МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВЕТЕРИНАРНОЇ МЕДИЦИНИ ТА БІОТЕХНОЛОГІЙ ІМ. С.З. ҐЖИЦЬКОГО
ФАКУЛЬТЕТ МЕХАНІКИ, ЕНЕРГЕТИКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

першого (бакалаврського) рівня вищої освіти

на тему: **«Проєктування веб-системи управління проєктами з використанням методологій гнучкої розробки»**

Виконав: студент 4 курсу групи Кн-41

Спеціальності 122 «Комп'ютерні науки»
(шифр і назва)

Шиян Ростислав Віталійович
(Прізвище та ініціали)

Керівник: к.т.н., доцент Падюка Р.І.
(Прізвище та ініціали)

ДУБЛЯНИ-2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ПРИРОДОКОРИСТУВАННЯ
ФАКУЛЬТЕТ МЕХАНІКИ, ЕНЕРГЕТИКИ ТА
ІНФОРМАЦІЙНИХ ТЕХНЕОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Перший (бакалаврський) рівень вищої освіти
Спеціальність 122 «Комп'ютерні науки»

«ЗАТВЕРДЖУЮ»

Завідувач кафедри_____

д.т.н., проф. А. М. Тригуба
«____» _____2025 р.

ЗАВДАННЯ

на кваліфікаційну роботу студенту

Шияну Ростиславу Віталійовичу

1. Тема роботи: «Проєктування веб-системи управління проєктами з використанням методологій гнучкої розробки»

Керівник роботи Падюка Роман Іванович, доцент
затверджені наказом по університету від 25.02.2025 року № 123/к-с.

2. Строк подання студентом роботи 10.06.2025 р.
3. Вихідні дані до роботи: *вимоги до проектування інформаційних систем; методика проектування інформаційних систем управління проектами, методологія управління проектами інформаційних систем.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які необхідно розробити)
Вступ.
1. Аналіз предметної області.
2. Аналіз систем управління, що використовують AGILE-методології
3.Проєктування веб-системи управління проєктами з використанням методологій гнучкої розробки
4. Охорона праці та безпека в надзвичайних ситуаціях
Висновки та пропозиції.
Список використаної літератури.
5. Перелік ілюстраційного матеріалу (з точним зазначенням обов'язкових креслень): *Актуальність теми, мета та завдання дослідження, теоретичні засади Agile-методологій, практичне застосування Agile, порівняння Agile-сумісних систем,*

архітектура веб-системи, функціональні модулі системи, база даних та логіка обробки, результати реалізації системи.

6. Консультанти з розділів:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1, 2, 3	*Падюка Р.І., доцент кафедри ІТ*		
4	*Городецький І.М., доцент кафедри інженерної механіки*		

7. Дата видачі завдання 26 лютого 2025 р.

Календарний план

№ з/п	Назва етапів дипломного проекту	Терміни виконання етапів роботи	Примітка
1	*Написання першого розділу*	*26.02.25-20.03.25*	
2	*Виконання другого розділу та аркушів ілюстраційного матеріалу до нього*	*21.03-15.04.25*	
3.	*Виконання третього розділу та аркушів ілюстраційного матеріалу до нього*	*16.04-30.04.25*	
4.	*Написання розділу «Охорона праці»*	*01-15.05.25*	
5.	*Завершення оформлення розрахунково-пояснювальної записки та аркушів ілюстраційного матеріалу*	*15-30.05.25*	
6.	*Завершення роботи вцілому*	*01 - 10.06.25*	

Студент_____ Шиян Р. В.
(підпис)

Керівник роботи _____Падюка Р.І.
(підпис)

УДК 004.9 : 631.1

Проєктування веб-системи управління проєктами з використанням методологій гнучкої розробки. Шиян Р. В.. Кафедра ІТ – Дубляни, Львівський НУВМБ ім. С.З. Гжицького, 2025.

Кваліфікаційна робота: 67 с. текст. част., 19 рис., 2 табл., 11 арк. ілюстраційного матеріалу, 22 джерел.

Ефективне управління проектами, особливо в ІТ, промисловості, освіті та бізнесі, все більше покладається на гнучкі, прозорі системи, які включають гнучкі методики, такі як Scrum, Kanban і Lean, щоб адаптуватися до швидких технологічних змін. Сучасні веб-інструменти керування, такі як Jira, ClickUp, Wrike і Asana, підтримують ці підходи за допомогою настроюваних інтерфейсів, можливостей інтеграції та автоматизації, сприяючи ефективному розподілу завдань, моніторингу прогресу та взаємодії команди.

Розробка таких систем бере за основу не лише на технічні характеристики проєкту, але й акцентує увагу на важливості організаційної культури, підтримки управління та практик постійного вдосконалення для забезпечення успішного впровадження та масштабованості в різних секторах, включаючи зростаюче впровадження в Україні практик Agile.

Ключові слова: моніторинг, управління проектами, методологія гнучкої розробки, Agile, веб-система

Key words: monitoring, project management, agile development methodology, Agile, web system

ЗМІСТ

ВСТУП

Ефективне управління проєктами є ключовим чинником успішної реалізації будь-якої ініціативи в галузі ІТ, промисловості, освіти та бізнесу. З огляду на стрімкий розвиток цифрових технологій, зростає попит на інструменти, що забезпечують гнучке, адаптивне та прозоре керування командами й процесами розробки. Особливої актуальності набувають системи управління проєктами, що орієнтовані на застосування Agile-методологій — підходів, які дозволяють адаптувати процес розробки до змінних умов, підвищити продуктивність командної роботи та досягти високого рівня задоволеності замовника.

Сучасні потреби бізнесу вимагають створення програмних рішень, які не лише автоматизують рутинні процеси планування й контролю, а й інтегрують у собі гнучкі інструменти управління задачами, ролями, дедлайнами та взаємодією між учасниками проєкту. У зв’язку з цим виникає потреба у проєктуванні веб-орієнтованих систем управління проєктами, що поєднують доступність, масштабованість, зручність використання та підтримку методів гнучкої розробки — таких як Scrum, Kanban, Lean тощо.

Метою цієї кваліфікаційної роботи є проєктування веб-системи управління проєктами з орієнтацією на методології Agile, яка забезпечить можливості ефективного розподілу задач, моніторингу прогресу, взаємодії між членами команди та оперативного реагування на зміни вимог.

Актуальність теми обумовлена зростаючою популярністю гнучких методів управління проєктами у світовій ІТ-індустрії та недостатнім рівнем адаптації таких підходів у локальних програмних рішеннях. Розробка універсальної, доступної та ефективної системи, що відповідає потребам малих і середніх команд, сприятиме покращенню організації праці та підвищенню якості реалізованих проєктів.

РОЗДІЛ 1.
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Теоретичні засади Agile-методологій

Зародження Agile як концептуального підходу до управління проєктами стало результатом еволюційного пошуку ефективних методів розробки програмного забезпечення, які дозволяють адаптуватися до змін, швидко доставляти цінність клієнту і підтримувати постійну взаємодію в команді. До появи Agile більшість команд використовували традиційні методи, що базувалися на детальному плануванні і суворому дотриманні послідовних етапів, таких як вимоги, проєктування, реалізація, тестування та впровадження [4]. Ці підходи виявлялися недостатньо гнучкими в умовах швидко змінюваного середовища IT-індустрії.

У 2001 році в штаті Юта група провідних фахівців з розробки програмного забезпечення сформулювала Agile Manifesto — маніфест гнучкої розробки. У документі викладено чотири базові цінності: люди та взаємодії важливіші за процеси та інструменти; працююче програмне забезпечення важливіше за вичерпну документацію; співпраця з клієнтом важливіша за погодження умов контракту; реагування на зміни важливіше за дотримання плану [6]. Ці цінності доповнено дванадцятьма принципами, що спрямовують команди до адаптивності, гнучкості та постійного вдосконалення.

Agile — це не одна методологія, а ціла сукупність методів і практик, що реалізують ці принципи. Найбільш відомими серед них є Scrum, Kanban, Extreme Programming (XP), Lean, Crystal, DSDM та FDD. Методології мають спільну ідеологію, але відрізняються механізмами реалізації гнучкого підходу.

Scrum — одна з найпоширеніших Agile-методик, яка організовує роботу команди в ітераціях тривалістю 2–4 тижні (Schwaber & Sutherland, 2020). Команда працює над конкретним переліком завдань (backlog), регулярно проводить зустрічі — щоденні стендапи, планування спринту, огляд результатів

та ретроспективу. Канонічна структура Scrum дозволяє забезпечити ритмічну та передбачувану розробку, де кожна ітерація дає змогу згенерувати готову до використання версію продукту. Kanban, у свою чергу, базується на візуалізації потоку завдань та обмеженні кількості одночасно виконуваних задач, що допомагає зменшити час виконання робіт і виявити "вузькі місця" в процесі [2].

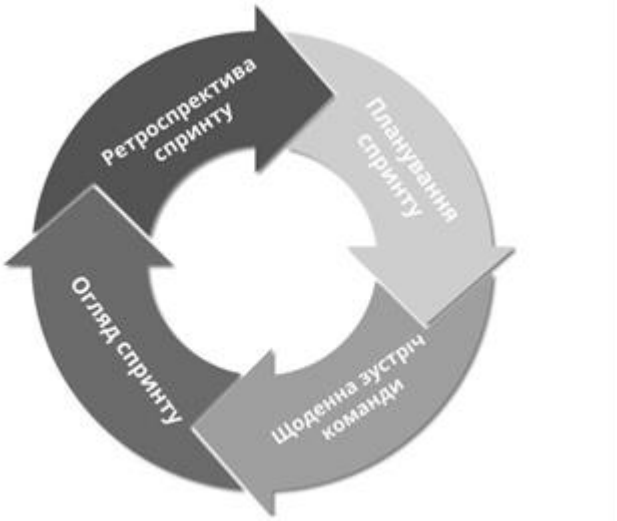


Рис. 1.1. Візуалізація циклу спринту.

Extreme Programming акцентує увагу на високій якості коду та технічній досконалості. Він запроваджує такі практики, як тестування, розробка через тестування (TDD), парне програмування та часті релізи. Постійне технічне вдосконалення, закладене в XP, дозволяє команді не лише зберігати стабільність програмного забезпечення, але й ефективно нарощувати функціонал без шкоди якості.

Lean, що походить із виробництва, сфокусований на максимізації цінності та усуненні втрат, що не приносять доданої вартості [7]. Філософія Lean орієнтована на оптимізацію всіх процесів, і її принципи, зокрема «відкладене прийняття рішень», «вбудована якість» і «загальна оптимізація», лягли в основу багатьох Agile-практик.

Рис. 1.2. Схема Lean розробки.

Методології Crystal і DSDM більше орієнтовані на адаптацію методів до умов конкретного проєкту, враховуючи його розмір, складність і критичність. Вони дозволяють варіювати правила та артефакти залежно від характеристик команди та домену, забезпечуючи баланс між формальністю та гнучкістю. Наприклад, Crystal дозволяє уникнути перевантаження документацією в малих командах, зберігаючи при цьому ефективну комунікацію.

Варто підкреслити, що Agile не є лише сукупністю технік чи процедур — це насамперед культурна трансформація організації. Впровадження Agile передбачає переосмислення ролей, управлінських функцій, способів взаємодії всередині команд та між ними. Центральне місце в цій трансформації посідає ідея довіри до команди та розподіленої відповідальності. Традиційна вертикальна ієрархія поступається місцем горизонтальним зв'язкам, де важливішим є спільне прийняття рішень, ніж контроль зверху.

Важливо зазначити, що впровадження Agile не завжди означає повну відмову від традиційних підходів. Існують гібридні моделі, які поєднують елементи PMBOK або PRINCE2 з гнучкими практиками. Це дозволяє зберегти певний рівень планування та формалізації в поєднанні з гнучкістю та

клієнтоорієнтованістю [10]. Такі інтегровані моделі набувають популярності у великих компаніях та в державному секторі, де необхідне одночасне забезпечення звітності, відповідності регламентам та адаптивності в реалізації проєктів.

Одним із важливих напрямів сучасного осмислення Agile є його застосування поза межами IT. Хоча методологія зародилася у сфері розробки програмного забезпечення, її принципи універсальні. У контексті управління змінами, розробки нових продуктів, інноваційної діяльності — Agile демонструє високу ефективність, дозволяючи швидко адаптуватися до змінних умов і забезпечувати сталі результати в умовах невизначеності.

Важливо також усвідомлювати, що Agile — не панацея. Це інструмент, ефективність якого залежить від контексту, культури організації, готовності команд до самостійності та відкритості до змін. Впровадження Agile без належної підготовки, навчання та підтримки з боку керівництва часто призводить до зворотного ефекту — зниження продуктивності, хаосу в роботі та демотивації співробітників. Саме тому багато організацій використовують Agile-коучинг, фасилітацію та інші інструменти підтримки, що сприяють поступовій та стійкій трансформації.

Добре, продовжую розширення, інтегруючи тему застосування Agile у різних сферах, зокрема освіті, медицині та державному управлінні, а також розкриваю деякі ключові виклики та перспективи методології.

Розширюючи межі застосування Agile, слід підкреслити, що сьогодні гнучкі методології успішно адаптують у різноманітних сферах діяльності поза традиційною IT-індустрією. У сфері освіти Agile сприяє формуванню динамічних, орієнтованих на учня навчальних середовищ, де навчальні програми та методики викладання постійно адаптуються до індивідуальних потреб та швидких змін у знаннях і технологіях. Використання Scrum чи Kanban у школах і університетах дозволяє педагогам краще планувати навчальний процес, а студентам — більш усвідомлено підходити до власного навчання. Цей підхід

покращує мотивацію учнів, розвиває критичне мислення, сприяє формуванню навичок співпраці та самоменеджменту, що є ключовими у сучасному світі [7].

У медицині Agile допомагає швидше реагувати на потреби пацієнтів, оптимізувати клінічні процеси та розробку медичного програмного забезпечення, а також впроваджувати інноваційні підходи до лікування та організації медичної допомоги. Гнучкі методики дозволяють мультидисциплінарним командам лікарів, медсестер і технічних фахівців швидко координувати свої дії, обмінюватися інформацією та оперативно адаптувати клінічні протоколи до нових наукових даних і технологічних змін. В умовах пандемій та інших криз Agile став важливим інструментом для швидкої перебудови медичних сервісів і прийняття рішень в умовах невизначеності.

Державне управління — ще одна сфера, де Agile набуває поширення, особливо в контексті цифрової трансформації урядових послуг. В умовах високої бюрократії і консервативних структур застосування гнучких методів допомагає підвищити прозорість, підзвітність і ефективність роботи органів влади. Agile-підходи стимулюють активну взаємодію між державними службовцями, громадськістю та іншими стейкхолдерами, сприяючи формуванню політик, орієнтованих на реальні потреби населення. Багато країн запроваджують Agile-практики в розробці і модернізації державних інформаційних систем, що дозволяє пришвидшити запуск нових сервісів і краще адаптувати їх до вимог користувачів [4].

Ключовим викликом при впровадженні Agile у будь-якій сфері є збереження балансу між гнучкістю та необхідністю формалізації. У державних організаціях, зокрема, існує потреба у суворому дотриманні нормативів і процедур, що іноді суперечить швидкості та адаптивності Agile. Подолання цих протиріч потребує грамотного управління змінами, підтримки з боку керівництва та поступового навчання персоналу.

Перспективним напрямом розвитку Agile є його поєднання з іншими сучасними концепціями управління, такими як DevOps, що інтегрує розробку і експлуатацію програмних продуктів, або Design Thinking, який орієнтується на

глибоке розуміння користувачів і креативне вирішення проблем. Такий синтез дозволяє максимально розкрити потенціал Agile, забезпечуючи комплексний підхід до створення цінності та інновацій.

В підсумку, Agile-методології трансформували підхід до управління проєктами, ставши не лише набором практик, а філософією організаційної культури, що підтримує гнучкість, інновації і постійний розвиток. Застосування Agile в різних сферах життя і бізнесу демонструє його універсальність і здатність допомагати організаціям успішно функціонувати в умовах динамічних змін і високої невизначеності.

1.2 Практичне застосування Agile в управлінні проєктами

Практичне застосування Agile-методологій свідчить про їхню здатність не лише адаптуватися під мінливі умови бізнес-середовища, а й трансформувати самі підходи до організації роботи, співпраці і досягнення цілей. Основні Agile-фреймворки — Scrum, Kanban, Extreme Programming — демонструють високу ефективність у різних проєктних контекстах, дозволяючи командам швидко орієнтуватися в пріоритетах, підтримувати високу якість продукту та максимально залучати замовника у процес створення цінності.

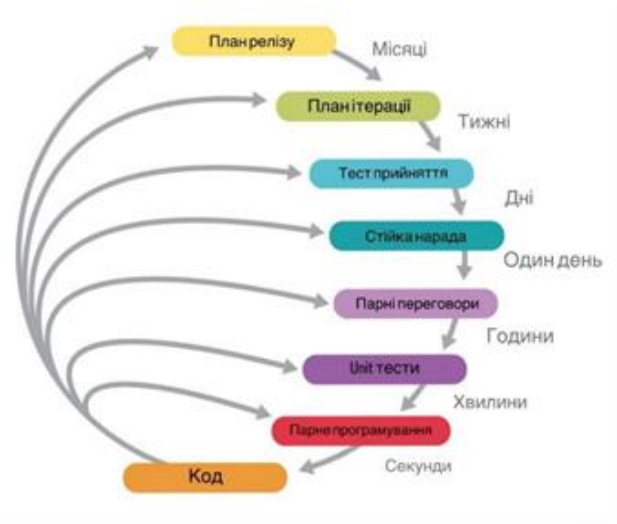


Рис. 1.3. Візуалізація Extreme Programming

В основі Scrum лежить концепція ітераційної роботи, де кожен спринт закінчується створенням потенційно готового продукту або його частини, що дає

змогу отримати швидкий зворотній зв’язок від користувачів і оперативно адаптувати подальші дії. Ролі Product Owner, Scrum Master і команда розробників чітко розподіляють відповідальність, що сприяє ефективній комунікації і мінімізує конфлікти (Schwaber & Sutherland, 2020). Постійні зустрічі — планування, щоденні стендапи, огляд результатів і ретроспектива — створюють механізми для усунення проблем і впровадження покращень.

Kanban, на відміну від Scrum, не має жорстких часових рамок і більше орієнтується на безперервний потік роботи. Візуалізація завдань на дошці Kanban дозволяє всім членам команди та зацікавленим сторонам бачити стан проєкту у режимі реального часу. Обмеження кількості завдань у роботі (Work in Progress) знижує багатозадачність і збільшує продуктивність (Anderson, 2010). Цей підхід є особливо ефективним у сферах із частими змінами пріоритетів та потребою швидкого реагування, наприклад, у службах підтримки або медіа.

Рис. 1.4. Візуалізація Kanban-дошки.

Extreme Programming (XP) розширює класичні Agile-принципи, фокусуючись на технічних практиках. Регулярне парне програмування, розробка через тестування (TDD), безперервна інтеграція та часті релізи забезпечують високу якість коду і зменшують ризики появи дефектів (Beck, 1999). XP допомагає командам не лише швидко адаптуватися, а й підтримувати стабільність і надійність програмного забезпечення.

У великих організаціях часто застосовують масштабовані Agile-моделі, такі як SAFe (Scaled Agile Framework), LeSS (Large Scale Scrum) або Nexus. Вони розроблені для координації багатьох команд, що працюють над складними проєктами, зберігаючи при цьому ключові Agile-принципи гнучкості та інтерактивності (Leffingwell, 2018). Масштабування включає організаційні зміни, узгодження процесів і більш формалізовані механізми управління залежностями між командами.

Застосування Agile у практиці часто супроводжується впровадженням спеціалізованих інструментів. Jira, Trello, Asana, Monday.com — це лише кілька прикладів систем, що підтримують планування, відстеження завдань і аналітику виконання. Вони дозволяють збирати дані про продуктивність команд, визначати «вузькі місця» процесу, підвищувати прозорість і відповідальність.

Важливим аспектом є роль Agile-коучів, які допомагають командам і організаціям перейти від традиційних підходів до гнучких методологій. Вони сприяють зміні корпоративної культури, навчають правильному використанню інструментів і практик, допомагають долати внутрішній опір і непорозуміння (Denning, 2018). Без цього процес переходу може бути формальним і не давати очікуваних результатів.

У багатьох відомих компаніях, таких як Spotify, Google, Amazon та Microsoft, Agile став базовою парадигмою розробки продуктів. Spotify, наприклад, впровадила унікальну організаційну структуру, що складається зі скводів, племен і гільдій, яка дозволяє масштабувати Agile і підтримувати автономію команд при координації на рівні всієї компанії (Kniberg & Ivarsson, 2012). Такі підходи показали, що Agile можна ефективно використовувати не лише на рівні окремих команд, а й масштабувати на великі організації.

Таким чином, практична реалізація Agile-методологій демонструє їхню високу ефективність і гнучкість у широкому спектрі бізнес-сценаріїв і організаційних структур. Водночас вона вимагає системного підходу, підтримки з боку керівництва та культури відкритості до змін.

1.3 Переваги, виклики та перспективи розвитку Agile-методологій

Переваги Agile-методологій очевидні і підтверджуються як емпіричними дослідженнями, так і численними кейсами з практики. Головною перевагою є здатність швидко реагувати на зміни вимог і ринкових умов, що у сучасному бізнесі має вирішальне значення. Замість жорсткого слідування початковому плану команди мають можливість коригувати пріоритети, що зменшує ризики створення продукту, який не відповідає очікуванням користувачів [7].

Клієнтоорієнтованість є ще однією суттєвою перевагою Agile. Постійна взаємодія з замовником, демонстрація робочих продуктів і регулярне отримання зворотного зв’язку допомагають створювати цінність і покращувати якість кінцевого результату. Це сприяє не лише задоволенню клієнта, а й формуванню довгострокових партнерських відносин.

Адаптивність Agile дозволяє командам швидко долати непередбачені проблеми та враховувати нові ідеї, що виникають у процесі роботи. Постійні ретроспективи сприяють культурі безперервного вдосконалення, де помилки розглядаються як джерело навчання, а не як підстави для покарань. Такий підхід зміцнює командний дух і мотивацію учасників проєкту.

Водночас впровадження Agile стикається з певними викликами. Не всі організації готові до радикальної трансформації управлінських процесів і корпоративної культури. Часто відсутність належної підтримки з боку керівництва, низький рівень розуміння методології або брак досвіду призводять до поверхневого впровадження Agile — так званого «фреймворк-агілу», коли використовуються лише зовнішні атрибути, а глибинні принципи і цінності ігноруються [4].

Традиційна ієрархія, бюрократія і формалізм у великих організаціях можуть гальмувати впровадження гнучких практик. Особливо це помітно у державних структурах або консервативних корпораціях, де зміни сприймаються з недовірою. Для подолання цих бар’єрів потрібен комплексний підхід —

навчання, підтримка змін на всіх рівнях, створення середовища довіри і відкритості.

Масштабування Agile для великих проєктів і організацій є окремою складною темою. Моделі SAFe, LeSS, Nexus допомагають поєднувати гнучкість з необхідною координацією та управлінням залежностями, проте їхнє впровадження вимагає значних ресурсів і часу [6]. Крім того, не існує універсального рецепту, і кожна організація має підбирати модель, що відповідає її контексту.

Щодо перспектив розвитку, Agile продовжує розширювати межі свого застосування. Використання Agile у державному управлінні, освіті, медицині свідчить про універсальність підходу. Зростає роль інтеграції Agile з іншими сучасними концепціями, зокрема DevOps, що забезпечує безперервну інтеграцію і доставку програмного забезпечення, а також з Design Thinking, що акцентує увагу на користувачеві і креативному вирішенні проблем [7].

У майбутньому значну роль відіграватиме штучний інтелект. Інтелектуальні системи зможуть автоматизувати планування, прогнозувати ризики, надавати рекомендації щодо пріоритизації завдань та аналізувати продуктивність команд у режимі реального часу. Це дозволить підвищити ефективність Agile-процесів і зробить їх ще більш адаптивними та точними.

Таким чином, Agile-методології вже сьогодні є ключовим фактором успіху для багатьох організацій. Вони не лише змінюють спосіб розробки продуктів і управління проєктами, а й формують нову культуру роботи, орієнтовану на гнучкість, співпрацю та безперервний розвиток.

1.4. Аналіз успішних кейсів впровадження Agile-методологій в Україні

Впровадження Agile-методологій у вітчизняних компаніях відображає загальносвітову тенденцію переходу до більш гнучких і адаптивних моделей управління проєктами. Український IT-сектор, який є одним із найбільш

динамічних у регіоні Східної Європи, став своєрідним плацдармом для застосування Agile-підходів, що зумовлено як вимогами ринку, так і внутрішніми організаційними процесами. За даними аналітичного звіту Ukraine IT Report 2023, понад 78% українських IT-компаній використовують Agile-методології у своїй діяльності, причому найбільш популярними залишаються Scrum і Kanban (Ukraine IT Report, 2023).

Одним із яскравих прикладів успішного впровадження Agile є компанія SoftServe — найбільший український IT-аутсорсер, який налічує понад 12 000 працівників по всьому світу. Компанія розпочала масштабне впровадження Scrum ще у середині 2010-х, поступово трансформуючи підходи до розробки продуктів і сервісів. Згідно з внутрішнім дослідженням SoftServe (2019), після переходу на Agile у проектах скоротився середній час виходу нових продуктів на ринок на 30%, а рівень задоволеності клієнтів зріс на 25%. Agile сприймається тут як філософія організаційної культури, що сприяє відкритості, прозорості і залученості кожного члена команди.

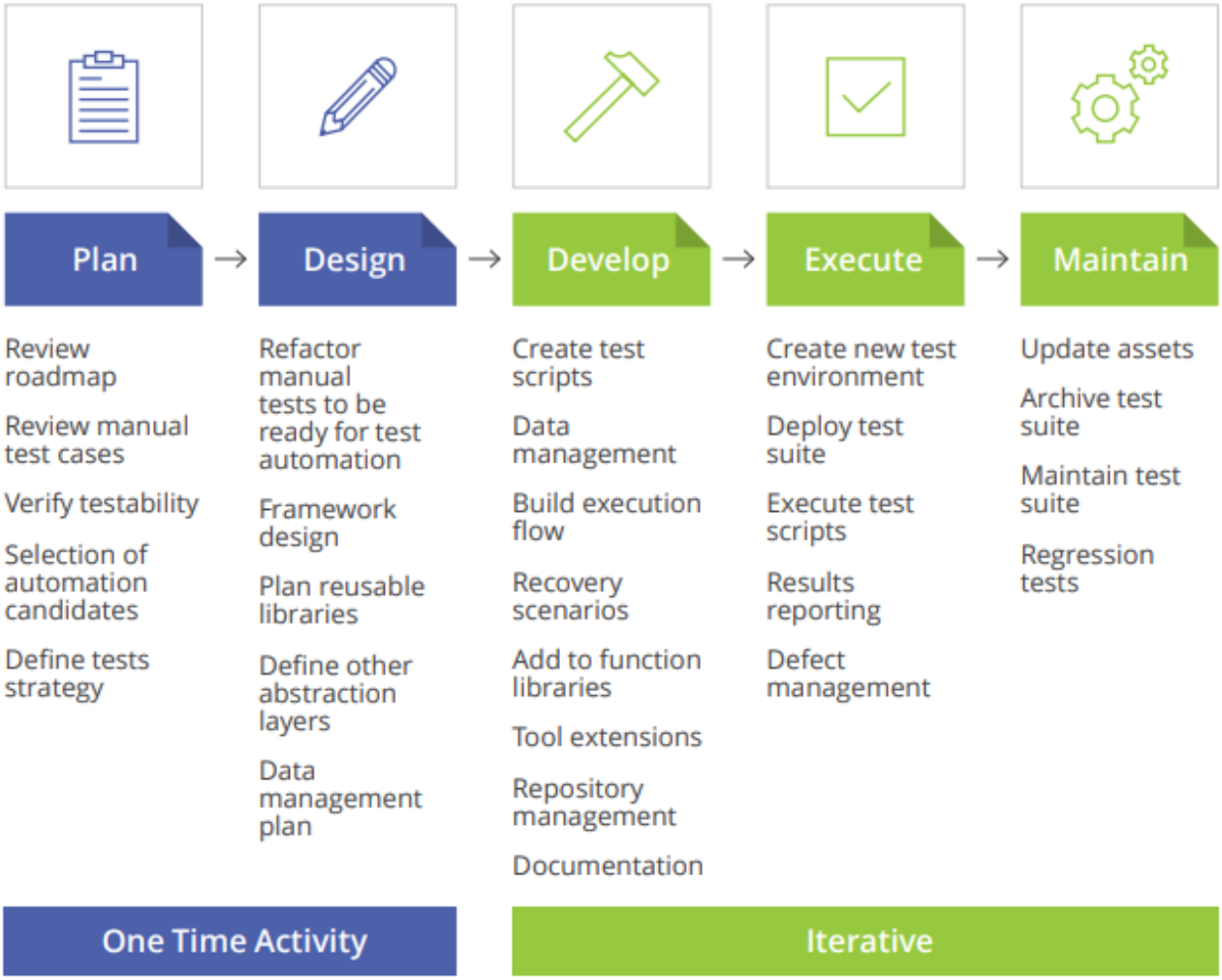


Рис. 1.5 Автоматизація тестування в Agile процесі SDLC [7]

Ще одним прикладом є компанія EPAM Systems, яка в Україні підтримує стандарти світового рівня і використовує комбіновані Agile-практики, включно зі Scrum, Kanban та XP. EPAM успішно масштабує Agile у великих мультидисциплінарних командах, що працюють над проєктами для фінансового сектору, телекомунікацій та енергетики. Згідно з річним звітом EPAM за 2021 рік, понад 85% проєктів компанії в Україні використовували гнучкі методології, що дозволило підвищити продуктивність команд на 20% та знизити кількість дефектів у кінцевому продукті на 18% (EPAM, 2021).

Варто згадати і невеликі українські стартапи, які завдяки Agile швидко вийшли на міжнародні ринки. Так, освітня платформа Prometheus, що функціонує з 2016 року, активно використовує Scrum і Lean Startup методології, що допомогло швидко тестувати ідеї, адаптувати курси під запити користувачів

і створити міцну ком’юніті навколо платформи. За статистикою Prometheus (2020), застосування Agile дозволило скоротити час розробки нових курсів з 4 місяців до 2,5, а кількість активних користувачів платформи за три роки зросла на 150%, що стало наслідком оперативного реагування на потреби ринку та зворотній зв’язок студентів.

У державному секторі також починаються експерименти з Agile, хоч і повільніше, ніж у приватному бізнесі. Українське Міністерство цифрової трансформації реалізує проєкти, зокрема «Дія», за принципами Agile, що дозволяє швидко впроваджувати нові сервіси та оперативно реагувати на зворотний зв’язок користувачів. За офіційними даними Мінцифри, застосування гнучких методів управління дозволило скоротити час розробки нових функцій платформи на 35% у порівнянні з традиційними підходами (Мінцифри, 2022). Це свідчить про потенціал Agile у підвищенні ефективності державних IT-проєктів.

Аналіз вказаних кейсів свідчить, що ключовими факторами успішного впровадження Agile в Україні є: підтримка з боку керівництва, готовність до культурних змін, інвестиції в навчання команд та належна інструментальна підтримка. Згідно з опитуванням Ukrainian Agile Community (2023), 67% респондентів вважають, що найбільшою перешкодою є недостатнє розуміння Agile-принципів на рівні топ-менеджменту, а 55% відзначають опір змінам серед співробітників. Водночас 72% вважають, що навчання і менторство є ключовими чинниками успішної трансформації.

Попри позитивні результати, українські компанії стикаються і з викликами. Серед основних проблем – формальне впровадження Agile без зміни мислення, надмірна бюрократія в організаціях та нестача досвіду. Важливою тенденцією є поширення коучингу, тренінгів і сертифікаційних програм, які допомагають командам уникнути цих пасток і ефективно адаптувати методології. За даними Ukrainian Agile Association, кількість сертифікованих Agile-фахівців в Україні зросла на 40% за останні три роки, що свідчить про підвищення рівня професіоналізму у цій сфері (UAA, 2023).

Отже, досвід України підтверджує, що Agile-методології можуть стати потужним драйвером розвитку підприємств у різних сферах діяльності, сприяють підвищенню конкурентоспроможності та інноваційності, а також відповідають вимогам сучасного динамічного ринку. Водночас для подальшого поширення Agile необхідно посилювати освітню підтримку, підвищувати культуру змін і забезпечувати інтеграцію гнучких підходів у стратегічні цілі організацій.

РОЗДІЛ 2.
АНАЛІЗ СИСТЕМ УПРАВЛІННЯ, ЩО ВИКОРИСТОВУЮТЬ AGILE-МЕТОДОЛОГІЇ

2.1. Сучасні типи Agile-сумісних систем управління та їх функціональні особливості

Agile-методологія кардинально змінила підходи до управління проєктами. Однією з ключових умов її ефективного впровадження є застосування спеціалізованих цифрових систем, які дозволяють організувати прозору, ітеративну та адаптивну модель роботи. Згідно з принципами Agile Manifesto, система управління має не лише підтримувати гнучке планування, а й забезпечувати постійну комунікацію між усіма учасниками проєкту, а також швидко реагувати на зворотний зв’язок користувачів. У цьому контексті виник новий клас систем — Agile-сумісні платформи управління проєктами.

Основними функціональними складовими таких систем є організація backlog’у, формування спринтів, розподіл ролей та прав доступу, підтримка Kanban- і Scrum-дошок, інструменти для планування завдань, відстеження часу, ведення звітності та аналітики. До того ж більшість сучасних платформ інтегруються з DevOps-практиками, CI/CD-пайплайнами, системами зберігання коду (GitHub, GitLab), API для автоматизації та підключення до сторонніх сервісів. Особливо важливими стали дашборди для моніторингу продуктивності, аналітики навантаження та прогнозування ризиків на основі історичних даних.

Із розширенням масштабів використання Agile-систем ці рішення адаптуються до різноманітних галузей — від IT до логістики, від медичних проєктів до стартапів. Такі платформи відіграють не лише роль диспетчера завдань, але й сприяють трансформації культури організації: забезпечують прозорість процесів, заохочують самоорганізацію команд і підтримують гнучке прийняття рішень.

Серед характеристик сучасних Agile-систем варто виділити:

- ітеративність і підтримку Scrum/Kanban-механік;
- автоматичне створення інкрементів і ведення журналів змін;
- адаптацію до масштабу команди (від малих груп до тисяч користувачів);
- широкі можливості кастомізації під бізнес-процеси організації;
- хмарні й локальні версії для різного рівня безпеки.

Використання таких систем безпосередньо сприяє підвищенню ефективності, мінімізації ризиків затримки проєктів та зростанню командної відповідальності за результат.

2.2 Огляд систем управління проєктами, що використовують Agile методології

Jira від компанії Atlassian є найбільш популярною системою для організації роботи за методологіями Scrum та Kanban. Система дозволяє створювати backlog, планувати спринти, візуалізувати прогрес на дошках, створювати кастомізовані робочі процеси (workflows), вести історію змін і формувати звіти. Її перевагами є висока гнучкість, масштабованість для великих організацій, інтеграція з Confluence, Bitbucket, Slack та іншими інструментами. Проте система може бути складною для новачків і потребує часу на навчання, а також належить до платних рішень з обмеженням безкоштовного плану для малих команд.

Wrike

Wrike — платформа, орієнтована на співпрацю між крос-функціональними командами. Підтримує Agile-підходи, але має й розширений набір функцій для управління ресурсами, фінансами та документообігом. Її цінують за зручну візуалізацію проєктів, часову шкалу (Gantt), інтеграцію з Google Workspace та Microsoft 365. Wrike підійде для середніх і великих компаній, однак у безкоштовному варіанті суттєво обмежені функції аналітики та кастомізації.

ClickUp

ClickUp — універсальна платформа, що дозволяє працювати в різних форматах (списки, дошки, календарі, таймлайни, діаграми Ганта). Вона підтримує функціонал Scrum/Kanban-дошок, OKR, ведення цілей, автоматизацію робочих процесів, голосування всередині завдань. ClickUp поєднує простоту Trello з функціональністю Jira. Основна перевага — висока кастомізованість інтерфейсу та приваблива ціна. Недоліком може бути перевантаженість інтерфейсу через наявність великої кількості модулів.

Trello

Trello залишається одним із найпростіших інструментів для візуалізації проєктів у форматі Kanban-дошки. Його переваги — простота, доступність, інтуїтивно зрозумілий інтерфейс. Він підходить для невеликих команд або неформальних проєктів. Недоліком є обмеженість звітності, відсутність підтримки складних ієрархій завдань та аналітики.

Asana

Asana поєднує функціональність Kanban і Scrum із потужним інтерфейсом для менеджменту цілей, дотримання дедлайнів, автоматизації задач. Вона популярна серед маркетингових та HR-команд. Переваги — простота налаштувань, інтеграції з понад 200 сервісами. До недоліків належать висока вартість повного функціоналу та обмежена аналітика в базових тарифах.

Таким чином, вибір платформи управління проєктами має ґрунтуватися на масштабі організації, рівні зрілості Agile-культури, технічних потребах та структурі команд.

2.3 Інтеграція штучного інтелекту, DevOps та автоматизації в Agile-системах управління

Одним із найпомітніших векторів розвитку Agile-інструментів останнього десятиліття є інтеграція зі штучним інтелектом (ШІ), DevOps-практиками та засобами автоматизації. Ці компоненти суттєво підвищують

ефективність процесів управління проєктами, скорочують людські помилки, сприяють адаптивному прийняттю рішень та забезпечують гнучке масштабування IT-продуктів.

Штучний інтелект усе активніше застосовується у вигляді віртуальних помічників для постановки завдань, автоматичного розподілу навантаження між членами команди, прогнозування термінів виконання спринтів, а також у створенні рекомендацій на основі попередніх спринтів або динаміки velocity. У системах, таких як ClickUp або Asana, реалізовані алгоритми, що виявляють критичні залежності між задачами, затримки у виконанні, а також прогнозують вплив цих факторів на загальний проєкт.

У рамках DevOps-інтеграції платформи на кшталт Jira, GitLab або Azure DevOps надають можливість повної синхронізації розробки, тестування та розгортання. Так, команди можуть у режимі реального часу отримувати аналітику з пайплайнів CI/CD, моніторити баги, фіксувати зміни в коді, а також швидко планувати нові ітерації відповідно до отриманих інсайтів. DevOps забезпечує безперервність Agile-циклів, трансформуючи окремі етапи проєкту в єдину автоматизовану систему зворотного зв’язку.

Автоматизація в сучасних системах управління охоплює створення правил тригерів, повторюваних задач, шаблонів проєктів і цілей. У Wrike чи ClickUp користувачі можуть налаштувати, наприклад, автоматичне оновлення статусу задачі після зміни підзадач, або надсилання сповіщення в Slack після завершення блоку. Це дозволяє знизити адміністративне навантаження, зосередивши увагу команди на ключових аспектах розробки.

Важливим є також використання API для побудови індивідуальних рішень. У більшості систем реалізовано відкритий API, що дає змогу будувати власні інтеграції з CRM, ERP, сервісами підтримки клієнтів, хмарними сховищами, що значно розширює можливості кастомізації та впровадження Agile в контексті конкретного бізнесу.

Таким чином, поєднання Agile, ШІ, DevOps і автоматизації відкриває новий рівень гнучкого управління, який базується не лише на людському досвіді,

а й на аналізі великих обсягів даних, прогнозуванні та машинному навчанні, що у підсумку дозволяє підвищити точність і темпи реалізації проєктів.

2.4 Роль культури управління в ефективному використанні Agile-систем

Впровадження Agile-систем в управління проєктами є не лише технічним або процесним завданням, а насамперед — питанням зміни організаційної культури. Культура Agile передбачає відкритість до змін, гнучкість мислення, довіру між учасниками команди, самоорганізацію та постійну самооцінку. Без зміни управлінської філософії навіть найсучасніші системи Jira, Wrike чи Trello можуть не дати очікуваного результату.

На практиці успішне впровадження Agile-систем починається із залучення керівництва. Лідери мають підтримувати ідеї гнучкості, орієнтації на результат замість процесу, відкритого діалогу та розподілу відповідальності. Відхід від командно-адміністративної моделі до сервісного лідерства дозволяє командам самостійно приймати рішення, динамічно адаптуватися до змін вимог і ефективно комунікувати всередині організації.

Важливе значення має також розвиток емоційного інтелекту в команді, що дозволяє формувати культуру зворотного зв’язку, мінімізувати конфлікти та підтримувати високу мотивацію. Agile-системи мають бути налаштовані не лише з технічної точки зору, а й на рівні комунікаційної прозорості: наявність єдиної інформаційної панелі (dashboard), каналів зворотного зв’язку, відкритих backlog’ів, доступу до звітів, а також дотримання єдиних стандартів виконання задач сприяє підвищенню довіри й відповідальності.

Не менш важливим є розвиток практик регулярної ретроспективи. В системах управління проєктами це можуть бути аналітичні модулі, що дозволяють оцінити ефективність спринтів, навантаження команд, динаміку velocity. Культура покращення через аналіз дозволяє уникнути повторення помилок, а також будувати командну стратегію на основі об’єктивних метрик.

Система винагород і оцінювання персоналу також має бути адаптована до Agile-контексту. Замість акценту на індивідуальних KPI варто впроваджувати метрики командної ефективності, customer satisfaction (CSAT), lead time тощо. Це стимулює не конкуренцію, а співпрацю в межах команди, що є сутністю Agile.

Таким чином, успіх застосування систем управління, які працюють за принципами Agile, значною мірою залежить від глибини впровадження відповідної культури в організаціях. Agile — це не лише про інструменти, а передусім — про людей і взаємодію між ними.

2.5 Вибір систем управління проєктами з використанням Agile-методологій

У процесі дослідження застосування Agile-підходів в управлінні проєктами особливу увагу було приділено аналізу сучасних цифрових систем, які забезпечують підтримку таких методологій. З-поміж найбільш поширених рішень, які активно впроваджуються в IT-компаніях України та світу, виокремлено чотири лідери ринку: Jira, ClickUp, Wrike та Asana. Для комплексного аналізу їх ефективності були відібрані ключові критерії, що мають безпосередній вплив на результативність Agile-проєктів: гнучкість кастомізації, інтеграції, зручність інтерфейсу, аналітика та автоматизація.

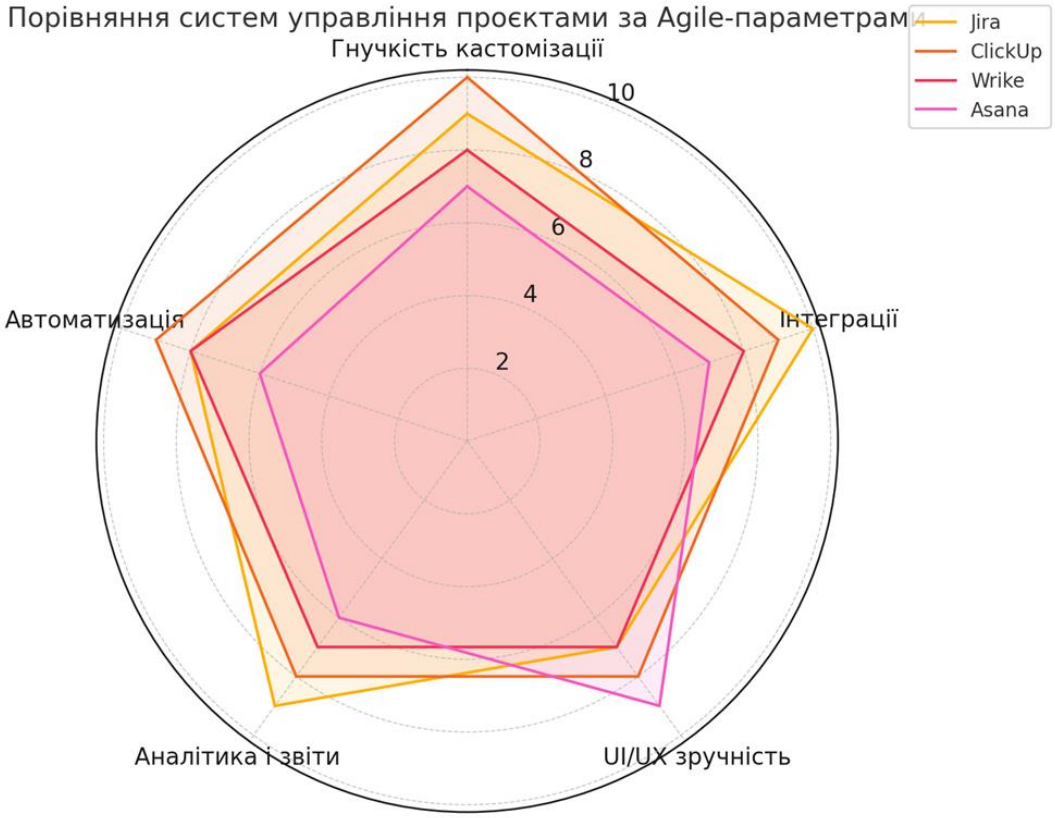


Рис. 2.1. Діаграма порівняння систем управління проєктами

Нами побудовано радарну діаграма, що ілюструє порівняння систем управління проєктами — Jira, ClickUp, Wrike та Asana — за п’ятьма ключовими критеріями: гнучкість кастомізації, інтеграції, зручність інтерфейсу, аналітика та звітність, а також автоматизація.

Результати оцінювання представлено у вигляді стовпчастої діаграми (див. Рис. 1), що наочно демонструють сильні та слабкі сторони кожного з розглянутих інструментів.

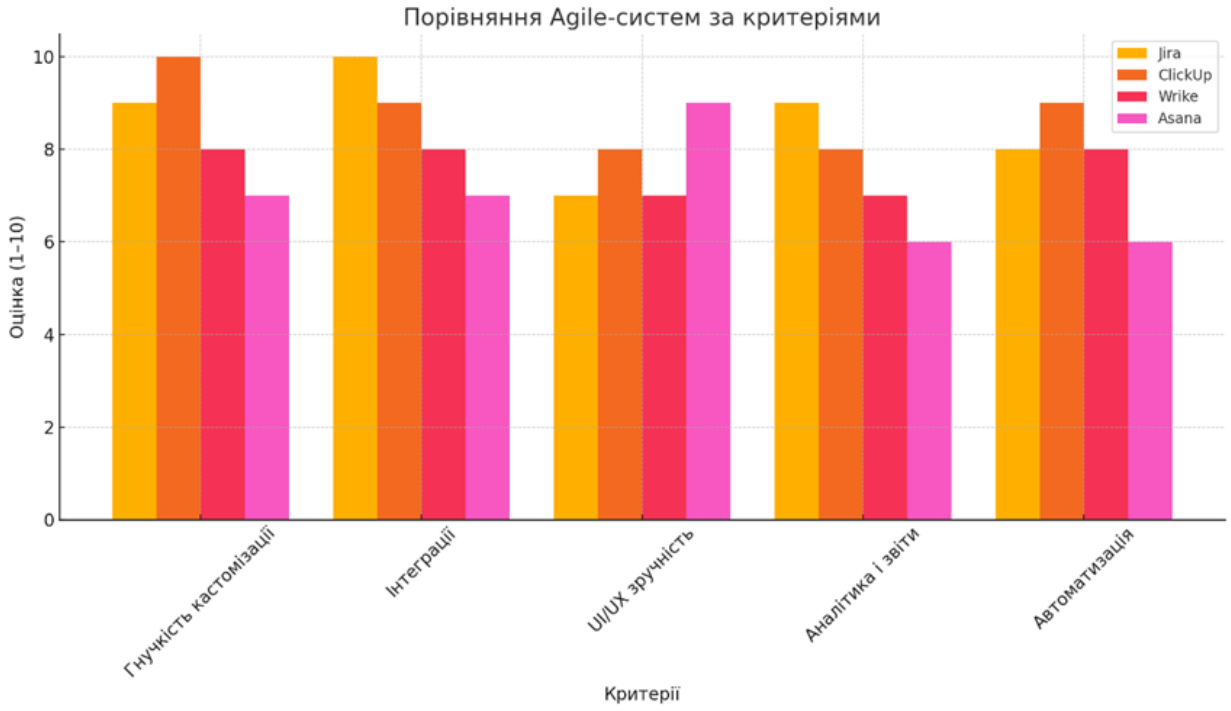


Рис. 2.2. Порівняння Agile-систем за критеріями

Система Jira отримала найвищі оцінки за критеріями інтеграції (9 балів із 10) та аналітики (також 9 балів), що робить її беззаперечним лідером для великих інженерних команд, які потребують глибокої взаємодії з іншими системами — CI/CD, DevOps, Bitbucket, GitHub тощо. Однак інтерфейс Jira лишається складним для нових користувачів, що потребує певного навчання на початковому етапі впровадження.

Інструмент ClickUp позиціонує себе як універсальна платформа, і це підтверджується результатами аналізу. Він лідирує в гнучкості кастомізації (9 балів) та автоматизації (також 9), що дозволяє командам адаптувати структуру завдань, статусів і форм взаємодії під свої конкретні процеси. Проте менш розвинуті функції аналітики (7 балів) можуть бути перешкодою для команд, орієнтованих на глибоку статистичну звітність.

Платформа Wrike показала стабільні оцінки на рівні 7–8 балів майже за всіма критеріями. Це свідчить про її збалансованість, а отже, вона є гарним вибором для організацій, що прагнуть підтримки як Agile-процесів, так і формалізованого менеджменту. Особливою перевагою є її можливості з колаборації у великих командах та побудови багаторівневих робочих структур.

Результати Asana підкреслюють її орієнтацію на зручність — 9 балів за критерієм інтерфейсу, що суттєво спрощує старт користувача. Проте низькі показники автоматизації та аналітики (по 6 балів) обмежують її ефективність у масштабних або технічно складних проєктах. Asana залишається переважно вибором для невеликих креативних команд, стартапів і неформальних робочих груп.

На основі отриманих даних можна зробити висновок, що Jira та ClickUp є найбільш доцільними для використання в IT-проєктах, орієнтованих на складну технічну реалізацію, багаторівневу структуру управління та інтеграцію з іншими середовищами. Wrike варто обирати в умовах потреби в універсальному інструменті з можливістю делегування, контролю ресурсів і бюджетування. Натомість Asana може бути рекомендована командам, де простота і швидкість впровадження є ключовими факторами.

Розглянуті системи не є взаємовиключними — у ряді випадків команди використовують гібридні моделі, наприклад, інтегруючи ClickUp з Jira або Wrike із зовнішніми BI-інструментами. Це дозволяє досягти максимальної відповідності робочим процесам організації без жорсткого обмеження однією платформою.

Таким чином, аналітичний підхід до вибору інструментів управління проєктами з використанням Agile дає змогу не лише обґрунтовано обирати платформу, але й оптимізувати її впровадження відповідно до структури, культури та специфіки команди.

РОЗДІЛ 3
ПРОЄКТУВАННЯ ВЕБ-СИСТЕМИ УПРАВЛІННЯ ПРОЄКТАМИ З ВИКОРИСТАННЯМ МЕТОДОЛОГІЙ ГНУЧКОЇ РОЗРОБКИ

3.1. Проектування архітектури системи та опис її функціональних модулів

Архітектура програмного забезпечення є основою будь-якої переможної системи програмного забезпечення, а плавність є лише одним із її впливів на зручність обслуговування, масштабованість, стабільність і безпеку протягом життєвого циклу системи. Найпершим кроком впровадження нової програмної системи є діаграма архітектури.

Програмне забезпечення матиме наступні модулі на основі базової функціональності:

Клас проекту. Цей модуль є одним із ключових, оскільки дозволяє користувачам запускати нові проекти та налаштовувати їх параметри. При створенні нового проекту користувачеві надається можливість ввести основну інформацію про проект.

Клас користувача. Модуль користувача в системах управління проектами відіграє важливу роль у забезпеченні ідентифікації та контролю доступу користувачів до функцій і ресурсів проекту.

Його основною функцією є реєстрація нових користувачів і автоматизація надання існуючим користувачам відповідних прав доступу.

Епічний клас. Еріс служить великим функціональним блоком або функціональною областю системи та може бути розкладено на менші завдання.

Клас спринт. Спринти створюються після завершення попереднього спринту або на початку проекту. Зазвичай спринт триває 1-4 тижні.

Клас квитка. Заявку можна створити для запланованих завдань, помилок, запитів або будь-якої іншої роботи, яка потребує уваги команди проекту.

Клас коментарів. Коментарі в системах управління проектами відіграють важливу роль у спілкуванні та документуванні робочого процесу команди проекту. Це надає можливість членам команди обмінюватися інформацією, висловлювати думки, запитання щодо роботи, відгуки щодо різних частин проекту.

Ми будемо використовувати діаграму класів UML для опису архітектури програми.

UML — це абревіатура від Unified Modeling Language [11]. Діаграми класів складають основу будь-якого об’єктно-орієнтованого рішення. Ці діаграми представляють класи в системі, атрибути та операції кожного класу, а також зв’язок між кожним класом. Як правило, у більшості інструментів моделювання клас ділиться на три секції. Назва знаходиться вгорі, атрибути посередині, а операції або методи внизу. У великих системах, що містять численні споріднені класи, класи групуються для формування діаграм класів. Різні типи стрілок вказують на різні відносини між класами [11].

Наведена діаграма класів, як на малюнку 3.1, представляє систему для управління проектами та спринтами, а також обробку епіків, проблем, користувачів і коментарів.

Опис класів та їхніх зв’язків наведено в Додатку А.

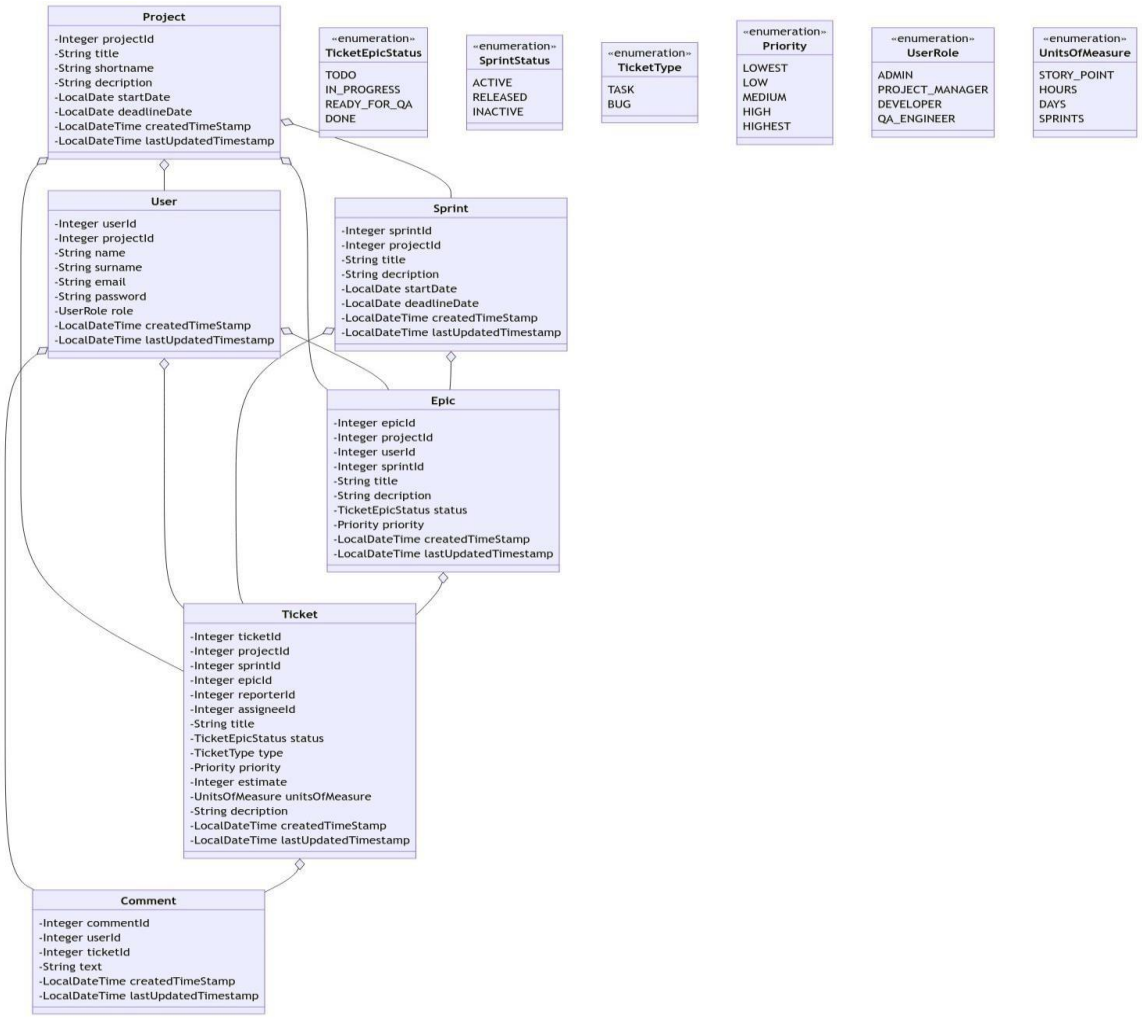


Рис. 3.1. Реалізація UML-діаграми класів.

3.2. Створення початкової структури проекту інформаційної системи управління проектами

Реалізація проекту використовуватиме інструмент автоматизації Maven разом із фреймворком Java. Spring Boot, платформа Java з відкритим вихідним кодом, значно полегшує складності, пов’язані з розгортанням масштабних монолітних веб-додатків або корпоративних додатків Java EE. Цей фреймворк, створений на основі Spring, пропонує ефективний підхід до налаштування та виконання програм.

Maven допомагає розробникам керувати та розуміти проекти, пропонуючи комплексну структуру для життєвого циклу збірки [13]. Щоб полегшити створення проекту Spring Boot, ми використовуємо Spring Initializr.

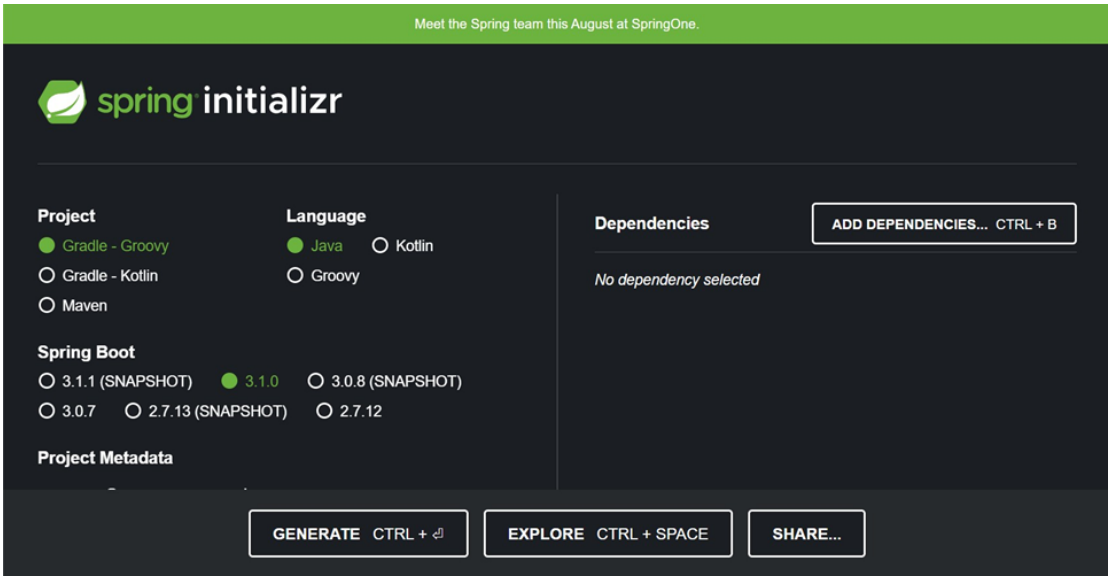


Рис 3.2. Платформу, призначену для створення проекту Spring Boot, можна знайти за адресою https://start.spring.io/.

Файл конфігурації pom.xml описує всі необхідні залежності для ефективної роботи програми. Приклад файлу pom.xml можна знайти в Додатку В. Приклад класу Application для виконання функцій програми, подано нище.

```
@Configuration
@SpringBootApplication
public class PMSoftwareApplication {
public static void main(String[] args) {
SpringApplication.run(PMSoftwareApplication.class, args);
}
}
```

3.3. Розробка контролерів для керування HTTP-запитами

Контролери в інфраструктурі Spring Boot відіграють вирішальну роль, служачи для керування запитами HTTP та надання відповідей, адаптованих до вимог програми. Їх основною функцією є виконання операцій над вхідними HTTP-запитами та подальше повернення відповідних відповідей. Виконуючи роль посередника, контролери сприяють взаємодії між клієнтами, такими як веб-браузери або мобільні програми, і серверною програмою. Нижче наведено

приклад дизайну класу контролера:

```
@RestController
@RequestMapping("/users") public class UserController {
// Методи для обробки HTTP-запитів
}
```

У Spring Boot контролери можуть керувати чотирма основними типами HTTP-запитів: GET, POST, PATCH і DELETE. Нижче наведено опис кожного із цих типів запитів.

1. Отримання даних із сервера здійснюється за допомогою запиту GET. У Spring Boot обробкою запитів GET керує анотація @GetMapping. Зазвичай цей метод надає дані клієнту у форматі JSON або у вигляді HTML-сторінки. Ілюстрація з нашого додатку:

```
@GetMapping("/get/{id}")
public ResponseEntity<UserDTO> getUser(@PathVariable(name =
"id") Integer userId) {
return ResponseEntity.ok(userService.getUser(userId));
}
```

Запит POST служить для створення нових ресурсів на сервері або передачі даних для обробки. У Spring Boot обробка запитів POST полегшена анотацією @PostMapping. Ця анотація дає змогу визначити метод контролера, призначений для керування запитами POST. Як правило, цей метод приймає дані від клієнта через параметри запиту або тіло запиту та виконує необхідні дії, такі як збереження даних у базі даних. Для ілюстрації:

```
@PostMapping("/create")
public ResponseEntity<UserDTO> createUser(@Valid
@RequestBody UserDTO
return ResponseEntity.ok(userService.createUser(user));
}
user) {
```

3. Запит PATCH служить для часткової зміни ресурсу на сервері. Це дає змогу змінювати певні поля чи властивості ресурсу, не впливаючи на інші поля. Щоб керувати запитами PATCH у Spring Boot, необхідно використовувати анотацію @PatchMapping. приклад:

```
@PatchMapping("/update")
public ResponseEntity<UserDTO> updateUser(@RequestBody
UserDTO user){ return
ResponseEntity.ok(userService.updateUser(user));
}
```

4. Запит DELETE призначений для видалення ресурсів із сервера. У Spring Boot для керування запитами DELETE використовується анотація @DeleteMapping. Цей метод приймає параметри, включаючи ідентифікатор ресурсу, призначеного для видалення, і виконує необхідні дії для видалення ресурсу з сервера. Наприклад:

```
@DeleteMapping("/delete")
public void deleteUser(@RequestBody UserDTO user) {
userService.deleteUser(user);
}
```

Контролери функціонують як проміжний рівень, який з'єднує вхідні запити та бізнес-логіку програми. Вони відрізняють логіку, пов'язану з обробкою запитів, від логіки, що стосується бізнес-правил і обробки даних.

Контролери дозволяють установлювати правила перевірки вхідних даних і керувати помилками. Вони полегшують регулювання інформації, отриманої від клієнта, і адресують недійсні або помилкові запити.

У наведеному вище прикладі в методі createUser(@Valid @RequestBody UserDTO user) наявність анотації @Valid означає, що певні поля можуть бути обмежені. Ці обмеження додатково окреслюються анотаціями, визначеними в класі UserDTO.

Шаблон проектування, відомий як DTO (Data Transfer Object), полегшує передачу даних між різними підсистемами програми. Крім того, важливо

включити анотацію @Validated як над класом контролера, так і над класом DTO. Подробиці щодо класів DTO можна знайти в Додатку В, а код для побудови контролерів наведено в Додатку Г.

3.4. Створення та підключення бази даних до інформаційної системи

Під час створення нашої системи управління проектами ми використовували реляційну базу даних MySQL. База даних цього типу організовує дані в окремі таблиці замість того, щоб консолідувати їх в єдине велике сховище. Структура бази даних упорядкована у фізичні файли, оптимізовані для продуктивності. Логічна модель даних, яка включає такі сутності, як таблиці даних, подання, рядки та стовпці, забезпечує універсальне середовище програмування. Користувачі встановлюють правила, які диктують зв’язки між різними полями даних, включаючи зв’язки «один-до-одного», «один-до-багатьох», унікальні, обов’язкові чи необов’язкові, а також «вказівники», що з’єднують різні таблиці [14].

Щоб створити базу даних у MySQL Workbench, необхідно підключитися до сервера бази даних. Після підключення ви перейдете до створення нової бази даних і введете сценарій, призначений для створення таблиць. Перелік цього сценарію можна знайти в Додатку Е.

Взаємодія з об’єктно-орієнтованим програмним забезпеченням разом із реляційними базами даних може виявитися трудомісткою та потребує багато часу. Hibernate служить рішенням Object Relational Mapping (ORM), спеціально розробленим для середовищ Java.

Інструмент ORM полегшує створення, маніпулювання та пошук даних. Ця методологія програмування встановлює відповідність між об’єктом і даними, що зберігаються в базі даних.

Hibernate служить бібліотекою бази даних, яка полегшує відображення класів Java у таблиці бази даних і відповідність типів даних Java типам даних

SQL. Крім того, він пропонує надійні інструменти для запитів і пошуку даних. Основна мета Hibernate — звільнити розробників від більшості рутинних завдань, пов'язаних із зберіганням даних [15].

Щоб встановити зв'язок між нашим проектом Maven і Hibernate with MySQL, необхідно включити залежності hibernate-core і mysql-connector-java у файл конфігурації pom.xml. Ці залежності охоплюють Hibernate ORM, який полегшує об'єктно-реляційне відображення, а також драйвер MySQL, який забезпечує підключення до бази даних MySQL.

Згодом необхідно налаштувати файл application.properties. У цьому файлі проекту Spring Boot необхідно вказати параметри, необхідні для встановлення підключення до бази даних MySQL.

```
      spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
datasource.hostname=localhost
spring.datasource.url=jdbc:mysql://${datasource.hostname}/pmsoftwa
re?serverTi
      mezone=UTC
      spring.datasource.username=root
spring.datasource.password=root server.port=8081
```

У наступних рядках pmsoftware означає назву нашої бази даних, ім'я користувача позначає користувача бази даних, а пароль означає пароль користувача для доступу до бази даних. Згодом необхідно налаштувати анотації Hibernate. Це передбачає створення класів сутностей, також відомих як моделі, які зберігатимуться в базі даних через Hibernate. Щоб встановити зв'язок із таблицями бази даних, важливо включити анотації Hibernate до цих класів, наприклад @Entity, @Table, @Id тощо. Нижче наведено приклад, що ілюструє дизайн класу Entity:

```
        @Entity
        @Table(name = "users") public class UserModel {
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        @Column(nullable = false, name = "user_id") private int
    userId;
        …
        }
```

Класи сутностей, які використовуються в нашому програмному забезпеченні, детально описані в Додатку Д. Щоб полегшити взаємодію з базою даних, ми використовуємо JpaRepository, інтерфейс, який міститься в бібліотеці Spring Data JPA, який спрощує операції з базою даних за допомогою методології об’єктно-реляційного відображення (ORM). Як одна з реалізацій сховища, він пропонує зручний засіб взаємодії з даними, що зберігаються в базі даних, уможливлюючи асоціацію об’єктів Java із записами в таблицях бази даних.

JpaRepository надає набір методів, розроблених для стандартних операцій з базою даних, включаючи збереження, оновлення, видалення та отримання даних. Ця функція дозволяє виконувати фундаментальні операції CRUD (створення, читання, оновлення, видалення) над об’єктами в базі даних без необхідності безпосереднього використання SQL-запитів.

Окрім типових операцій, JpaRepository також здатний виконувати більш складні запити. Нижче наведено ілюстрацію дизайну сховища:

```
      @Repository
      public interface UserRepository extends JpaRepository<User,
Long> {
      // Можна додати власні методи пошуку даних в базі
      }
```

Після успішного встановлення з’єднання з базою даних і супровідними бібліотеками, необхідними для взаємодії, ми переходимо до розробки класу служби, у якому ми викликаємо методи репозиторію для виконання операцій із залученням бази даних. Ілюстрація сервісного дизайну така:

```
       @Service
       public class UserServiceImpl {
       private final UserRepository userRepository;
       public UserServiceImpl(UserRepository userRepository) {
 this.userRepository = userRepository;
       }
       public User getUserById(Integer userId) {
       return userRepository.findById(userId).orElse(null);
       }
       // Можна додати інші методи та бізнес-логіку
       }
```

У цій простій ілюстрації служби сховище використовується для

отримання об’єкта User за допомогою його ідентифікатора. Детальну реалізацію всіх сервісів можна знайти в Додатку Е.

3.5. Функціональні можливості та результати роботи системи управління проектами

Додаток Postman використовується для оцінки програми та представлення результатів, а дані генеруються на сервері.

Щоб ініціювати новий запит, починаємо із вибору потрібного типу HTTP-запиту, який може містити такі параметри, як GET, POST, PUT або DELETE. Після цього введіть URL-адресу нашого API у призначене для цього поле «Введіть URL-адресу запиту» (рис. 3.3).

Рис. 3.3. Приклад HTTP-запиту в Postman.

У випадках, коли запит потребує передачі даних у тілі запиту, така конфігурація можлива. Для цього необхідно вибрати вкладку «Тіло» та вказати формат даних для передачі, наприклад JSON або XML.

Після відправлення запиту Postman надає отриману відповідь. Таким чином ми можемо переглянути статус запиту, а також заголовки, тіло відповіді та додаткові відомості.

Оновлення даних сервера показано на рис. 3.4, де зображено HTTP-запит на часткове оновлення даних щодо об’єкта проекту. Для цього використовується запит типу PATCH, що супроводжується зміною URL-адреси. У тілі запиту поля, призначені для оновлення, змінюються, що призводить до отримання змінених даних проекту.

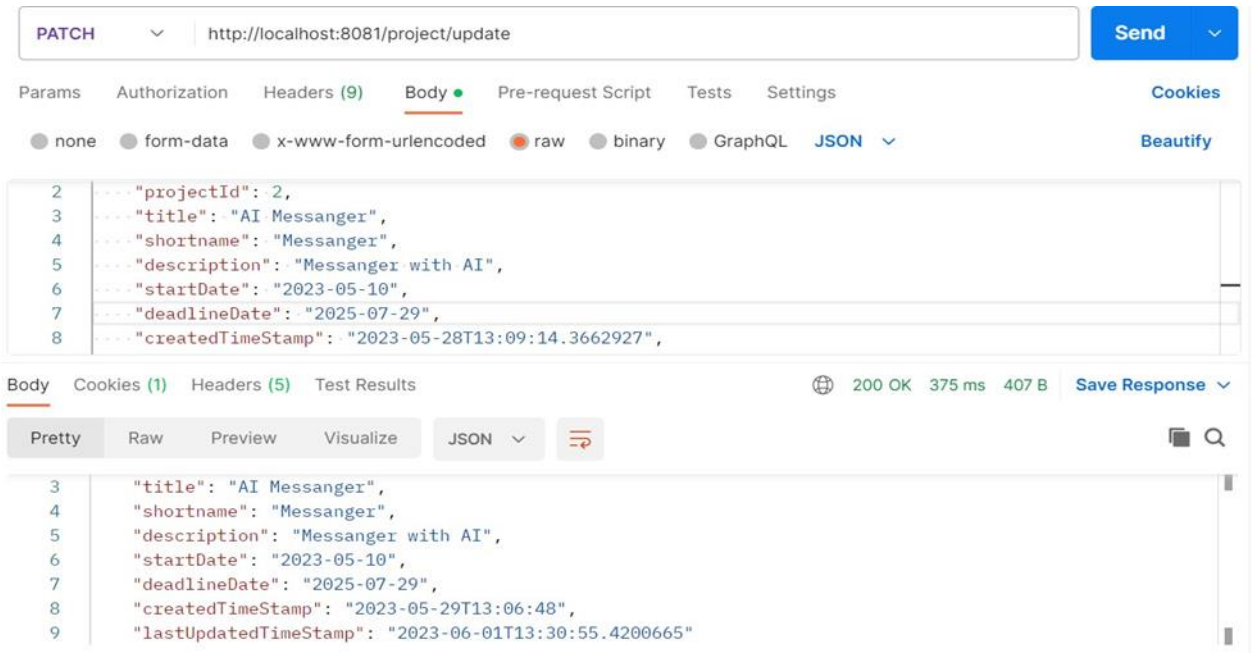


Рис. 3.4. HTTP-запит на оновлення даних проекту.

На рисунках 3.5 і 3.6 зображено дані в таблиці проектів до і після оновлення ресурсу під індексом 2.

project_id	title	shortname	project_description	start_date	deadline_date
1	Project Managment Sofrware	pmsoftware	API PM Software	2023-05-10 00:00:00	2025-07-29 00:00:00
2	messanger	messanger	smth	2023-01-19 00:00:00	2024-09-03 00:00:00
NULL	NULL	NULL	NULL	NULL	NULL

Рис. 3.5. Таблиця проектів до оновлення даних.

project_id	title	shortname	project_description	start_date	deadline_date
1	Project Managment Sofrware	pmsoftware	API PM Software	2023-05-10 00:00:00	2025-07-29 00:00:00
2	AI Messanger	Messanger	Messanger with AI	2023-05-10 00:00:00	2025-07-29 00:00:00
NULL	NULL	NULL	NULL	NULL	NULL

Рис. 3.6. Таблиця проектів після оновлення даних.

Процес видалення даних на сервері проілюструємо на прикладі об’єкта спринту. На малюнку 3.7 новий HTTP-запит зображено поруч зі зміненим типом запиту DELETE. Об’єкт, призначений для видалення, передається в тілі запиту.

Тіло відповіді позбавлене вмісту, а статус запиту вказує на 200 OK. Це означає, що видалення виконано успішно.

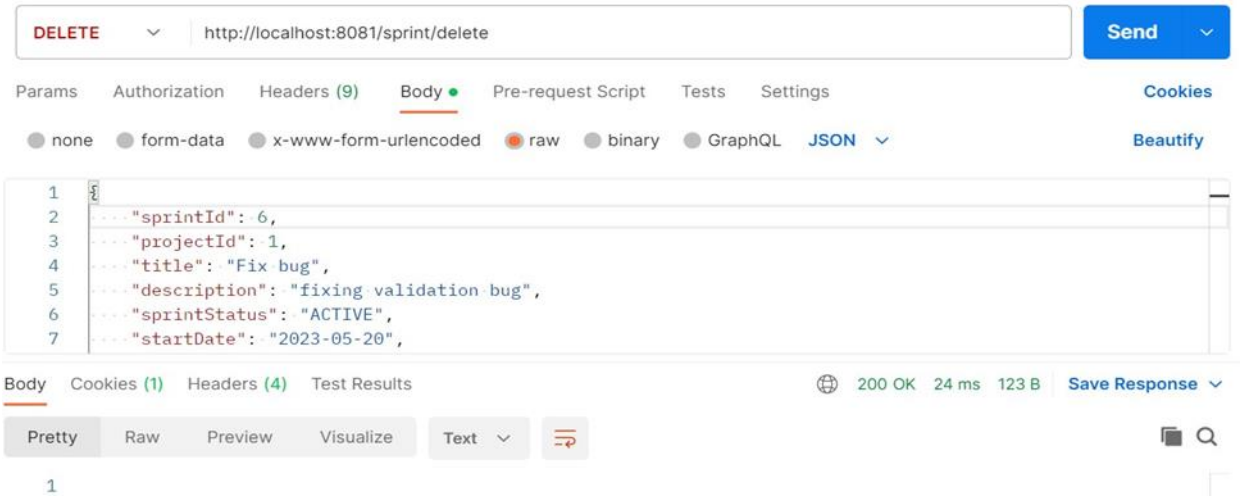


Рис. 3.7. Запит через HTTP на видалення даних спринту..

Вилучення інформації з сервера за допомогою фільтрів.

Рисунок 3.8 ілюструє процес пошуку користувачів за допомогою фільтра. Пошук користувачів ведеться за їхніми ролями. Щоб ініціювати це потрібно змінити тип запиту на GET, ввести нову URL-адресу та вказати потрібну роль у тілі запиту, щоб знайти існуючих користувачів.

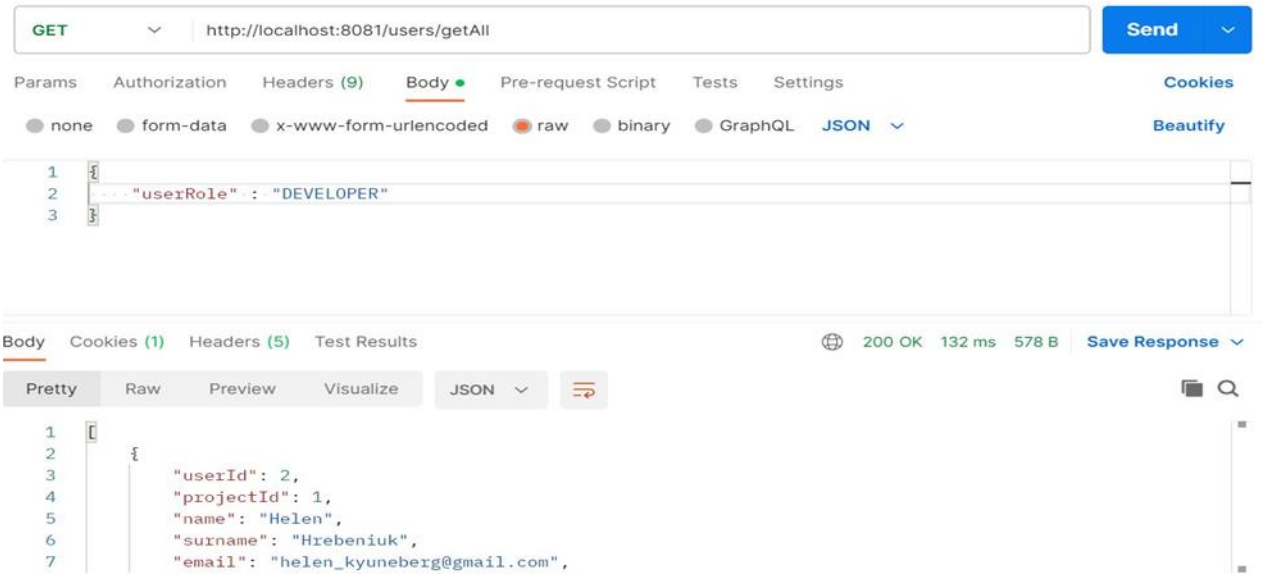


Рис. 3.8. HTTP-запит робиться для отримання даних користувача на основі вказаних фільтрів.

Нижче наведено повний набір результатів запиту. Як видно всі

користувачі маюсь роль “DEVELOPER”:


```
        {
        "userId": 2,
        "projectId": 1, "name": "Taras",
        "surname": "Shevchenko",
        "email": "headoflof@gmail.com", "password": "qwerty",
        "role": "DEVELOPER",
        "createdTimeStamp": "2025-05-28T13:15:54",
  'lastUpdatedTimeStamp": null
        },
        {
        "userId": 3,
        "projectId": 1, "name": "Rostyslav",
        "surname": "Shyan",
        "email": "r shyanan@gmail.com", "password": "Rost123",
        "role": "DEVELOPER",
        "createdTimeStamp": "2025-05-29T22:27:19",
  'lastUpdatedTimeStamp": null
        }
```


На малюнку 3.9 показано запит, який містить ідентифікатор і фільтр. Взявши об’єкт спринту як приклад, ми отримуємо спринти з бази даних за допомогою унікального ідентифікатора проекту «1» (який з’являється в кінці URL-адреси запиту) разом із значенням фільтра, що відповідає статусу спринту.

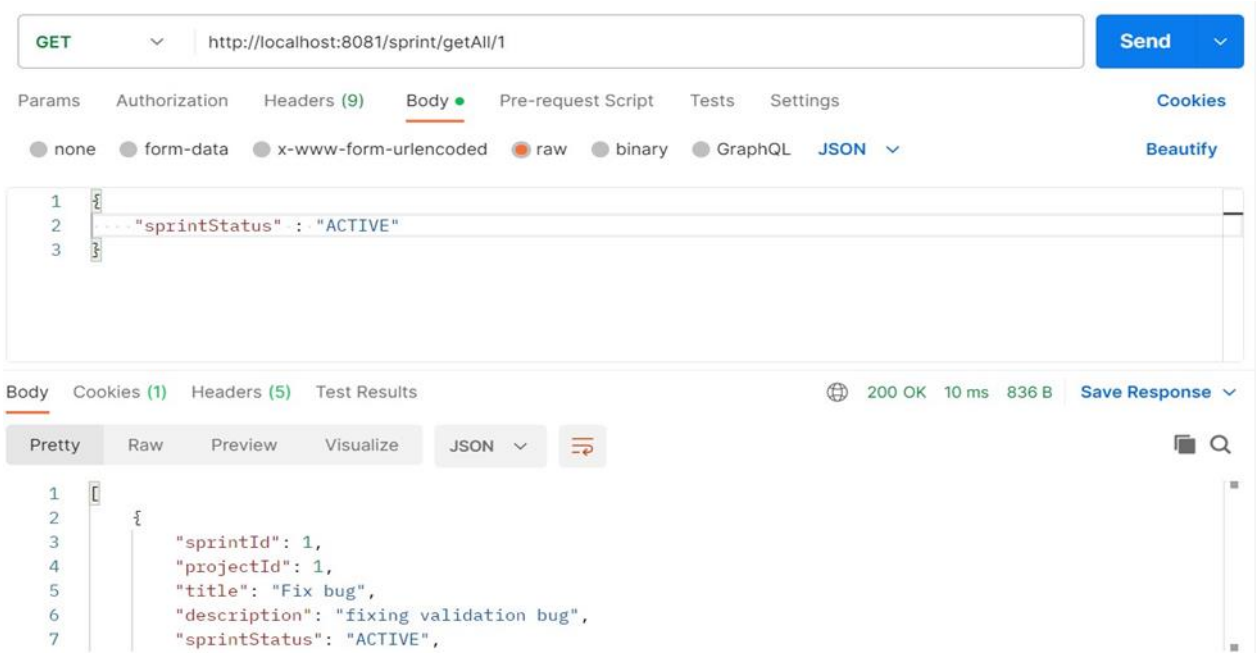


Рис. 3.9. HTTP-запит робиться для отримання даних користувача на основі вказаних фільтрів та ідентифікаційних номерів (ID).

Нижче наведено повний набір результатів запиту. Очевидно, що всі спринти класифікуються зі статусом “ACTIVE” та мають ідентифікатор проекту «1».


```
    {
      "sprintId": 1,
      "proj
ectId": 1,
"title":
"Fix bug",
      "description":
"fixing validation bug",
"sprintStatus":
"ACTIVE", "startDate":
"2025-05-20",
      "deadlineDate": "2025-05-29",
"createdTimeStamp": "2025-05-
28T13:43:57",
"lastUpdatedTimeStamp": null
    },
    {
      "sprintId": 3,
      "projectId": 1,
      "title":
"Handler
creation",
"description":
null,
"sprintStatus":
"ACTIVE",
"startDate":
"2025-05-20",
      "deadlineDate": "2025-06-02",
"createdTimeStamp": "2025-05-
28T13:59:44",
"lastUpdatedTimeStamp": null
    },
    {
      "sprintId": 7,
      "projectId": 1,
      "title":
"Validation
Bug",
"description":
null,
"sprintStatus":
"ACTIVE",
"startDate":
"2025-06-20",
      "deadlineDate": "2025-06-29",
"createdTimeStamp": "2025-06-
01T14:26:20",
"lastUpdatedTimeStamp": null
    }
```

3.6. Обробка помилок під час роботи системи

У Java обробка винятків відноситься до методу адресації та реагування на винятки (помилки), які можуть виникнути під час виконання програми. Ці винятки позначають непередбачені обставини або помилки, які виникають під час роботи програми.

Щоб керувати винятками в програмі Spring Boot, необхідно встановити клас, призначений як глобальний обробник винятків. Цей клас повинен бути анотований @RestControllerAdvice. Крім того, анотації @ResponseStatus і @ExceptionHandler повинні застосовуватися над методами, відповідальними за обробку винятків.

@ResponseStatus – ця анотація слугує для визначення статусу відповіді HTTP, який має бути передано клієнту, коли виникає певна виняткова ситуація. @ExceptionHandler – ця анотація визначає метод, який буде викликано для керування певним типом винятку. Нижче наведено короткий приклад коду, що демонструє використання вищезгаданих анотацій:

```
@RestControllerAdvice
public class GlobalExceptionHandler{
@ExceptionHandler(UserNotFoundException.class)
public ResponseEntity<Object> showUserAbsence() {
ExceptionResponse response = new
ExceptionResponse(); response.setMessage("User is not
found.");
return new ResponseEntity<>(response, HttpStatus.NOT_FOUND);|
}

// Інші обробники виключень
}
```

У цьому випадку, якщо виникне виняток типу UserNotFoundException, буде згенеровано відповідь зі статусом 404 Не знайдено та повідомленням «Користувач не знайдено». Нижче наведено різні варіанти результатів

виконання програми, коли використовуються неправильні дані:

1. На рис. 3.10 представлений варіант HTTP-запиту, у якому дата початку розробки проекту («startDate») показана після дати завершення («deadlineDate»), що не є ані логічним, ані точним.

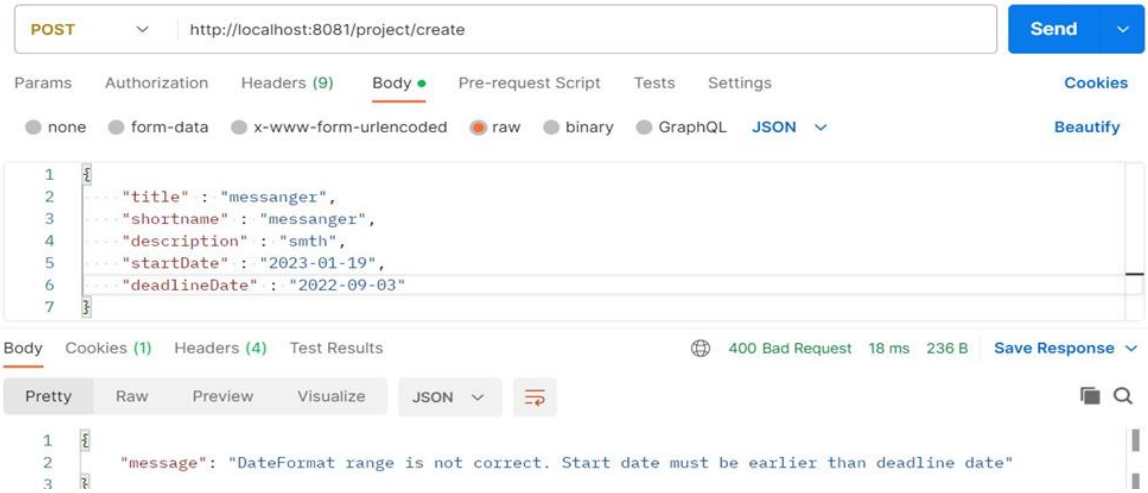


Рис. 3.10. HTTP-запит, який містить помилковий діапазон часу розробки програми.

2. На рисунку 3.11 показаний варіант запиту HTTP, який вказує адресу електронної пошти, яка вже є в базі даних системи. Це означає, що відповідний користувач уже зареєстрований і йому потрібно лише увійти.

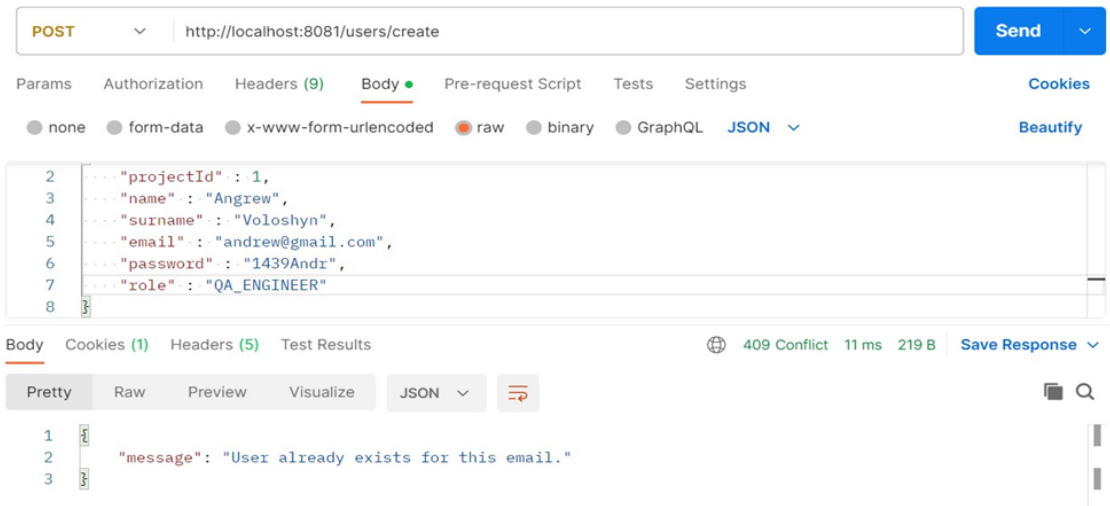


Рис. 3.12. HTTP-запит, який місить уже існуючу на сервері електронну скриньку.

3. На рис. 3.13 зображено варіант HTTP-запиту, у якому спринт

викликається за допомогою унікального ідентифікатора проекту, якого немає на сервері.

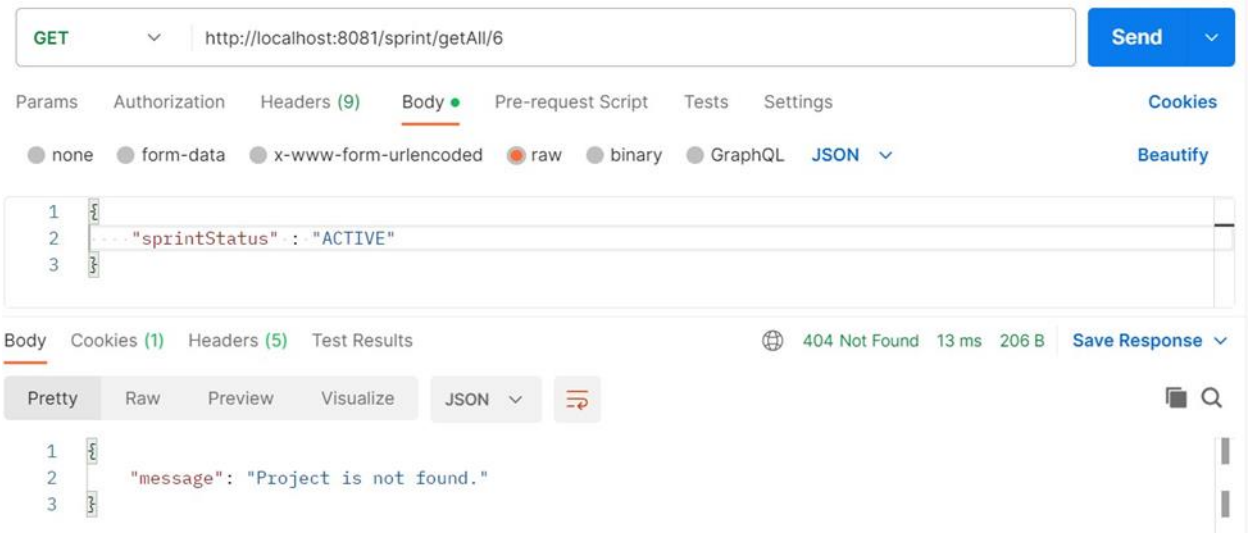


Рис. 3.13. HTTP-запит із неіснуючим проектом.

Основною концепцією та відмінністю створеної нами системи управління проектами є її представлення як незалежного API. Цей підхід пропонує розробникам зручні засоби доступу до повного спектру функціональних можливостей системи без необхідності орієнтуватися в складному інтерфейсі користувача. Таким чином, вони можуть легко та ефективно включати систему управління проектами у свої проекти.

Використовуючи цей API, розробники набувають гнучкості у виборі функцій, які найкраще відповідають їхнім вимогам. Вони мають можливість вибирати лише ті методи API, які відповідають їхнім конкретним потребам, ігноруючи інші. Такий підхід сприяє оптимізації використання ресурсів і сприяє ефективній взаємодії з системою управління проектом.

Створення окремого API для системи управління проектами підвищує ефективність процесу розробки. Використовуючи вже існуючі функції та модулі системи управління проектами, розробники можуть уникнути необхідності перепланування та зосередитися на вирішенні конкретних завдань, пов’язаних із їхніми проектами. Цей підхід не тільки прискорює терміни розробки, але й мінімізує тривалість, необхідну для досягнення бажаних цілей.

Ця система управління проектами пропонує наступні функціональні розширення, які розширять можливості ефективного управління проектами в майбутньому: Spring Security. Ця структура, надана Spring, допомагає в налаштуванні процесів доступу та автентифікації, відіграючи вирішальну роль у захисті програм [16].

Звітність виконує важливу функцію в системах управління проектами, оскільки полегшує моніторинг і оцінку прогресу проекту. Цей процес передбачає збір, аналіз і представлення інформації щодо важливих компонентів проекту, що дозволяє керівникам і зацікавленим сторонам отримати необхідні дані для прийняття обґрунтованих рішень і формулювання планів дій.

Пов’язування квитків стосується практики створення зв’язку між батьківським і допоміжним білетами в системах керування проектами. Батьківський квиток може являти собою основний запит або завдання, яке потребує вирішення, тоді як додатковий квиток позначає додатковий запит або завдання, пов’язане з батьківським білетом, яке потрібно вирішити незалежно.

Коментарі. У майбутньому можна буде додавати коментарі до низки об’єктів у системі керування проектами, включаючи завдання, спринти, проектні документи, зміни, помилки тощо. Ця можливість сприятиме розвитку контекстуальних обговорень і гарантуватиме збереження комунікації щодо конкретних елементів проекту.

Можливість позначати користувачів у коментарях покращить спілкування та взаємодію між учасниками проекту. Це особливо корисно в ситуаціях, коли коментарі можуть містити обширну інформацію або стосуватися конкретних питань, які вимагають уваги призначених користувачів.

Періодична фільтрація. Користувачі матимуть можливість фільтрувати спринти, епіки та квитки за певними часовими рамками, що дозволить їм швидко знаходити та перевіряти проекти, завдання чи інші елементи, які були створені, змінені чи завершені протягом визначеного періоду.

4. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1. Структурно-функціональний аналіз виробничого процесу та розроблення моделі травмонебезпечних ситуацій

У зображеннях процесів формування, виникнення аварій та виробничих травм усі випадкові події, що утворюють конкретну аварійну ситуацію, пов'язані між собою причинно-наслідковими зв'язками.

Метод логічного моделювання потенційних аварій, травм та катастроф відкриває можливість розробити досконалу систему управління ОП виробництва, яка базується на оперативному пошуку виробничих небезпек, їх глибокому аналізі й терміновому прийнятті заходів для усунення потенційних небезпек ще до виникнення травмонебезпечних та катастрофічних ситуацій. Деякі небезпечні ситуації в табл. 4.1.

Працівники, що обслуговують електрообладнання вениляційної системи, зобов'язані знати Правила безпечної експлуатації електроустановок споживачів відповідно до займаної посади або роботи, як вони виконують, і мати відповідну групу з електробезпеки [21,22].

Працівники, що порушили вимоги Правил безпечної експлуатації електроустановок, усуваються від роботи і несуть відповідальність (дисциплінарну, адміністративну, кримінальну) згідно з чинним законодавством. Такі працівники не допускаються до робіт в електроустановках без позачергової перевірки знань вимог правил безпечної експлуатації електроустановок.

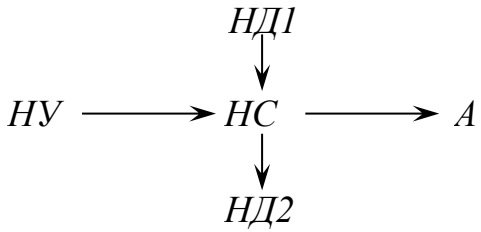
Забороняється допускати до роботи в електроустановках осіб, які не пройшли навчання і перевірку знань Правил безпечної експлуатації електроустановок.

Працівнику, який пройшов перевірку знань Правил безпечної експлуатації електроустановок, видається посвідчення встановленої форми.

Таблиця 4.1. Моделювання процесів формування та виникнення травмонебезпечних і аварійних ситуацій

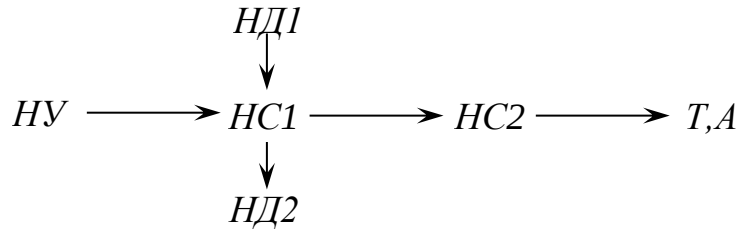
Вид робіт	Виробнича небезпека			Можливі наслідки	Заходи запобігання небезпечним ситуаціям
	Небезпечна умова (НУ)	Небезпечна дія (НД)	Небезпечна ситуація (НС)		
Використання механічної вентиляції	Оператор не перевірив обладнання НУ	Пошкоджений трубопровід мережі НД1 Закупорений трубопровід шланга НД2	Відмова вентиляційної системи (двигуна) НС	Аварія	Розвісити плакати, провести інструктажі із експлуатації обладнання системи

Модель процесу:



Використання електронних пристроїв регулювання	Пошкоджена ізоляція провідників з’єднання НУ	Пробій на корпус НД1 Коротке замикання НД2	Ураження людини електричним струмом НС1 Виведення обладнання із ладу НС2	Травма Аварія	Заміна провідників, установлення захисного обладнання (запобіжників, захист від ураження людини струмом) тощо

Модель процесу:



Посвідчення про перевірку знань працівника є документом, який засвідчує право на самостійну роботу в електроустановках на зазначеній посаді за фахом.

4.2. Вимоги техніки безпеки під час роботи обладнання та протипожежні заходи

Вимоги правил техніки безпеки перед початком роботи. Для початку роботи пов’язаної з вентиляцією вимикають рубильники або автоматичні вимикачі щита низької напруги, запирають шафу і вивішують попереджувальні плакати. Також повинні бути основні захисні засоби до яких належать такі, ізоляція яких надійно захищає від робочої напруги мережі і за допомогою яких можна дотикатися до струмопровідних частин, що перебувають під напругою, без небезпеки ураження електричним струмом (інструмент з ізольованими ручками, ізолюючі струмовимірювальні кліщі, діелектричні рукавиці).

Вимоги правил техніки безпеки під час роботи. Виконавши ці операції, надівають діелектричні рукавиці і за допомогою покажчика напруги перевіряють відсутність напруги на всіх фазах. Потім, приєднавши один кінець переносного заземлення до заземлюючого пристрою, накладають його на струмоведучі частини. Після цього остаточно приступають до роботи.

Вимоги правил техніки після закінчення роботи. Після закінчення роботи системи перед її вимиканням необхідно виконати такі технічні операції: перевірити надійність кріплення, зняти переносні тимчасові заземлення, відімкнути щит низької напруги і зняти плакати з техніки безпеки; якщо тимчасове переносне заземлення встановлене на лінії, його також треба зняти тощо [21].

Протипожежні заходи на об’єкті. Для запобігання пожеж на об’єкті розроблено організаційні, експлуатаційні, технічні режимного характеру, пожежно-евакуаційні, профілактичні заходи. До організаційних заходів

відносяться правила розміщення машин, що обслуговують приміщення, обладнання, матеріалів з дотримання певних проходів, не допускається захаращення приміщень, проходів і т.д.

4.3. Розрахунок штучного заземлення

Вибір штучного заземлення проводиться в залежності від характеру ґрунту і способу забивання стержнів [21]. Розраховуємо заземлюючий контур підстанції напругою 10/0,4 кВ з глухозаземленою нейтраллю. Характер ґрунту – чорнозем з $\rho=2\cdot10^4$ Ом·см. Кліматична зона – IV (K_c – 1,2, K_n – 1,5). Струм замикання на землю в мережі становить 50 А.

В відповідності з діючими правилами, опір заземлюючого пристрою повинен становити

$$R = \frac{125}{I_з} = \frac{125}{50} = 2,5 Ом , \qquad (4.1)$$

де $I_з$ – струм замикання на землю, А.

Приймаємо 3 Ом. Контур заземлення розміщуємо в ряд з а = 5 м, l = 2,5 м. В якості стержневого заземлювача приймаємо кутникові сталь 50х50х5 мм, а протяжного – пластинчасту сталь 40х4 мм.

Опір одиночного стержня становить:

$$R_o = 0.00318\rho \cdot K_c, Ом \qquad (4.2)$$

де K_c – коефіцієнт сезонності для стержневого заземлювача (K_c – 1,2).

$$R_o = 0.00318 \cdot 2 \cdot 10^4 \cdot 1.2 = 76.32 Ом .$$

Число стержнів приймаємо 15. При цьому коефіцієнт використання стержневих заземлювачів становить η_c = 0,7. Опір всіх стержнів розтікання струму становить:

$$R_c = \frac{R_o}{n \cdot \eta_c}, Ом , \qquad (4.3)$$

де *n* – число стержнів, шт.

$$Rc = \frac{76.32}{15 \cdot 0.7} = 7.3Ом .$$

Довжина протяжного заземлювача становить *l* = 35 м (3500 см); приймаємо *t* = 50 см, *b* = 0,4 см. Опір протяжного заземлювача становить:

$$R_{пр} = \frac{0,366}{l} \cdot \rho \cdot 2 \cdot \lg \frac{2 \cdot l^2}{t \cdot b}, Ом \qquad (4.4)$$

$$R_{пр} = \frac{0,366}{3500} \cdot 1,2 \cdot 10^4 \cdot 2 \cdot \lg \frac{2 \cdot 3500^2}{0,4 \cdot 50} = 3,2Ом$$

Коефіцієнт використання протяжного заземлювача η_n = 0,71. Дійсний опір протяжного заземлення становить:

$$R_n = \frac{R_{пр}}{\eta_n} = \frac{3,2}{0,71} = 4,5Ом \qquad (4.5)$$

Опір всього заземлюю чого пристрою становить:

$$R_u = \frac{R_c \cdot R_n}{R_c + R_n} = \frac{4.5 \cdot 7.3}{4.5 + 7.3} 2.78 < 3Ом \qquad (4.6)$$

Отже, число стержнів вибрано вірно.

4.4. Захист цивільного населення

Забезпечення захисту населення і території у разі загрози та виникнення надзвичайних ситуацій є одним з найважливіших завдань не лише підприємства, але й цілої держави. Актуальність проблеми забезпечення природо-техногенної безпеки населення і території зумовлена тенденціями зростання втрат людей і шкоди територіям, що спричиняються небезпечними природними явищами, промисловими аваріями і катастрофами.

Інженерний захист проводиться з метою виконання вимог ІТЗ із питань забудови міст, розміщення ПНО, будівлі будинків, інженерних споруд та інше.

Медичний захист проводиться для зменшення ступеня ураження людей, своєчасного надання допомоги постраждалим та їх лікування, забезпечення епідеміологічного благополуччя в районах надзвичайних ситуацій.

Біологічний захист включає своєчасне виявлення чинників біологічного зараження, їх характеру і масштабів, проведення комплексу адміністративно-господарських, режимно-обмежувальних і спеціальних протиепідемічних та медичних заходів.

Радіаційний і хімічний захист включає заходи щодо виявлення і оцінки радіаційної та хімічної обстановки, організацію і здійснення дозиметричного та хімічного контролю, розроблення типових режимів радіаційного захисту, забезпечення засобами індивідуального захисту, організацію і проведення спеціальної обробки.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

Ефективне управління проектами має вирішальне значення для успішної реалізації в різних сферах, таких як IT, промисловість, освіта та бізнес, особливо з огляду на швидкий розвиток цифрових технологій. Гнучкі методології, які підкреслюють гнучкість, адаптивність і безперервну взаємодію, стають все більш актуальними, дозволяючи командам оперативно реагувати на зміни умов і підвищувати рівень задоволеності клієнтів.

Сучасні програмні рішення спрямовані на автоматизацію рутинних процесів, одночасно підтримуючи керування завданнями, ролями, дедлайнами та командною взаємодією через веб-платформи, які включають гнучкі практики, такі як Scrum, Kanban і Lean. Теоретичні основи Agile висвітлюють його походження на початку 2000-х років, наголошуючи на таких цінностях, як люди над процесами та реагування на зміни за планом, а методики, такі як Scrum і Kanban, демонструють практичні переваги. Застосування Agile виходить за межі IT в освіту, охорону здоров’я та державне управління, де воно сприяє динамічному середовищу, швидкому реагуванню та підвищеній прозорості, незважаючи на такі виклики, як опір організації та потреба в культурних змінах.

Практичні впровадження в Україні, наприклад, показують, що Agile може підвищити продуктивність, скоротити час виходу на ринок і підвищити задоволеність клієнтів за умови підтримки керівництва, належного навчання та зміни організаційного мислення. Загалом, інтеграція Agile із сучасними інструментами управління, штучним інтелектом, DevOps і автоматизацією продовжує підвищувати свою ефективність, що робить його життєво важливим підходом для організацій, які прагнуть процвітати в цифровому світі, що постійно розвивається.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Anderson, D. J. (2010). Kanban: Successful evolutionary change for your technology business. Blue Hole Press.

2. Beck, K. (1999). Extreme Programming Explained: Embrace Change. Addison-Wesley.

3. Denning, S. (2018). The Age of Agile: How Smart Companies Are Transforming the Way Work Gets Done. AMACOM.

4. Kniberg, H., & Ivarsson, A. (2012). Scaling Agile @ Spotify with Tribes, Squads, Chapters & Guilds. Crisp.

5. Leffingwell, D. (2018). SAFe® 4.5 Reference Guide: Scaled Agile Framework® for Lean Enterprises. Addison-Wesley.

6. Rigby, D. K., Sutherland, J., & Takeuchi, H. (2016). Embracing Agile. Harvard Business Review, 94(5), 40-50.

7. Schwaber, K., & Sutherland, J. (2020). The Scrum Guide. Scrum.org. https://scrumguides.org

8. Krayovskyy, R. Make Big Data Work with Machine Learning https://www.softserveinc.com/files/whitepaper/agile-sdlc.pdf

9. Стаття «The Agile Coach. Atlassian's no-nonsense guide to agile development» [Electronic resource]. Режим доступу: https://www.atlassian.com/agile (дата звернення 15.05.2025).

10. Стаття «Top 5 main Agile methodologies: advantages and disadvantages» [Electronic resource]. Режим доступу: https://www.xpand-it.com/blog/top-5-agile- methodologies/ (дата звернення 15.05.2025).

11. Стаття «What Is a Scrum Sprint Cycle?» [Electronic resource]. Режим доступу: https://www.wrike.com/scrum-guide/faq/what-is-a-scrum-sprint-cycle (дата звернення 16.05.2025). – Назва з екрана.

12. Стаття «Agile Team Roles: What They Are and What They Do» [Electronic resource]. Режим доступу: https://projectmanagementacademy.net/resources/blog/agile- team-roles what-they-are-and-what-they-do/ (дата звернення 16.05.2025).

13. Стаття «Agile Project Management - What is it and how to get started? How agile methodologies can work for your software team» [Electronic resource]. Режим доступу: https://www.atlassian.com/agile/project-management#:~:text=Agile%20project%20management%20is%20an,customer%20feedba ck%20with%20every%20iteration (дата звернення 20.05.2025).

14. Стаття «What is JIRA? Overview and Full Guide» [Electronic resource]. Режим доступу до ресурсу: https://appmaster.io/blog/what-is-jira (дата звернення 20.05.2025).

15. Стаття «What is Wrike? Overview of Wrike benefits» [Electronic resource]. Режим доступу: https://elearningindustry.com/directory/elearning-software/wrike (дата звернення 20.05.2025).

16. Стаття «About ClickUp» [Electronic resource]. Режим доступу: https://www.softwareadvice.com/project-management/clickup-profile/

17. Стаття «ClickUp Overview: How It Works & Why Use It for Project Management» [Electronic resource]. Режим доступу: https://friday.app/p/what-is-clickup (дата звернення 22.05.2025).

18. Стаття «Software architecture diagramming and patterns» [Electronic resource]. Режим доступу: https://www.educative.io/blog/software-architecture- diagramming-and-patterns (дата звернення 23.05.2025).

19. Стаття «UML Diagram Types Guide: Learn About All Types of UML Diagrams with Examples» [Electronic resource]. Режим доступу: https://creately.com/blog/diagrams/uml-diagram-types-examples/#ClassDiagram

20. Стаття «Spring Boot» [Electronic resource]. Режим доступу: https://spring.io/projects/spring-boot (дата звернення 28.05.2025).

21. Пістун І. П., Тимочко В.О., Городецький І. М.. Березовецький А. П. Охорона праці (гігієна праці та виробнича санітарія): навч. посібн. / за ред. І.П. Пістуна. Ч. І. Львів: Тріада плюс, 2011. 224 с.

22 Пістун І. П., Тимочко В.О., Городецький І. М.. Березовецький А. П. Охорона праці (гігієна праці та виробнича санітарія): навч. посібн. / за ред. І.П. Пістуна. Ч. ІІ. Львів: Тріада плюс, 2011. 274 с.

ДОДАТКИ

ДОДАТОК А

Розшифровка UML-діаграми класів

- *Project (Проект):* Представляє проект. Має зв'язок один-до-багатьох з класами User (Користувач), Sprint (Спрінт), Epic (Епік) та Ticket (Завдання). Містить різні атрибути, такі як projectId (ідентифікатор проекту), title (назва), shortname (скорочена назва), description (опис), startDate (дата початку), deadlineDate (дата завершення), createdTimeStamp (час створення) та lastUpdatedTimestamp (останній час оновлення).

- *User (Користувач):* Представляє користувача системи. Має зв'язок один-до- багатьох з класами Epic (Епік), Ticket (Завдання) та Comment (Коментар). Містить атрибути, такі як userId (ідентифікатор користувача), projectId (ідентифікатор проекту), name (ім'я), surname (прізвище), email (електронна пошта), password (пароль), role (роль), createdTimeStamp (час створення) та lastUpdatedTimestamp (останній час оновлення). Перерахування UserRole (Роль користувача) представляє різні ролі, які користувач може мати.

- *Sprint (Спрінт):* Представляє спрінт в проекті. Має зв'язок один-до-багатьох з класами Epic (Епік) та Ticket (Завдання). Містить атрибути, такі як sprintId (ідентифікатор спрінту), projectId (ідентифікатор проекту), title (назва), description (опис), startDate (дата початку), deadlineDate (дата завершення), createdTimeStamp (час створення) та lastUpdatedTimestamp (останній час оновлення). Перерахування
`SprintStatus` (Статус спрінту) представляє різні статуси спрінту.

- *Epic (Епік):* Представляє епік в проекті. Має зв'язок багато-до-одного з класами Project (Проект), User (Користувач) та Sprint (Спрінт). Містить атрибути, такі як epicId (ідентифікатор епіка), projectId (ідентифікатор проекту), userId (ідентифікатор користувача), sprintId (ідентифікатор спрінту), title (назва), description (опис), status (статус), priority (пріоритет), createdTimeStamp (час створення) та lastUpdatedTimestamp (останній час оновлення). Перерахування TicketEpicStatus (Статус епіка та завдання) представляє різні статуси епіка, а перерахування Priority (Пріоритет) представляє різні рівні пріоритету.

- *Ticket (Завдання):* Представляє заявку в проекті. Має зв'язок багато-до-одного з класами Project (Проект), Sprint (Спрінт), Epic (Епік), User (Користувач). Містить атрибути, такі як ticketId (ідентифікатор завдання), projectId (ідентифікатор проекту), sprintId (ідентифікатор спрінту), epicId (ідентифікатор епіка), reporterId (ідентифікатор користувача, що створює завдання), assigneeId (ідентифікатор користувача, що виконує завдання), title (назва), status (статус), type (тип), priority (пріоритет), estimate (оцінка), unitsOfMeasure (одиниці виміру оцінки), description (опис), createdTimeStamp (час створення) та lastUpdatedTimestamp (останній час оновлення). Перерахування TicketType (Тип заявки) представляє різні типи заявок, а перерахування UnitsOfMeasure (Одиниці виміру) представляє різні одиниці виміру для оцінки заявок.

- *Comment (Коментар):* Представляє коментар, зроблений користувачем до заявки. Має зв'язок багато-до-одного з класами User (Користувач) та Ticket (Завдання). Містить атрибути, такі як commentId (ідентифікатор коментаря), userId (ідентифікатор користувача), ticketId (ідентифікатор завдання), text (текст коментаря), createdTimeStamp (час створення) та lastUpdatedTimestamp (останній час оновлення).

Діаграма класів містить кілька перерахувань, які визначають конкретні набори значень для певних атрибутів. Ось розшифровка кожного перерахування.

- *TicketEpicStatus (Статус епіка та завдання):* Представляє статус епіка і завдання. Має чотири можливі значення: TODO (В роботі), IN_PROGRESS (В процесі), READY_FOR_QA (Готово для тестування) та DONE (Виконано).

- *SprintStatus (Статус спрінту):* Представляє статус спрінту. Має три можливі значення: ACTIVE (Активний), RELEASED (Випущений) та INACTIVE (Неактивний).

- *TicketType (Тип заявки):* Представляє тип заявки. Має два можливі значення: TASK (Завдання) та BUG (Помилка/проблема).

- *Priority (Пріоритет):* Представляє рівень пріоритету завдання або епіка. Має п'ять можливих значень: LOWEST (Найнижчий), LOW (Низький), MEDIUM (Середній), HIGH (Високий) та HIGHEST (Найвищий). Ці значення вказують на відносну важливість або терміновість заявки або епіка.

- *UserRole (Роль користувача):* Представляє роль користувача в системі. Має чотири можливі значення: ADMIN (Адміністратор), PROJECT_MANAGER (Менеджер проекту), DEVELOPER (Розробник) та QA_ENGINEER (Тестувальник). Ці значення вказують на різні ролі, які користувач може мати в системі управління проектами.

- *UnitsOfMeasure (Одиниці виміру):* Представляє одиниці виміру для оцінки завдань. Має чотири можливі значення: HOURS (Години), DAYS (Дні) та SPRINTS (Спрінти), STORY_POINT. Оцінка за допомогою story point базується на загальному розумінні складності завдання, ураховуючи фактори, такі як технічні вимоги, ризики, необхідні ресурси та знання, що потрібні для його виконання.

ДОДАТОК Б
Фрагмент коду з файлу файлу pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <parent>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-parent</artifactId>
                <version>2.7.8</version>
                <relativePath/>
        </parent>
        <groupId>com.diploma</groupId>
        <artifactId>pmsoftware</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <name>pmsoftware</name>
        <properties>
                <java.version>11</java.version>
        </properties>
        <dependencies>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-web</artifactId>
                          </dependency>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-data-jpa</artifactId>

        </dependency>
        <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-test</artifactId>
                <scope>test</scope>
                <exclusions>
                        <exclusion>
                                <groupId>org.junit.vintage</groupId>
                                <artifactId>junit-vintage-engine</artifactId>
                        </exclusion>
                </exclusions>
        </dependency>
```

Додаток В

Фрагмент коду з файлу ProjectModel.java

```
@RequiredArgsConstructor
@Getter
@Setter
@Entity
@Table(name =
"project") public class
ProjectModel {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(nullable = false, name =
    "project_id") private int projectId;
    @Column(nullable = false, name =
    "title") private String title;
    @Column(name =
    "shortname") private String
    shortname;
    @Column(name =
    "project_description") private String
    description;
    @Column(name =
    "start_date") private LocalDate
    startDate;
    @Column(name =
    "deadline_date") private LocalDate
   deadlineDate;
    @Column(nullable = false, name =
   "created_time_stamp") private LocalDateTime
```

Додаток В
Фрагмент коду з файлу SprintModel.java

```
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(nullable = false, name =
    "sprint_id") private int sprintId;
    @Column(nullable = false, name =
    "project_id") private int projectId;
    @Column(nullable = false, name =
    "title") private String title;
    @Column(name =
    "sprint_description") private String
    description;
    @Column(nullable = false, name =
    "sprint_status") private SprintStatus
    sprintStatus;
    @Column(name =
    "start_date") private LocalDate
    startDate;
    @Column(name =
    "deadline_date") private LocalDate
    deadlineDate;
    @Column(nullable = false, name =
    "created_time_stamp") private LocalDateTime
    createdTimeStamp;
    @Column(name =
    "last_updated_time_stamp") private
    LocalDateTime lastUpdatedTimeStamp;
}

 @RestController
```

Додаток Г
Фрагмент коду з EpicComtroller.java

```
@RequestMapping("/epic
") public class
EpicController {
   private final EpicServiceImpl epicService;
   public EpicController(EpicServiceImpl
      epicService) { this.epicService = epicService;
   }
   @PostMapping("/create")
   public ResponseEntity<EpicDTO> createEpic(@Valid @RequestBody EpicDTO
   epic)
      return ResponseEntity.ok(epicService.create(epic));
   }
   @GetMapping("/getByFilter")
   public ResponseEntity<List<EpicDTO>> getByFilter(@RequestBody Filter
      filter) { return ResponseEntity.ok(epicService.getAllByFilter(filter));
   }
   @GetMapping("/getAllByProject/{project-id}")
   public ResponseEntity<List<EpicDTO>>
getAllByProjectId(@PathVariable(name = "project-id") Integer projectId,
@RequestBody Filter filter) {
      return ResponseEntity.ok(epicService.getAllByProject(projectId, filter));
   }
   @GetMapping("/getAllBySprint/{sprint-id}")
   public ResponseEntity<List<EpicDTO>>
getAllBySprintId(@PathVariable(name = "sprint-id") Integer sprintId,
@RequestBody Filter filter) {
      return ResponseEntity.ok(epicService.getAllBySprint(sprintId, filter));
   }
   @GetMapping("/get/{epic-id}")
     public ResponseEntity<EpicDTO> getEpic(@PathVariable(name = "epic-
id") Integer epicId) {
      return ResponseEntity.ok(epicService.getEpic(epicId));
   }
   @DeleteMapping("/delete")
   public void deleteEpic(@RequestBody EpicDTO epic) {
      epicService.delete(epic);
   @PatchMapping("/update")
   public ResponseEntity<EpicDTO> updateEpic(@RequestBody EpicDTO
   }
```

Додаток Ґ
Фрагмент коду з SQL запиту на створення таблиць в базі даних

```
CREATE TABLE IF NOT EXISTS project(
project_id int NOT NULL auto_increment PRIMARY KEY, title
varchar(255) NOT NULL,
shortname varchar(255),
project_description varchar(255),
start_date datetime, deadline_date
datetime,
created_time_stamp datetime NOT NULL,
last_updated_time_stamp datetime
);
CREATE TABLE IF NOT EXISTS users(
user_id int NOT NULL auto_increment PRIMARY KEY,
project_id int,
user_name varchar(255) NOT NULL, surname
varchar(255) NOT NULL, email varchar(255)
NOT NULL,
user_password varchar(255) NOT NULL, user_role
varchar(255), created_time_stamp datetime NOT
NULL, last_updated_time_stamp datetime,
FOREIGN KEY (project_id) REFERENCES project(project_id)
);
CREATE TABLE IF NOT EXISTS sprint(
sprint_id int NOT NULL auto_increment PRIMARY KEY,
project_id int NOT NULL,
title varchar(255) NOT NULL,
sprint_description varchar(400),
sprint_status varchar(255) NOT NULL,
start_date datetime,
deadline_date datetime,
created_time_stamp datetime NOT
NULL, last_updated_time_stamp
datetime,
FOREIGN KEY (project_id) REFERENCES project(project_id)
);
CREATE TABLE IF NOT EXISTS epic(
epic_id int NOT NULL auto_increment PRIMARY KEY,
project_id int NOT NULL,
user_id int NOT NULL,
sprint_id int NOT NULL,
title varchar(255) NOT NULL,
```

Додаток Д
Фрагмент коду з файлу ProjectModel.java

```
@RequiredArgsConstructor
@Getter
@Setter
@Entity
@Table(name =
"project") public class
ProjectModel {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(nullable = false, name =
    "project_id") private int projectId;
    @Column(nullable = false, name =
    "title") private String title;
    @Column(name =
    "shortname") private String
    shortname;
    @Column(name =
    "project_description") private String
    description;
    @Column(name =
    "start_date") private LocalDate
    startDate;
    @Column(name =
    "deadline_date") private LocalDate
    deadlineDate;
    @Column(nullable = false, name =
    "created_time_stamp") private LocalDateTime
    createdTimeStamp;
    @Column(name =
    "last_updated_time_stamp") private
    LocalDateTime lastUpdatedTimeStamp;
}
```

Додаток Е

Фрагмент коду з файлу TicketService.java

```
public interface TicketService {
   TicketDTO create(TicketDTO
   ticket);
   List<TicketDTO> getAllByProject(Integer projectId, Filter filter);
   List<TicketDTO> getAllBySprint(Integer sprintId, Filter filter);
   List<TicketDTO> getAllByEpic(Integer epicId, Filter filter);
   List<TicketDTO> getAllByReporter(Integer reporterId, Filter filter);
   List<TicketDTO> getAllByAssignee(Integer assigneeId, Filter filter);
   TicketDTO getTicket(Integer id);
   void deleteTicket(TicketDTO ticket);
   TicketDTO updateTicket(TicketDTO
   ticket);
   }
```

Додаток Є

Фрагмент коду з файлу GlobalExceptionHandler.java

```
@RestControllerAdvice
public class GlobalExceptionHandler {
   @ResponseStatus(HttpStatus.BAD_REQUEST)
   @ExceptionHandler(MethodArgumentNotValidException.class) public
   ResponseEntity<Object>
handleValidatorException(MethodArgumentNotValidException exception) {
      Map<String, String> errors = new HashMap<>();
      exception.getBindingResult().getAllErrors().forEach((error -> {
         String fieldName = ((FieldError) error).getField(); String
         errorMessage = error.getDefaultMessage();
         errors.put(fieldName, errorMessage);
      }));
      return new ResponseEntity<>(errors, HttpStatus.BAD_REQUEST);
   @ExceptionHandler(UserAlreadyExistsException.clas
   s) public ResponseEntity<Object>
   showUserExistence() {
      ExceptionResponse response = new ExceptionResponse();
      response.setMessage("User already exists for this email.");
      return new ResponseEntity<>(response, HttpStatus.CONFLICT);
   }
   @ExceptionHandler(UserNotFoundException.cl
   ass) public ResponseEntity<Object>
   showUserAbsence() {
      ExceptionResponse response = new ExceptionResponse();
      response.setMessage("User is not found.");
```