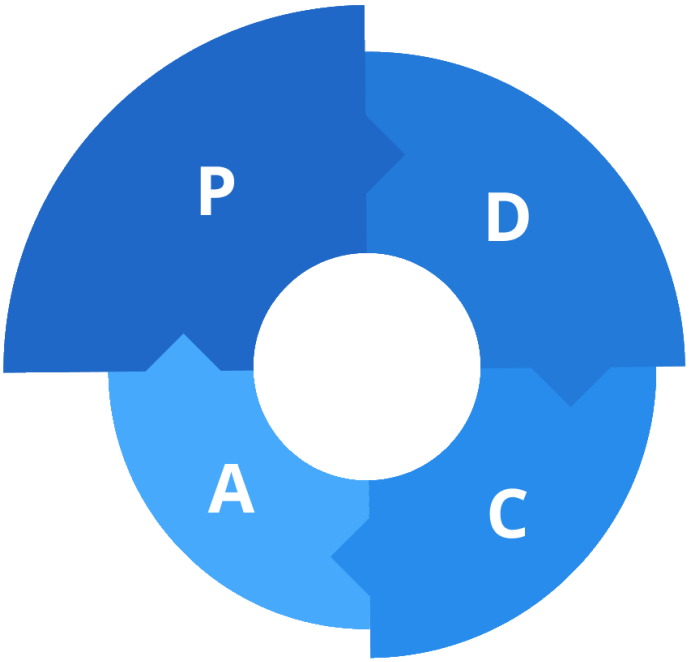


Рисунок 2.1 – Цикл PDCA

Цикл PDCA забезпечує постійне вдосконалення процесів, що є основним принципом ISO 9001. Він дозволяє організаціям систематично аналізувати свої дії, виявляти слабкі місця та впроваджувати покращення.

У програмних проєктах ISO 9001 відіграє ключову роль саме в управлінні якістю – допомагає командам фокусуватися на вимогах клієнтів, забезпечувати їх виконання та формувати прозорі процеси розробки. Це особливо важливо на етапі планування, коли необхідно чітко визначити очікування замовника та забезпечити відповідність продукту цим очікуванням упродовж усього життєвого циклу [8, 9, 21].

Метод PDCA, який є основою ISO 9001, знаходить широке застосування в управлінні якістю програмних проєктів. На етапі планування (Plan) команда аналізує потреби користувачів, формує вимоги до програмного забезпечення, створює архітектуру та розробляє план робіт. Під час виконання (Do) відбувається безпосередня реалізація запланованих дій – програмування, інтеграція компонентів, проведення проміжних перевірок. На етапі перевірки (Check) команда оцінює результат – здійснює тестування, аналізує якість коду, перевіряє відповідність встановленим вимогам. Завершальний етап – дії (Act) – передбачає впровадження змін, таких як усунення дефектів, покращення процесів чи оновлення документації, що сприяє постійному вдосконаленню якості.

Крім цього, впровадження принципів ISO 9001 сприяє покращенню комунікації між учасниками проєкту. Завдяки чіткому документуванню вимог, рішень та відповідальностей, усі члени команди мають доступ до актуальної інформації. Це особливо важливо в умовах Agile або розподілених команд, де відкритість і прозорість є критичними факторами успішного управління якістю програмного продукту.

2.2 ISO/IEC 25010:2023 – Модель якості програмного забезпечення

ISO/IEC 25010 є міжнародним стандартом, який визначає модель якості програмного забезпечення [10, 21]. Цей стандарт є оновленою версією ISO/IEC 9126 і надає детальну структуру для оцінки якості програмних продуктів. Він розглядає якість з двох основних перспектив: якість у використанні та якість

продукту. Ця модель є важливим інструментом для розробників, тестувальників та менеджерів проєктів, оскільки дозволяє систематично оцінювати та покращувати якість програмного забезпечення.

Модель ISO/IEC 25010 включає вісім основних характеристик якості, які поділяються на дві категорії: якість продукту та якість у використанні. Кожна характеристика містить підхарактеристики, що дозволяють детально оцінити різні аспекти якості (рисунок 2.2).

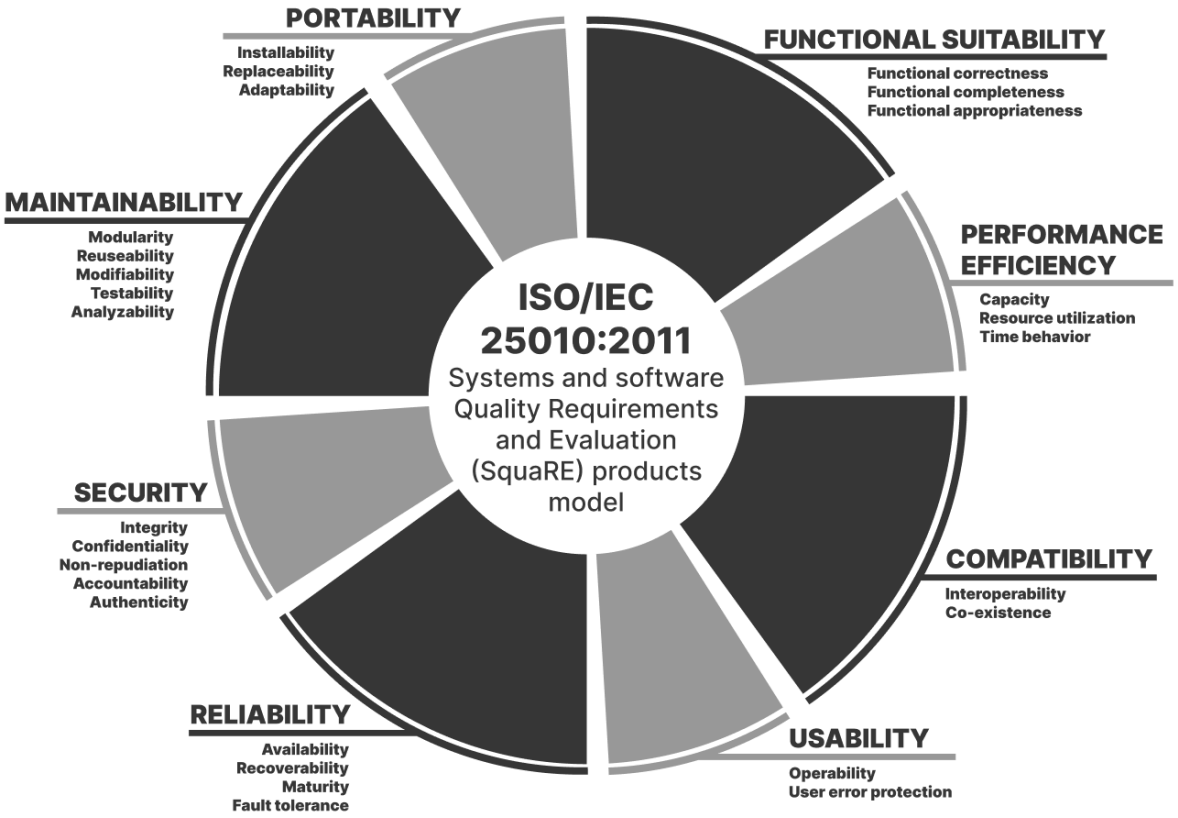


Рисунок 2.2 – Модель якості ISO/IEC 25010

Якість продукту [11]:

– функціональність – здатність програмного забезпечення виконувати заявлені функції відповідно до вимог користувачів;

– надійність – здатність продукту працювати стабільно та без збоїв у різних умовах;

– зручність використання – простота та інтуїтивність інтерфейсу, яка дозволяє користувачам ефективно використовувати програмний продукт;

– ефективність – здатність програмного забезпечення виконувати свої функції швидко та з мінімальним використанням ресурсів;

– сумісність – здатність продукту працювати в різних середовищах та взаємодіяти з іншими системами;

– безпека – здатність програмного забезпечення захищати дані та ресурси від несанкціонованого доступу;

– підтримка – легкість внесення змін та вдосконалень у програмний продукт;

– переносимість – здатність програмного забезпечення працювати в різних середовищах без значних змін.

Якість у використанні:

– ефективність у використанні – здатність користувачів досягати своїх цілей з мінімальними зусиллями;

– продуктивність у використанні – здатність користувачів виконувати завдання швидко та ефективно;

– задоволеність користувачів – рівень задоволеності користувачів від використання програмного продукту;

– відсутність ризиків – здатність продукту мінімізувати ризики для користувачів та їхнього оточення.

Модель ISO/IEC 25010 є важливим інструментом для оцінки та покращення якості програмного забезпечення. Вона дозволяє командам визначати ключові аспекти якості, які потребують уваги, та розробляти стратегії для їх покращення [10, 11]. Наприклад:

– на етапі планування проєкту модель може використовуватися для визначення вимог до якості продукту;

– під час розробки вона допомагає оцінювати якість окремих компонентів та виявляти потенційні проблеми;

– на етапі тестування модель слугує основою для оцінки відповідності продукту вимогам.

2.3 ISO/IEC/IEEE 12207:2017 – Процеси життєвого циклу програмного забезпечення

Стандарт ISO/IEC/IEEE 12207 описує процеси життєвого циклу програмного забезпечення, включаючи планування, розробку, тестування, впровадження та підтримку. Він забезпечує структурований підхід до управління проєктами, що особливо важливо для великих та складних програмних систем [12, 21].

Основні процеси ISO/IEC/IEEE 12207 (рисунок 2.3) [12, 21]:

– перша категорія – процеси узгодження. Стосуються взаємодії між організаціями, які виступають у ролі замовників (acquirers) та постачальників (suppliers) програмних систем. Ці процеси забезпечують укладення угод, які дозволяють обом сторонам отримувати вигоду та підтримувати бізнес-стратегії. Узгоджувальні процеси можуть застосовуватися як поза межами життєвого циклу проєкту, так і під час його реалізації. Вони також можуть використовуватися всередині однієї організації для визначення відповідальностей між різними функціональними підрозділами. Ці процеси забезпечують чітке розуміння цілей проєкту, вимог до продукту чи послуг, а також розподіл обов’язків між замовником і постачальником. Наприклад, процес придбання (acquisition process) включає визначення вимог до програмного продукту та вибір постачальника, тоді як процес постачання (supply process) забезпечує відповідність продукту вимогам замовника;

– друга категорія – організаційні процеси сприяння проєкту. Спрямовані на забезпечення ресурсів, необхідних для реалізації проєктів, які відповідають потребам та очікуванням зацікавлених сторін. Ці процеси діють на стратегічному рівні та стосуються управління бізнесом організації, розподілу ресурсів, управління ризиками та покращення процесів. Організаційні процеси створюють середовище, в якому реалізуються проєкти. Вони включають встановлення процесів та моделей життєвого циклу, надання ресурсів (людських, фінансових тощо), а також визначення та моніторинг показників

якості програмних систем та інших результатів проєктів. Ці процеси однаково важливі як для комерційних, так і для неприбуткових організацій, оскільки всі вони несуть відповідальність перед зацікавленими сторонами та стикаються з ризиками у своїй діяльності;

– третя категорія – процеси технічного управління. Стосуються управління ресурсами та активами, виділеними організацією, для виконання угод та досягнення цілей проєкту. Вони включають планування витрат, термінів та результатів, контроль виконання заходів для забезпечення відповідності планам та критеріям ефективності, а також виявлення та вибір корективних дій для усунення недоліків у прогресі та досягненнях. Ці процеси використовуються для створення та виконання технічних планів проєкту, управління інформацією в команді, оцінки технічного прогресу щодо планів розробки програмних систем, продуктів чи послуг, а також для підтримки прийняття рішень. Процеси технічного управління застосовуються під час виконання кожного технічного процесу;

– четверта категорія – технічні процеси. Охоплюють технічні дії, які виконуються протягом усього життєвого циклу програмного забезпечення. Вони трансформують потреби зацікавлених сторін у продукт чи послугу, які забезпечують стабільну роботу та задоволення вимог клієнтів. Технічні процеси застосовуються для створення та використання програмних систем, незалежно від їх форми (модель чи готовий продукт) та етапу життєвого циклу. Вони включають аналіз бізнес-потреб, визначення вимог, проектування архітектури, розробку, тестування, впровадження, верифікацію, валідацію та підтримку програмного продукту.

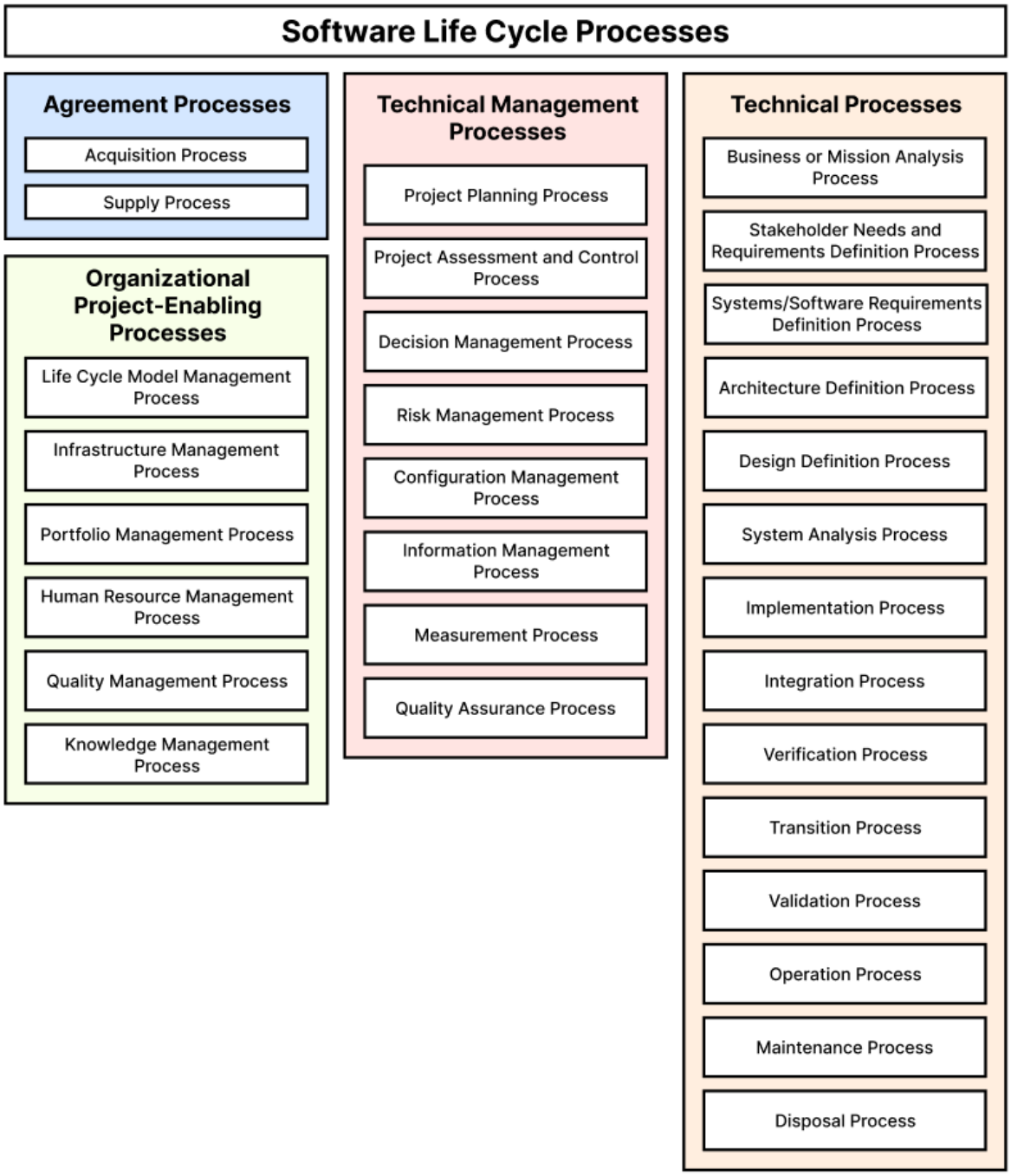


Рисунок 2.3 – Процеси життєвого циклу

2.4 ISO/IEC 27001:2022 – Система управління інформаційною безпекою (ISMS)

ISO/IEC 27001 – найбільш визнаний у світі стандарт для створення, впровадження та постійного удосконалення системи управління інформаційною безпекою (ISMS) [13]. Документ задає вимоги до політик,

процесів, ресурсів і контролів, які забезпечують конфіденційність, цілісність та доступність інформаційних активів організації, а також інтегрує управління ризиками в загальну систему менеджменту компанії [13].

Впровадження ISMS за ISO/IEC 27001 починається з визначення контексту організації та зацікавлених сторін, після чого виконується систематичне оцінювання ризиків: ідентифікація інформаційних активів, аналіз загроз і вразливостей, розрахунок потенційного впливу та ймовірності. Результатом стає план обробки ризиків і «Заява про застосовність» (SoA), де перераховано вибрані контролі Annex A. Наступні етапи включають впровадження політик і процедур, внутрішній аудит, аналіз з боку керівництва та зовнішній сертифікаційний аудит, який підтверджує відповідність вимогам стандарту та допускає видачу сертифіката (рисунок 2.4) [13, 14].

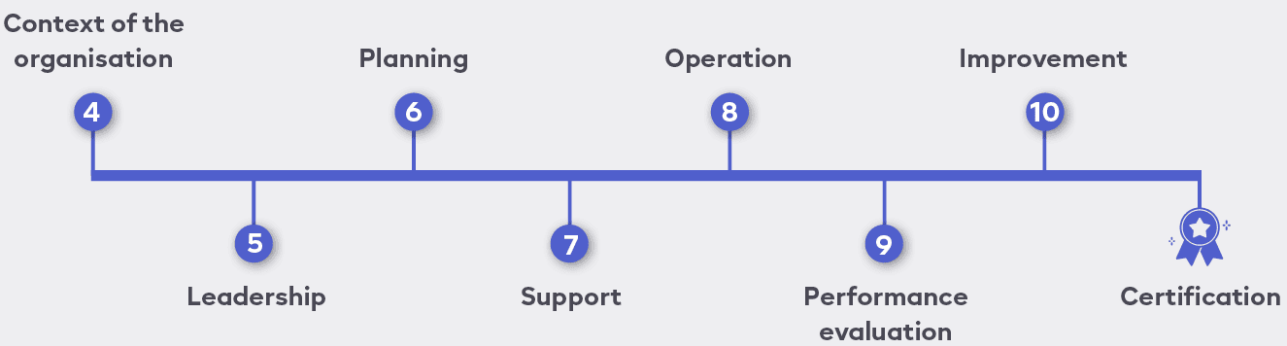


Рисунок 2.4 – Вимоги та структура стандарту

Хоча ISO/IEC 27001 фокусується на інформаційній безпеці, його інтеграція в систему управління якістю ПЗ забезпечує цілісний підхід: від вимог до безпеки на етапі проєктування до захисту даних користувачів під час експлуатації. Впровадження ISMS підтримує вимоги ISO 9001 щодо управління ризиками та документування процесів, а також доповнює модель ISO/IEC 25010, розширюючи атрибут «безпека» конкретними мірами контролю. Таким чином, ISO/IEC 27001 слугує невід'ємним елементом комплексної стратегії забезпечення якості та довіри до програмного продукту.

3 МЕТОДИ ВПРОВАДЖЕННЯ СТАНДАРТІВ ISO У ПРОГРАМНИХ ПРОЄКТАХ

3.1 Планування якості в програмних проєктах

Планування якості є невід'ємною складовою управління програмними проєктами. Воно охоплює визначення цілей, встановлення вимог та критеріїв якості, а також розробку стратегій і заходів для їх реалізації. Ефективний підхід до планування дозволяє створити продукт, що відповідає очікуванням користувачів, зменшити ризики та уникнути значних витрат, пов'язаних із виправленням помилок на пізніх етапах розробки [16, 17].

Основні елементи планування якості:

– перший елемент – це визначення цілей якості, які повинні бути чіткими, вимірними та відповідати потребам користувачів і замовників. Цілі якості можуть включати такі аспекти, як стабільність продукту, швидкість роботи, зручність використання та відповідність вимогам безпеки;

– другий елемент – це визначення вимог до якості, які формуються на основі аналізу потреб замовників, очікувань користувачів та вимог стандартів. Вимоги до якості повинні бути детально документовані та узгоджені з усіма зацікавленими сторонами. Вони слугують основою для розробки критеріїв оцінки якості, які дозволяють вимірювати рівень відповідності продукту встановленим вимогам;

– третій елемент – це розробка стратегії забезпечення якості, яка включає вибір методів, інструментів та підходів для досягнення цілей якості. Наприклад, це може бути автоматизоване тестування, код-рев'ю, впровадження міжнародних стандартів або використання методологій управління якістю. Важливим аспектом стратегії є розподіл відповідальності між членами команди, що забезпечує чітке розуміння ролей та обов'язків у процесі забезпечення якості;

– четвертий елемент – це розробка плану якості, який є основним документом, що описує, як будуть забезпечуватися цілі якості. План якості включає такі аспекти, як графік тестування, критерії прийняття продукту, процедури контролю якості та механізми виявлення та усунення дефектів. Він також містить інформацію про необхідні ресурси та інструменти для забезпечення якості продукту.

Планування якості має важливе значення на кожному етапі життєвого циклу програмного проєкту. На початковій стадії воно допомагає визначити ключові вимоги, що стає основою для подальшої роботи. У процесі детального планування розробляються стратегії, що дозволяють уникнути проблем на наступних етапах. Під час реалізації продукту управління якістю забезпечує відповідність програмного коду встановленим стандартам. На фінальному етапі планування допомагає оцінити досягнуті результати та визначити напрями для покращення у майбутніх проєктах.

3.2 Інтеграція стандартів ISO у процеси розробки ПЗ

Інтеграція стандартів ISO у процеси розробки програмного забезпечення є важливим кроком для забезпечення високої якості продуктів та відповідності міжнародним вимогам. Стандарти ISO, такі як ISO 9001, ISO/IEC/IEEE 12207, ISO/IEC 25010 та ISO/IEC 27001, надають чіткі рекомендації щодо організації процесів розробки, управління якістю та оцінки продуктивності [6, 8, 10, 12, 13]. Їх інтеграція дозволяє компаніям систематизувати свою роботу, мінімізувати ризики та забезпечити стабільну якість програмного забезпечення [16, 17].

Інтеграція стандартів ISO у процеси розробки програмного забезпечення включає кілька ключових етапів (рисунок 3.1):

– перший етап – це вивчення вимог стандартів та їх адаптація до потреб організації. На цьому етапі необхідно визначити, які саме стандарти будуть використовуватися, та розробити план їх впровадження. Наприклад, ISO 9001

може бути використаний для побудови системи управління якістю, тоді як ISO/IEC/IEEE 12207 – для організації процесів життєвого циклу програмного забезпечення [8, 12];

– другий етап – це розробка політики та процедур, які відповідають вимогам стандартів. Це включає створення документів, які описують процеси розробки, тестування, управління ризиками та інші аспекти роботи. Наприклад, політика якості може включати вимоги до проведення код-рев'ю, автоматизованого тестування та оцінки якості продукту. Важливим аспектом цього етапу є забезпечення розуміння та підтримки з боку всіх учасників проєкту;

– третій етап – це впровадження стандартів у повсякденну роботу команди. Це включає навчання співробітників, впровадження інструментів для управління якістю та проведення регулярних аудитів для оцінки відповідності вимогам стандартів. Наприклад, для впровадження ISO/IEC 25010 може бути використаний інструмент для оцінки якості програмного забезпечення, який дозволяє вимірювати такі характеристики, як функціональність, надійність та зручність використання [10];

– четвертий етап – це моніторинг та вдосконалення процесів. Після впровадження стандартів необхідно регулярно оцінювати їх ефективність та впроваджувати заходи для покращення. Це може включати проведення ретроспектив, аналіз результатів тестування та оновлення політик і процедур. Наприклад, якщо аудит виявить невідповідність процесів вимогам ISO 9001, необхідно розробити план для їх усунення та впровадити коректив заходи. ISO/IEC 27001 допомагає забезпечити необхідний рівень безпеки [8, 13].

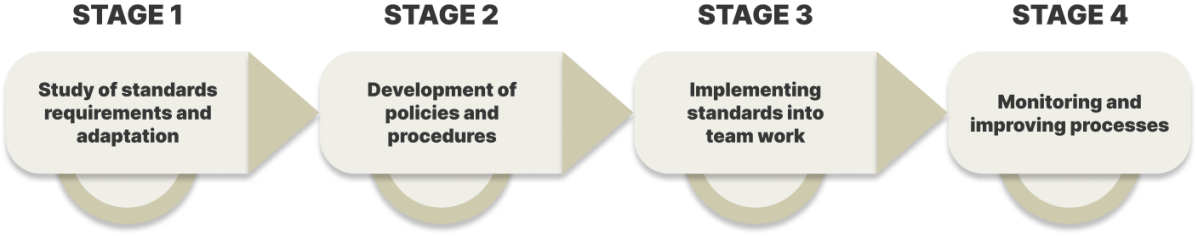


Рисунок 3.1 – Ключові етапи інтеграції

Приклади інтеграції стандартів ISO:

– ISO 9001. Використання цього стандарту дозволяє створити систему управління якістю, яка забезпечує стабільну якість продуктів та задоволеність клієнтів. Наприклад, компанія може впровадити процедури для планування якості, контролю процесів та проведення аудитів [8];

– ISO/IEC/IEEE 12207. Цей стандарт допомагає організувати процеси життєвого циклу програмного забезпечення, включаючи планування, розробку, тестування та підтримку. Наприклад, компанія може використовувати ISO/IEC 12207 для розробки детальних планів робіт та контролю виконання завдань [12];

– ISO/IEC 25010. Використання цього стандарту дозволяє оцінювати якість програмного забезпечення за такими характеристиками, як функціональність, надійність та зручність використання. Наприклад, компанія може використовувати ISO/IEC 25010 для розробки критеріїв оцінки якості продукту та проведення тестування [10];

– ISO/IEC 27001. Цей стандарт забезпечує системний підхід до захисту інформації та управління ризиками в IT-середовищі. Наприклад, компанія може використовувати ISO/IEC 27001 для впровадження політик безпеки, контролю доступу, шифрування даних і регулярного оцінювання вразливостей, що особливо важливо при зберіганні метрик якості та персональних даних користувачів у дашбордах чи інших програмних продуктах [13].

3.3 Інструменти та технології для забезпечення відповідності стандартам ISO

Сучасний арсенал інструментів та технологій для забезпечення відповідності стандартам ISO в управлінні якістю програмних проєктів охоплює широкий спектр рішень, що дозволяють ефективно впроваджувати вимоги міжнародних стандартів у практику розробки [1, 2]. Ці інструменти можна розглядати через призму їх функціонального призначення та

відповідності конкретним стандартам ISO.

Важливу роль у забезпеченні відповідності стандарту ISO 9001 відіграють системи управління якістю (QMS) [8]. Такі платформи як Qualio, Q-Pulse, Sparta TrackWise, Intelex або MasterControl надають комплексні рішення для документування процесів, управління невідповідностями та проведення аудитів. Вони дозволяють автоматизувати більшість вимог ISO 9001, зокрема управління документацією, відстеження виправлювальних дій та аналіз даних для прийняття управлінських рішень.

Для реалізації вимог ISO/IEC/IEEE 12207 щодо управління життєвим циклом ПЗ незамінними стають інструменти ALM (Application Lifecycle Management) [12]. Такі системи як IBM Engineering Lifecycle Management, Jira + Confluence, Azure DevOps забезпечують комплексне управління вимогами, тестуванням та релізами, дотримуючись принципів стандарту. Вони особливо корисні для великих проєктів з високими вимогами до трасованості змін.

Технології автоматизованого тестування, такі як Selenium, JMeter або TestComplete, відіграють ключову роль у забезпеченні відповідності вимогам якості згідно ISO/IEC 25010 [10]. Ці інструменти дозволяють систематично перевіряти функціональність, продуктивність та інші характеристики якості, зафіксовані в стандарті. Особливе значення має їх інтеграція в CI/CD-ланцюжки, що дозволяє реалізувати принцип безперервного контролю якості.

Для забезпечення відповідності вимогам інформаційної безпеки згідно ISO/IEC 27001 використовуються спеціалізовані інструменти статичного та динамічного аналізу коду, що дозволяють виявляти вразливості ще до потрапляння системи в продуктивне середовище [13, 14]. До таких інструментів належать SonarQube, Checkmarx, Fortify, Snyk, Veracode та інші рішення. Вони допомагають впровадити контроль безпеки на етапах розробки, інтеграції та тестування, що повністю відповідає вимогам стандарту щодо впровадження технічних та організаційних заходів контролю, вказаних у Annex A до ISO/IEC 27001. Застосування цих інструментів також сприяє управлінню ризиками безпеки шляхом виявлення критичних уразливостей, що можуть призвести до

компрометації інформаційних активів.

Важливим аспектом є інтеграція цих інструментів у єдине середовище управління якістю. Це дозволяє реалізувати принципи безперервного вдосконалення, закладені в ISO 9001, на практиці.

Ефективне використання цих інструментів вимагає не лише технічної інтеграції, але й адаптації процесів роботи команди. Важливо, щоб вибір конкретних рішень базувався на аналізі потреб проєкту та чіткому розумінні того, які саме вимоги стандартів ISO вони допомагають закрити. Тільки комплексний підхід, що поєднує технології, процеси та компетенції команди, дозволяє повною мірою реалізувати переваги стандартів ISO в управлінні якістю програмних проєктів.

4 АНАЛІЗ ВПЛИВУ СТАНДАРТІВ ISO НА ЯКІСТЬ ПРОГРАМНИХ ПРОЄКТІВ

4.1 Методологія оцінки ефективності

Оцінювання ефективності впровадження методів управління якістю згідно зі стандартами ISO – важлива складова дослідження, яка дає змогу не просто перевірити відповідність формальним вимогам, а й зрозуміти, чи дійсно обрані підходи сприяють реальному підвищенню якості програмного забезпечення. Це також дозволяє виявити, як такі підходи впливають на зменшення ризиків і рівень задоволеності користувачів [8].

Для такої оцінки застосовується комплексний підхід, що поєднує кількісні та якісні показники. Отримані результати порівнюються з попередньо визначеними цілями, що дає можливість зробити обґрунтовані висновки щодо ефективності впроваджених рішень.

Першим етапом є вибір критеріїв оцінювання. Вони мають відповідати вимогам міжнародних стандартів, зокрема ISO 9001, ISO/IEC 25010 та ISO/IEC/IEEE 12207 [8, 10, 12]. До основних критеріїв, як правило, належать:

– ступінь відповідності програмного продукту функціональним і нефункціональним вимогам [10];

– кількість виявлених і усунених помилок на різних етапах життєвого циклу [12];

– рівень автоматизації процесів контролю якості [12];

– своєчасність і повнота виконання заходів із управління ризиками [8];

– кількість запропонованих і реалізованих покращень, ініційованих на основі зворотного зв’язку [8].

Після визначення критеріїв наступним етапом є перехід до збору фактичних даних. Це може бути як технічна документація (звіти про тестування, результати аудитів, дані з систем управління проєктами), так і

думки учасників процесу – через опитування або інтерв’ю з користувачами й членами команди. У процесі збору важливо забезпечити об'єктивність, достовірність та простежуваність інформації [8].

Після збору даних виконується кількісна та якісна оцінка результатів. Кількісна оцінка передбачає аналіз таких метрик, як:

– щільність дефектів (Defect Density) [28];

– середній час реагування на помилки [8];

– відсоток тестів, що завершилися успішно [8];

– рівень покриття вимог під час тестування (Requirements Coverage) [23].

У якісному аналізі ключову роль відіграють фактори, які важко виміряти чисельно, але які суттєво впливають на загальну якість: наскільки внутрішні процеси відповідають стандартам, чи чітко задокументовані процедури, як команда реагує на зміни та наскільки активно вона залучена до покращення процесів [25].

Завершальним кроком є інтерпретація результатів. Це дозволяє оцінити, які саме методи виявилися дієвими, де є простір для покращень, а також які аспекти чинної моделі управління якістю можуть потребувати доопрацювання. Таким чином забезпечується відповідність принципу постійного вдосконалення, на якому базується ISO 9001 [8, 25].

4.2 Порівняльний аналітичний огляд реалізації стандартів

Порівняльний аналіз (таблиця 4.1) дає змогу глибше зрозуміти, як саме реалізуються різні стандарти ISO у контексті управління якістю програмного забезпечення. Такий огляд дозволяє оцінити не лише ефективність кожного зі стандартів, а й їхню здатність адаптуватися до потреб сучасних IT-проєктів і специфіки розробки. У центрі уваги – основні міжнародні стандарти, що мають безпосереднє відношення до якості в розробці ПЗ: ISO 9001, ISO/IEC 25010, ISO/IEC/IEEE 12207 та ISO/IEC 27001 [8, 10, 12, 13].

Стандарт ISO 9001, хоча й є загального призначення, активно

використовується в IT-сфері завдяки своїй структурованості, фокусі на клієнтоорієнтованість та принципі постійного вдосконалення. Його універсальний характер є одночасно перевагою і викликом: аби застосувати його у розробці програмного забезпечення, часто потрібна адаптація – зокрема, уточнення процедур тестування, перевірки якості та особливостей життєвого циклу продукту [8].

Інший стандарт – ISO/IEC/IEEE 12207 – створено спеціально для потреб програмної інженерії. Він охоплює всі етапи життєвого циклу ПЗ – від планування і проєктування до супроводу та виведення з експлуатації. Стандарт чітко визначає ролі, дії, результати робіт, що дає змогу структуровано організовувати процеси, навіть у масштабних або складних проєктах. Проте його впровадження може бути непростим: для невеликих команд або проєктів, що працюють за гнучкими методологіями, він може виявитися надмірно ресурсоємним [12].

Стандарт ISO/IEC 25010 акцентує увагу на якості самого продукту – програмного забезпечення. Він визначає вісім ключових характеристик, серед яких: функціональна відповідність, надійність, зручність у користуванні, продуктивність, безпека, сумісність, легкість супроводу й переносимість. Це корисний орієнтир під час визначення критеріїв оцінки ПЗ, але для його ефективного застосування необхідно впроваджувати чіткі метрики й інтегрувати їх у процедури тестування та контролю якості [10].

Окрему увагу заслуговує стандарт ISO/IEC 27001, що регламентує вимоги до управління інформаційною безпекою. Він встановлює систему політик і заходів щодо захисту даних, контролю доступу, використання криптографії, реагування на інциденти та управління ризиками. Цей стандарт особливо важливий для проєктів, які мають справу з конфіденційною інформацією або функціонують у критичних галузях [13]. У поєднанні з ISO 9001 та ISO/IEC 25010 він дозволяє створити комплексну систему управління якістю, у якій безпека виступає важливою складовою загальної довіри до програмного продукту.

Таблиця 4.1 – Порівняльна таблиця основних відмінностей між стандартами

Стандарт	Сфера охоплення	Основна перевага	Потенційне обмеження
ISO 9001	Система управління якістю	Гнучкість і адаптивність до будь-якої галузі	Недостатня деталізація процесів у сфері IT
ISO/IEC/IEEE 12207	Життєвий цикл ПЗ	Повне охоплення процесів розробки	Складність впровадження в невеликих командах
ISO/IEC 25010	Якість програмного продукту	Чіткі характеристики якості ПЗ	Необхідність додаткових інструментів для вимірювання
ISO/IEC 27001	Інформаційна безпека ПЗ	Захист даних, контроль доступу, управління ризиками	Потребує інтеграції з іншими стандартами для повноти

Результати порівняльного огляду свідчать, що найкращих результатів у сфері управління якістю можна досягти шляхом комбінованого підходу, за якого загальна модель ISO 9001 доповнюється процесними вимогами ISO/IEC 12207, характеристиками продукту згідно з ISO/IEC 25010 та вимогами інформаційної безпеки за ISO/IEC 27001 [8, 10, 12, 13]. Такий інтегрований підхід дозволяє адаптувати управління якістю до специфіки конкретного проєкту, одночасно підвищуючи ефективність, безпеку та довіру до результату розробки.

4.3 Метрики оцінки якості ПЗ після впровадження ISO

Після впровадження стандартів ISO у процеси розробки програмного забезпечення особливу роль відіграє кількісна оцінка рівня досягнутої якості [6, 8]. Вона не лише підтверджує ефективність обраних підходів, а й забезпечує систематичний контроль відповідності продукту встановленим вимогам. З цією метою використовуються спеціалізовані метрики якості, які охоплюють широкий спектр характеристик – від технічних аспектів функціонування до якості процесів розробки та ступеня задоволеності користувачів [10, 27].

У межах реалізації стандартів ISO (зокрема ISO/IEC 25010 та ISO 9001) виділяють кілька основних категорій метрик [8, 10, 27, 23]:

– метрики функціональної відповідності – дозволяють оцінити, наскільки програмний продукт виконує заявлені функції. До типових показників належать: відсоток реалізованих функціональних вимог, кількість виявлених функціональних дефектів, а також рівень покриття тестами ключових сценаріїв;

– метрики надійності – характеризують стабільність роботи системи та її здатність протистояти збоям. Найпоширенішими є показники середнього часу між відмовами (MTBF), частоти критичних помилок і коефіцієнта відновлення після збоїв;

– метрики зручності використання (usability) – вимірюють комфортність взаємодії з програмним забезпеченням з точки зору користувача. Серед них – рівень задоволеності, кількість звернень до служби підтримки, час виконання типових дій та частота помилок під час роботи з інтерфейсом;

– метрики продуктивності – відображають ефективність використання ресурсів і швидкодію системи. До них належать час відгуку при виконанні основних операцій, рівень споживання оперативної пам’яті та завантаження процесора;

– метрики супроводжуваності та змінюваності – дозволяють оцінити, наскільки просто оновлювати або модифікувати продукт після його випуску. Важливими індикаторами є середній час внесення змін, кількість помилок після

оновлень, а також складність коду, зокрема за метрикою цикломатичної складності;

– метрики задоволеності зацікавлених сторін – відображають оцінку якості з боку клієнтів і кінцевих користувачів. До таких показників належать результати опитувань, індекс лояльності Net Promoter Score (NPS) та аналіз зворотного зв’язку.

Використання цих метрик забезпечує об’єктивне уявлення про якість програмного забезпечення, підвищує прозорість і керованість процесів розробки [23]. Крім того, постійний моніторинг показників сприяє реалізації принципу безперервного вдосконалення – одного з фундаментальних положень стандарту ISO 9001 [8]. У результаті метрики стають не лише засобом контролю, а й важливим стратегічним інструментом розвитку якості в IT-проєктах [23].

5 РОЗРОБКА ПРОГРАМНОГО РІШЕННЯ ДЛЯ МОНІТОРИНГУ ЯКОСТІ ПРОГРАМНИХ ПРОЄКТІВ

5.1 Вибір інструментів та технологій для розробки дашборду

У рамках розробки системи моніторингу метрик якості програмного забезпечення відповідно до міжнародних стандартів ISO, особливо таких як ISO/IEC 25010, ISO 9001, ISO/IEC 12207 та ISO/IEC 27001, ключову роль відіграє ефективна візуалізація та інтерактивна аналітика [8, 10, 12, 13]. Саме тому було прийнято рішення про розробку спеціалізованого дашборду – інформаційної панелі, яка дозволяє наочно відслідковувати критичні показники якості, проводити порівняльний аналіз та формувати управлінські звіти для відповідальних осіб.

Під час проектування дашборду було визначено низку функціональних вимог, які обумовили вибір відповідних інструментів та технологічного стеку:

– підтримка роботи через веб-інтерфейс для кросплатформеності та доступності;

– можливість зберігання та обробки структурованих даних (проєкти, категорії метрик, значення показників, користувачі);

– надійна реалізація контролю доступу та автентифікації;

– можливість масштабування, поділу на компоненти та подальшої модифікації;

– інтеграція з API та підтримка експорту звітів у форматах PDF/CSV;

– простота у розгортанні на локальному сервері без використання складних DevOps-підходів.

Визначимо наступний стек технологій, враховуючи зазначені вимоги.

Мова програмування та серверна частина: JavaScript (Node.js) + Express. Для реалізації серверної логіки та побудови API було обрано платформу Node.js у зв'язці з фреймворком Express.js. Ця комбінація забезпечує високу

продуктивність, масштабованість та гнучкість при реалізації REST-архітектури. Express дозволяє легко створювати маршрути, підключати middleware, реалізовувати аутентифікацію, обробляти запити до бази даних тощо.

Система управління базами даних: PostgreSQL. Для зберігання структурованих даних про проєкти, категорії якості, метрики та користувачів було використано реляційну СУБД PostgreSQL. Вона є надійною, безпечною, масштабованою та підтримує роботу з складними SQL-запитами.

Клієнтська частина: React. React було обрано як основну бібліотеку для створення інтерфейсу користувача. Вона забезпечує високу продуктивність, повторне використання компонентів, ефективне оновлення DOM та хорошу інтеграцію з бібліотеками для візуалізації даних. Завдяки React можливо реалізувати гнучкий і динамічний дашборд, який оновлюється в реальному часі на основі даних, отриманих із серверного API.

Система маршрутизації: React Router. Для реалізації навігації між сторінками (головна, аналітика, адміністрування, вхід) використовується бібліотека React Router. Вона дозволяє керувати шляхами, реалізовувати захист маршрутів (наприклад, для адміністративної панелі), а також створювати зручну SPA (Single Page Application) навігацію.

Візуалізація даних: Recharts. Для побудови графіків та діаграм використовується бібліотека Recharts, що базується на D3.js. Вона забезпечує побудову інтерактивних, адаптивних та легко кастомізованих графіків, що ідеально підходить для реалізації ключового елемента дашборду – візуального представлення метрик якості.

Автентифікація та безпека: JWT (JSON Web Token). Для реалізації автентифікації користувачів та контролю доступу було використано протокол JWT. Він дозволяє створювати токени доступу, які зберігаються на клієнтській стороні та перевіряються сервером. Усі запити до API, які потребують авторизації, супроводжуються токеном, що підвищує безпеку доступу до адміністративних функцій та даних.

Генерація звітів: jsPDF + jsPDF-AutoTable. Окремою функціональністю

дашборду є можливість експорту звітів за метриками у форматах PDF та CSV. Для цього використовуються бібліотеки jsPDF та jsPDF-AutoTable, які дозволяють створювати PDF-документи з таблицями безпосередньо на клієнтській частині. Експортована інформація може слугувати підтвердженням дотримання критеріїв якості згідно зі стандартами ISO та бути використаною у внутрішньому або зовнішньому аудиті.

Інтерфейсні компоненти: Tailwind CSS та shadcn/ui. Tailwind CSS забезпечує гнучку утилітарну стилізацію, що дозволяє швидко створювати адаптивні, мінімалістичні та сучасні інтерфейси. Додатково застосовується бібліотека shadcn/ui для швидкого створення естетично привабливих компонентів (наприклад, карток, таблиць, фільтрів), які відповідають принципам UX-дизайну.

Середовище розробки та запуску: Windows 10 + Node.js + PostgreSQL. Розробка та тестування дашборду здійснювались у середовищі Windows 10 із використанням Node.js (v18.12.1), PostgreSQL (v17.4-1), менеджера пакетів npm та середовища Visual Studio Code. Всі інструменти є безкоштовними та кросплатформеними, що дозволяє легко перенести проєкт на будь-яке інше середовище (наприклад, Linux-сервер або хмарну інфраструктуру).

Таким чином, обраний набір інструментів забезпечує реалізацію сучасного, адаптивного та функціонального дашборду, який дозволяє реалізувати практичні аспекти стандартів ISO в управлінні якістю програмних проєктів. Він підтримує візуалізацію ключових метрик, дозволяє приймати рішення на основі даних та підвищує прозорість у процесі забезпечення якості на всіх етапах життєвого циклу ПЗ.

5.2 Архітектура та функціональні можливості дашборду

Розроблений дашборд для управління якістю програмних проєктів на основі стандартів ISO має модульну архітектуру, що забезпечує його масштабованість, гнучкість, розширюваність та простоту супроводу. Його

структурна організація побудована за принципами Model-View-Controller (MVC), що дозволяє чітко відокремити логіку представлення даних, обробку запитів користувача та роботу з інформаційною моделлю. Це забезпечує високу адаптивність та зручність підтримки системи.

Система дашборду має класичну трирівневу архітектуру, яка забезпечує логічне розділення функціональності на клієнтську, серверну та рівень зберігання даних:

– клієнтська частина (frontend) – побудована з використанням бібліотеки React, клієнтська частина відповідає за взаємодію з користувачем. Вона реалізує інтерфейс дашборду, графіки, таблиці, фільтри, форми введення даних, а також функції авторизації та експорту звітів. Вся взаємодія з сервером відбувається через REST API;

– серверна частина (backend) – реалізована на Node.js з використанням Express. Сервер відповідає за обробку запитів клієнта, автентифікацію користувачів, валідацію даних, взаємодію з базою даних через Pool pg та формування API-відповідей. Завдяки модульній побудові, сервер легко масштабувати та доповнювати новими можливостями;

– рівень бази даних (database) – уся інформація про проєкти, категорії якості, метрики, користувачів та історію змін зберігається у реляційній базі даних PostgreSQL. Структура бази побудована відповідно до принципів нормалізації та забезпечує цілісність і несуперечливість даних.

У функціональному сенсі дашборд виконує кілька ключових задач, що орієнтовані на підтримку процесу управління якістю програмного забезпечення. Основні функціональні можливості включають:

– управління проєктами. Користувачі можуть створювати нові програмні проєкти, надавати їм назву та опис, редагувати та видаляти при необхідності. Кожен проєкт є окремим об’єктом для оцінювання якості;

– категоризація за ISO/IEC 25010. Метрики організовані за 8-ма ключовими категоріями якості (надійність, функціональна придатність, зручність використання тощо), які відповідають структурі стандарту ISO/IEC

25010 [10]. Це дозволяє формувати повноцінну картину якості продукту;

– додавання та редагування метрик. Для кожного проєкту можна додавати метрики вручну, вказуючи числове значення, опис, дату фіксації та категорію. Це дозволяє адаптувати оцінювання до специфіки конкретного програмного проєкту;

– графічна аналітика. На сторінці аналітики реалізовано побудову графіків (BarChart, LineChart, PieChart), які візуалізують розподіл метрик за категоріями або у динаміці. Завдяки цьому управлінці можуть легко виявляти проблемні аспекти та слідкувати за прогресом;

– експорт звітів. Застосунок підтримує формування звітів у форматах PDF та CSV. У PDF-версії автоматично додається таблиця з метриками, яка містить дані, зібрані з інтерфейсу. CSV-файл підходить для подальшого аналізу в Excel або BI-системах;

– автентифікація та обмеження доступу. Для захисту адміністративних функцій реалізовано логін-систему на основі JWT. Користувачі без авторизації можуть переглядати лише загальнодоступну аналітику, тоді як створення та редагування даних потребує входу;

– збереження та інтеграція даних. Всі введені дані зберігаються у PostgreSQL та можуть бути використані як доказова база відповідності стандартам під час аудиту чи сертифікації за ISO;

– масштабованість і підтримка розширення. Завдяки модульній архітектурі проєкт легко розширити: можна додати нові типи графіків, інтегрувати системи авторизації на базі OAuth, підключити BI-інструменти або інші ISO-показники, наприклад з ISO/IEC 27001 [13, 14].

Кожен компонент дашборду взаємодіє із сервером через RESTful API. Запити формуються з використанням бібліотеки axios. API має захищені маршрути для автентифікованих користувачів (через JWT middleware), а також відкриті маршрути для публічного перегляду звітів і графіків. Завдяки цьому дотримується принцип розділення доступу та безпеки.

Загалом, дашборд є ефективним інструментом для керівників проєктів,

розробників, аналітиків з якості та інших зацікавлених осіб, які прагнуть здійснювати моніторинг і контроль відповідності програмного продукту вимогам ISO. Його модульність дозволяє у подальшому розширювати функціональність шляхом інтеграції з іншими стандартами або метриками без необхідності кардинальної перебудови архітектури системи.

5.3 Реалізація дашборду для моніторингу метрик якості

Розробка дашборду для моніторингу метрик якості програмного забезпечення стала ключовим етапом практичної частини дослідження, адже дозволила поєднати теоретичні знання зі стандартів управління якістю із реальними інструментами та механізмами контролю якості. Цей інструмент не лише демонструє можливість системного моніторингу характеристик якості програмного продукту, але й забезпечує умови для підвищення прозорості процесів розробки, прийняття рішень на основі метрик, а також забезпечення відповідності принципам міжнародних стандартів ISO.

Основна мета реалізації полягала в створенні програмного продукту, який дозволяє в інтерактивному режимі:

– візуалізувати показники якості ПЗ;

– фіксувати й аналізувати динаміку змін метрик;

– порівнювати значення з нормативними порогами;

– виявляти відхилення у процесі розробки та обслуговування;

– підтримувати процеси прийняття рішень на основі даних;

– генерувати звіти для внутрішнього контролю та зовнішнього аудиту.

Під час проєктування та реалізації дашборду були враховані вимоги та рекомендації одразу кількох ключових стандартів у сфері якості.

ISO/IEC 25010 (Software Product Quality Model) – реалізовано через структуру категорій і метрик якості (функціональна придатність, надійність, зручність використання, безпека тощо) [10]. Кожна метрика в системі прив’язана до відповідної характеристики з ISO/IEC 25010, що дозволяє

відстежувати їхні зміни в динаміці та аналізувати відхилення.

ISO 9001 (Quality Management Systems) – дотримано через побудову логіки запису, обліку, перегляду та верифікації значень метрик [8]. Функціонал експорту звітів, історія змін, валідація введених даних і обмеження доступу на основі ролей – усе це підтримує вимоги до документування та контролю якості процесів.

ISO/IEC/IEEE 12207 (Software Life Cycle Processes) – реалізовано завдяки підтримці життєвого циклу проєкту, який відображається у дашборді: від створення проєкту до його завершення [12]. Система дозволяє на кожному етапі життєвого циклу контролювати відповідні метрики, фіксувати критичні точки, а також формувати звітність за результатами фази.

ISO/IEC 27001 (Information Security Management) – дотримано шляхом впровадження автентифікації користувачів, захисту API через JWT, обмеження доступу до редагування критичних даних, логування дій користувачів та загального принципу мінімізації доступу [13, 14]. Це особливо важливо, коли дашборд використовується у командній роботі або з конфіденційною інформацією.

5.3.1 Створення серверної частини (Backend)

На початковому етапі реалізації дашборду було прийнято рішення про розділення клієнтської та серверної частини системи на два окремих проєкти (рисунок 5.1). Такий підхід базується на принципах сучасної архітектури клієнт-сервер, яка забезпечує масштабованість, гнучкість у розгортанні та полегшує супровід системи.

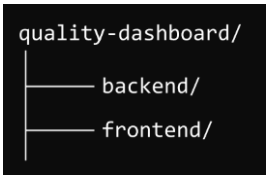


Рисунок 5.1 – Структура кореневої директорії

Першим кроком стало створення серверної частини – бекенду, який відповідає за обробку запитів, взаємодію з базою даних, виконання бізнес-логіки та забезпечення безпеки API. Саме цей рівень архітектури є ключовим для зберігання даних про проєкти, категорії, метрики, оцінки та користувачів, а також для реалізації авторизації згідно з вимогами стандарту ISO/IEC 27001.

Для побудови бекенду було обрано стек JavaScript на базі Node.js та фреймворк Express.js. Основним аргументом такого вибору стала висока продуктивність Node.js, підтримка асинхронної обробки запитів, велика екосистема npm-пакетів, а також можливість забезпечити єдність мови програмування на клієнтському та серверному рівні. Express.js, у свою чергу, дозволяє швидко створювати RESTful API, підтримує middleware-архітектуру, що спрощує впровадження валідації, логування, автентифікації тощо.

Для підключення до бази даних PostgreSQL обрано офіційний драйвер pg, а саме клас Pool, який дозволяє створювати пул з’єднань до бази. Такий підхід є більш контрольованим і ефективним у невеликих проєктах або коли важливо мати повний контроль над SQL-запитами, що відповідає принципам прозорості, які задекларовані в ISO 9001.

Серверна частина була організована у вигляді модульного Node.js-застосунку. Було створено окремі директорії для моделей, контролерів та маршрутів (рисунок 5.2).

Кожна з директорій відповідає за окрему логічну частину застосунку: controllers обробляють бізнес-логіку, models взаємодіють з базою, routes реєструють API-маршрути.

Розробку серверної частини було розпочато зі створення нового Node.js-проєкту. Для цього у терміналі була виконана команда:

– npm init -y.

Ця команда автоматично створила файл package.json, у якому зберігається основна інформація про проєкт: назва, версія, головний файл (index.js), ліцензія та інші параметри. Завдяки використанню ключа -y, було прийнято всі значення за замовчуванням без необхідності вводити їх вручну.

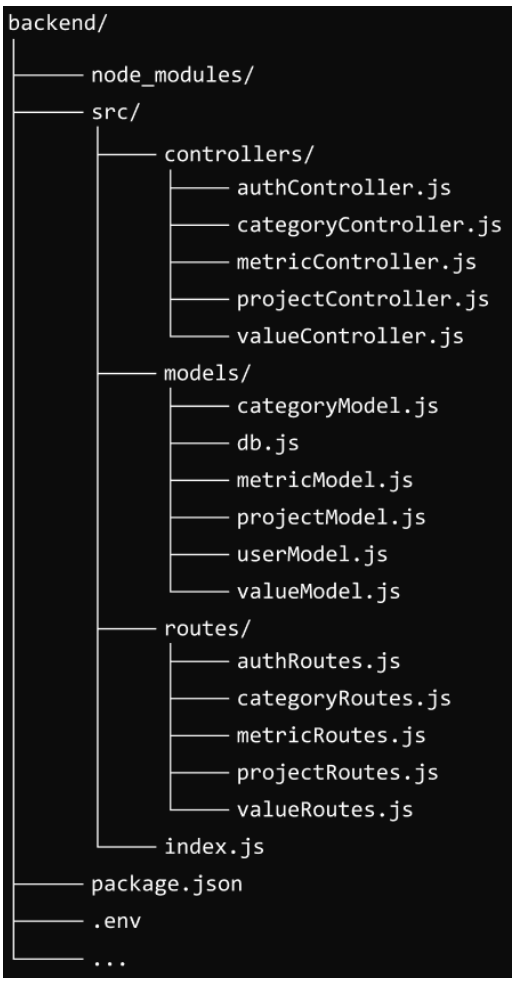


Рисунок 5.2 – Структура директорії «backend»

Наступним кроком стало встановлення необхідних залежностей, які забезпечують роботу серверного застосунку. У терміналі було введено:

– npm install express sequelize pg pg-hstore cors jsonwebtoken bcrypt dotenv.

Після виконання команди розпочався процес завантаження та встановлення бібліотек з репозиторію npm. На екрані з’явилися повідомлення про успішне встановлення кожного пакета, а також інформація про встановлення додаткових залежностей. Структура проєкту доповнилася папкою node_modules/, де фізично зберігаються всі встановлені пакети, а файл package.json автоматично оновився, включивши нові залежності до секції dependencies. Також, було створено файл .env який містить конфіденційні дані для доступу до бази.

Після ініціалізації проєкту та встановлення необхідних пакетів розробка

бекенду почалась з базового налаштування веб-сервера. Метою цього кроку є створення точки входу для усіх клієнтських запитів. Сервер слухає HTTP-запити на певному порту та делегує їх відповідним обробникам (рисунок 5.3).

Було створено файл index.js, у якому за допомогою бібліотеки express створено сервер, що слухає порт 5000 (лістинг 5.1). Для забезпечення обробки JSON-запитів було підключено middleware express.json(), а для розв'язання проблем із політикою CORS – пакет cors.

Лістинг 5.1 – Налаштування базового сервера Express

```
const express = require("express");
const cors = require("cors");
const dotenv = require("dotenv");

dotenv.config();

const app = express();
app.use(cors());
app.use(express.json());

app.get("/api/ping", (req, res) => res.json({ message: "OK" }));
const PORT = process.env.PORT || 5000;
app.listen(PORT, () => {
  console.log(`Сервер працює на порту ${PORT}`);
});
```

Рисунок 5.3 – Запуск сервера

Наступним кроком стало налаштування з'єднання з базою даних PostgreSQL, що є основою для зберігання всіх сутностей дашборду. Для цього було створено окремий модуль db.js у директорії models, у якому через клас

Pool створюється пул з’єднань на основі конфігурації з .env (лістинг 5.2).

Лістинг 5.2 – Підключення до бази даних PostgreSQL

```
const { Pool } = require("pg");
const dotenv = require("dotenv");

dotenv.config();

const pool = new Pool({
  user: process.env.DB_USER,
  host: process.env.DB_HOST,
  database: process.env.DB_NAME,
  password: process.env.DB_PASSWORD,
  port: process.env.DB_PORT,
});
module.exports = pool;
```

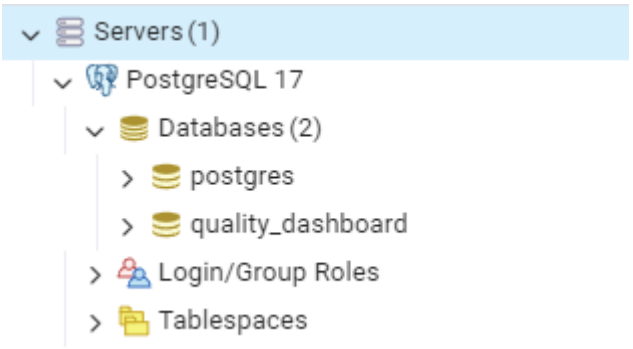


Рисунок 5.4 – Підключена БД «quality_dashboard»

Після налаштування бази розпочато реалізацію API-маршрутів. Першим кроком стало створення маршруту для управління проєктами (models/projectModel.js) (лістинг 5.3).

Лістинг 5.3 – Модель для отримання та створення проєктів

```
const pool = require("./db");

const ProjectModel = {
  getAll: async () => {
    const result = await pool.query("SELECT * FROM projects ORDER
BY id");
    return result.rows;
  },
```

```
  create: async ({ name, description }) => {
    const result = await pool.query(
       "INSERT INTO projects (name, description) VALUES ($1, $2)
RETURNING *",
      [name, description]
    );
    return result.rows[0];
  },
};

module.exports = ProjectModel;
```

Щоб надати клієнтам можливість взаємодіяти з моделлю Project, було створено контролер projectController.js, який обробляє логіку запитів (лістинг 5.4). Тут реалізовано два основні методи: отримання списку всіх проєктів (GET /api/projects) та створення нового проєкту (POST /api/projects).

Лістинг 5.4 – Контролер обробляє логіку запитів

```
const ProjectModel = require("../models/projectModel");

const getProjects = async (req, res) => {
  try {
    const projects = await ProjectModel.getAll();
    res.json(projects);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
};

const createProject = async (req, res) => {
  try {
    const { name, description } = req.body;
      if (!name) return res.status(400).json({ error: "Name is
required" });

       const newProject = await ProjectModel.create({ name,
description });
    res.status(201).json(newProject);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
};

module.exports = {
  getProjects,
  createProject,
};
```

Після реалізації контролера було необхідно зареєструвати маршрути, які викликатимуть відповідні функції. Для цього було створено файл routes/projectRoutes.js (лістинг 5.5), у якому з використанням Express Router описано доступні методи.

Лістинг 5.5 – Оголошення маршруту

```
const express = require("express");
const router = express.Router();
const {
  getProjects,
  createProject,
} = require("../controllers/projectController");

router.get("/", getProjects);
router.post("/", createProject);

module.exports = router;
```

Фінальним кроком стало підключення projectRoutes.js до основного сервера, аби маршрути стали активними (лістинг 5.6, рисунок 5.5).

Лістинг 5.6 – Імпорт маршруту у index.js

```
const projectRoutes = require("./routes/projectRoutes");
app.use("/api/projects", projectRoutes);
```

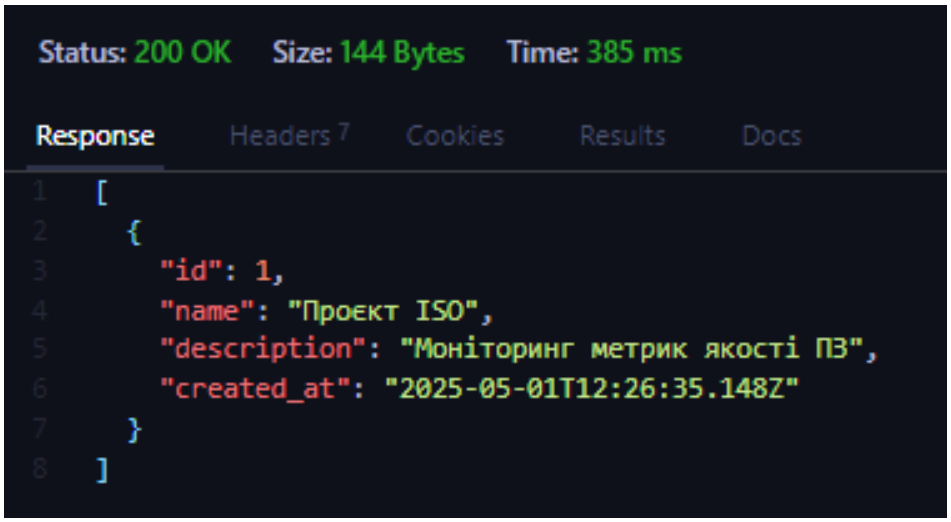


Рисунок 5.5 – Відповідь Thunder Client на GET http://localhost:5000/api/projects

5.3.2 Налаштування бази даних PostgreSQL

Після завершення розробки серверної частини дашборду логічним наступним кроком стало проєктування структури бази даних. Оскільки між сутностями, що стосуються проєктів, метрик, категорій якості, оцінок та користувачів, існують чіткі реляційні зв’язки, виникла потреба у використанні надійного реляційного сховища. Таке сховище повинне забезпечувати збереження критично важливої інформації з дотриманням вимог до цілісності, структурованості й безпеки даних. Цей підхід цілком відповідає принципам, закладеним у стандарті ISO/IEC 12207, зокрема щодо управління архітектурою даних протягом усього життєвого циклу програмного забезпечення [12].

У ролі системи управління базами даних було обрано PostgreSQL – завдяки її відкритому коду, високій продуктивності, підтримці транзакцій, строгій типізації даних та активній спільноті користувачів. Крім того, PostgreSQL має вбудовані механізми для забезпечення безпеки, ведення журналів змін, а також потужний SQL-інтерфейс, який дає змогу ефективно реалізовувати складні аналітичні запити.

Замість використання ORM (Object-Relational Mapping), що іноді може додавати надмірної складності у невеликих проєктах, було прийнято рішення використовувати бібліотеку pg (PostgreSQL client for Node.js), зокрема, об’єкт Pool для керування з’єднаннями. Це дозволяє мати більший контроль над запитами, гнучкість у структурі коду та високу продуктивність при прямому виконанні SQL.

Перед безпосереднім створенням таблиць було розроблено логічну модель, яка визначає основні сутності, атрибути та зв’язки між ними. В основу моделі покладено концепції ISO/IEC 25010, що класифікує якість ПЗ на вісім характеристик, які лягли в основу таблиці категорій [10].

У результаті були спроєктовані наступні таблиці:

– projects – таблиця з інформацією про проєкти, що оцінюються;

– quality_categories – категорії якості згідно з ISO/IEC 25010;

– quality_metrics – специфічні показники, які належать до певної категорії;

– metric_values – числові значення метрик, прив’язані до проєктів.

Ці таблиці мають між собою чітко визначені зв’язки «один до багатьох» та зовнішні ключі, що забезпечує цілісність даних (рисунок 5.9):

– projects ↔ metric_values: один проєкт має багато значень метрик;

– metrics ↔ metric_values: одна метрика має багато вимірювань;

– categories ↔ metrics: одна категорія має багато метрик.

Після проєктування логіки схеми база даних була створена за допомогою SQL-скриптів у середовищі pgAdmin 4 (лістинг 5.7).

Лістинг 5.7 – Скрипт створення таблиць

```
CREATE TABLE IF NOT EXISTS projects (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    description TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
CREATE TABLE IF NOT EXISTS quality_categories (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    description TEXT
);
CREATE TABLE IF NOT EXISTS quality_metrics (
    id SERIAL PRIMARY KEY,
    category_id INTEGER REFERENCES quality_categories(id),
    name VARCHAR(100) NOT NULL,
    unit VARCHAR(20),
    target_value NUMERIC,
    warning_threshold NUMERIC,
    critical_threshold NUMERIC,
    description TEXT
);
CREATE TABLE IF NOT EXISTS metric_values (
    id SERIAL PRIMARY KEY,
    project_id INTEGER REFERENCES projects(id),
    metric_id INTEGER REFERENCES quality_metrics(id),
    value NUMERIC NOT NULL,
    measured_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Після запуску вищезазначеного SQL-коду в pgAdmin база даних набула

необхідної структури (рисунок 5.6), що дозволило перейти до безпосередньої реалізації з'єднання із сервером.

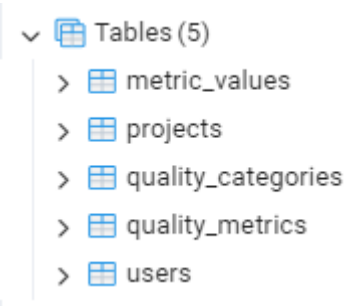


Рисунок 5.6 – Дерево таблиць

Для відповідності стандарту ISO/IEC 25010, до таблиці categories було додано 9 характеристик якості ПЗ (лістинг 5.8):

– надійність;

– функціональна придатність;

– зручність використання;

– продуктивність;

– сумісність;

– безпека;

– супроводжуваність;

– переносимість;

– модульність.

Лістинг 5.8 – SQL-приклад наповнення:

```
INSERT INTO categories (name, description) VALUES
('Надійність', 'Стійкість до помилок і відновлення після збоїв'),
('Безпека', 'Захист інформації та запобігання несанкціонованому доступу'), ...;
```

Для підключення до бази даних з Node.js-сервера було використано об'єкт Pool із бібліотеки pg, що забезпечує пул з'єднань для багаторазового використання під час виконання SQL-запитів. Тепер, у будь-якому контролері

можна було використовувати pool.query(...) для взаємодії з базою (лістинг 5.9, 5.10).

Лістинг 5.9 – Приклад отримання всіх проєктів

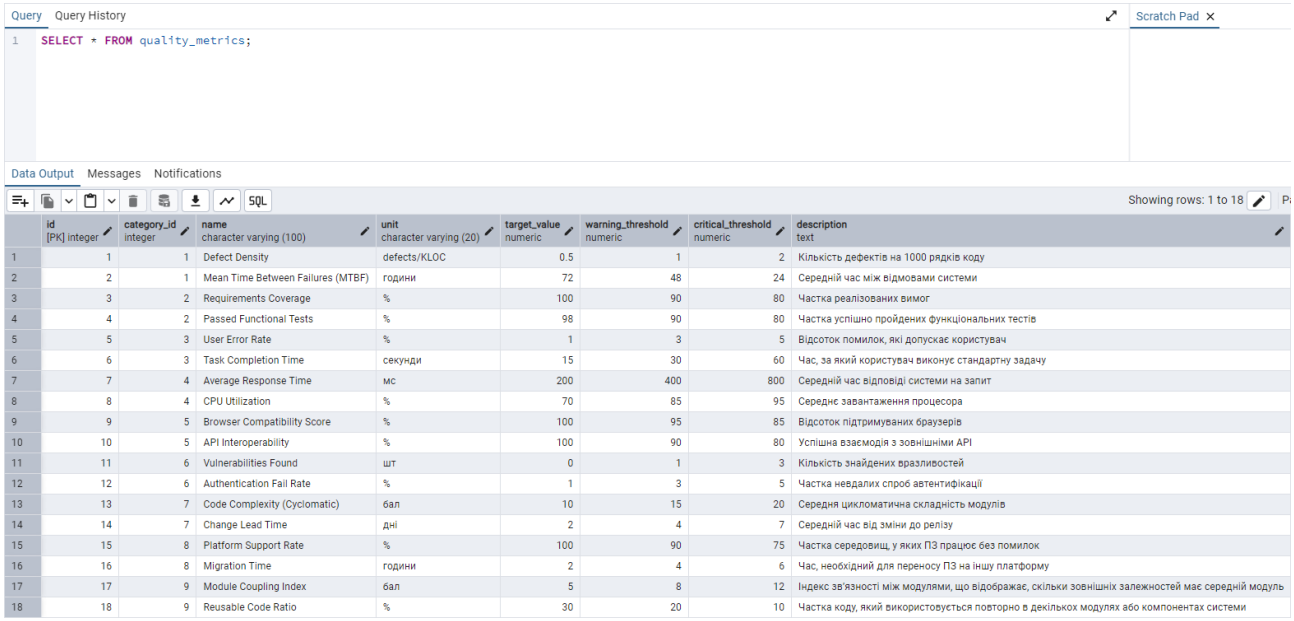
```
const pool = require("../db");

exports.getAllProjects = async (req, res) => {
  const result = await pool.query("SELECT * FROM projects ORDER BY
created_at DESC");
  res.json(result.rows);
};
```

Лістинг 5.10 – Приклад створення запису

```
exports.createProject = async (req, res) => {
  const { name, description } = req.body;
  const result = await pool.query(
    "INSERT INTO projects (name, description) VALUES ($1, $2)
RETURNING *",
    [name, description]
  );
  res.status(201).json(result.rows[0]);
};
```

Це дозволило повністю реалізувати CRUD-операції для основних таблиць дашборду (рисунок 5.7, 5.8).



Query | Query History

```
SELECT * FROM quality_metrics;
```

Scratch Pad

Data Output | Messages | Notifications

Showing rows: 1 to 18

	id [PK] integer	category_id integer	name character varying (100)	unit character varying (20)	target_value numeric	warning_threshold numeric	critical_threshold numeric	description text
1	1	1	Defect Density	defects/KLOC	0.5	1	2	Кількість дефектів на 1000 рядків коду
2	2	1	Mean Time Between Failures (MTBF)	години	72	48	24	Середній час між відмовами системи
3	3	2	Requirements Coverage	%	100	90	80	Частка реалізованих вимог
4	4	2	Passed Functional Tests	%	98	90	80	Частка успішно пройдених функціональних тестів
5	5	3	User Error Rate	%	1	3	5	Відсоток помилок, які допускає користувач
6	6	3	Task Completion Time	секунди	15	30	60	Час, за який користувач виконує стандартну задачу
7	7	4	Average Response Time	мс	200	400	800	Середній час відповіді системи на запит
8	8	4	CPU Utilization	%	70	85	95	Середнє завантаження процесора
9	9	5	Browser Compatibility Score	%	100	95	85	Відсоток підтримуваних браузерів
10	10	5	API Interoperability	%	100	90	80	Успішна взаємодія з зовнішніми API
11	11	6	Vulnerabilities Found	шт	0	1	3	Кількість знайдених вразливостей
12	12	6	Authentication Fail Rate	%	1	3	5	Частка невдалих спроб автентифікації
13	13	7	Code Complexity (Cyclomatic)	бал	10	15	20	Середня цикломатична складність модулів
14	14	7	Change Lead Time	дні	2	4	7	Середній час від зміни до релізу
15	15	8	Platform Support Rate	%	100	90	75	Частка середовищ, у яких ПЗ працює без помилок
16	16	8	Migration Time	години	2	4	6	Час, необхідний для переносу ПЗ на іншу платформу
17	17	9	Module Coupling Index	бал	5	8	12	Індекс зв'язності між модулями, що відображає, скільки зовнішніх залежностей має середній модуль
18	18	9	Reusable Code Ratio	%	30	20	10	Частка коду, який використовується повторно в декількох модулях або компонентах системи

Рисунок 5.7 – Приклад виконання запиту SELECT * FROM quality_metrics

Рисунок 5.8 – Фрагмент отриманих метрик з Thunder Client як альтернатива pgAdmin

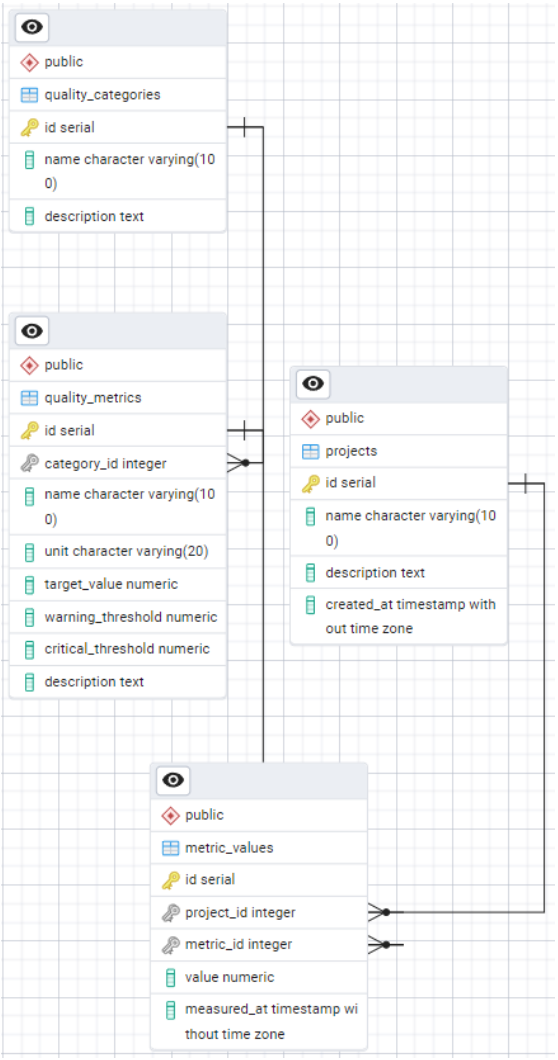


Рисунок 5.9 – ERD схеми

5.3.3 Реалізація клієнтської частини (Frontend)

Після побудови серверної логіки й підключення до бази даних наступним етапом стало створення клієнтської частини дашборду – інтерактивного веб-інтерфейсу, за допомогою якого користувач може взаємодіяти з системою (рисунок 5.10). Відповідно до архітектурного принципу розподілу обов'язків, бекенд відповідає за зберігання та обробку даних, а фронтенд – за відображення інформації, валідацію вводу та взаємодію з користувачем (рисунок 5.18). Цей етап особливо важливий у контексті стандарту ISO/IEC 25010 [10], який включає зручність використання (usability), візуальну доступність, продуктивність інтерфейсу та функціональну придатність.

У якості основної бібліотеки для реалізації клієнтського інтерфейсу було обрано React.js. Такий вибір обґрунтовано тим, що React дозволяє створювати компонентно-орієнтовані інтерфейси, легко працює з REST API, підтримує керування станом застосунку, а також має велику екосистему супровідних бібліотек (Recharts, React Router, axios, jwt-decode тощо).

Для стилізації було використано Tailwind CSS – утилітарний фреймворк для швидкого й адаптивного оформлення, а також shadcn/ui для готових компонентів з сучасним дизайном.

React-застосунок було створено за допомогою інструменту create-react-app, після чого організовано структуру директорій таким чином:

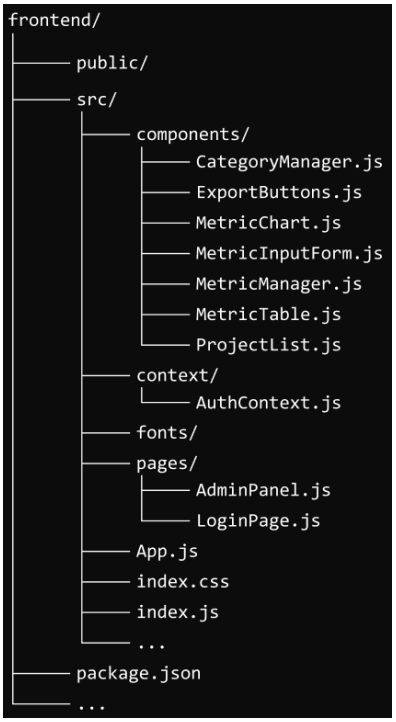


Рисунок 5.10 – Структура директорії frontend

Кожен модуль відповідає за окремий аспект: перелік проєктів (лістинг 5.11), таблицю метрик, форму логіну тощо. Це спрощує масштабування та підтримку інтерфейсу.

Для зручної та асинхронної роботи з API використовувалась бібліотека axios.

Лістинг 5.11 – Приклад отримання переліку проєктів

```
import axios from "axios";

const API_URL = "http://localhost:5000/api/projects";

const fetchProjects = async () => {
  try {
    const res = await axios.get(API_URL);
    setProjects(res.data);
  } catch (err) {
    console.error("Помилка при завантаженні проєктів:", err);
  }
};
```

У відповідному компоненті ProjectList.js дані виводяться за допомогою методу useEffect (лістинг 5.12) і зберігаються у стані.

Лістинг 5.12 – Хук

```
useEffect(() => {
  api.get('/projects').then((res) => {
    setProjects(res.data);
  });
}, []);
```

Кожен запит супроводжується перевіркою токена авторизації, якщо потрібно (для захищених маршрутів).

Основні компоненти інтерфейсу:

– CategoryManager – форма для додавання нової категорії;

– ProjectList – показує список усіх створених проєктів, дозволяє обрати активний проєкт;

– MetricTable – відображає таблицю метрик із зазначенням категорії, значення, цільового показника, порогів і дати вимірювання;

– MetricInputForm – форма для додавання нового значення метрики;

– MetricChart – компонент із графіком динаміки певної метрики;

– MetricManager – форма для додавання нових метрик;

– ExportButtons – кнопки для експорту у форматах CSV та PDF;

– LoginForm / AuthContext – система авторизації збереження токена,

обмеження доступу до адміністративної частини;

– AdminPanel – панель керування категоріями та метриками, доступна лише авторизованим користувачам.

Для реалізації маршрутизації було використано бібліотеку react-router-dom (лістинг 5.13).

Лістинг 5.13 – Реалізована структура у App.js

```
<Routes>
            <Route path="/" element={<> <ProjectList />
<MetricInputForm /> <MetricTable /> <MetricChart /> </> } />
            <Route path="/login" element={<LoginPage />} /> <Route
path="/admin" element={ <AdminRoute> <AdminPanel /> </AdminRoute>
} />
</Routes>
```

Захист маршруту /admin реалізовано через компонент ProtectedRoute, який перевіряє наявність JWT-токена в контексті.

Інтерфейс забезпечує валідацію введених даних: обов’язкові поля, допустимі значення, попередження про помилки (лістинг 5.14, рисунок 5.12). Усі повідомлення реалізовані українською мовою, відповідно до вимог ISO 9001 (доступність для цільової аудиторії) (рисунок 5.11).

Лістинг 5.14 – Форма для введення метрики

```
<form onSubmit={handleSubmit} className="space-y-4">
  <select
    className="w-full p-2 border border-gray-300 rounded"
    value={projectId}
    onChange={(e) => setProjectId(e.target.value)} >
    <option value="">Оберіть проєкт</option>
    {projects.map((p) => (
      <option key={p.id} value={p.id}>
        {p.name}
      </option>
    ))}
  </select>
  <select
    className="w-full p-2 border border-gray-300 rounded"
    value={metricId}
    onChange={(e) => setMetricId(e.target.value)} >
    <option value="">Оберіть метрику</option>
    {metrics.map((m) => (
```

```
        <option key={m.id} value={m.id}>
          {m.name} ({m.unit}) - {m.category_name}
        </option>
      ))}
    </select>

    <input
      type="number"
      step="0.01"
      className="w-full p-2 border border-gray-300 rounded"
      placeholder="Значення"
      value={value}
      onChange={(e) => setValue(e.target.value)}
    />
    <button
      type="submit"
      className="bg-blue-600 text-white px-4 py-2 rounded hover:bg-
blue-700"
    >
      Додати значення
    </button>
</form>
```

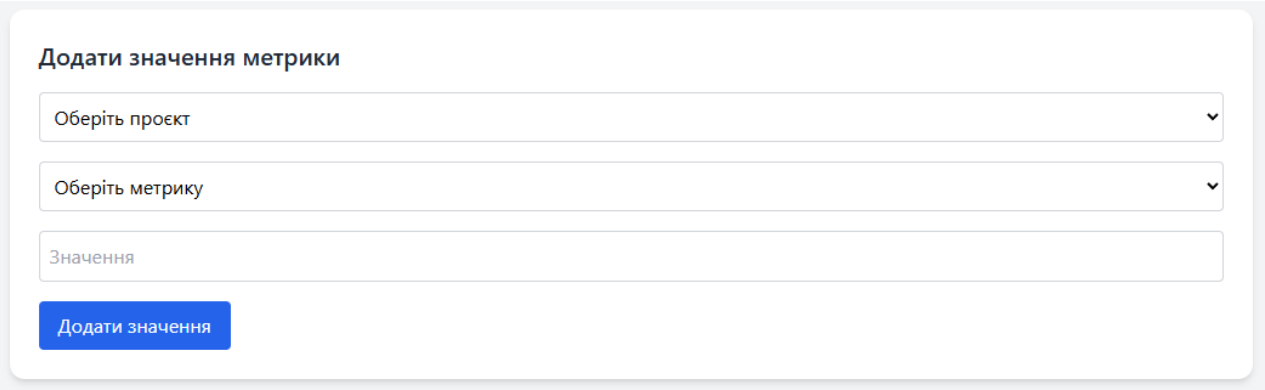


Рисунок 5.11 – Форма додавання нового значення метрики

	id [PK] integer	category_id integer	name character varying (100)	unit character varying (20)	target_value numeric	warning_threshold numeric	critical_threshold numeric	description text
1	1	1	Defect Density	defects/KLOC	0.5	1	2	Кількість дефектів на 1000 рядків коду
2	2	1	Mean Time Between Failures (MTBF)	години	72	48	24	Середній час між відмовами системи
3	3	2	Requirements Coverage	%	100	90	80	Частка реалізованих вимог
4	4	2	Passed Functional Tests	%	98	90	80	Частка успішно пройдених функціональних тестів
5	5	3	User Error Rate	%	1	3	5	Відсоток помилок, які допускає користувач
6	6	3	Task Completion Time	секунди	15	30	60	Час, за який користувач виконує стандартну задачу
7	7	4	Average Response Time	мс	200	400	800	Середній час відповіді системи на запит
8	8	4	CPU Utilization	%	70	85	95	Середнє завантаження процесора
9	9	5	Browser Compatibility Score	%	100	95	85	Відсоток підтримуваних браузерів
10	10	5	API Interoperability	%	100	90	80	Успішна взаємодія з зовнішніми API
11	11	6	Vulnerabilities Found	шт	0	1	3	Кількість знайдених вразливостей
12	12	6	Authentication Fail Rate	%	1	3	5	Частка невдалих спроб автентифікації
13	13	7	Code Complexity (Cyclomatic)	бал	10	15	20	Середня цикломатична складність модулів
14	14	7	Change Lead Time	дні	2	4	7	Середній час від зміни до релізу
15	15	8	Platform Support Rate	%	100	90	75	Частка середовищ, у яких ПЗ працює без помилок
16	16	8	Migration Time	години	2	4	6	Час, необхідний для переносу ПЗ на іншу платформу
17	17	9	Module Coupling Index	бал	5	8	12	Індекс зв'язності між модулями, що відображає, скільки зовнішніх залежностей має середній модуль
18	18	9	Reusable Code Ratio	%	30	20	10	Частка коду, який використовується повторно в декількох модулях або компонентах системи

Рисунок 5.12 – Додані метрики до відповідних категорій

Основу інтерфейсу становить таблиця метрик, що містить (лістинг 5.15):

– назву метрики;

– категорію якості (згідно з ISO/IEC 25010);

– поточне значення;

– цільове значення;

– попереджувальний та критичний пороги;

– дату вимірювання.

Компонент MetricTable.js побудовано з використанням HTML-таблиці, стилізованої через Tailwind CSS. Було реалізовано сортування за колонками, динамічне підсвічування (наприклад, червоний колір, якщо значення метрики нижче критичного порогу).

Лістинг 5.15 – Зовнішній вигляд таблиці

```
<table className="w-full border-collapse text-sm">
  <thead className="bg-gray-100">
    <tr>
      <th className="p-2 border">Метрика</th>
      <th className="p-2 border">Категорія</th>
      <th className="p-2 border">Значення</th>
      <th className="p-2 border">Ціль</th>
      <th className="p-2 border">Дата</th>
    </tr>
  </thead>
  <tbody>
    {metrics.map((m) => (
      <tr key={m.id}>
        <td className="p-2 border">{m.metric_name}</td>
        <td className="p-2 border">{m.category_name}</td>
        <td className={`p-2 border ${getColor(m)}`}>{m.value}</td>
        <td className="p-2 border">{m.target_value}</td>
        <td                 className="p-2                 border">{new
Date(m.measured_at).toLocaleDateString()}</td>
      </tr>
    ))}
  </tbody>
</table>
```

Щоб забезпечити аналітику в контексті конкретного проєкту, реалізовано компонент ProjectSelector, що дозволяє обрати поточний проєкт і динамічно оновити таблицю метрик.

Щоб користувач візуально розрізняв, наскільки поточне значення відповідає цілі, було реалізовано динамічне форматування кольорів:

– зелений – значення ≥ цільового;

– жовтий – значення між попереджувальним і цільовим;

– червоний – значення ≤ критичного порогу.

Це дозволяє швидко виявляти проблемні зони без необхідності аналізувати числові значення вручну (лістинг 5.16).

Побудова таблиці метрик з підсвіткою порушень порогів (рисунок 5.14).

Лістинг 5.16 – Виділення кольором

```
const getStatusColor = (val, warn, crit) => {
  if (val >= crit) return "bg-red-100 text-red-800";
  if (val >= warn) return "bg-yellow-100 text-yellow-800";
  return "bg-green-100 text-green-800";
};
```

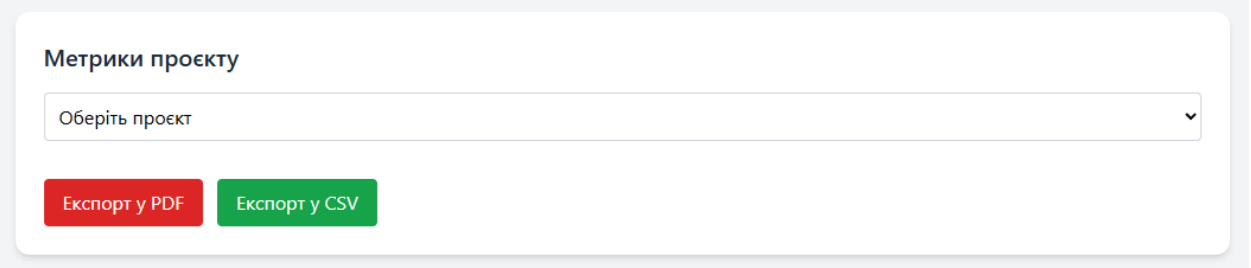
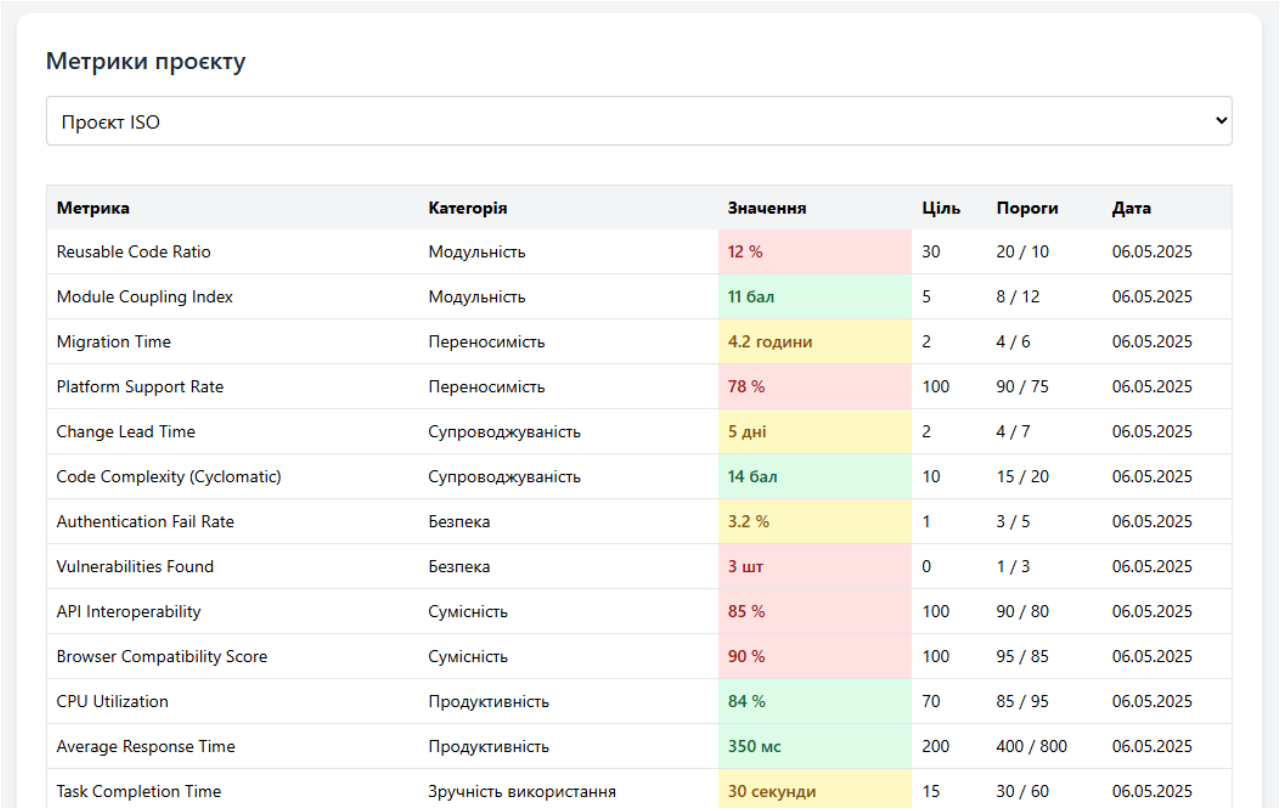


Рисунок 5.13 – Зона вибору проєкту та відображення таблиці значень

Для забезпечення повноцінного моніторингу якості програмних проєктів важливо не лише представити метрики у вигляді таблиці, а й візуалізувати їх динаміку з плином часу. Це дозволяє користувачам бачити тенденції – наприклад, як змінюється показник maintainability або reliability протягом кількох тижнів чи місяців. Такі графіки значно підвищують зрозумілість даних, прискорюють прийняття рішень і відповідають вимогам стандартів якості до звітності та відстеження змін.



Метрики проєкту

Проєкт ISO

Метрика	Категорія	Значення	Ціль	Пороги	Дата
Reusable Code Ratio	Модульність	12 %	30	20 / 10	06.05.2025
Module Coupling Index	Модульність	11 бал	5	8 / 12	06.05.2025
Migration Time	Переносимість	4.2 години	2	4 / 6	06.05.2025
Platform Support Rate	Переносимість	78 %	100	90 / 75	06.05.2025
Change Lead Time	Супроводжуваність	5 дні	2	4 / 7	06.05.2025
Code Complexity (Cyclomatic)	Супроводжуваність	14 бал	10	15 / 20	06.05.2025
Authentication Fail Rate	Безпека	3.2 %	1	3 / 5	06.05.2025
Vulnerabilities Found	Безпека	3 шт	0	1 / 3	06.05.2025
API Interoperability	Сумісність	85 %	100	90 / 80	06.05.2025
Browser Compatibility Score	Сумісність	90 %	100	95 / 85	06.05.2025
CPU Utilization	Продуктивність	84 %	70	85 / 95	06.05.2025
Average Response Time	Продуктивність	350 мс	200	400 / 800	06.05.2025
Task Completion Time	Зручність використання	30 секунди	15	30 / 60	06.05.2025

Рисунок 5.14 – Фрагмент таблиці з відображеними значеннями метрик

Для реалізації графіків була обрана бібліотека Recharts, яка є високорівневою бібліотекою для побудови інтерактивних діаграм у React (лістинг 5.17). Вона базується на D3.js, але має значно простіший синтаксис і добре інтегрується у JSX-код. Її основні переваги:

– підтримка адаптивної верстки (ResponsiveContainer);

– широкий набір діаграм (лінійні графіки, гістограми, кругові діаграми тощо);

– можливість накладення декількох серій даних;

– підтримка підказок (tooltip), легенд, координатних сіток.

Було створено окремий компонент MetricChart, який відповідає за відображення графіка значень обраної метрики протягом часу. Його логіка включає:

– завантаження історії значень метрики через запит на API: /api/metric_values?metric_id=...;

– побудова графіка з урахуванням дати вимірювання по осі X та значення по осі Y;

– візуальне відображення порогових значень – наприклад, горизонтальна лінія, яка позначає цільове або критичне значення.

Лістинг 5.17 – Кодова реалізація графіка (фрагмент JSX)

```
<ResponsiveContainer width="100%" height={300}>
  <LineChart data={metricData}>
    <CartesianGrid strokeDasharray="3 3" />
    <XAxis dataKey="date" />
    <YAxis domain={[0, 'dataMax + 1']} />
    <Tooltip />
    <Legend />
    <Line    type="monotone"    dataKey="value"    stroke="#8884d8"
name="Значення" />
    <ReferenceLine          y={targetValue}          stroke="green"
strokeDasharray="3 3" label="Ціль" />
    <ReferenceLine          y={criticalValue}          stroke="red"
strokeDasharray="3 3" label="Критичне" />
  </LineChart>
</ResponsiveContainer>
```

Ключові елементи:

– LineChart – основний контейнер для лінійного графіка;

– Line – лінія, що відображає реальні значення;

– ReferenceLine – горизонтальні орієнтири для порогів.

Перед побудовою графіка отримані з сервера дані трансформуються у формат, зручний для осі X (лістинг 5.18).

Лістинг 5.18 – Форматування дати

```
const formattedData = rawData.map((d) => ({ ...d,
  date: new Date(d.measured_at).toLocaleDateString(),
}));
```

Користувач має можливість перемикатись між різними метриками, після чого графік оновлюється автоматично (рисунок 5.15, 5.16).

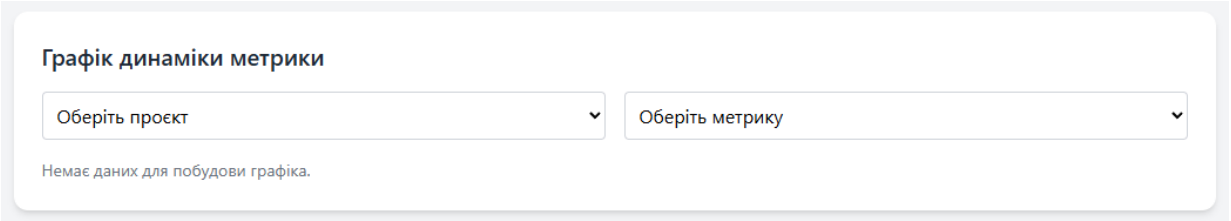


Рисунок 5.15 – Зона відображення графіка динаміки змін метрик

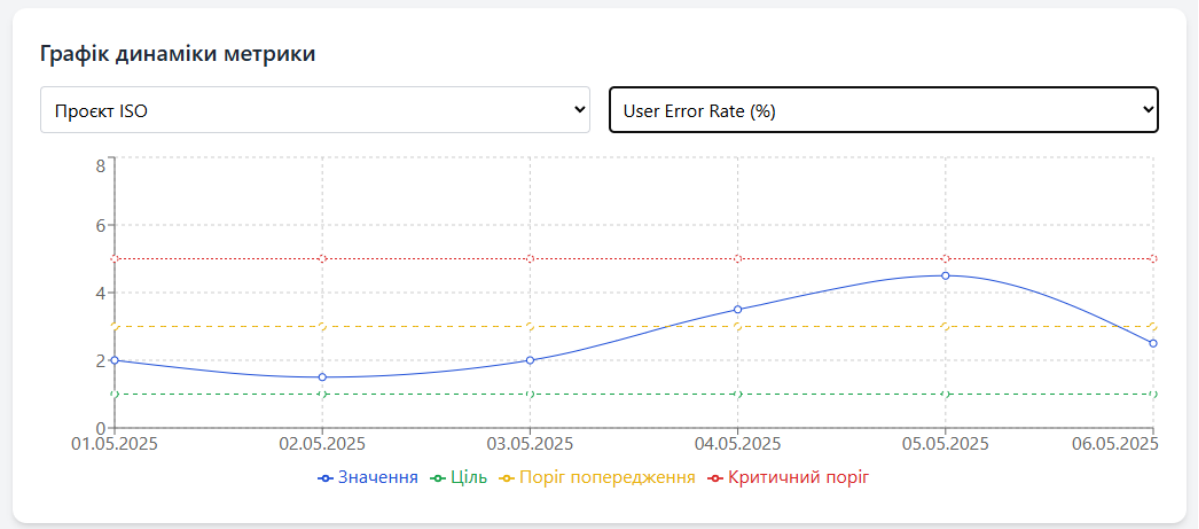


Рисунок 5.16 – Приклад графіка зміни значення для метрики User Error Rate

Додаткові елементи візуалізації (рисунок 5.17):

– Tooltip – з’являється при наведенні миші на точку графіка, показує точну дату та значення;

– Legend – пояснює, що саме позначено на графіку: реальні значення, цільове значення, критичний поріг.

Окрім лінійного графіка, також тестувались:

– BarChart – гістограма;

– AreaChart – залітаєна діаграма;

– PieChart – для загального розподілу категорій.

Однак, з точки зору моніторингу динаміки метрик з часом, найкращим виявився саме LineChart, тому він і був обраний як основний.

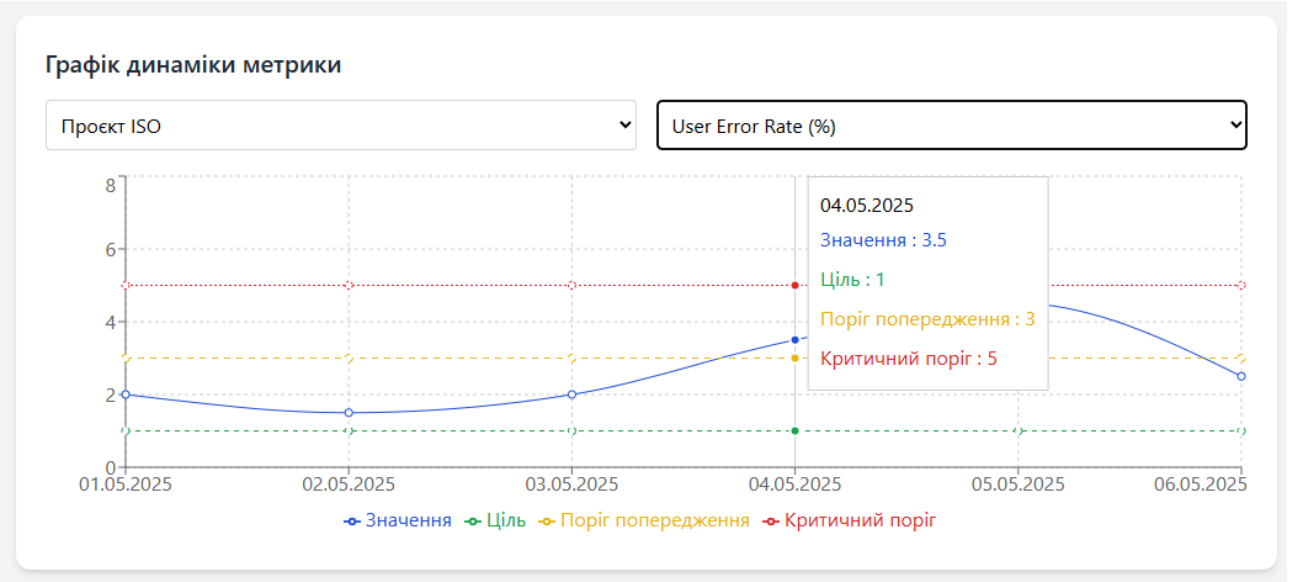


Рисунок 5.17 – Графік з підказками й легендою

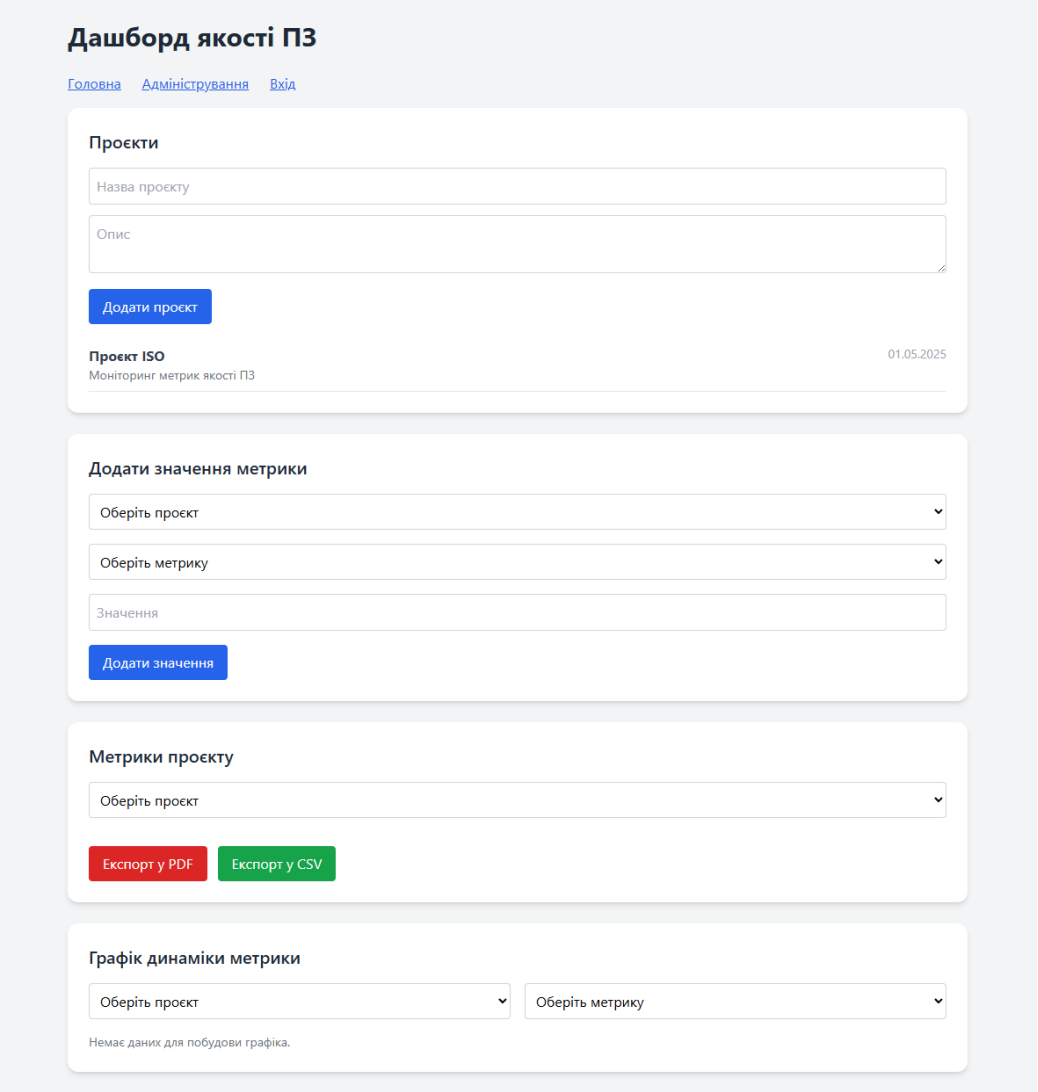


Рисунок 5.18 – Головна сторінка дашборду з переліком проєктів

5.3.4 Експорт звітів у PDF та CSV

У рамках управління якістю програмних проєктів особливу роль відіграє можливість формування звітності. Звіти мають бути доступними у форматі, який легко передати зацікавленим сторонам: керівникам, зовнішнім аудиторам, технічним спеціалістам. Така потреба повністю відповідає стандартам ISO 9001 та ISO/IEC 25010, які наголошують на важливості документування якості, відтворюваності результатів оцінювання, а також здатності довести факт відповідності характеристик якості цільовим показникам [8, 10].

Після реалізації базових функцій інтерфейсу й візуалізації метрик наступним логічним етапом стало додавання механізмів експорту – у формати PDF (лістинг 5.20) для друкованої/офіційної звітності та CSV (лістинг 5.21) для подальшого аналізу в Excel або BI-системах.

Для реалізації функції експорту в PDF було обрано бібліотеку jsPDF у поєднанні з плагіном jspdf-autotable (лістинг 5.19). Цей зв’язок дозволяє створювати документ, що містить як текстові елементи, так і таблиці з колонками, адаптованими до розміру сторінки.

Усі функції експорту було винесено в окремий компонент ExportButtons.js, який приймає на вхід дані метрик (data) та назву проєкту (projectName). Це дозволяє використовувати його в будь-якому місці інтерфейсу – наприклад, під таблицею метрик або поруч із графіком.

Лістинг 5.19 – Імпорт бібліотек

```
import jsPDF from "jspdf";
import autoTable from "jspdf-autotable";
import Papa from "papaparse";
import { saveAs } from "file-saver";
import "../fonts/Roboto-normal";
```

Лістинг 5.20 – Експорт у PDF

```
const exportPDF = () => {
    const doc = new jsPDF();
    doc.setFont("Roboto");
    doc.text(`Звіт для проєкту: ${projectName}`, 14, 16);
```

```
    autoTable(doc, {
        startY: 20,
        head: [["Метрика", "Категорія", "Значення", "Ціль",
"Попередження", "Критичний", "Дата"]],
        body: data.map((m) => [
        m.metric_name,
        m.category_name,
        `${m.value} ${m.unit}`,
        m.target_value,
        m.warning_threshold,
        m.critical_threshold,
        new Date(m.measured_at).toLocaleDateString(),
        ]),
        styles: {
        font: "Roboto", // <-- тут явно вказуємо шрифт
        fontStyle: "normal",
        fontSize: 10, // можна змінити
        },
        headStyles: {
        font: "Roboto",
        fontStyle: "normal"
        },
        bodyStyles: {
        font: "Roboto",
        fontStyle: "normal"
        }
    });
    doc.save(`report_${projectName}.pdf`);
};
```

Щоб PDF-файли коректно відображали кириличні символи (зокрема українську мову), було виконано генерацію спеціального шрифту у форматі .ttf з подальшою конвертацією його у формат, сумісний із jsPDF.

Для цього використовувалась утиліта ttf2js, яка дозволяє перетворити TrueType-шрифт у JavaScript-модуль.

Лістинг 5.21 – Експорт у CSV

```
const exportCSV = () => {
    const csv = Papa.unparse(
      data.map((m) => ({
        metric: m.metric_name,
        category: m.category_name,
        value: m.value,
        unit: m.unit,
        target: m.target_value,
        warning: m.warning_threshold,
        critical: m.critical_threshold,
```

```
        date: new Date(m.measured_at).toLocaleDateString(),
      }))
    );
    const blob = new Blob([csv], { type: "text/csv;charset=utf-8;"
});
    saveAs(blob, `report_${projectName}.csv`);
  };

  return (
    <div className="flex gap-3 mt-4">
      <button
        onClick={exportPDF}
        className="bg-red-600 text-white px-4 py-2 rounded
hover:bg-red-700"
      >
        Експорт у PDF
      </button>
      <button
        onClick={exportCSV}
        className="bg-green-600 text-white px-4 py-2 rounded
hover:bg-green-700"
      >
        Експорт у CSV
      </button>
    </div>
  );
```

Компонент ExportButtons (лістинг 5.22) було інтегровано в MetricTable.js, де відображається перелік метрик. Це логічне розташування, оскільки саме ця таблиця є основою для звіту (рисунок 5.19).

Лістинг 5.22 – Додана кнопка до форми з таблицею метрик

```
<ExportButtons data={metrics} projectName={projects.find(p => p.id
=== parseInt(selectedProject))?.name || "проект"} />
```

Рисунок 5.19 – Кнопки експорту

Таким чином, користувач у будь-який момент може натиснути відповідну кнопку й одразу завантажити звіт у бажаному форматі.

Особливості реалізації:

– таблиця у PDF будується динамічно, з урахуванням даних, отриманих через REST API;

– формат дати автоматично перетворюється у локалізований вигляд;

– під час формування CSV дотримується стандартна структура з заголовками, що дозволяє одразу імпортувати файл у Excel;

– усі файли називаються динамічно за назвою проєкту (звіт_<ім’я>.pdf).

У структурі звіту враховується можливість мультимовної локалізації (рисунок 5.20, 5.21). У подальшому можна автоматично змінювати заголовки колонок відповідно до мови інтерфейсу, що відповідає вимогам ISO 9001 щодо інтернаціоналізації та доступності звітності для різних учасників проєкту.

Звіт для проєкту: Проєкт ISO

Метрика	Категорія	Значення	Ціль	Попередження	Критичний	Дата
Reusable Code Ratio	Модульність	12 %	30	20	10	06.05.2025
Module Coupling Index	Модульність	11 бал	5	8	12	06.05.2025
Migration Time	Переносимість	4.2 години	2	4	6	06.05.2025
Platform Support Rate	Переносимість	78 %	100	90	75	06.05.2025
Change Lead Time	Супроводжуваність	5 дні	2	4	7	06.05.2025
Code Complexity (Cyclomatic)	Супроводжуваність	14 бал	10	15	20	06.05.2025
Authentication Fail Rate	Безпека	3.2 %	1	3	5	06.05.2025
Vulnerabilities Found	Безпека	3 шт	0	1	3	06.05.2025
API Interoperability	Сумісність	85 %	100	90	80	06.05.2025
Browser Compatibility Score	Сумісність	90 %	100	95	85	06.05.2025
CPU Utilization	Продуктивність	84 %	70	85	95	06.05.2025
Average Response Time	Продуктивність	350 мс	200	400	800	06.05.2025
Task Completion Time	Зручність використання	30 секунди	15	30	60	06.05.2025
User Error Rate	Зручність використання	2.5 %	1	3	5	06.05.2025
Passed Functional Tests	Функціональна придатність	83 %	98	90	80	06.05.2025
Requirements Coverage	Функціональна придатність	85 %	100	90	80	06.05.2025
Mean Time Between Failures (MTBF)	Надійність	30 години	72	48	24	06.05.2025
Defect Density	Надійність	0.6 defects/KLOC	0.5	1	2	06.05.2025
Reusable Code Ratio	Модульність	9 %	30	20	10	05.05.2025
Module Coupling Index	Модульність	7.8 бал	5	8	12	05.05.2025
Migration Time	Переносимість	5.5 години	2	4	6	05.05.2025
Platform Support Rate	Переносимість	85 %	100	90	75	05.05.2025
Change Lead Time	Супроводжуваність	6 дні	2	4	7	05.05.2025
Code Complexity (Cyclomatic)	Супроводжуваність	17 бал	10	15	20	05.05.2025
Authentication Fail Rate	Безпека	4.9 %	1	3	5	05.05.2025

Рисунок 5.20 – Фрагмент сформованого PDF-файлу

metric	category	value	unit	target	warning	critical	date
Reusable Code Ratio	Модульність	12	%	30	20	10	06.05.2025
Module Coupling Index	Модульність	11	бал	5	8	12	06.05.2025
Migration Time	Переносимість	4.2	години	2	4	6	06.05.2025
Platform Support Rate	Переносимість	78	%	100	90	75	06.05.2025
Change Lead Time	Супроводжуваність	5	дні	2	4	7	06.05.2025
Code Complexity (Cyclomatic)	Супроводжуваність	14	бал	10	15	20	06.05.2025
Authentication Fail Rate	Безпека	3.2	%	1	3	5	06.05.2025
Vulnerabilities Found	Безпека	3	шт	0	1	3	06.05.2025
API Interoperability	Сумісність	85	%	100	90	80	06.05.2025
Browser Compatibility Score	Сумісність	90	%	100	95	85	06.05.2025
CPU Utilization	Продуктивність	84	%	70	85	95	06.05.2025
Average Response Time	Продуктивність	350	мс	200	400	800	06.05.2025
Task Completion Time	Зручність використання	30	секунди	15	30	60	06.05.2025
User Error Rate	Зручність використання	2.5	%	1	3	5	06.05.2025
Passed Functional Tests	Функціональна придатність	83	%	98	90	80	06.05.2025
Requirements Coverage	Функціональна придатність	85	%	100	90	80	06.05.2025
Mean Time Between Failures (MTBF)	Надійність	30	години	72	48	24	06.05.2025
Defect Density	Надійність	0.6	defects/KLOC	0.5	1	2	06.05.2025
Reusable Code Ratio	Модульність	9	%	30	20	10	05.05.2025
Module Coupling Index	Модульність	7.8	бал	5	8	12	05.05.2025
Migration Time	Переносимість	5.5	години	2	4	6	05.05.2025
Platform Support Rate	Переносимість	85	%	100	90	75	05.05.2025
Change Lead Time	Супроводжуваність	6	дні	2	4	7	05.05.2025
Code Complexity (Cyclomatic)	Супроводжуваність	17	бал	10	15	20	05.05.2025
Authentication Fail Rate	Безпека	4.9	%	1	3	5	05.05.2025
Vulnerabilities Found	Безпека	4	шт	0	1	3	05.05.2025
API Interoperability	Сумісність	89	%	100	90	80	05.05.2025
Browser Compatibility Score	Сумісність	93	%	100	95	85	05.05.2025
CPU Utilization	Продуктивність	88	%	70	85	95	05.05.2025
Average Response Time	Продуктивність	600	мс	200	400	800	05.05.2025
Task Completion Time	Зручність використання	50	секунди	15	30	60	05.05.2025
User Error Rate	Зручність використання	4.5	%	1	3	5	05.05.2025
Passed Functional Tests	Функціональна придатність	88	%	98	90	80	05.05.2025
Requirements Coverage	Функціональна придатність	89	%	100	90	80	05.05.2025
Mean Time Between Failures (MTBF)	Надійність	40	години	72	48	24	05.05.2025
Defect Density	Надійність	1.1	defects/KLOC	0.5	1	2	05.05.2025

Рисунок 5.21 – Фрагмент сформованого CSV-файлу

5.3.5 Управління категоріями та метриками

На етапі створення повноцінного дашборду виникла необхідність у реалізації інтерфейсу для адміністративного керування: додавання, редагування як самих метрик якості, так і їхніх категорій. Такий функціонал є важливою складовою, оскільки структура метрик не є фіксованою: вона може змінюватись залежно від проєкту, стандартів чи фази життєвого циклу.

У відповідності до принципів ISO/IEC 12207, що передбачають адаптивність інструментів управління якістю, було вирішено реалізувати

окрему адміністративну панель, доступ до якої мають лише авторизовані користувачі з відповідними правами [12].

У межах цієї функціональності було реалізовано:

– редагування й створення категорій якості відповідно до моделі ISO/IEC 25010;

– створення нових метрик, їх зв’язок із категоріями, встановлення одиниць вимірювання, цільових значень і порогів;

– валідацію введення й збереження змін у базу даних через API.

Центральною точкою адміністрування виступає компонент «AdminPanel», який відображає дві основні частини:

– управління категоріями (CategoryManager);

– управління метриками (MetricManager).

Інтерфейс побудований за принципом "два розділи на одній сторінці", що дозволяє одночасно переглядати всі категорії та метрики.

Структура функціоналу управління категоріями:

– виведення поточних категорій (запит до /api/categories);

– форма для додавання нової категорії (лістинг 5.23);

– валідація на порожні значення;

– автоматичне оновлення після створення.

Лістинг 5.23 – Фрагмент коду додавання категорії

```
const fetchCategories = () => {
  axios.get(API).then((res) => setCategories(res.data));
};
const handleSubmit = (e) => {
  e.preventDefault();
  if (!name) return;
  axios
    .post(API, { name, description: desc })
    .then(() => {
      setName("");
      setDesc("");
      fetchCategories();
    });
};
```

Кожна категорія має лише назву та опис. Після додавання або редагування – категорії миттєво оновлюються без перезавантаження сторінки (рисунок 5.22). Перевірка на дублікати реалізується на рівні бази даних (через UNIQUE або ручну перевірку на бекенді).

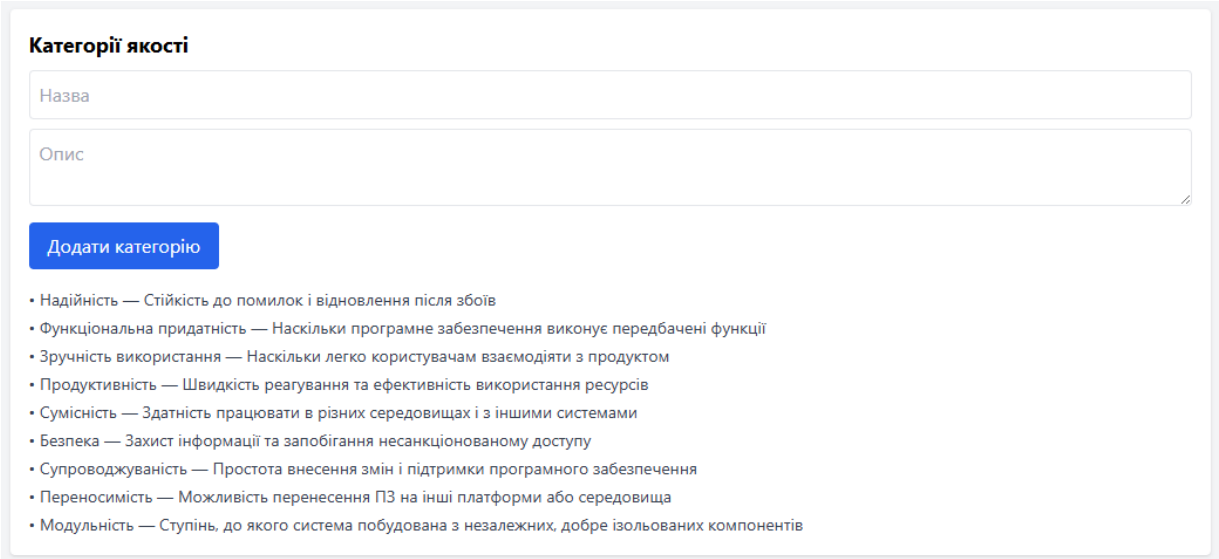


Рисунок 5.22 – Форма для додавання категорії з переліком вже існуючих

Структура функціоналу управління метриками:

– завантаження всіх категорій для вибору;

– форма для створення нової метрики (лістинг 5.24);

– заповнення параметрів: назва, опис, одиниця вимірювання, ціль, попереджувальне й критичне значення;

– вибір категорії, до якої метрика належить;

– запис у базу даних через API /api/metrics.

Лістинг 5.24 – Фрагмент коду додавання метрики

```
const fetchAll = () => {
  axios.get(API_CATS).then((res) => setCategories(res.data));
  axios.get(API_METRICS).then((res) => setMetrics(res.data));
};
const handleChange = (e) => {
  setForm({ ...form, [e.target.name]: e.target.value });
};
```

```
const handleSubmit = (e) => {
  e.preventDefault();
  axios.post(API_METRICS, form).then(() => {
    setForm({
      name: "",
      description: "",
      unit: "",
      target_value: "",
      warning_threshold: "",
      critical_threshold: "",
      category_id: "",
    });
    fetchAll();
  });
};
```

Кожна метрика чітко зв’язується з однією категорією якості (рис. 5.23). Зберігаються цільові значення та пороги для подальшої візуальної інтерпретації (кольори графіків, попередження). Одиниці вимірювання зберігаються як текст (%, сек, бал, тощо), що дозволяє застосовувати систему до різних типів метрик.

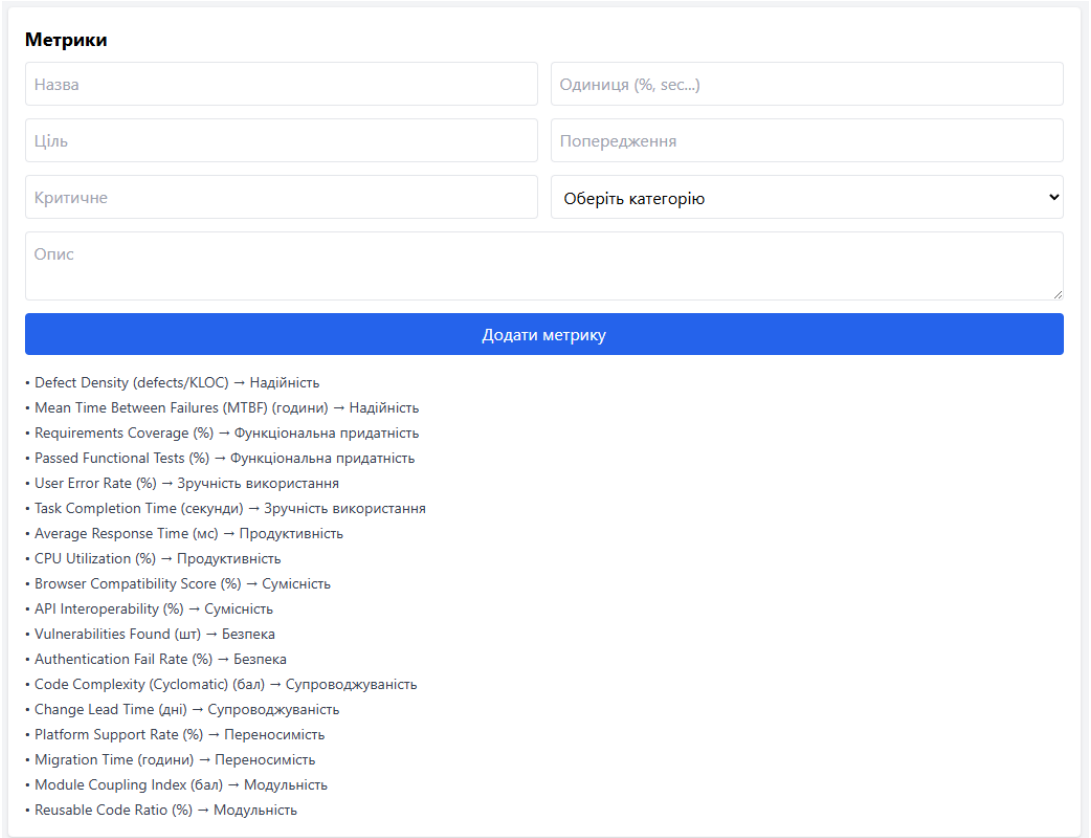


Рисунок 5.23 – Форма для створення нової метрики та переліком всіх метрик із прив’язкою до категорій

Оскільки створення й редагування категорій і метрик є адміністративною функцією, доступ до AdminPanel обмежено. Всі запити до /api/categories і /api/metrics у режимі запису перевіряються через JWT-токен, який зберігається у localStorage і передається в заголовку Authorization.

5.3.6 Автентифікація користувачів

На фінальному етапі реалізації дашборду постало завдання забезпечити контроль доступу до критично важливих функцій системи – зокрема, до адміністративної панелі, де здійснюється редагування категорій якості та створення нових метрик (рисунок 5.27). Відкрита доступність цієї частини інтерфейсу могла б призвести до несанкціонованих змін у системі оцінювання якості програмного забезпечення. Відповідно до міжнародного стандарту ISO/IEC 27001, який регламентує політики інформаційної безпеки, одним з основних принципів є забезпечення автентифікації користувачів та обмеження доступу відповідно до ролей [13, 14].

Саме тому логічним продовженням попередніх етапів стало впровадження повноцінної системи автентифікації на базі JWT (JSON Web Token) – безпечного й сучасного підходу, що дозволяє здійснювати авторизацію без збереження сесій на сервері.

Автентифікація необхідна для захисту маршруту /admin та компонентів CategoryManager та MetricManager. Створення системи ролей для подальшого масштабування. Також, забезбечення безпечного зберігання токена авторизації та здійснення перевірки автентичності токена для захищених API.

Для створення користувачів та зберігання даних авторизації необхідна ще одна таблиця БД (лістинг 5.25, рисунок 5.24).

Лістинг 5.25 – Створення таблиці users

```
CREATE TABLE IF NOT EXISTS users (
    id SERIAL PRIMARY KEY,
    username VARCHAR(50) UNIQUE NOT NULL,
```

```
    password TEXT NOT NULL,
    role VARCHAR(20) DEFAULT 'user'
);
```

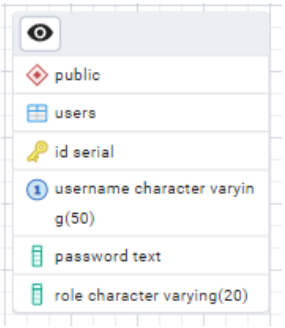


Рисунок 5.24 – ERD таблиці users

Для хешування паролів використовується бібліотека bcrypt, а для генерації токенів – jsonwebtoken (лістинг 5.26).

Лістинг 5.26 – Контролер авторизації

```
const jwt = require("jsonwebtoken");
const bcrypt = require("bcrypt");
const userModel = require("../models/userModel");

const SECRET = "super_secret_key";

const register = async (req, res) => {
  const { username, password } = req.body;
  const existing = await userModel.getByUsername(username);
   if (existing) return res.status(400).json({ error: "Користувач
вже існує" });

  const hashed = await bcrypt.hash(password, 10);
    const user = await userModel.createUser({ username,
hashedPassword: hashed, role: "admin" });
  res.status(201).json(user);
};
const login = async (req, res) => {
  const { username, password } = req.body;
  const user = await userModel.getByUsername(username);
   if (!user) return res.status(400).json({ error: "Невірний логін
або пароль" });
  const valid = await bcrypt.compare(password, user.password);
  if (!valid) return res.status(400).json({ error: "Невірний логін
або пароль" });
   const token = jwt.sign({ id: user.id, username: user.username,
role: user.role }, SECRET, {
```

```
    expiresIn: "2h",
  });
  res.json({ token });
};
module.exports = { register, login };
```

Маршрут /api/auth/login обробляє запит і повертає токен користувачу.

Для реалізації на стороні клієнта було створено глобальний контекст авторизації (AuthContext.js), який зберігає токен у localStorage, розшифровує дані користувача й надає доступ до функцій login() та logout() (лістинг 5.27).

Лістинг 5.27 – Фрагмент коду реалізації контексту

```
import jwtDecode from "jwt-decode";

export const AuthProvider = ({ children }) => {
  const [auth, setAuth] = useState({ token: null, user: null });

  const login = (token) => {
    const decoded = jwtDecode(token);
    localStorage.setItem("token", token);
    setAuth({ token, user: decoded });
  };
  const logout = () => {
    localStorage.removeItem("token");
    setAuth({ token: null, user: null });
  };
  ...
};
```

Токен додається до всіх запитів через axios або через відповідні заголовки.

На окремій сторінці реалізовано форму входу (LoginPage), яка надсилає логін і пароль на бекенд. У разі успішної автентифікації зберігається токен, і користувача перенаправляють на /admin (лістинг 5.28).

Лістинг 5.28 – Перевірка даних авторизації

```
const [username, setUsername] = useState("");
const [password, setPassword] = useState("");
const { login } = useAuth();
const navigate = useNavigate();
```

```
const handleSubmit = async (e) => {
  e.preventDefault();
  try {
                        const          res          =          await
axios.post("http://localhost:5000/api/auth/login", {
      username,
      password,
    });
    login(res.data.token);
    navigate("/admin");
  } catch (err) {
    alert("Невірні дані входу");
  }
};
```

За допомогою React Router реалізовано компонент ProtectedRoute, який перевіряє, чи автентифікований користувач і чи має роль admin (лістинг 5.29).

Лістинг 5.29 – Використання у App.js

```
<Route
  path="/admin"
  element={
    <AdminRoute>
      <AdminPanel />
    </AdminRoute>
  }
/>
```

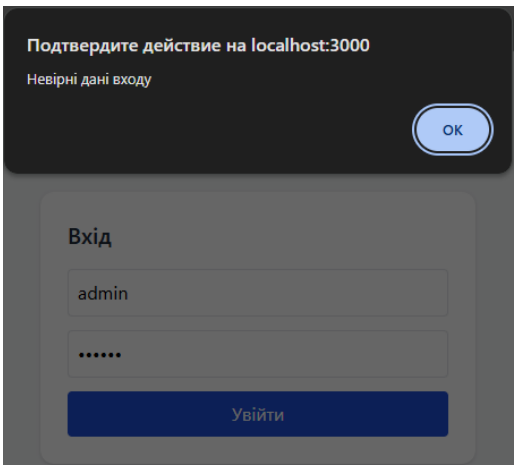


Рисунок 5.25 – Повідомлення про некоректні введені дані

У шапці сторінки виводиться інформація про поточного користувача (лістинг 5.30, рисунок 5.26).

Лістинг 5.30 – Блок відображення користувача

```
{auth.user && (
  <div className="text-sm text-gray-600 mb-4">
    Вітаю, {auth.user.username} ({auth.user.role}) -{" "}
    <button onClick={logout} className="text-blue-700 underline">
      Вийти
    </button>
  </div>
)}
```

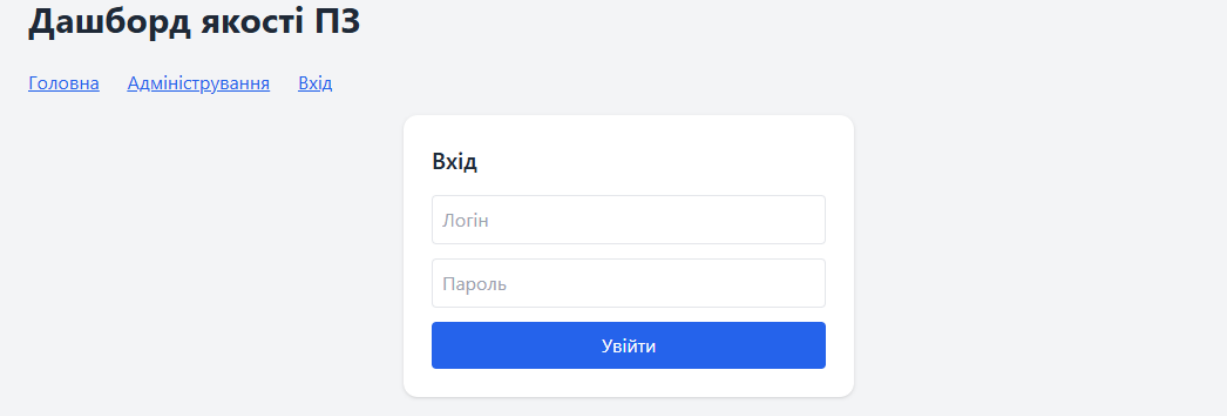


Рисунок 5.26 – Інтерфейс авторизації

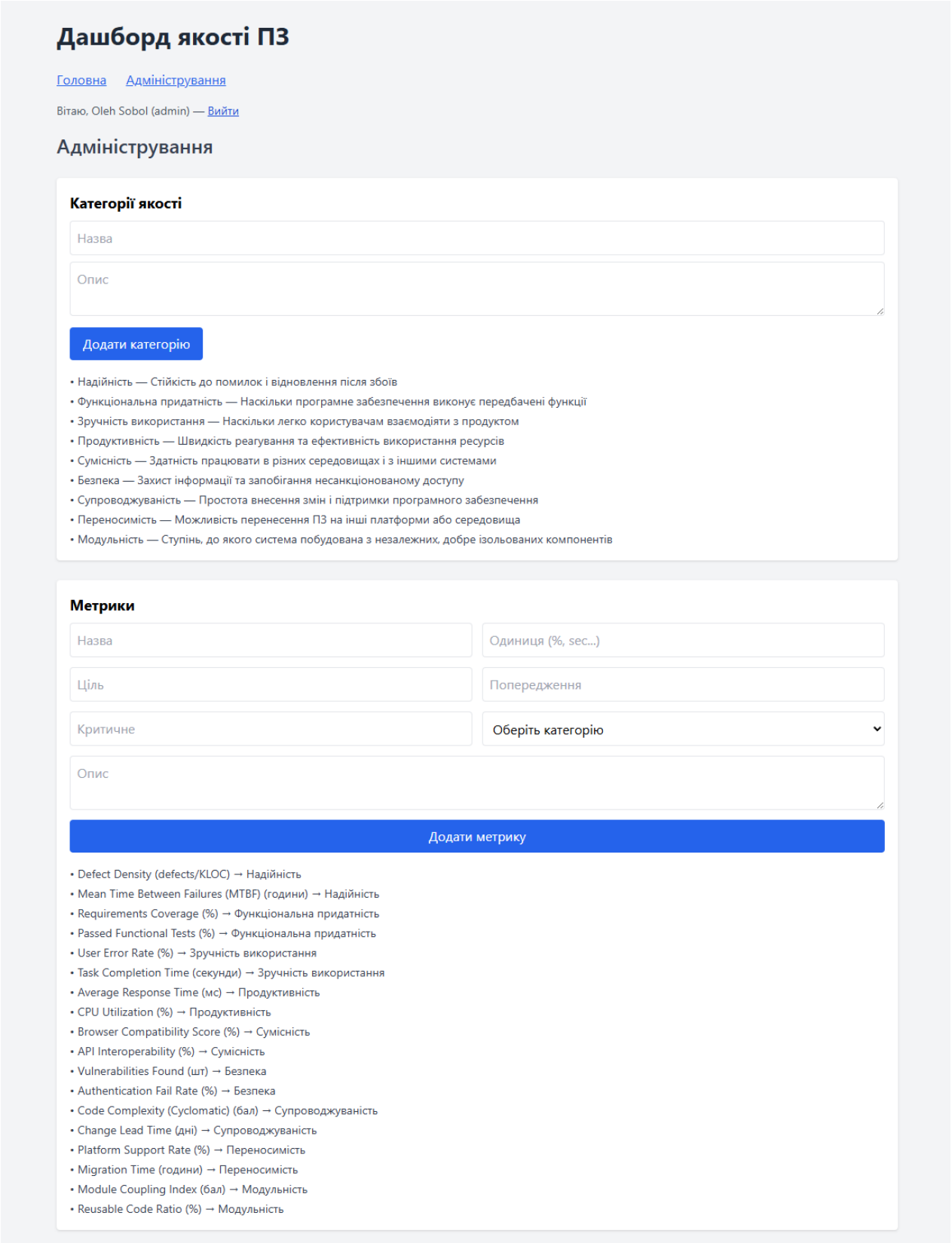


Рисунок 5.27 – Захищена сторінка /admin, доступна лише після входу

5.4 Тестування та оцінка ефективності дашборду

Після завершення розробки та впровадження всіх функціональних компонентів дашборду наступним логічним етапом стало тестування системи з метою перевірки її стабільності, коректності обробки даних, зручності використання та відповідності вимогам стандартів ISO. Тестування дозволило верифікувати працездатність системи в реальних умовах та забезпечити обґрунтовану оцінку її ефективності як інструменту для управління якістю програмного забезпечення [6, 18, 19].

Метою тестування було перевірити повноту реалізації функціоналу, правильність взаємодії між компонентами, стійкість до некоректного вводу, а також відповідність дашборду характеристикам якості згідно з моделлю ISO/IEC 25010: функціональна придатність, продуктивність, зручність використання, сумісність і безпека [10].

Тестування проводилося у середовищі Windows 10 з використанням браузера Google Chrome, pgAdmin 4, інструментів розробника (DevTools), а також засобів журналювання бекенду. Було підготовлено перелік ключових сценаріїв, що охоплюють усі основні можливості системи: авторизацію, управління проєктами, категоріями, метриками, візуалізацію, експорт звітів і захист доступу (таблиця 5.1).

Таблиця 5.1 – Результати тестування

№	Функція	Тест	Очікуваний результат	Результат
1	Авторизація	Вхід з коректними даними	Перехід до /admin, доступ до адміністрування	Пройдено
2		Вхід з некоректним паролем	Повідомлення про помилку входу	Пройдено
3		Спроба доступу до /admin без авторизації	Перенаправлення на /login	Пройдено

Продовження таблиці 5.1

№	Функція	Тест	Очікуваний результат	Результат
4	Проєкти	Створення нового проєкту через форму	Новий проєкт збережено та відображено	Пройдено
5		Видалення проєкту з інтерфейсу	Проєкт зник з таблиці і бази	Пройдено
6	Категорії якості	Додавання нової категорії	Категорія з’являється в списку	Пройдено
7	Метрики	Створення нової метрики з параметрами	Метрика збережена і прив’язана до категорії	Пройдено
8		Введення некоректних даних (порожні поля, нулі)	Валідація з попередженням	Пройдено
9	Значення метрик	Додавання числового значення метрики	Значення збережено і з’явилось в таблиці	Пройдено
10		Відображення значення у графіку	Побудова графіка без помилок	Пройдено
11		Виділення кольором критичних значень	Колірний фон відповідає порогам	Пройдено
12	Експорт звітів	Експорт у PDF	Документ завантажено, дані присутні	Пройдено
13		Експорт у CSV	Дані доступні для відкриття в Excel	Пройдено
14	Безпека та доступ	Запит на API без токена	Відмова у доступі	Пройдено

У результаті перевірки було встановлено, що всі функціональні модулі дашборду працюють коректно, обробляють дані згідно з очікуваннями,

забезпечують стійкість до помилок вводу та захищеність від несанкціонованого доступу.

Оцінка зручності використання проводилась на основі власного експертного аналізу та базових критеріїв моделі ISO/IEC 25010 [10]. Інтерфейс дашборду показав високу інтуїтивність у навігації, логічне розташування елементів, швидке реагування на дії користувача та читабельну візуалізацію метрик. Усі повідомлення про помилки й успішні дії виводяться одразу, що забезпечує своєчасний зворотний зв’язок.

Продуктивність перевірялась на прикладі внесення великої кількості значень метрик (понад 100 записів). Рендеринг графіків і таблиць залишався стабільним, час відповіді API не перевищував 200-300 мс, а побудова графіків – до 1 с. Це свідчить про придатність системи до використання у реальних умовах із більшими обсягами даних.

На основі тестування можна зробити висновок, що реалізований дашборд:

– відповідає принципам ISO/IEC 25010 – функціональність, зручність, продуктивність, безпека [10];

– відповідає ISO 9001 – результати вимірювання документуються, доступні для перегляду/завантаження [8];

– виконує вимоги ISO/IEC 12207 – охоплено повний цикл: планування, реалізація, супровід, контроль [12];

– реалізує положення ISO/IEC 27001 – через механізми авторизації та контроль доступу [13].

5.5 Обмеження, ризики та потенційні напрями покращення

Реалізація дашборду для моніторингу метрик якості програмного забезпечення, описана в попередніх підпунктах, продемонструвала можливість створення інтерактивної, модульної системи оцінювання, яка повністю відповідає вимогам міжнародних стандартів, зокрема ISO/IEC 25010, ISO 9001, ISO/IEC 12207 та ISO/IEC 27001 [8, 10, 12, 13]. Однак, як і будь-яка система на

початковому етапі розвитку, вона має певні обмеження, потенційні ризики при впровадженні, а також напрямки для подальшого вдосконалення.

Одним із головних архітектурних обмежень системи є те, що вона на поточному етапі передбачає одночасне обслуговування одного користувача або дуже обмеженої кількості користувачів у локальному або невеликому внутрішньому середовищі. Хоча використання PostgreSQL і Node.js дозволяє масштабувати застосунок, його теперішня конфігурація, зокрема спосіб зберігання токена в localStorage, є придатною переважно для невеликих інсталяцій або освітніх цілей. У разі розгортання у виробничому середовищі або на рівні підприємства буде необхідно врахувати вимоги до безпечнішого керування сесіями (наприклад, через httpOnly cookies або централізовану систему автентифікації).

Ще одним функціональним обмеженням є відсутність повноцінного редагування або видалення метрик та їх значень. У чинній реалізації акцент зроблено на створення та перегляд, однак із точки зору життєвого циклу даних у системі управління якістю важливою є можливість актуалізації, виправлення або повного вилучення помилково внесеної інформації. Це питання також тісно пов’язане з реалізацією журналювання дій (audit trail), яке на цьому етапі не впроваджене, але є критично важливим у контексті відповідності ISO 9001 та ISO/IEC 27001 [8, 13].

Окрему увагу слід звернути на обмеження, пов’язані з візуалізацією. Поточна реалізація дозволяє побудову простих діаграм (стовпчикових, кругових, лінійних) для загальної аналітики, однак не передбачає гнучких панелей порівняння між проєктами, трендів у розрізі кількох характеристик одночасно, або інтерактивних фільтрів для візуального аналізу великої кількості даних. Для покращення цієї частини варто розглянути інтеграцію з аналітичними платформами, наприклад Power BI або Metabase, або впровадження кастомного drag-and-drop конструктора звітів на клієнтському боці.

У технічному аспекті до потенційних ризиків можна віднести відсутність

механізмів резервного копіювання бази даних. У разі використання системи у виробничому середовищі без належного бекапу можуть виникнути незворотні втрати інформації. Це має бути враховано під час впровадження, особливо у сфері управління якістю, де збереження історії змін є одним з основних вимог до відповідності стандартам.

Серед організаційних ризиків варто згадати можливість недостатнього навчання персоналу, що працює з дашбордом. Незважаючи на інтуїтивність інтерфейсу, неправильна інтерпретація метрик, недбале заповнення або відсутність єдиного підходу до вимірювання можуть призвести до викривлення реальної картини якості продукту. Таким чином, впровадження подібного інструменту повинно супроводжуватись інструкцією користувача, методологією оцінювання та відповідним контролем з боку менеджменту якості.

У контексті розвитку системи варто передбачити її подальшу модульну еволюцію. Перспективним напрямом є розширення набору метрик відповідно до підкатегорій ISO/IEC 25010, реалізація автоматичного імпорту результатів із зовнішніх інструментів тестування (наприклад, CI/CD pipelines або систем unit-тестування), а також додавання можливості порівняння проєктів між собою за динамікою окремих показників [10].

У майбутньому дашборд може стати складовою частиною більшої інформаційної системи управління якістю – інтегрованої з корпоративною CRM, системою керування вимогами, дефектами або навіть з системою управління ризиками. Такий підхід дозволить сформувати замкнутий цикл контролю якості на всіх етапах життєвого циклу програмного продукту, що повністю відповідає стратегічним підходам стандарту ISO/IEC 12207 [12].

Таким чином, незважаючи на обмеження, реалізований дашборд є повноцінною базою для подальшого розвитку, гнучким інструментом моніторингу та аналітики, який вже сьогодні здатний підтримати діяльність з управління якістю програмних проєктів відповідно до міжнародних вимог і стандартів.

ВИСНОВКИ

У межах кваліфікаційної роботи проведено комплексне дослідження методів управління якістю програмних проєктів з урахуванням міжнародних стандартів ISO – зокрема, ISO/IEC 25010, ISO 9001, ISO/IEC 12207 та ISO/IEC 27001. Робота поєднує теоретичний аналіз моделей якості та стандартів у сфері розробки програмного забезпечення з практичною реалізацією інструменту моніторингу – інтерактивного дашборду.

У теоретичній частині розглянуто сучасні підходи до забезпечення якості ПЗ, досліджено структуру життєвого циклу програмного продукту відповідно до ISO/IEC/IEEE 12207, а також класифікацію характеристик якості за ISO/IEC 25010. Особливу увагу приділено питанням процесного управління, надійності документації, важливості зворотного зв’язку з кінцевими користувачами та застосуванню кількісних метрик для прийняття управлінських рішень. Обґрунтовано доцільність автоматизації процесів збору, аналізу та візуалізації показників якості, що особливо актуально в умовах багатокомандної розробки, DevOps-практик і зростаючих вимог до інформаційної безпеки.

Практична частина проєкту передбачала створення вебзастосунку – інтерактивного дашборду для моніторингу метрик якості програмного забезпечення. Система дає змогу класифікувати метрики згідно з категоріями ISO/IEC 25010, відстежувати їхню динаміку у графічному вигляді, формувати звіти у форматах PDF та CSV, а також адмініструвати проєктні параметри. Серверна частина реалізована з використанням Node.js, Express і PostgreSQL; клієнтська – на базі React, Tailwind CSS і Recharts. Для роботи з базою даних замість ORM-рішень застосовано бібліотеку pg із використанням пулу з’єднань (Pool), що забезпечило гнучке управління SQL-запитами й спростило архітектуру системи.

Серед основних функцій дашборду реалізовано: управління проєктами, створення й редагування категорій якості, додавання та перегляд метрик,

побудову діаграм, експорт даних, а також захист адміністративного інтерфейсу за допомогою JWT-аутентифікації. Проведене тестування підтвердило відповідність функціональним вимогам, стабільність роботи та працездатність системи. Оцінка користувацького досвіду засвідчила високий рівень зручності використання (usability), що відповідає вимогам стандарту ISO/IEC 25010.

Попри окремі обмеження – зокрема відсутність функцій редагування або видалення записів і базову масштабованість – розроблена система є придатною для використання в навчальних, дослідницьких та внутрішньокорпоративних цілях. Її архітектура передбачає можливість адаптації до різних сценаріїв, подальше розширення функціональності та використання як основи для створення повноцінної системи управління якістю програмного забезпечення.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Лузан І. В., Луценко І. С. Система управління якістю як фактор підвищення конкурентоспроможності підприємства // Актуальні проблеми економіки та управління : збірник наукових праць молодих вчених. Київ, 2012. Т. 6. С. 1–6.

2. Дорош М.С., Ребенок А.В. Інтеграція систем управління проектами в систему організації на різних етапах розвитку // Управління проектами та розвиток виробництва : збірних наукових праць. Луганськ: вид во СНУ ім. В.Даля. Луганськ, 2009. Т 4. С. 21.

3. ISO/IEC 9126. Information Technology. – Software Quality Characteristics and metrics.

4. Соммервілл І. Інженерія програмного забезпечення. 6-те вид., доп. Київ: Видавничий дім «Вільямс», 2002. 624 с.

5. Ліпаєв В.В. Програмна інженерія. Методологічні основи / В.В. Ліпаєв – М.: Teic, 2006. 608 с.

6. ISO: Global standards for trusted goods and services.

7. ISO/IEC/IEEE 29119-1:2022 – Software and systems engineering — Software testing.

8. ISO 9001:2015 – Quality management systems – Requirements.

9. Зосим М. Цикл Демінга (The Deming Cycle), він же - Цикл Шухарта, Цикл PDCA / PDSA. Maxzosim. 07.09.2022. URL: https://www.maxzosim.com/deming-cycle-pdca/ (дата звернення: 13.03.2025).

10. ISO/IEC 25010:2023 – Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – Product quality model.

11. Lamatrice G. Gli indicatori di qualita del software secondo ISO/IEC 25010. Impresa24web. INFORMATICA AZIENDALE. 10.03.2020. URL: https://www.impresa24web.it/gli-indicatori-di-qualita-del-software-secondo-iso-iec-25010/ (дата звернення: 13.03.2025).

12. ISO/IEC/IEEE 12207:2017 – Systems and software engineering – Software life cycle processes.

13. ISO/IEC 27001:2022 – Information security, cybersecurity and privacy protection — Information security management systems — Requirements.

14. What is ISO 27001? A detailed, simple, and straightforward guide. URL: https://www.controlcase.com/what-is-iso-27001/ (дата звернення: 20.04.2025).

15. Systems, Software and Services Process Improvement : 27th European Conference, Eurospi 2020, Düsseldorf, Germany, september 9–11, 2020, proceedings / M. Yilmaz та ін. ; Communications in Computer and Information Science, 2020. 617–628 с.

16. 7 Steps to Follow for ISO Management System Implementation. Quality-assurance. URL: https://quality-assurance.com.au/blog/7-steps-to-follow-for-iso-management-system-implementation/ (дата звернення: 13.03.2025).

17. IEEE Std 730-2014 – IEEE Standard for Software Quality Assurance Processes.

18. Канер С., Фолк Д., Нгуен Е. К. Тестування програмного забезпечення : Фундаментальні концепції менеджменту бізнес-додатків. 2-ге вид., Київ : ДіаСофт, 2001. 544 с.

19. IEEE Std 829-2008 – IEEE Standard for Software and System Test Documentation.

20. McConnell S. Code Complete: A Practical Handbook of Software Construction. 2nd edition. Washington : Microsoft Press, 2004. 916 с.

21. Запорожець О., Соболь О., Гриньов Д., Зиков І. Інтеграція стандартів ISO в управління програмними проєктами // Системи управління, навігації та зв’язку. Збірник наукових праць. 2025. Т. 1. № 79. С. 31–36.

23. Test Coverage Metrics – A Complete Overview. URL: https://testsigma.com/blog/test-coverage-metrics/ (дата звернення: 19.04.2025).

24. Usability Metrics : Nielsen Norman Group. URL: https://www.nngroup.com/articles/usability-metrics/ (дата звернення:

22.04.2025).

25. ISO 9001 – Clause 9 – Performance Evaluation : ISMS.online. URL: https://www.isms.online/iso-9001/clause-9-performance-evaluation/ (дата звернення: 20.04.2025).

26. The Ultimate Guide to ISO 27001. URL: What is ISO/IEC 27001, The Information Security Standard (дата звернення: 20.04.2025).

27. Bourque P., Fairley R.E. Guide to the Software Engineering Body of Knowledge - SWEBOK V3.0 : IEEE and IEEE Computer Society, 2014.