МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ПРИРОДОКОРИСТУВАННЯ

ФАКУЛЬТЕТ МЕХАНІКИ, ЕНЕРГЕТИКИ
ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

другого (магістерського) рівня вищої освіти

на тему: **“ Імплементація Agile-технологій в командній координації розробки ІТ-проектів ”**

Виконав: ст. гр. Іт-61_____

Спеціальності 126 – «Інформаційні системи та технології»_____
(шифр і назва)

_____Тучапський Денис Юрійович_____
(прізвище та ініціали)

Керівник: __к.т.н., доц. Луб П.М._____
(прізвище та ініціали)

Рецензент: _____
(прізвище та ініціали)

ДУБЛЯНИ-2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ПРИРОДОКОРИСТУВАННЯ

ФАКУЛЬТЕТ МЕХАНІКИ, ЕНЕРГЕТИКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

другий (магістерський) рівень вищої освіти
126 – «Інформаційні системи та технології»

"ЗАТВЕРДЖУЮ"
Завідувач кафедри_____
д.т.н., проф. А.М. Тригуба
"____"_____2024 р.

ЗАВДАННЯ

на кваліфікаційну роботу студенту

Тучапському Денису Юрійовичу

1. Тема роботи: «Імплементація Agile-технологій в командній координації розробки ІТ-проектів»

Керівник роботи Луб Павло Миронович, к.т.н., доцент
Затверджені наказом по університету 12 вересня 2024 року № 616/к-с.

2. Строк подання студентом роботи 06.12.2024 р.

3. Початкові дані до роботи: 1. Науково-технічна і довідкова література. 2. Традиційні та гнучкі методології планування та координації розробки ПЗ. 3. Функціональні кейси Java. 4. Методологія обміну даними на основі реляційної MySQL. 5. Функціонал реалізований Postman.

4. Зміст розрахунково-пояснювальної записки:

1. Аналіз завдань із розвитку технологій командної розроби ІТ-проектів

2. Комунікації в командній розробці та методологія Agile

3. Реалізація Agile-орієнтованої моделі командної розробки та управління ІТ-проектами

4. Практичне використання розробки

5. Охорона праці та безпека в надзвичайних ситуаціях.

Висновки та пропозиції.

Бібліографічний список.

Додатки.

5. Перелік графічного матеріалу: 1 та 2 – Тема, мета, завдання роботи; 3 – Аналіз головних понять управління ІТ-проектами; 4 – Аналіз традиційних методологій управління; 5 – Аналіз гнучких методологій управління; 6 – Дошки для планування робіт; 7 – Унікальність та відмінність розробки; 8 – Діаграма архітектури програмного засобу; 9 – Розгортання корпоративного веб-додатку; 10 – Багатофункціональна платформа Postman для роботи з API; 11 – Клієнт-серверна взаємодія; 12 – Обробка виключень; 13 – Висновки.

6. Консультанти з розділів:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1, 2, 3, 4	*Луб П.М., доцент кафедри інформаційних технологій*		
5	*Городецький І.М., доцент кафедри фізики, інженерної механіки та безпеки виробництва*		

7. Дата видачі завдання 12 вересня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту	Строк виконання етапів роботи	Приміт-ка
1.	*Написання першого розділу та означення головних завдань роботи*	12.09 - 01.10.24	
2.	*Виконання другого розділу та формування головних показників для розрахунків*	12.09 - 01.10.24	
3.	*Виконання третього розділу, розрахунків та розробка листів*	01.10 - 01.11.24	
4.	*Написання розділу: «Охорона праці та безпека в надзвичайних ситуаціях»*	01.10 - 01.11.24	
6.	*Завершення оформлення розрахунково-пояснювальної записки та аркушів графічної частини*	01.11 - 01.12.24	
7.	*Завершення роботи в цілому*	01-10.12.24	

Студент _____ Тучапський Д.Ю.
(підпис)

Керівник роботи _____ Луб П.М.
(підпис)

УДК: 004.4'2:005.35

Кваліфікаційна робота: 72 с. текст. част., 31 рис., 1 табл., 13 слайдів, 29 джерел.

Імплементація Agile-технологій в командній координації розробки IT-проектів. Тучапський Денис Юрійович. Кафедра IT. – Дубляни, Львівський НУП, 2024.

Проаналізовано завдання із розвитку технологій командної розроби IT-проектів. Зокрема, виконано аналіз особливостей IT-проектів, традиційних технологій розробки IT-проектів, а також гнучких технологій розробки IT-проектів.

Означено комунікації в командній розробці та методологію Agile. Наведено особливості комплексних інструментів для командного управління IT-проектами. Наведено систему координації командного управління проектами на основі методології Agile. Акцентовано на унікальності запропонованої системи комунікації розробників IT-продуктів.

Представлено реалізацію Agile-орієнтованої моделі командної розробки та управління IT-проектами. Для цього, розроблено архітектуру та базові функціональні кейси програми реалізовані в Java. Представлено реалізацію логічної моделі обміну даними на основі реляційної MySQL. Описано застосування функціоналу програмного засобу який інтегрований в Postman.

Наведено практичне використання розробки. Зокрема, розкрито суть клієнт-серверної взаємодії застосунку та наведено приклади обробки виключень взаємодії між клієнтом та сервером.

Ключові слова: Agile, командна розробка, координація, управління, IT-проекти, API, клієнт, сервер.

ЗМІСТ

ВСТУП

Високі темпи накопичення інформації, вимоги мобільності, оперативності поширення та надійності зберігання даних, швидкості та достовірності їх обробки спонукають розвиток апаратних засобів, системних та прикладних оболонок і пакетів. На сьогоднішній день 97 % населення світу користується мобільними телефонами, а кількість комп'ютерів у світі перевищує 1361 мільйонів і продовжує зростати. При цьому приблизно один раз в півтора року подвоюються показники основних технічних характеристик апаратних засобів, один раз в два-три роки змінюються покоління програмного забезпечення і один раз на п'ять-сім років змінюється база стандартів, інтерфейсів та протоколів. Разом з тим, темп кількісного зростання інформаційних систем значно перевищує темп підготовки спеціалістів, здатних ефективно працювати з ними.

Сучасні організації та фахівці в галузі інформаційних технологій стикаються з різноманітними проектами, що вимагають систематизації, контролю та оптимізації. Технології управління проектами відіграють ключову роль, надаючи набір методологій, інструментів і стратегій, які допомагають досягти поставлених цілей і впоратися з викликами сучасного світу IT.

Існує безліч методологій та підходів із управління проектами, однак найбільш гнучкими в проактивними є так звані «гнучкі» методології управління. Зокрема, Agile – це методологія управління проектами та розроблення програмного забезпечення, що ґрунтується на гнучкості, ітеративності та активній взаємодії із замовником. Цей підхід застосовують у проектах, де потрібна висока готовність до швидких змін і до адаптації впродовж виконання проекту. Це робить методологію Agile особливо корисною для проектів, де необхідне швидке розроблення, часті зміни або важлива взаємодія із замовником.

Мета роботи – розробка каркасу та реалізація програмної системи для командної розробки й управління IT-проектами побудованої за гнучкою методологією Agile.

Завдання роботи:

- проаналізувати традиційні та гнучкі технології розробки IT-проектів;
- охарактеризувати комплексні інструменти для командного управління розробкою;
- розкрити унікальність та архітектуру запропонованої системи комунікації розробників IT-проектів;
- реалізувати систему управління засобами автоматизації Java та фреймворку Spring Boot;
- структурувати клієнт-серверну взаємодію за стосунку.

Об'єктом роботи є показники програмної системи, головні функціональні блоки, архітектура системи, зв’язки в командній розробці та управління IT-проектами.

Предметом роботи є показники роботи системи, її складові, обмін даними між клієнтом та сервером, головні елементи програмного коду.

Новизна роботи:

- проаналізовано Agile методології та визначено їх особливості;

- розроблено архітектуру програмного забезпечення та спосіб побудови системи для командної розробки й управління IT-проектами побудованої за гнучкою методологією Agile;

- реалізовано головні функціональні елементи (каркас) системи, контролери для обробки HTTP-запитів, обробки виключень тощо.

Практичне значення одержаних результатів. Запропонована система має ознаки інноваційності – дозволяє розробникам працювати із функціоналом управління IT-проектами без зайвих труднощів та витрат часу. Вона забезпечуватиме ефективне управління IT-проектами з використанням Agile підходу, дозволяючи командам розробників працювати швидше, гнучкіше та результативніше.

Розроблена UML-діаграма класів відображає структуру системи управління IT-проектами. Її реалізація передує процесу створення Spring Boot проекту, реалізації контролерів, репозиторіїв, сервісів та обробників помилок. Всі ці

компоненти взаємодіють між собою та забезпечують високу якість та ефективність системи командного управління ІТ-проектами.

Загалом, реалізація системи командного управління ІТ-проектами на Java з використанням Agile методологій є важливим внеском у галузь розробки ПЗ. Вона демонструє, що шлях до успішного управління ІТ-проектами може бути спрощений та оптимізований за допомогою сучасних інструментів та технологій.

РОЗДІЛ 1
АНАЛІЗ ЗАВДАНЬ ІЗ РОЗВИТКУ ТЕХНОЛОГІЙ КОМАНДНОЇ РОЗРОБКИ ІТ-ПРОЕКТІВ

1.1. Аналіз особливостей ІТ-проектів

Поняття "*проект*" об'єднує різноманітні види діяльності, що характеризуються рядом загальних ознак, найбільш загальними з яких є наступні [1, 6, 8, 17]:

- спрямованість на досягнення конкретних цілей, визначених результатів;
- координоване виконання численних взаємозалежних дій;
- обмеженість у ресурсах та у часі з певним початком і кінцем.

Відмінність проекту від виробничої діяльності полягає в тому, що проект є одноразовою, нециклічною діяльністю.

Рисами проекту є:

- спрямованість на досягнення конкретних цілей;
- координоване виконання взаємозалежних дій;
- обмеженість у часі з чітко визначеними початком та завершенням;
- унікальність дій по плануванню, виконанню робіт та результатів.

Результатом реалізації ІТ-проекту є – програмні комплекси, програмно-технічні системи, інформаційно-аналітичні системи, системи автоматизації проектування та інші

Отже, ***ІТ-проект*** – це цілеспрямоване, заздалегідь опрацьоване і заплановане створення або модернізація технологічних та бізнес-процесів на основі розробки програмних та програмно-технічних систем, технічної та організаційної документації для них, а також управлінських рішень і заходів щодо їх виконання (рис. 1.1).

Проект в ІТ це:

1. План робіт по розробці програмно-технічного забезпечення.

2. Система дій та методів технічної підтримки виконання робіт по розробці програмно-технічного забезпечення.

3. Система дій та методів по забезпеченню підтримки ресурсів (фінансових, людських, технічних та ін.) для виконання робіт по розробці програмно-технічного забезпечення.

4. Контроль та моніторинг строків виконання робіт відповідно до плану.

5. Контроль дотримання якості виконання робіт відповідно до плану.

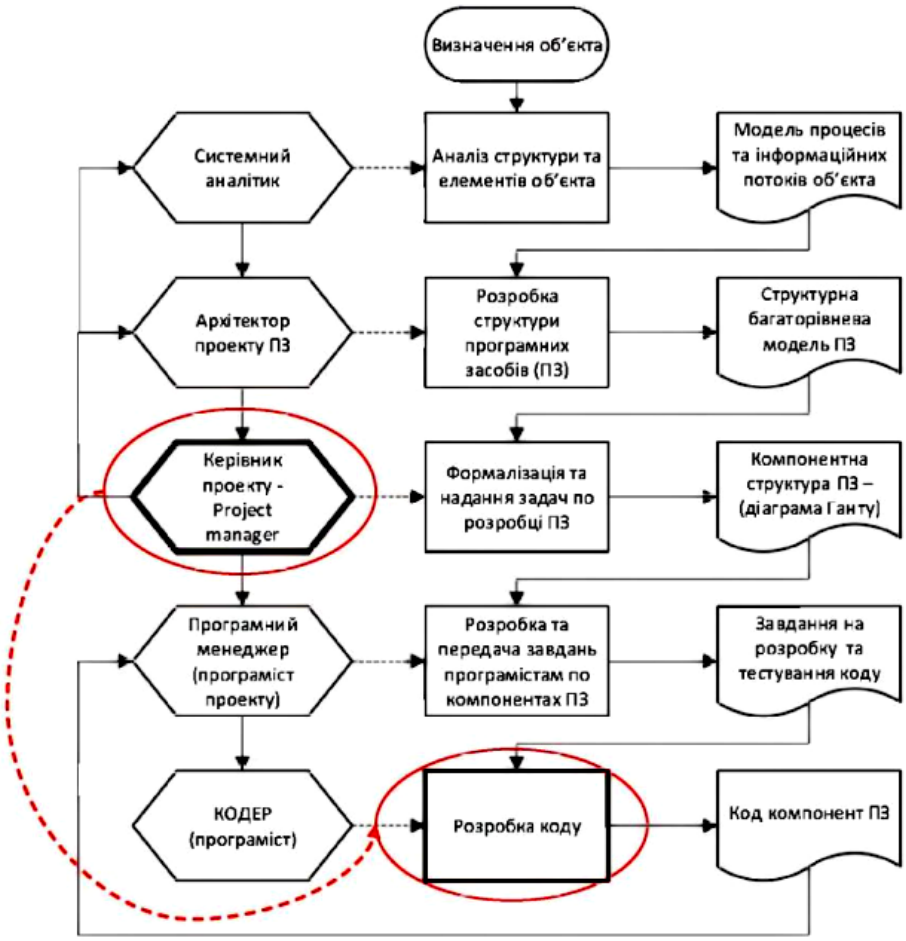

Рис. 1.1 – Роль та місце управління проектами в розробці програмного забезпечення [1, 8]

Проект включає в себе задум (проблему), засоби його реалізації (вирішення проблеми) і одержувані в процесі реалізації результати.

Особливі риси проекту [6, 17]:

1. ***Спрямованість на досягнення окреслених цілей.*** Чітка постановка кінцевої мети проекту сприяє його успішній реалізації за умови правильного

формулювання проміжних взаємозалежних цілей. Реалізація проекту означає послідовне досягнення цілей з найбільш низького рівня до вищого, тобто до досягнення кінцевої мети.

2. ***Координоване виконання взаємозалежних дій.*** Одні дії необхідно виконувати паралельно, інші – послідовно, і будь-яке порушення порядку їх виконання може поставити під загрозу виконання проекту взагалі.

3. ***Обмеженість в часі.*** Запорукою успішної реалізації проекту є оптимальний розподіл зусиль і ресурсів у часі, який забезпечується приведенням в порядок послідовності виконання робіт і заходів в межах проектної діяльності. На відміну від виробничої системи проект є одноразовою, а не циклічною діяльністю.

4. ***Унікальність.*** Кожен проект має відмінні риси і ознаки. Не існує ідентичних проектів, навіть якщо вони передбачають виконання однакових дій.

У тому випадку, коли в якості результатів реалізації проекту виступають деякі фізичні об'єкти (програмні та програмно-технічні засоби, виробничі комплекси та ін.), визначення проекту може бути пов'язано з певними обмеженнями на його розробку та виконання. Потрійне обмеження (рис 1.2) – це терміни чи час виконання, вартість робіт за проектом та зміст чи об’єм робіт проекту.

Рис. 1.2 – Потрійне обмеження ІТ-проектів

В певних галузях, таких, як авіаційна, космічна або оборонна промисловість, створювані ІТ об'єкти є настільки складними, що робота над ними здійснюється не в складі проектів, а в складі програм, які можна визначити, як

сукупність проектів або проект, що відрізняється особливою складністю створюваної продукції і/або методів управління його здійсненням.

Портфель проектів, як група проектів пов'язаних єдиним планом та фінансуванням. Папка проектів (рис 1.3), як група проектів пов’язаних будь якими груповими признаками (географічними, джерелами фінансування, замовниками, користувачами, виконавцями, періодом часу тощо).

Рис. 1.3 – Типи сукупностей ІТ-проектів

Водночас, розробка нових проектів завжди йде паралельно із функціонуванням ІТ-компаній – відбувається поточна операційна діяльність та реалізація ІТ-проектів (рис. 1.4).

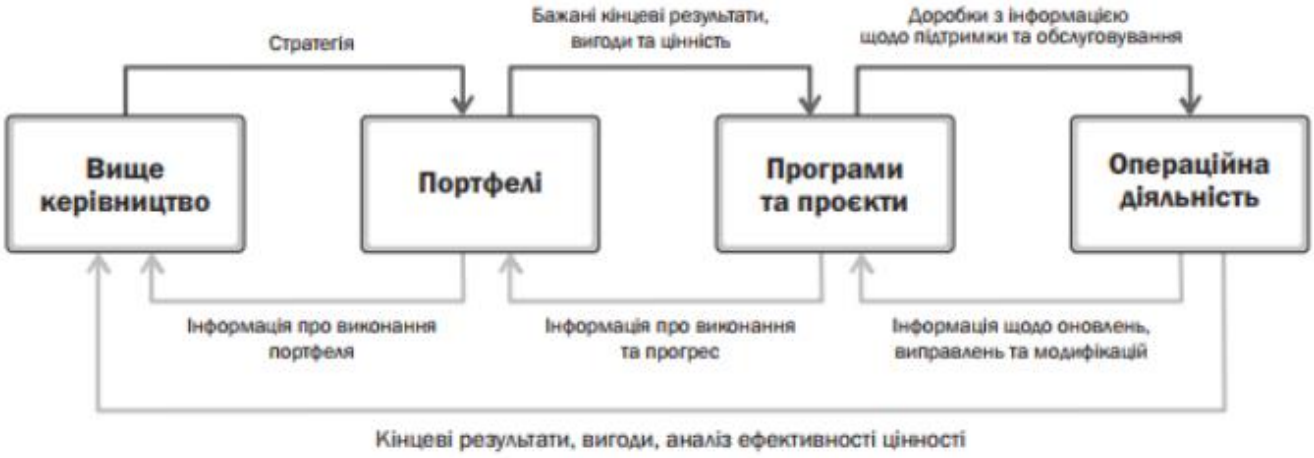

Рис. 1.4 – Поточна операційна діяльність та процеси проектування [1, 17]

Отже, проект – це цілеспрямоване, обґрунтоване та сплановане створення або модернізація програмно-технічних засобів, програмних комплексів, технічної та організаційної документації для них, управлінських рішень у рамках матеріальних, фінансових, трудових та інших ресурсів, що виділені для реалізації проекту.

1.2. Аналіз традиційних технологій розробки ІТ-проектів

Сучасні системи розробки ІТ-продуктів досягли таких величезних масштабів і складності, що їхнє створення вимагає долучати до роботи цілі команди фахівців різного профілю: програмісти, аналітики, системні адміністратори, тестувальники і кінцеві користувачі [2]. Через таку велику кількість залучених осіб ускладнюється координація робіт, а для полегшення цього процесу компанії дотримуються певних моделей управління життєвим циклом розробки ПЗ.

Традиційними моделями життєвого циклу ПЗ найчастіше називають (рис. 1.5):

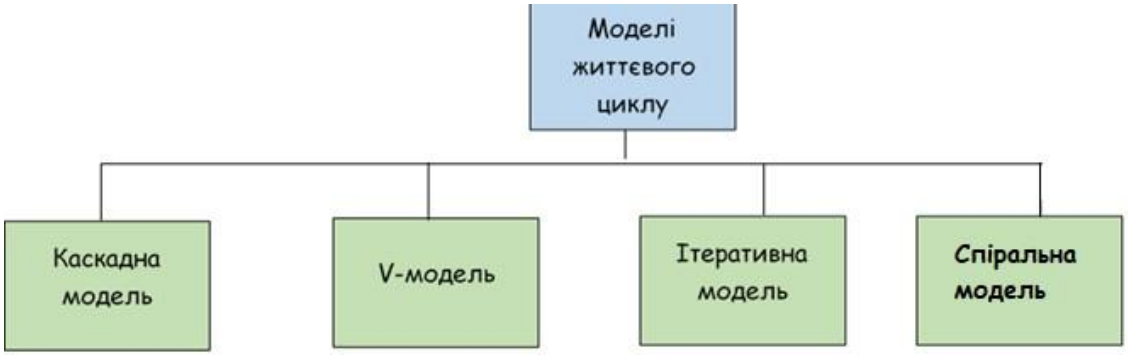

Рис. 1.5. – Класифікація моделей життєвого циклу ПП [6]

Каскадна (або ***водоспадна, Waterfall, лінійно-послідовна***) ***модель*** є найстарішою, найпростішою і найвідомішою моделлю побудови багаторівневого процесу розробки ПЗ. Вона є дуже легкою і простою для розуміння, але, в той же час, занадто ідеалістичною і є не настільки практичною [7]. Сьогодні каскадна модель застосовується переважно великими компаніями і лише для великих і складних проектів, які передбачають всеосяжний контроль ризиків – вона є дуже важливою, оскільки всі інші моделі життєвого циклу розробки ПЗ базуються саме на ній.

Ця модель ділить життєвий цикл на деякий набір фаз, кожну з яких можна розпочати лише після завершення попередньої, тобто вихід однієї фази є входом для наступної. І ось за такої ідеї, процес розвитку є «водоспадом» з послідовним

потоком фаз, звідки і походить назва (рис. 1.6).

Сьогодні використовую усі із зазначених моделей. Зокрема, каскадна модель надалі використовується в світових компаніях, ітераційна модель є яскравим представником еволюційного погляду, але, в той же час, є єдиною моделлю, яка приділяє явну увагу аналізу та попередження ризиків [10, 13].

Слід підкреслити, що кожне розроблене ПЗ відрізняється і вимагає відповідного підходу до розробки ПЗ, який слід застосовувати, спираючись на внутрішні та зовнішні фактори.

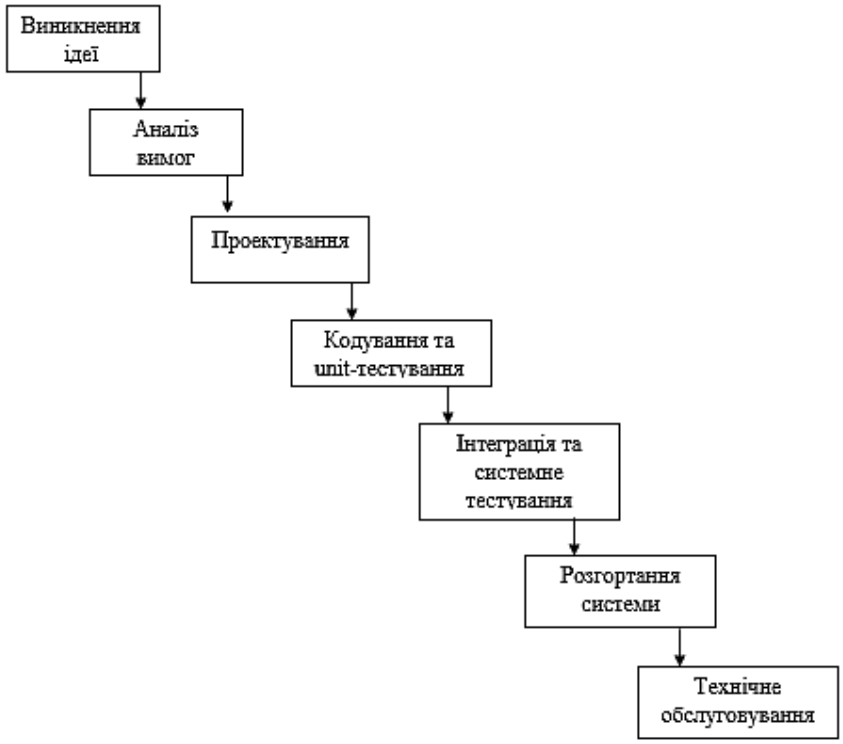

Рис. 1.6. – Каскадна модель життєвого циклу ПЗ

V-модель є «покращеною» версією розглянутої вище каскадної моделі розробки ПЗ, де виконання процесів відбувається послідовно у, так званій, V-подібній формі. Вона також відома як модель верифікації та валідації [25]:

- *верифікація* – це статична практика перевірки документів, дизайну, архітектури, коду тощо, яка включає техніку статичного аналізу (огляд), виконану без виконання коду, та відповідає на питання «Чи робимо ми продукт правильно?».

- *валідація* – це процес оцінки кінцевого продукту, необхідно перевірити, чи відповідає ПЗ очікуванням і вимогам клієнта, яка включає техніку динамічного аналізу та відповідає на питання «Чи робимо ми продукт правильним?».

Кожен етап моделі супроводжується контролем поточного прогресу, що дає можливість оцінити наскільки можливо та доцільно переходити на наступний рівень проекту згідно із моделлю [25]. При цьому, ще зі стадії написання вимог починається процес тестування та продовжується для кожного наступного етапу відповідно до того, які очікувані результати та критерії входу/виходу прописано у сформованому тест-плані (рис. 1.7).

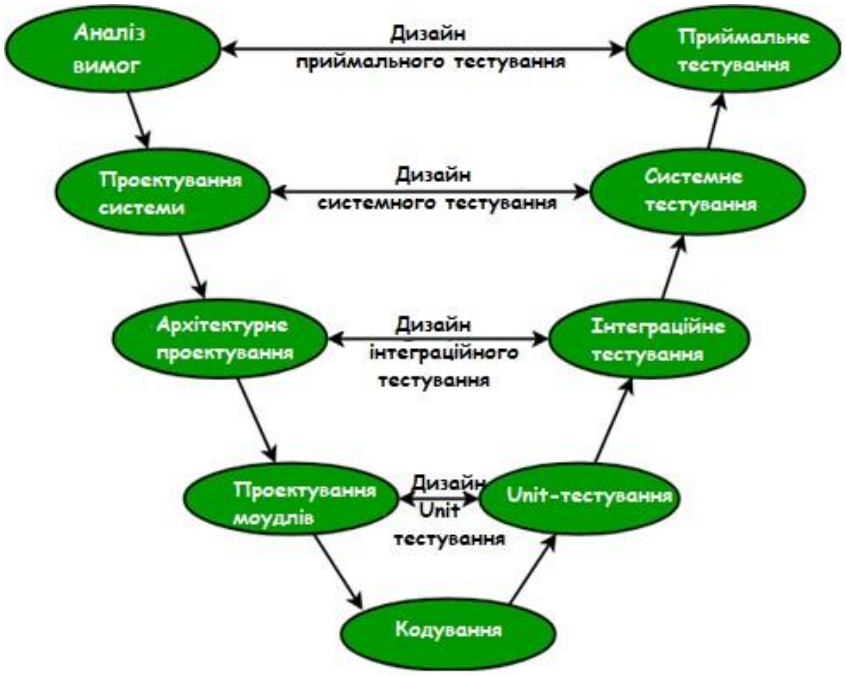

Рис. 1.7 – V-модель життєвого циклу розробки ПЗ

Ітеративна розробка ПЗ – модель, яка передбачає поетапний процес, в якому кожна фаза додає функціональність до розроблюваного ПЗ та включає свій незалежний набір заходів з розробки та тестування [13, 25]. Дана модель, по суті, є перехідною (від каскадної до Agile) моделлю розробки ПЗ і, на думку багатьох фахівців, оптимальною. Такий підхід до розробки ПЗ виконується шляхом паралельного виконання робіт, що включає в себе процеси безперервного аналізу отриманих в ході цього результатів і коригуванням попередніх етапів роботи.

Згідно основної ідеї цього методу, система розроблюється повторюваними

циклами (ітеративно) та невеликими «порціями» за один раз (інкрементно).

Життєвий цикл розробки ПП розбивається на міні-цикли (ітерації) замість єдиної послідовності етапів, кожен з яких включає свої власні етапи аналізу вимог, проектування, реалізації і завершується тестуванням, інтеграцією та створенням працюючої частини системи. Таким чином, система поступово збільшується крок за кроком і може приймати «товарний вигляд» за 10-15 ітерацій (рис. 1.8).

Перевагою даної моделі, перед попередніми розглянутими традиційними, є те, що вона не вимагає повного обсягу вимог для початку розробки [13]. Вважається за достатнє – наявність вимог до базової частини функціоналу, а вже на наступних ітераціях, вимоги доповнюються, модифікуються і призводять до розширення функціоналу системи.

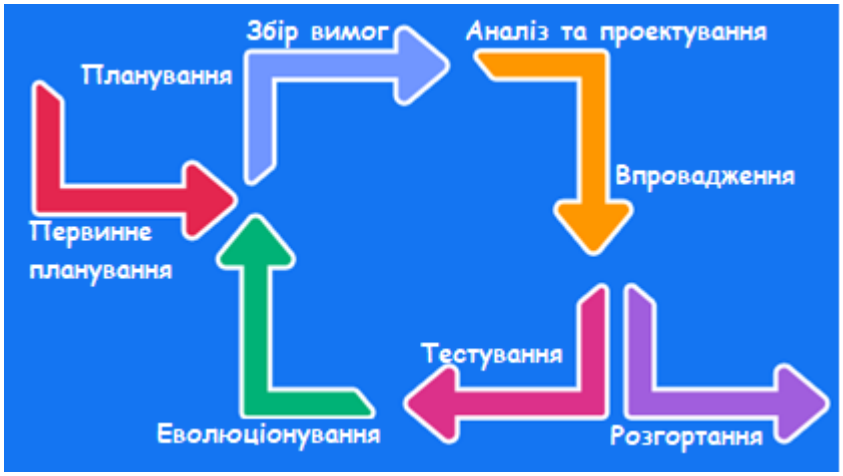

Рис. 1.8 – Ітеративна розробка ПЗ

Спіральна модель – модель розробки ПЗ, яка забезпечує підтримку управління ризиками, на схематичному відображенні дана модель відображається у вигляді спіралі (фази) із безліччю петель, кількість яких може варіюватися в залежності від проекту [6]:

- радіус спіралі – витрати (вартість) проекту на даний момент;
- кутова розмірність – прогрес, досягнутий на даний момент на поточній фазі.

Тому спіральна модель – модель, яка є подібною до «поступового» розвитку системи, який було приведено вище, але з більшим акцентом на аналізі ризиків

(рис. 1.9).

Ризик, в даному випадку, розглядається як несприятлива ситуація і можливість її впливу на успішність завершення проекту ПЗ. І спіральна модель передбачає управління цими невідомими та несприятливими ситуаціями після початку проекту шляхом розробки їхнього «прототипу» на кожному етапі розробки ПЗ [18].

Спіральну модель ще іноді носить іншу назву – ***мета-модель***, внаслідок того, що вона включає у себе всі інші розглянуті вище моделі.

Вона поєднує ідею ітеративного розвитку із систематичними, контрольованими аспектами моделі водоспаду, тому що є комбінацією ітеративної моделі процесу розвитку та моделі водоспаду з дуже високим акцентом на аналізі ризиків.

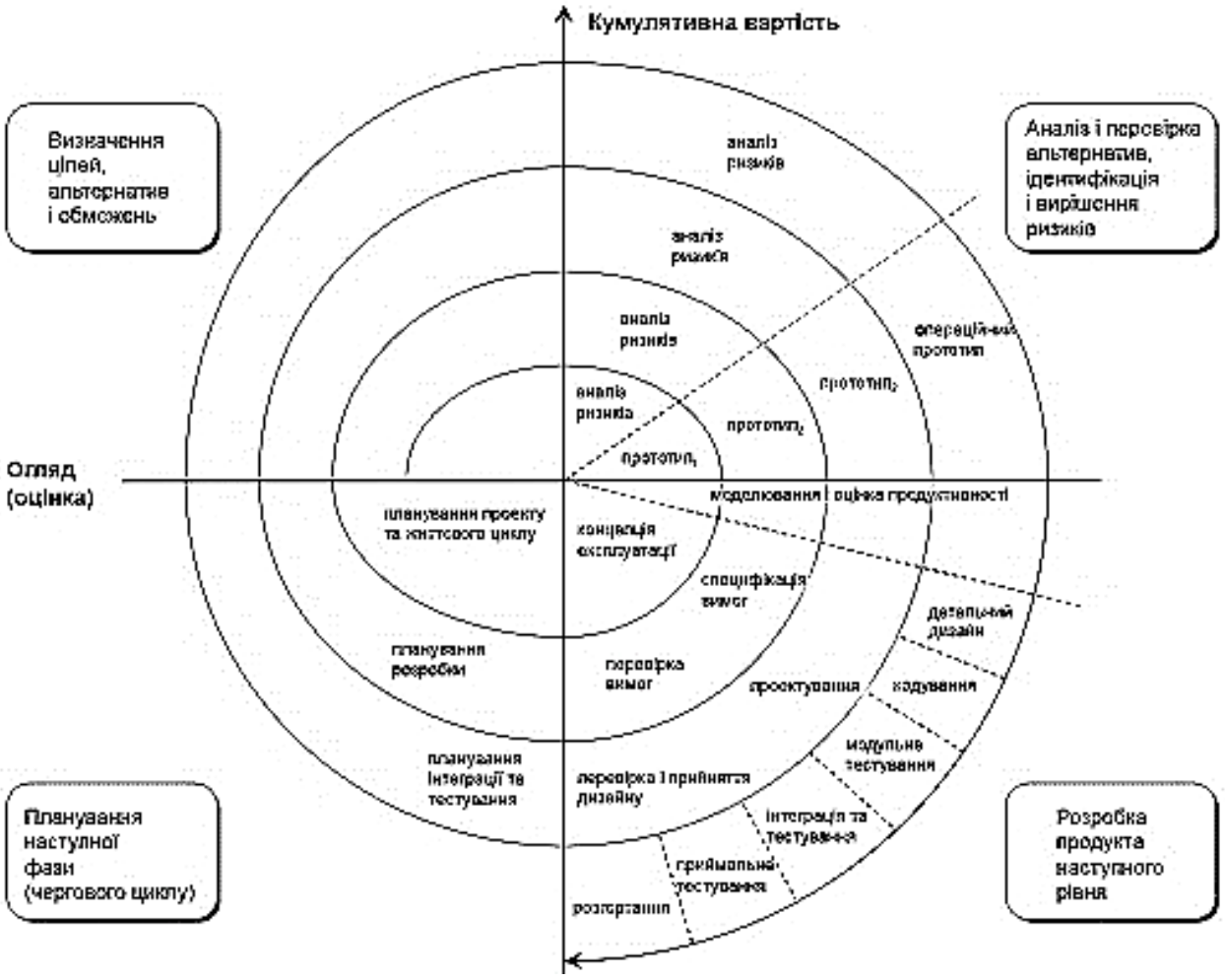

Рис. 1.9 – Спіральна модель життєвого циклу ПЗ

Спіральна модель є широко застосованою в індустрії ПЗ й наступні тези відображають типове використання спіральної моделі [18]:

- обмеженість бюджету;
- важливість оцінки ризику;
- складні та динамічні вимоги;
- вимога поетапного випуску ПЗ;
- великий проект.

1.3. Аналіз гнучких технологій розробки ІТ-проектів

Гнучкі технології розробки ПЗ скеровані на безперервний зворотній зв'язок і врахування зміни у вимогах протягом життєвого циклу ПЗ, підтримувати тісну співпрацю між клієнтами і розробниками, а також забезпечувати перспективні програмні функції [8] (рис. 1.10).

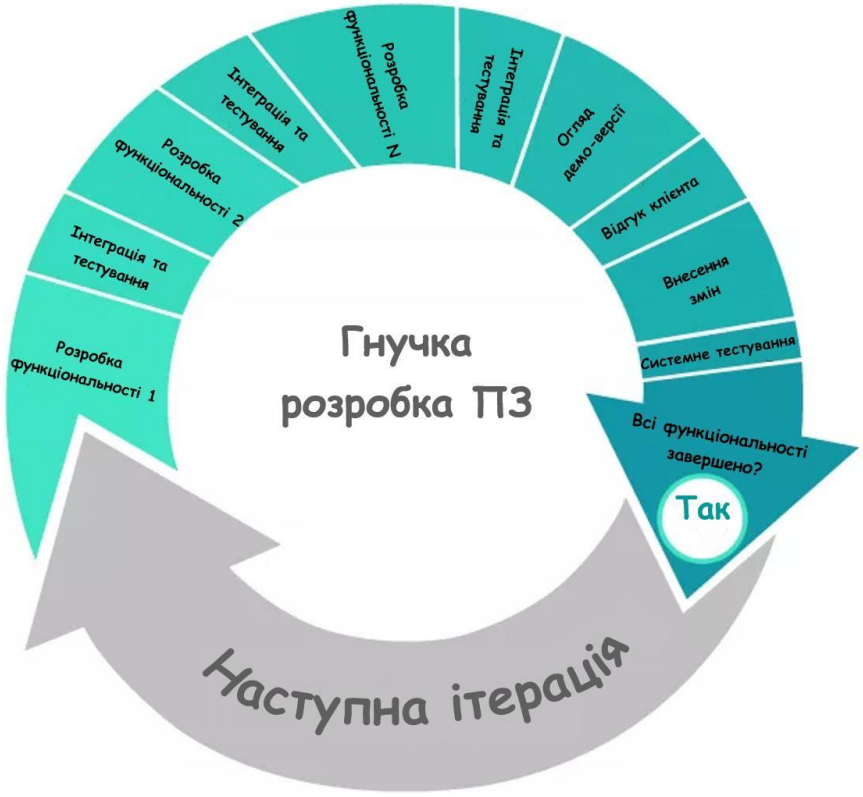

Рис. 1.10 – Життєвий цикл гнучкої розробки ПЗ

Маніфест ***Agile*** – база, на якій побудовані методи гнучких технологій розробки ПП. Він був опублікований групою розробників ПЗ та консультантів у 2001 році [3]:

Ми постійно відкриваємо для себе більш досконалі методи розробки ПЗ, займаючись розробкою безпосередньо і допомагаючи в цьому іншим. Завдяки виконаній роботі ми змогли усвідомити, що [3]:

Люди і взаємодія *важливіше процесів та інструментів;*

Працюючий продукт *важливіше вичерпної документації;*

Співпраця з замовником *важливіше узгодження умов контракту;*

Готовність до змін *важливіше проходження попереднім планом.*

Гнучкі методології можна визначити як групу процесів розробки ПЗ, які є [3] (рис. 1.11):

- ***ітераційними*** як спроба вирішення проблем шляхом пошуку послідовних наближень до рішення (наприклад, розроблена повна система змінює функціональність підсистем з кожним випуском внаслідок оновлення вимог);
- ***поступовими*** як підхід, який передбачає розподіл системи на функціональні підсистеми та додавання нової функціональності до загальної системи з кожним випуском;
- ***самоорганізуючими*** як концепція, що дає команді можливість організовувати самостійно процес для найкращого результату завдання (наприклад, взаємодія в команді, динаміка роботи, робочий час команда самостійно розподіляє для найліпшого вирішення);
- ***виникаючими*** як досвід навчання з кожного проекту, внаслідок того, що кожен проект організовується по-різному, застосовуючи ітеративні, поступові, самоорганізуючі та новітні методи.

Деякі з відомих методів гнучких технологій розробки ПП – це екстремальне програмування (XP), Scrum, кристальні методології Crystal Clear, Kanban, розробка керована функціями, динамічний метод розробки [4]. Зокрема, розглянемо три найвідоміші методи – Scrum, XP, Kanban.

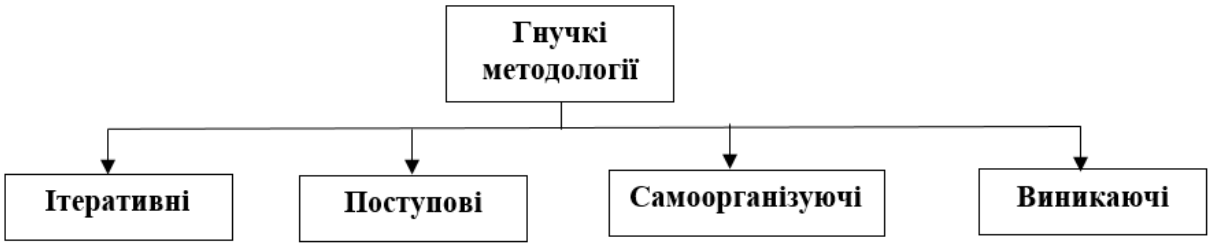

Рис. 1.11 – Визначення гнучких методологій

Scrum – один із провідних методів гнучких технологій з розробки ПЗ, який є ітеративним та інкрементним підходом до керування проектами. Назва методу – це однойменний термін з гри регбі, де він означає «повернення м'яча поза грою» через командні зусилля [5]. В проектах із Scrum-ом є основними наступні ролі.

Перша роль – це *Scrum-майстер*. Він є тренером, який допомагає команді правильно використовувати процес Scrum для виконання процесів на найвищому рівні. Від традиційного менеджера проекту його відрізняє те, що ця роль не забезпечує щоденне керівництво командою і не призначає завдання окремим особам.

Власник продукту представляє бізнес, клієнтів або користувачів, і направляє команду до правильної мети, аби створити якісний продукт [19]. Робиться це шляхом створення бачення продукту із подальшою передачею цього бачення команді через відставання продукту.

Третя і остання роль – *команда Scrum*. Оскільки людина, яка б вирішувала хто і яке завдання буде виконувати відсутня, то дана команда є такою, що саморганізовується. Ці питання по розподілу задач і зайнятості вирішуються командою в цілому. Scrum-команда є функціональною, що означає, що кожен повинен взяти участь в розробці від ідеї до реалізації [19].

Scrum акцентує увагу на тому, що "визначені та повторювані процеси працюють лише для вирішення визначених та повторюваних проблем із визначеними та повторюваними людьми у визначених та повторюваних середовищах", а це, як очевидно, неможливо. Саме тому, для вирішення

наведеної проблеми процесів, проект розділяється на, так званні, ітерації (спринти).

Scrum-процес ділиться на три частини: *фаза аналізу*, *фаза розробки і фаза огляду* (рис. 1.12).

Рис. 1.12 – Структура процесу Scrum

Спринти завершуються оглядом результатів клієнтом. Дані результати оцінюються і визначається чи були досягнуті цілі, встановлені для спринту.

Extreme programming. Іншим популярним методом гнучких технологій розробки ПЗ є екстремальне *програмування (XP)*. Як і інші методи гнучких технологій, він працює завдяки об’єднанню всієї команди за допомогою простих практик із включенням достатнього зворотного зв’язку, а також виступає за коротку ітерацію і часті випуски робочого коду з метою підвищення продуктивності [11]. XP був розроблений для команд, що включають 8-10 учасників, які працюють з об'єктно-орієнтованою мовою програмування.

XP-процес починається зі створення історій користувача замовником, що описують невеликі одиниці функціональності і які можна реалізувати в середньому за тиждень-два процесу тестування та кодування. Замовник, виходячи

із кошторису, який йому надають, пріоритезує історії. Реалізація функцій проходить ітераційно, що передбачає поставки замовнику кожні два тижні. Поступово система зростає у функціональності, по частинах, паралельно керуючись замовником на оглядах [11].

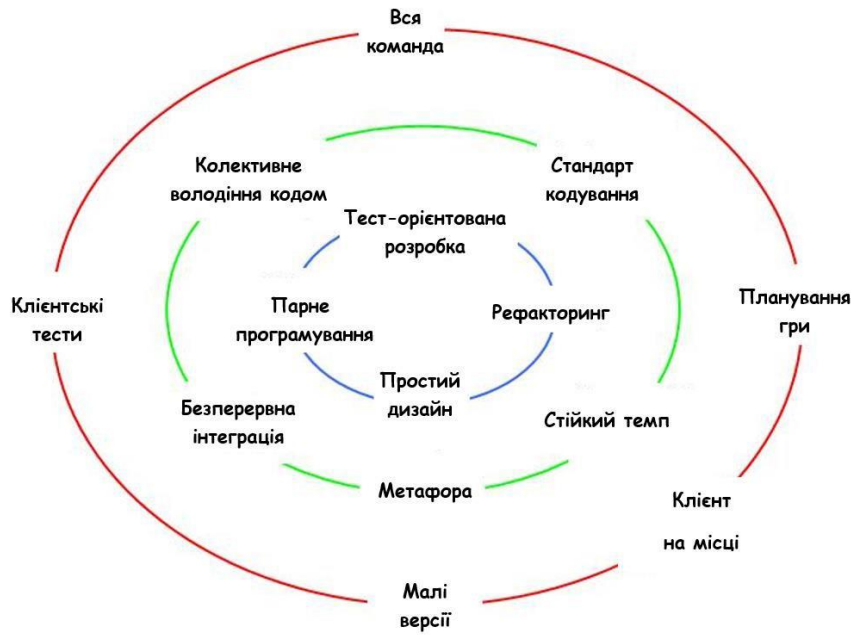

Рис. 1.13 – Практики екстремального програмування

Kanban – це метод гнучких технологій, який допомагає здійснювати процеси проектування, управління та вдосконалення потокових систем. Його назва походить внаслідок того, що в основі нього лежить використання канбан-механізмів візуальної сигналізації, які дають змогу контролювати готовність для ПЗ. Він дотримується принципу, що робота безперервно протікає через систему, а не організовується у різні часові рамки [14].

Цей метод придатний для використання у ситуаціях, де необхідна організація знань, коли робота є непередбачуваною та коли алгоритм роботи передбачає інкрементну поставку продукту замовнику, як тільки він буде готовий.

Дуже пізнаваною практикою Kanban є дошка, яка є обов’язковим елементом даного підходу. Вона завжди знаходиться у вільному доступі, тому кожен член команди в будь-який час бачить на якому етапі перебуває завдання. Kanban дошка підійде і реальна, і віртуальна: можна використовувати просту

коркову або програми-трекери на кшталт Trello, Jira, оскільки вона підлаштовується під будь-який процес і застосовується в будь-якій області з метою скласти список справ [14].

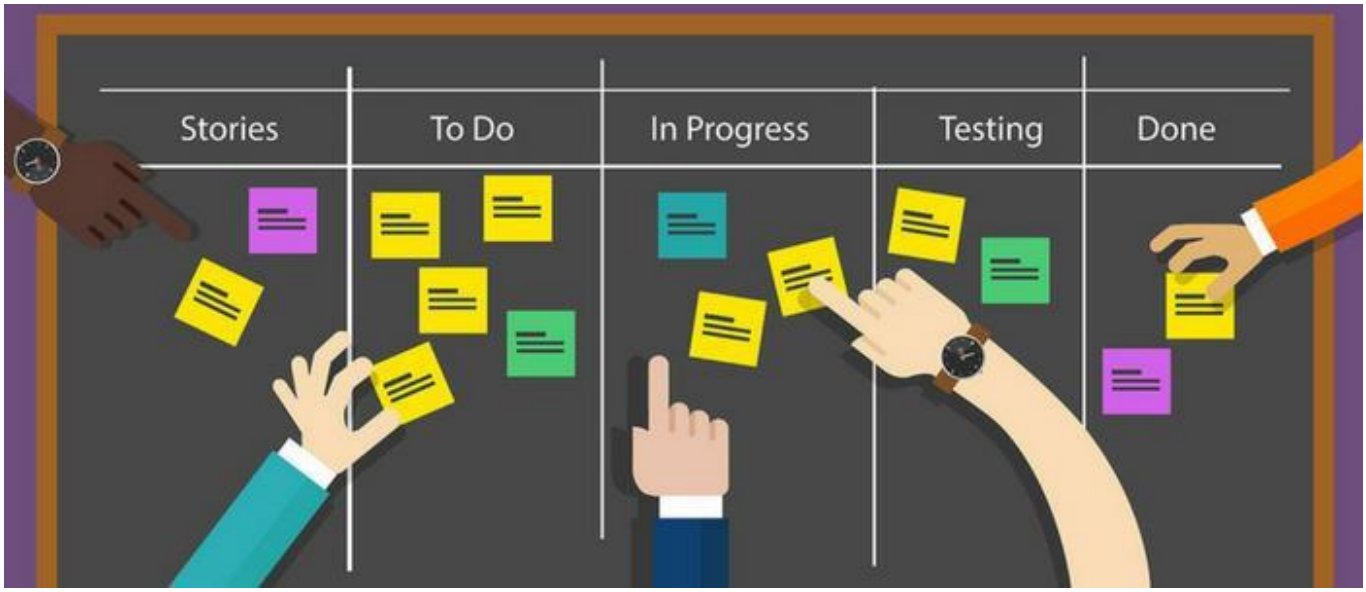

Рис. 1.14 – Kanban дошка

Отже, вся робота у Kanban є єдиним потоком і кожна система у процесі роботи відрізняється етапами.

Деякі команди об'єднують ідеали Scrum і Kanban в Scrumban. Зі Scrum беруться спринти фіксованою тривалості і ролі, а з Kanban – концентрацію на тривалості циклу і обмеження кількості одночасно виконуваних завдань. Проте, на етапах адаптації та гнучкої трансформації настійливо рекомендується вибрати ту чи іншу методику і деякий час слідувати виключно їй, адже, час для експериментів знайдеться завжди.

РОЗДІЛ 2
КОМУНІКАЦІЇ В КОМАНДНІЙ РОЗРОБЦІ ТА МЕТОДОЛОГІЯ AGILE

2.1. Комплексні інструменти для командного управління IT-проектами

Сьогодні використовують різноманітні інструменти для управління робочим процесом у різних командах розробки. Вони створювалися як рішення для відстеження завдань та виявлення помилок. Зокрема, розглянемо декілька варіантів дошок для планування роботи IT-команди – вони підходять для різних випадків, від управління вимогами і сценаріями тестування до agile-розробки ПЗ.

ClickUp – це найвідоміший інструмент для підвищення продуктивності.

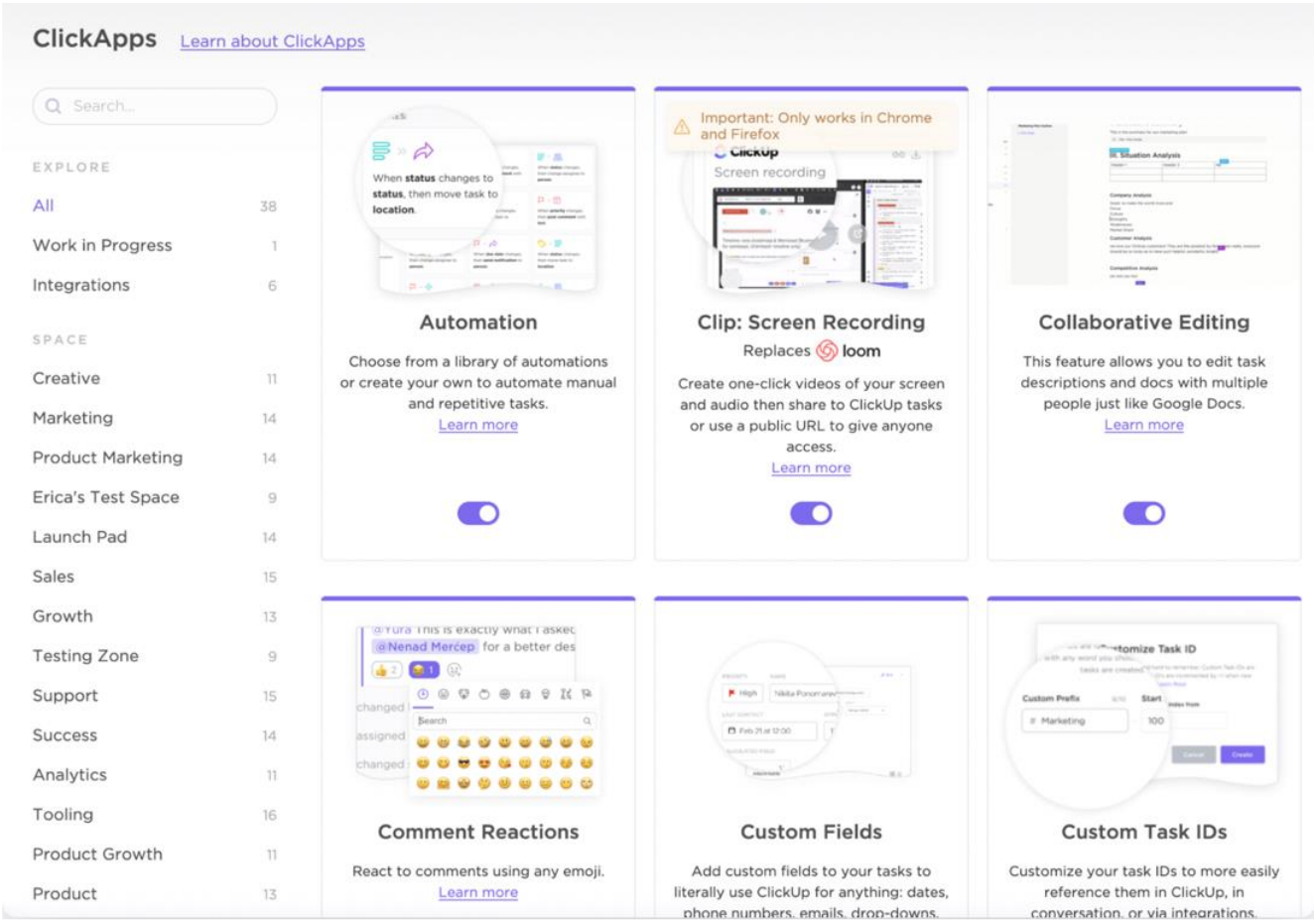

Рис. 2.1 – Головна сторінка дошки ClickUp

ClickUp використовують усі види бізнесу по всьому світу. ClickUp надає безкоштовну версію з такими функціями, як призначені коментарі, списки справ,

контрольні списки завдань, над багато редагування тексту, багатозадачна панель інструментів, прості/користувацькі статуси, спринти, цілі тощо (рис. 2.1). ClickApps дозволяє налаштувати роботу групи в кожному окремому просторі. Використовуючи цей інструмент вмикають пріоритети, теги, спеціальні поля та синхронізувати свої простори з робочим процесом. Ті, кому присвоєно звання адміністратора, мають право вмикати/вимикати ClickApps у двох розділах платформи.

Binfire — це комплексний інструмент управління проектами, який може використовувати команда інженерів та програмного забезпечення. Він надає функції керування завданнями, дошку Канбан, інтерактивну діаграму Ганта тощо. Він також включає повний набір функцій спільної роботи, необхідних для керування малими та великими проектами (рис. 2.2). Binfire створює віртуальний офіс, де члени команди можуть працювати віддалено та ефективно співпрацювати один з одним.

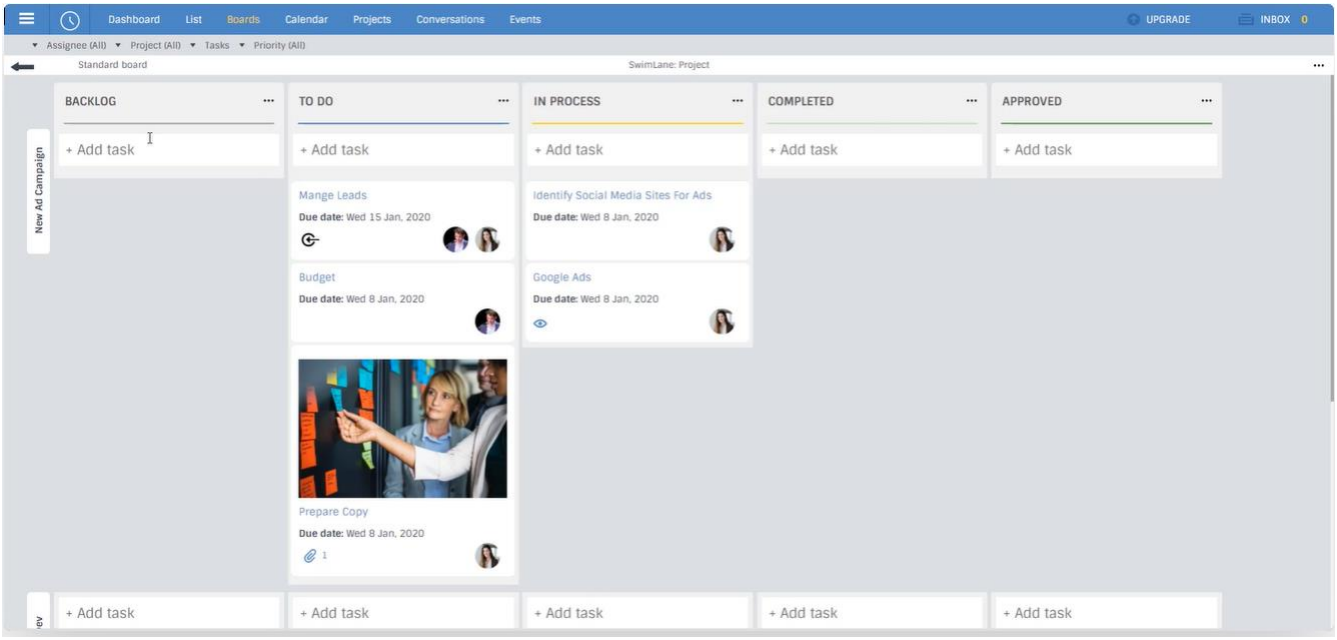

Рис. 2.2 – Інтерфейс дошки Binfire

Basecamp досить ефективно спрямовує людей з різними ролями до спільної мети. Він забезпечує модель ціноутворення з оплатою по мірі використання без будь-якого контракту (рис. 2.3). Є річний пакет для тих, хто бажає отримати

повний набір функцій системи. Вартість не залежить від кількості користувачів, тому можна залучити стільки людей, скільки потрібно.

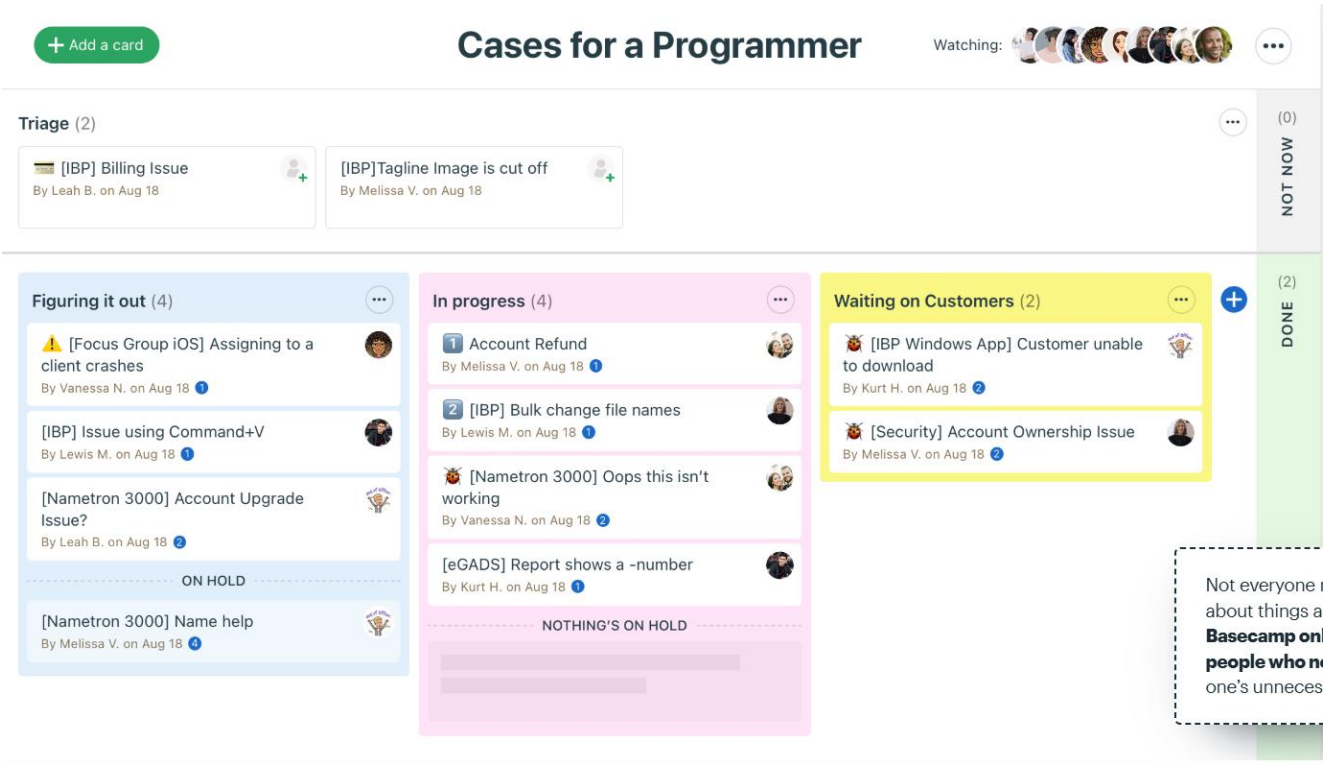

Рис. 2.3 – Інтерфейс вікна дошки Basecamp

Він дозволяє ефективно делегувати завдання та з легкістю відстежувати прогрес. Не має значення, чи знаходиться команда в різних часових поясах. Basecamp дозволяє дистанційно керувати проектами, що значно покращує швидкість роботи та продуктивність команди управління проектами.

Pivotal Tracker представляє списки відставання, функції та виправлення, щоб допомогти вашій команді вибрати, над чим працювати далі. Буде видно, як швидко працює команда розробників гнучкої програми, причому швидкість команди розраховується на основі очок історії, завершених на кожній ітерації.

Pivotal Tracker навіть допомагає планувати свої ітерації за допомогою свого інструмента відстеження (рис. 2.4), який дає змогу перервати керований обсяг роботи.

Asana стала досить популярним інструментом управління проектами. Вона полегшує комунікацію та співпрацю всієї команди управління проектами. Функції Asana включають кілька робочих просторів, можливість додавати призначених і

вкладення до завдань, відстеження завдань, спільну роботу в реальному часі та можливість коментувати завдання.

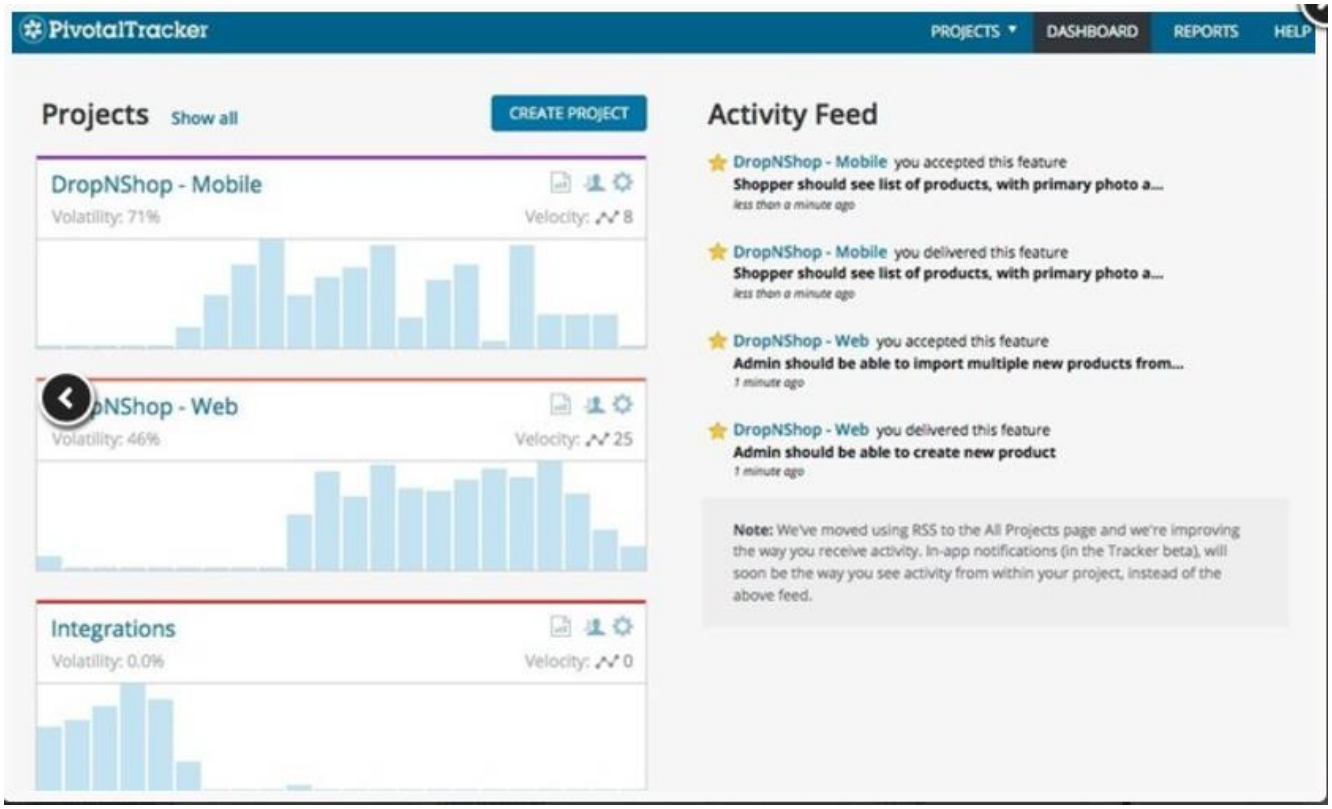

Рис. 2.4 – Дошка Pivotal Tracker

Користувачі навіть можуть бачити завдання та пріоритети своїх колег під час завантаження. Це тип прозорості, який потрібен кожній організації та гнучкому проекту. Навіть можна відстежувати хід виконання проектів і завдань із різних пристроїв і браузерів. Немає необхідності покладатися на сторонні програми або електронну пошту для корпоративного спілкування, коли запущена Asana.

Kanbanize — ще один популярний інструмент керування проектами. Дозволяє командам візуалізувати ключові ініціативи та розбивати їх на робочі елементи, комбінуючи прості у використанні дошки канбан. Надає обмеження роботи, фільтри, доступ на основі ролей і спеціальні поля, щоб ви та ваші команди могли легко візуалізувати роботу так, як завгодно. Використовуючи дошки канбан, щоб створити різні користувацькі робочі процеси або функцію робочого процесу на шкалі часу, щоб змінити їх. Він також дозволяє відстежувати

час, щоб побачити години, витрачені на виконання завдань або проектів, а також аналітику для моніторингу ефективності.

Jira — це перш за все програмне забезпечення для управління проектами, але воно почало своє життя в 2002 році як платформа для відстеження проблем для розробників програмного забезпечення.

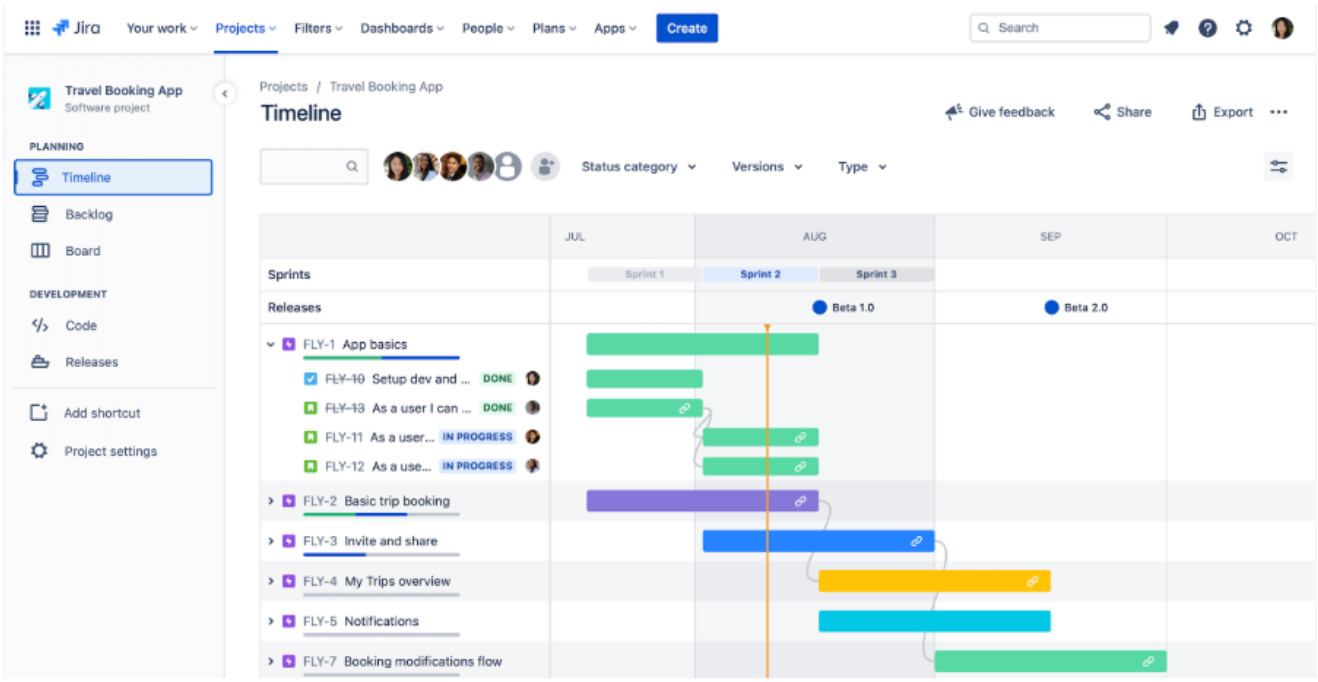

Рис. 2.5 – Дошка Pivotal Tracker

Зараз він пропонується в трьох окремих пакетах:

• Jira Core: основна платформа управління проектами Jira;

• Jira Software: пропонує всі функції Jira Core, але також включає додаткову функціональність Agile;

• Jira Service Desk: призначений для IT-фахівців або інших форм служби підтримки [13].

2.2. Побудова системи координації командного управління проектами на основі методології Agile

Ефективність використання Agile методології у системах управління

командної розробки залежить від багатьох факторів, але загалом цей підхід є дуже ефективним для багатьох організацій та проектів.

Гнучкість: Agile підходить для проектів, що потребують постійних змін та швидкої реакції на них. Це забезпечується за допомогою коротких циклів розробки, що дозволяють командам швидко адаптуватися до нової інформації та вносити зміни у проект.

Співпраця та комунікація: Agile сприяє активній співпраці та комунікації між членами команди, клієнтами та зацікавленими сторонами. Це досягається за допомогою регулярних стендапів, планування спринтів та ретроспектив, що забезпечує прозорість, спільне розуміння цілей та зниження ризику невірного розуміння вимог.

Постійний зворотній зв'язок: Agile методології акцентують значення постійного зворотного зв'язку. За допомогою демонстрацій функціональності після кожного спринту, команда отримує цінний відгук від клієнтів та користувачів, що дозволяє швидше вносити зміни та покращувати якість продукту.

Висока якість продукту: Agile методології засновані на принципах постійного тестування, автоматизованого тестування та неперервної інтеграції. Це дозволяє командам виявляти й виправляти помилки на ранніх етапах, покращувати якість продукту та забезпечувати високу клієнтську задоволеність.

Складність планування: Agile підхід акцентується на змінних вимогах, що може ускладнити планування проекту і прогнозування графіка та обсягу робіт. Відсутність жорсткого графіка може також ускладнити комунікацію з управлінням проекту та клієнтами, які можуть бути звиклими до традиційних методів управління.

Вимоги до дисципліни команди: Agile методології вимагають високого рівня дисципліни, самоорганізації та відповідальності з боку команди. Це може бути викликом для нових команд або команд із обмеженим досвідом використання технології Agile.

Відсутність деталізації вимог: Agile наголошує на постійній взаємодії з

клієнтом та зміні вимог. Проте, це може призвести до недостатньої деталізації вимог на початкових етапах проекту. Це може ускладнити розробку плану та оцінку зусиль, особливо в проектах із великим масштабом або коли потрібно враховувати строгі регуляторні вимоги.

Не підходить для всіх проектів: Agile методології можуть бути менш ефективними, якщо проект має жорсткі вимоги до графіка, бюджету або безпеки, або якщо проект потребує чітко визначених вимог і планування на початкових етапах.

Отже, кожен проект і команда мають свої особливості, і ефективність Agile методологій може значно варіюватись залежно від цих факторів. Успіх використання гнучких методологій залежить від правильного вибору методології, її адаптації до потреб проекту та команди, а також від підтримки та керівництва організації.

2.3. Унікальність запропонованої системи комунікації розробників IT-продуктів

Унікальність запропонованої системи командної розробки та управління проектами полягає в тому, що вона реалізована як окремий API. Це дасть користувачам-розробникам зручний спосіб використання всіх функціональних можливостей системи без необхідності працювати з складним користувацьким інтерфейсом. Це дозволяє їм інтегрувати систему управління проектами у свої проекти швидко та ефективно. Тоді, розробники отримають гнучкість у виборі того, які функції вони хочуть використовувати. Вони можуть вибрати лише ті API-методи, які їм потрібні для їх конкретних потреб та ігнорувати інші. Це дозволяє оптимізувати використання ресурсів і забезпечити ефективну роботу із системою командного управління IT-проектами.

Окремий API для системи управління проектами також прискорює процес розробки. Розробники можуть скористатися готовими функціями та модулями

системи управління проектами, що дозволяє їм уникнути повторної розробки та зосередитися на вирішенні специфічних завдань своїх проектів. Це збільшує швидкість розробки та скорочує час досягнення мети. В запропонованій системі управління передбачаються такі розширення функціональності, що в майбутньому зможуть надати більше можливостей якісно керувати проектом:

Spring Security. Структура, надана Spring, яка допомагає налаштувати доступ і процес автентифікації – відіграє дуже важливу роль у забезпеченні безпеки програм.

Звітність. В системах управління проектами звітність відіграє важливу роль у процесі контролю та оцінки прогресу проектів. Це процес збору, аналізу та представлення інформації про ключові аспекти проекту, що дозволить керівникам та зацікавленим сторонам отримувати необхідні дані для прийняття рішень та встановлення планів дій.

Ticket linking (Зв’язок між тікетами). В системі управління проектами та використовується поняття "зв'язок між тікетами" для встановлення зв'язку між основним тікетом та підтікетом (sub ticket). Основний тікет може бути, наприклад, головним запитом чи задачею, яку треба вирішити. Підтікет є додатковою задачею, яка пов'язана з основним квитком і потребує окремого виконання.

Коментарі усюди. Коментарі можуть бути додані до різних об'єктів у системі управління проектами, таких як задачі, спринти, проектні документи, зміни, помилки тощо. Це дозволить створювати контекстні обговорення та забезпечить збереження комунікації відносно конкретного елемента проекту.

Можливість відмічати користувачів в коментарях. Це дозволить полегшити комунікацію та спілкування між учасниками проекту, особливо у випадах, коли коментарі можуть містити багато інформації або стосуються конкретних питань, які потребують уваги конкретних користувачів.

Фільтрація за періодом. Можливість фільтрувати спринти, епіки та тікети за періодом дозволить користувачам швидко знаходити та переглядати проекти, завдання або інші елементи, які були створені, змінені або закриті протягом певного періоду часу.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ AGILE-ОРІЄНТОВАНОЇ МОДЕЛІ КОМАНДНОЇ РОЗРОБКИ ТА УПРАВЛІННЯ ІТ-ПРОЕКТАМИ

3.1. Архітектура та базові функціональні кейси програми реалізовані в Java

Архітектура ПЗ є основою будь-якої успішної програмної системи. Вона впливає на все, починаючи від зручності обслуговування, масштабованості, стабільності та безпеки протягом життєвого циклу системи. Першим кроком до впровадження нової програмної системи є діаграма архітектури [4, 7, 17].

Базовий функціонал розроблюваного ПЗ включатиме наступні модулі:

Клас Project (Проект). Цей модуль дозволяє користувачам почати нові проекти та налаштовувати їх параметри. Під час створення нового проекту, користувач отримує можливість ввести основну інформаціє про проект.

Клас User (Користувач). Модуль користувача в системах управління проектами відіграє важливу роль у забезпеченні ідентифікації та керування доступом користувачів до функцій та ресурсів проекту. Його основна мета полягає у реєстрації нових користувачів та автоматизації вже зареєстрованих для надання їм відповідних прав доступу.

Клас Epic (Епік). Надає великий блок початкового функціоналу, або функціональної області системи і може бути розбитий на менші задачі.

Клас Sprint (Спринт). Створення *Спринта* розпочинається після завершення попереднього Спринта, або на початку проекту, який триває від 1 до 4 тижнів.

Клас Ticket (Тікет). Тікет може бути створений для запланованих завдань, помилок, побажань або будь-якої іншої роботи, яка вимагає уваги команди проекту.

Клас Comments (Коментарі). Коментарі в системах управління проектами відіграють важливу роль у спілкуванні та документуванні робочих процесів команди проекту. Вони надають можливість членам команди обмінюватися

інформацією, висловлювати думки, задавати питання та надавати коментарі до різних аспектів проекту.

Для представлення архітектури програми розробленої системи використано UML *(Unified Modeling Language)* діаграму класового типу й уніфіковану мову моделювання [7, 25].

Class diagrams (Діаграми класів) є основним будівельним блоком будь-якого об’єктно-орієнтованого рішення. Він показує класи в системі, атрибути та операції кожного класу та зв’язок між кожним класом. У більшості інструментів моделювання клас складається з трьох частин.

Розроблена діаграма класів (рис. 3.1) побудована як система для управління проектами, спрінтами, епіками, заявками, користувачами та коментарями.

- *Project* (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): 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* (Спринт): Має зв'язок один-до-багатьох з класами Epic (Епік) та Ticket (Завдання). Містить атрибути 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.

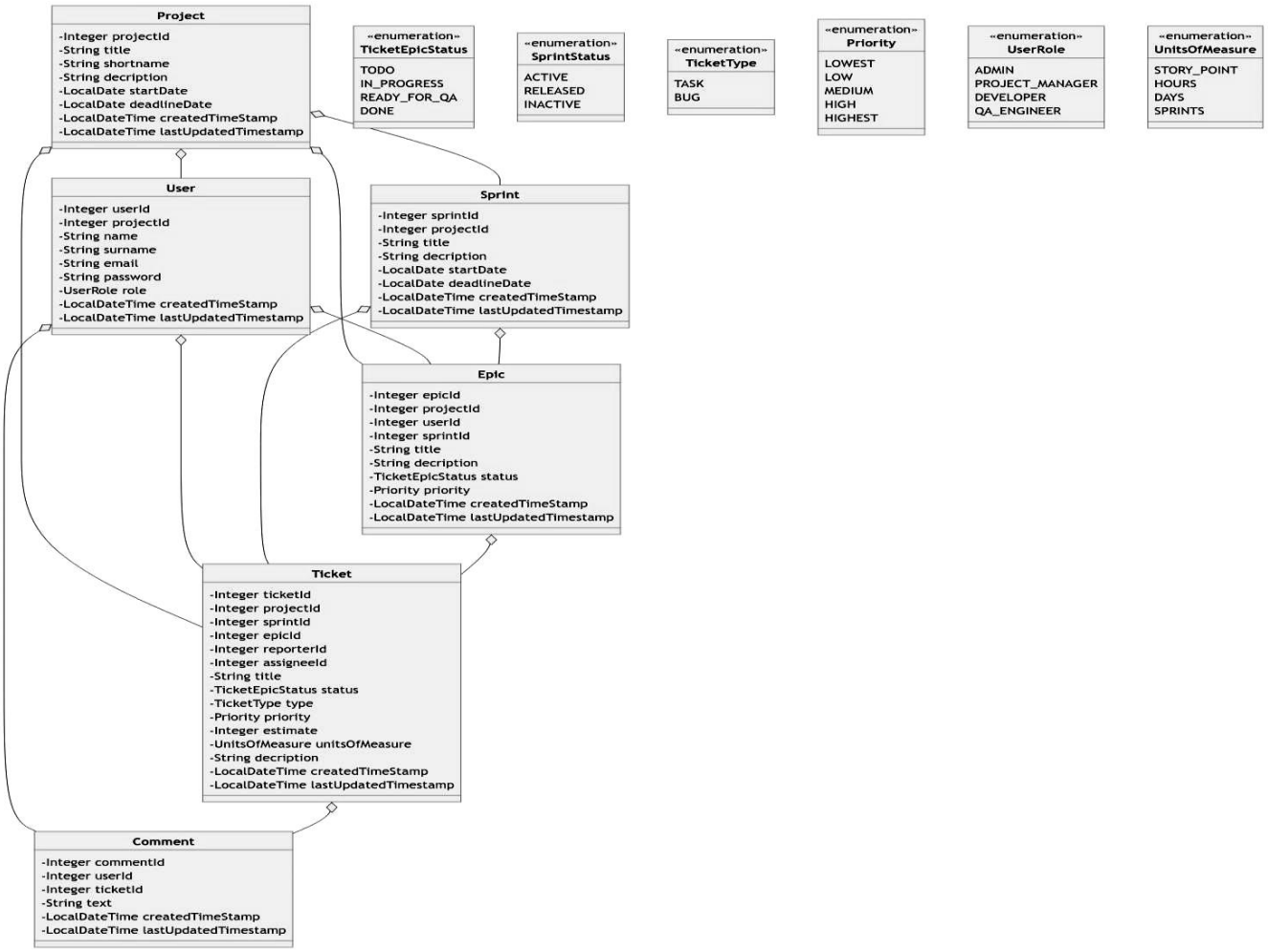

Рис. 3.1. Структура UML-діаграми класів

- *Epic* (Епік): 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): 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): 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): 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): 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): 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): 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.

Реалізація системи управління проектами виконана на підставі засобів автоматизації Maven та Java у фреймворку Spring Boot.

Spring Boot — це фреймворк Java з відкритим вихідним кодом, що спрощує завдання розгортання корпоративних веб-додатків Java. Це проект, побудований на основі Spring framework, який забезпечує ефективний спосіб налаштування та запуску програм [25].

Maven — це інструмент управління проектами та розуміння, який надає розробникам повну структуру життєвого циклу збірки.

Для легкого створення Spring Boot проекту використовуємо spring initializr (рис. 3.2). Зокрема опишемо структуру коду: у файлі конфігурації **pom.xml** описуємо всі необхідні залежності для роботи застосунку.

Приклад pom.xml файлу наведено в додатку А.

Створюємо Aplication клас, для запуску програми:

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

Spring Boot controllers є важливою складовою частиною фреймворку Spring Boot, і вони використовуються для обробки HTTP-запитів і надання відповідей на ці запити у відповідності до потреб додатку.

Головна мета контролерів полягає у виконанні дій відповідно до вхідних HTTP-запитів і поверненні відповідей на ці запити. Контролери забезпечують взаємодію між клієнтами (такими як веб-браузери, або мобільні додатки) та

серверним додатком.

Приклад оформлення класу-контролера:

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

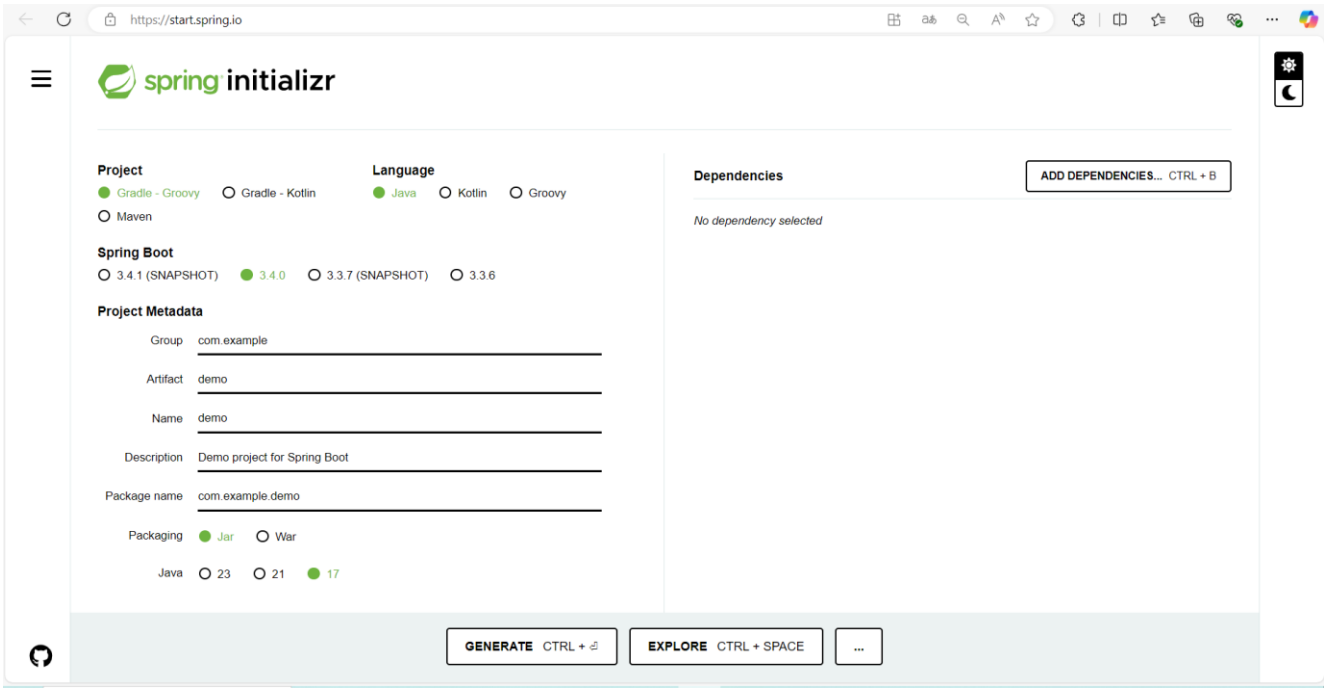

Рис 3.2. Сайт для швидкого розгортання Spring Boot проектів [25]

У Spring Boot існують чотири основні типи HTTP-запитів: *GET*, *POST*, *PATCH* та *DELETE*, які можна обробляти за допомогою контролерів.

Опис кожного з цих запитів:

1. *GET*-запит використовується для отримання даних з сервера. У Spring Boot для обробки GET-запитів використовується анотація `*@GetMapping*`. Зазвичай цей метод повертає дані клієнту у форматі JSON, або HTML сторінку:

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

2. *POST*-запит використовується для створення нових ресурсів на сервері, або відправки даних для обробки. У Spring Boot для обробки POST-запитів використовується анотація `*@PostMapping*`. Ця анотація дозволяє визначити метод контролера, який оброблятиме POST запити. Зазвичай цей метод отримує дані від клієнта через параметри запиту, або тіло запиту і виконує відповідні дії, наприклад, зберігає дані в базі даних:

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

}
```

3. *PATCH*-запит використовується для часткового оновлення ресурсу на сервері. Він дозволяє вносити зміни лише до певних полів, або властивостей ресурсу, не торкаючись інших полів. Для обробки PATCH-запитів у Spring Boot використовує анотацію *@PatchMapping*:

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

}
```

4. *DELETE*-запит використовується для видалення ресурсів на сервері. У Spring Boot для обробки DELETE-запитів використовується анотація `*@DeleteMapping*`. Цей метод приймає параметри, наприклад, ідентифікатор ресурсу, який потрібно видалити, і виконує відповідні дії для видалення ресурсу з сервера:

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

Контролери служать як проміжний рівень між вхідними запитами і бізнес-логікою додатку. Вони розділяють логіку, пов'язану з обробкою запитів, від

логіки, пов'язаної з бізнес-правилами та обробкою даних.

Контролери дозволяють встановлювати правила валідації для вхідних даних та обробляти помилки. Вони дають можливість контролювати інформацію, що надходить від клієнта, і реагувати на некоректні запити.

В прикладі вище в методі createUser(@Valid @RequestBody UserDTOuser анотація @Valid вказує на те, що деякі поля можуть мати обмеження. Обмеження теж вказуються анотаціями, але вже в класі UserDTO.

DTO (Data Transfer Object) — шаблон проектування, який виконується для передачі даних між підсистемами програми.

Також над класом контролера та класом DTO потрібно вказати анотацію @Validated. Класи DTO та контролери наведені в EpicComtroller.java (Додаток А).

3.2. Реалізація логічної моделі обміну даними на основі реляційної MySQL

Для реалізації запропонованої системи управління проектами використана *реляційна база даних MySQL*, яка зберігає дані в окремих таблицях, а не в одному великому сховищі. Структура бази даних організована у фізичні файли, які оптимізовані для швидкості обміну даними. Логічна модель даних, з об'єктами, такими як таблиці даних, представлення, рядки і стовпці, пропонує гнучке програмне середовище. Користувач встановлює правила, що регулюють відношення між різними полями даних, такі як один до одного, один до багатьох, унікальне, обов'язкове або необов'язкове, і "вказівники" між різними таблицями [4].

Для створення бази даних в MySQL Workbench у першу чергу необхідно підключитися до сервера баз даних, тоді створити нову базу і вписати скрипт створення таблиць. Лістинг скрипту наданий в EpicComtroller.java (Додаток А).

Робота з об’єктно-орієнтованим ПЗ та з реляційними базами даних є громіздкою та займає багато часу. Нами використано ***Hibernate*** – це рішення для

об’єктного/реляційного відображення (ORM – Object Relational Mapping) у середовищі Java. Інструмент ORM спрощує створення даних, маніпулювання ними та доступ до них. Це техніка програмування, яка відображає об’єкт на дані, що зберігаються в базі даних.

Фактично, Hibernate – це бібліотека для роботи із базами даних, яка не тільки дозволяє відображати класи Java у таблиці бази даних і типи даних Java у типи даних SQL, але й надає засоби для запиту та пошуку. Головна мета Hibernate – звільнити розробника від більшості рутинних задач, пов'язаних з збереженням даних.

Для підключення до нашого Maven проекту Hibernate та MySQL додаємо до файлу конфігурації pom.xml залежності hibernate-core та msql-connector-java. Ці залежності включають Hibernate ORM для роботи з об'єктно-реляційним відображенням та драйвер MySQL для з'єднання з базою даних MySQL.

Далі потрібно налаштувати файлу application.properties. У цьому файлі проекту Spring Boot необхідно вказати налаштування для 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}/pmsoftware?s
erverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=root
server.port=8081
```

У цих рядках *pmsoftware* – назва бази даних, *username* – користувач бази даних, *password* – пароль користувача бази даних.

Далі необхідно налаштувати Hibernate-анотації. Для цього створюються *Entity* класи, які будуть зберігатися в базі даних за допомогою Hibernate. Потрібно додати анотації Hibernate до цих класів, такі як *@Entity, @Table, @Id* та інші, щоб вказати взаємозв'язок з таблицями бази даних.

Оформлення Entity класу наступний:

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

Для взаємодії коду із базою даних використовуємо JpaRepository, що є інтерфейсом у бібліотеці Spring Data JPA, який дозволяє легко працювати з базами даних за допомогою підходу ORM (Object-Relational Mapping). Він є однією з реалізацій репозиторіїв і надає зручний спосіб взаємодії з даними в базі даних, включаючи зв'язування об'єктів Java з записами в таблицях бази даних.

JpaRepository містить набір методів для стандартних операцій з базою даних, таких як збереження (save), оновлення (update), видалення (delete) та отримання (get) даних. Це дозволяє виконувати основні CRUD-операції (створення, читання, оновлення, видалення) з об'єктами в базі даних без прямої роботи з SQL-запитами.

Крім стандартних операцій, JpaRepository також підтримує виконання складніших запитів. Приклад оформлення репозиторію:

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

Після успішного підключення бази даних та допоміжних бібліотек для взаємодії з нею, створюємо *Service* клас, в якому викликаємо методи репозиторію для виконання операцій з базою даних. Приклад оформлення сервісу:

```
@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
}
```

В цьому прикладі простого сервісу, використовується репозиторій для отримання об'єкта User за його ідентифікатором.

3.3. Функціонал програмного засобу інтегрований в Postman

Розгортання проектів із ПЗ, що передбачають командну роботу потребують рішень які дають змогу легко взаємодіяти із серверами та додатками, інтерпретувати їхні відповіді, створювати тести і документувати API. Таку концепцію застосовує інструмент Postman.

Postman – це багатофункціональна платформа, яка допомагає розробникам і тестувальникам на всіх етапах роботи з API: від створення і тестування до документування та інтеграції. Основна мета платформи полягає у створенні зручного способу взаємодії з API для розробників. Сьогодні це один із найпопулярніших інструментів для роботи з API.

Цей інструмент надає широкий набір функцій. Ось основні його можливості [17]:

- *Тестування API*: розробники можуть створювати і запускати тести на JavaScript, перевіряючи різні аспекти відповідей від сервера, такі як статус-коди і вміст.
- *Відправлення запитів*: зручний інтерфейс для створення і відправлення запитів до API різних типів, включно з GET, POST, PUT, DELETE та іншими.
- *Аналіз відповідей*: постмендозволяє аналізувати відповіді від сервера в зручному форматі (JSON, XML та інші), допомагаючи зрозуміти, як API відповідає на запити.

- *Спрощення розробки та тестування API*: надає зручний інтерфейс для надсилання запитів, написання тестів і аналізу відповідей, спрощуючи процес розробки та тестування API для програмістів.

Зокрема, нами використовується програмний застосунок Postman для перевірки роботи нашої програми та представлення результатів.

Отже, опишемо процес створення даних на сервері. Для нового запиту обираємо тип HTTP-запиту, який потрібно виконати, наприклад GET, POST, PUT або DELETE. Далі потрібно ввести URL-адресу нашого API в поле "Enter request URL".

Якщо запит потребує передачі даних у тілі запиту, то можна налаштувати це. Потрібно обрати вкладку "Body" та дані, які потрібно передати, наприклад JSON, або XML (рис. 3.3).

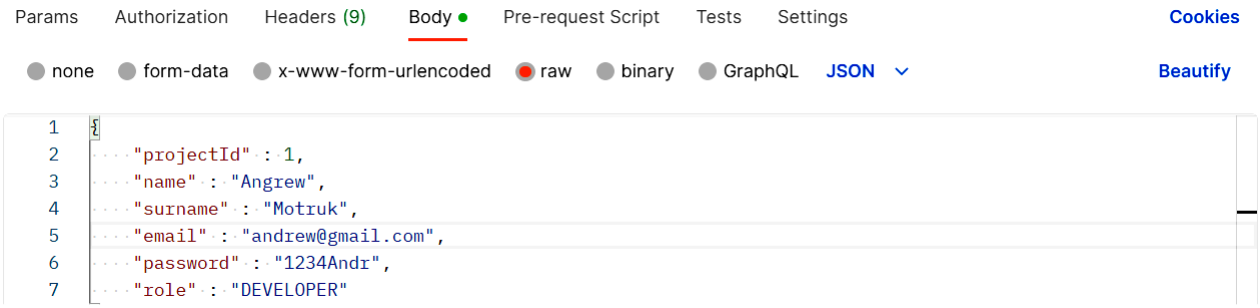

Рис. 3.3. Спосіб передачі даних у тілі запиту

Після відправки запиту Postman відображуємо отриману відповідь зі статусом запиту, заголовками, тілом відповіді та іншу інформацію. У результаті формується перелік створених користувачів в базі даних.

user_id	project_id	user_name	surname	email	user_password	user_role	created_time_
1	1	Julia	Voloshina	juliavoloshina02@gmail.com	1234Juli	ADMIN	2023-05-28 10:
2	1	Helen	Hrebeniuk	helen_kyuneberg@gmail.com	123Helen	DEVELOPER	2023-05-28 10:
3	1	Angrew	Motruk	andrewmmmm@gmail.com	1234Andr	DEVELOPER	2023-05-29 19:

Рис. 3.4. Таблиця переліку зареєстрованих користувачів

Для оновлення даних на сервері для тої чи іншої команди використовується HTTP-запит. Для цього використовують запит типу PATCH, а також змінюємо URL-адресу. В тілі змінюємо поля, що підлягають під оновлення, та отримуємо результат змінених даних проекту.

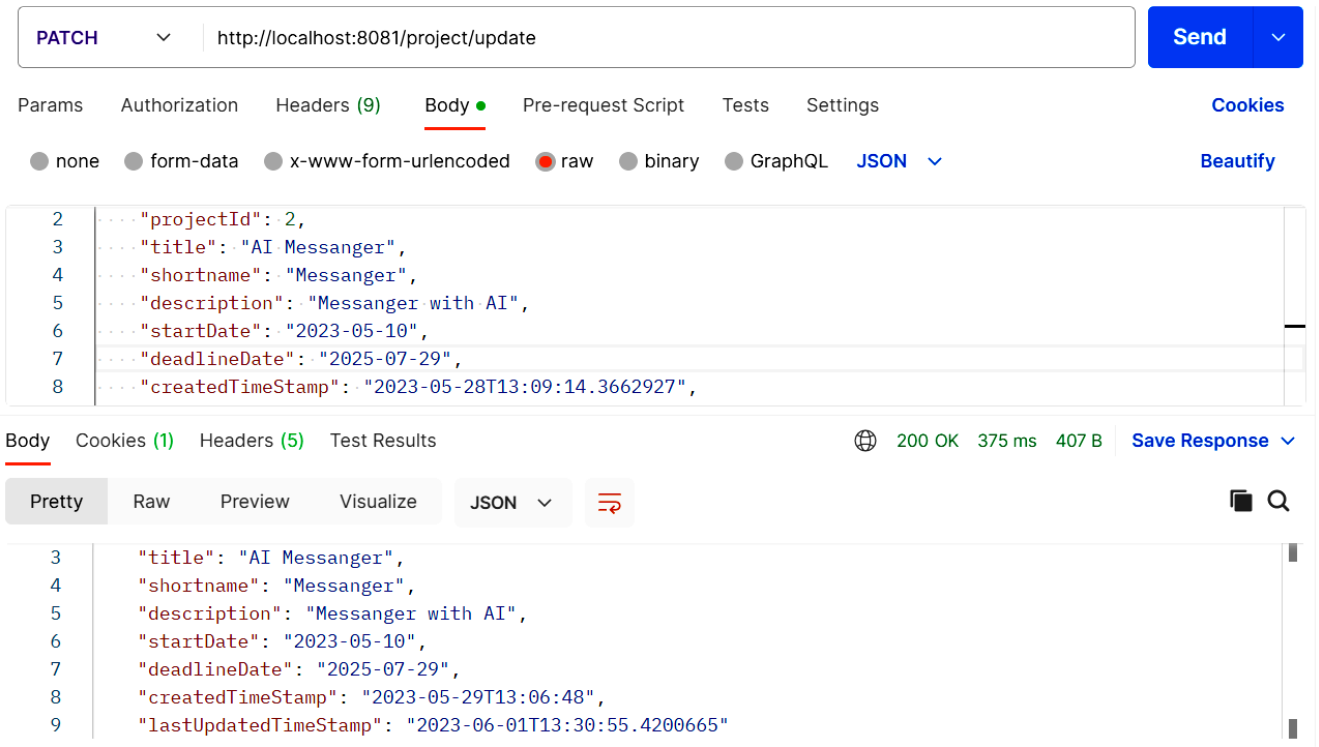

Рис. 3.5. HTTP-запит на оновлення даних проекту

Для видалення даних на сервері використовують схожий підхід. Наведемо приклад для об’єкта Спринта. На рис. 3.6 зображено новий HTTP-запит, а також змінений тип DELETE-запиту. В тіло передаємо об’єкт, який ми хочемо видалити. Тіло результату пусте, і статус запиту "200 OK". Це означає, що видалення пройшло успішно.

Рис. 3.6. HTTP-запит на видалення даних Спрінта

Отже, написання та виконання завдань розробки у Postman дає змогу розробникам автоматизувати процес перевірки працездатності API. Це робить процес тестування ефективнішим і дає змогу виявляти помилки в API на ранніх стадіях розробки:

- *Мова тестування*: Postman використовує мову JavaScript для написання тестів, що робить її гнучким інструментом для створення різноманітних тестових сценаріїв (на освітній платформі FoxmindED ви можете ознайомитися з програмою стартового курсу JavaScript Start і записатися на навчання).
- *Засоби Postman*: є вбудовані засоби для написання та виконання тестів. Це включає в себе функції, такі як tests, pm.response, pm.expect та інші, які дозволяють створювати різноманітні перевірки відповідей API.

Postman допомагає поліпшити якість API на різних етапах розробки:

- *Проектування і прототипування*: створення прототипів API за допомогою колекцій запитів для швидкої перевірки функціональності до початку активної розробки.
- *Розробка і налагодження*: надсилання запитів до API і перевірка його функціональності, а також швидке налагодження за допомогою інструментів Postman.
- *Тестування та автоматизація*: створення і виконання тестів для перевірки працездатності API, а також автоматизація тестування за допомогою Newman та інтеграція в CI/CD-пайплайни.
- *Документація API*: автоматична генерація документації API на основі колекцій запитів, що спрощує процес створення та оновлення документації.

Загалом, інструмент допомагає поліпшити якість і надійність API, даючи змогу розробникам тестувати кожен аспект функціональності, швидко перевіряти зміни, робити API зрозумілішим для команди та користувачів, а також мінімізувати ризики виникнення проблем і помилок.

РОЗДІЛ 4
ПРАКТИЧНЕ ВИКОРИСТАННЯ РОЗРОБКИ

4.1. Клієнт-серверна взаємодія застосунку

Архітектура клієнт-сервер є одним із архітектурних шаблонів програмного забезпечення та є домінуючою концепцією у створенні розподілених мережних застосунків і передбачає взаємодію та обмін даними між ними [1, 3]. Вона передбачає такі основні компоненти:

- набір серверів, які надають інформацію або інші послуги програмам, які звертаються до них;
- набір клієнтів, які використовують сервіси, що надаються серверами;
- мережа, яка забезпечує взаємодію між клієнтами та серверами.

Сервери є незалежними один від одного. Клієнти (комп’ютери-користувачі) також функціонують паралельно і незалежно один від одного. Немає жорсткої прив'язки клієнтів до серверів. Більш ніж типовою є ситуація, коли один сервер одночасно обробляє запити від різних клієнтів; з іншого боку, клієнт може звертатися то до одного сервера то до іншого. Клієнти мають знати про доступні сервери, але можуть не мати жодного уявлення про існування інших клієнтів.

Модель клієнт-серверної взаємодії визначається перш за все розподілом обов'язків між клієнтом та сервером. Логічно можна відокремити три рівні операцій:

- рівень представлення даних, який по суті являє собою інтерфейс користувача і відповідає за представлення даних користувачеві і введення від нього керуючих команд;
- прикладний рівень, який реалізує основну логіку застосунку і на якому здійснюється необхідна обробка інформації;
- рівень управління даними, який забезпечує зберігання даних та доступ до них.

Дворівнева клієнт-серверна архітектура передбачає взаємодію двох програмних модулів — клієнтського та серверного. В залежності від того, як між ними розподіляються наведені вище функції, розрізняють:

- модель тонкого клієнта, в рамках якої вся логіка застосунку та управління даними зосереджена на сервері. Клієнтська програма забезпечує тільки функції рівня представлення;
- модель товстого клієнта, в якій сервер тільки керує даними, а обробка інформації та інтерфейс користувача зосереджені на стороні клієнта. Товстими клієнтами часто також називають пристрої з обмеженою потужністю: кишенькові комп'ютери, мобільні телефони та ін.

Отже, для отримання даних зі серверу (з використанням фільтрів) використовуватимемо GET-запити. На рис. 4.1 показано спосіб пошуку користувачів із заданими фільтрами. Пошук користувачів відбувається за їх роллю.

Застосовуємо тип запиту GET. Першочергово генеруємо нову URL-адресу і в тілі запиту вказуємо роль, за якою хочемо знайти внесених у юбазу даних користувачів.

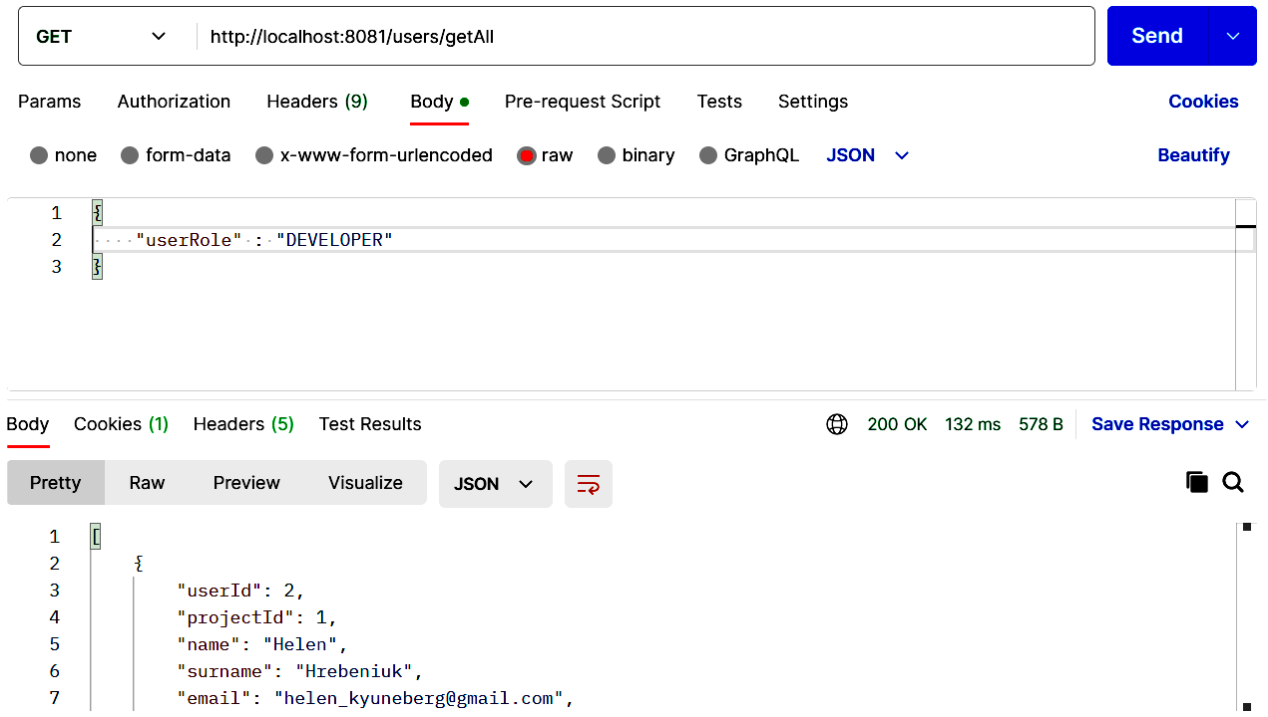

Рис. 4.1 – HTTP-запит на отримання даних користувача за вказаним фільтром

Повне тіло результату запиту матиме наступний вигляд (усі користувачі маюсь роль “DEVELOPER”):


```
{
"userId": 2,
"projectId": 1,
"name": "Helen",
"surname": "Hrebeniuk",
"email": "helen_kyuneberg@gmail.com",
"password": "123Helen",
"role": "DEVELOPER",
"createdTimeStamp": "2023-05-28T13:15:54",
"lastUpdatedTimeStamp": null
},
{
"userId": 3,
"projectId": 1,
"name": "Angrew",
"surname": "Motruk",
"email": "andrewmmmm@gmail.com",
"password": "1234Andr",
"role": "DEVELOPER",
"createdTimeStamp": "2023-05-29T22:27:19",
"lastUpdatedTimeStamp": null
}
```


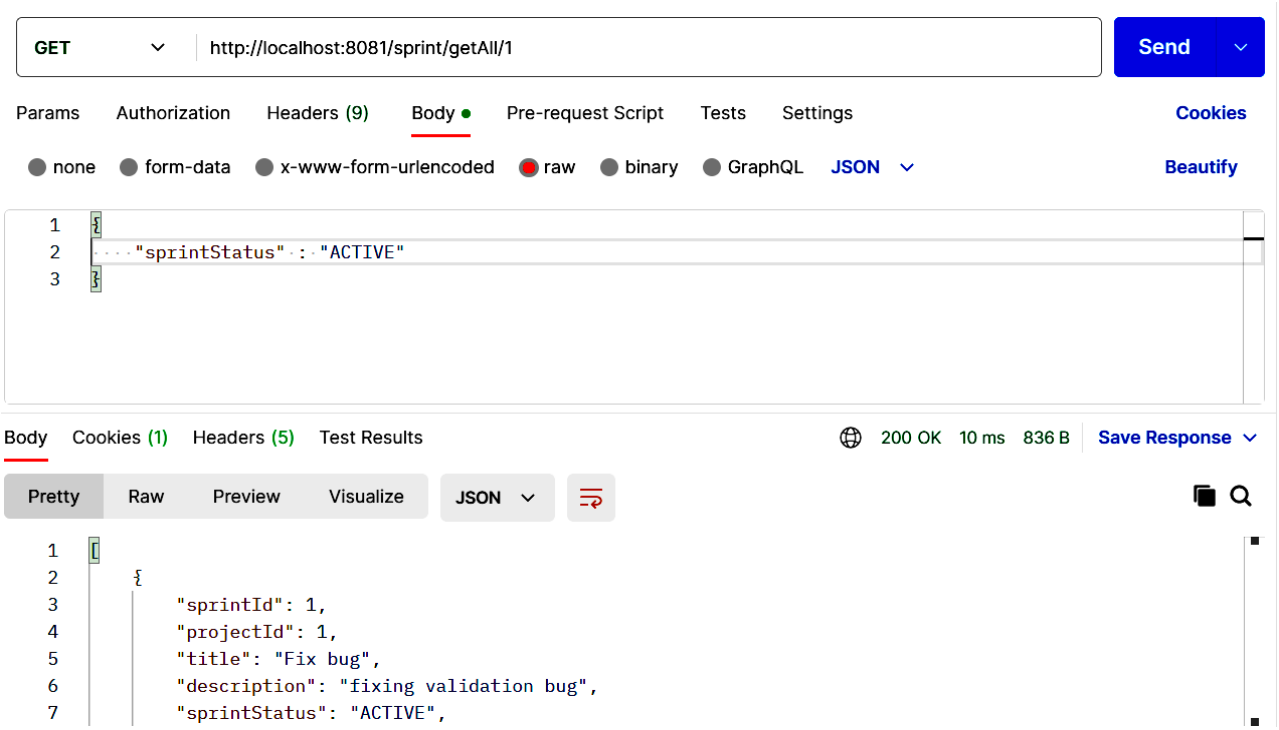

Рис. 4.2 – HTTP-запит на отримання даних користувача за фільтром та ID

Для отримання даних зі сервера з використанням фільтрів та унікального ідентифікатора ID необхідно дещо змінити (рис. 4.2). Для окремого об’єкта Спринта, беремо із бази даних Спринти з унікальним ідентифікатором проекту «1» (що прописується останнім в URL-адресі запиту) та значенням фільтру sprint status, тобто статус Спринта.

Як видно в повному тілі відповіді сервера на запит усі Спринти маюсь статус “ACTIVE” та ідентифікатор проекту «1»:


```
{
"sprintId": 1,
"projectId": 1,
"title": "Fix bug",
"description": "fixing validation bug",
"sprintStatus": "ACTIVE",
"startDate": "2023-05-20",
"deadlineDate": "2023-05-29",
"createdTimeStamp": "2023-05-28T13:43:57",
"lastUpdatedTimeStamp": null
},
{
"sprintId": 3,
"projectId": 1,
"title": "Handler creation",
"description": null,
"sprintStatus": "ACTIVE",
"startDate": "2023-05-20",
"deadlineDate": "2023-06-02",
"createdTimeStamp": "2023-05-28T13:59:44",
"lastUpdatedTimeStamp": null
},
{
"sprintId": 7,
"projectId": 1,
"title": "Validation Bug",
"description": null,
"sprintStatus": "ACTIVE",
"startDate": "2023-06-20",
"deadlineDate": "2023-06-29",
"createdTimeStamp": "2023-06-01T14:26:20",
"lastUpdatedTimeStamp": null
}
```

Отже, робота запропонованої системи має свою практичне підтвердження. Клієнт-серверна взаємодія застосунку передбачає використання стандартних запитів із відповідними фільтрами та ID.

4.2. Обробка виключень взаємодії між клієнтом та сервером

Під час розробки й використання ПЗ виникнення помилок та непередбачених ситуацій у роботі програми називають винятком (***exception***). Виключення можуть виникати внаслідок неправильних дій користувача, відсутності ресурсу на диску, або втрати з'єднання з сервером по мережі тощо. Причинами *exception* під час виконання програми можуть бути помилки програмування, або неправильне використання API. Тобто, програма має чітко знати, як діяти в такій ситуації. Для цього в Java передбачено механізм винятків.

Орім веб-служб REST, можна також використовувати Spring MVC для обробки динамічного вмісту HTML. Spring MVC підтримує різні технології шаблонизації, включно з Thymeleaf, FreeMarker та JSP. До того ж, багато інших шаблонізаторів містять власні засоби інтеграції зі Spring MVC.

Spring Boot передбачає підтримку автоконфігурації для наступних шаблонизаторів:

- FreeMarker;
- Groovy;
- Thymeleaf;
- Mustache.

Якщо один із цих шаблонизаторів використовується зі стандартною конфігурацією, шаблони автоматично підбираються з *src/main/resources/templates*.

За замовчуванням Spring Boot передбачає відображення */error*, яке обробляє всі помилки адекватним чином, і воно реєструється як "глобальна" сторінка помилок у контейнері ***сервлетів***. Для машинних клієнтів воно створює

відповідь у форматі JSON з докладним описом помилки, кодом стану HTTP та повідомленням про виключення. Для браузерних клієнтів існує подання помилок "whitelabel", яке візуалізує ті ж дані у форматі HTML (щоб налаштувати його, додають `View`, що дозволяє `error`).

Зокрема, ***Java Servlet API*** – стандартизований API, призначений для реалізації на сервері та роботі з клієнтом за схемою запит-відповідь.

Сервлет – це клас, який вміє отримувати запити від клієнта та повертати йому відповіді. Так, сервлети Java – саме ті елементи, за допомогою яких будується клієнт-серверна архітектура.

Існує низка властивостей `server.error`, які можна встановити, якщо потрібно персоналізувати налаштування логіки обробки помилок за замовчуванням. Щоб повністю замінити логіку роботи за замовчуванням, можна реалізувати `ErrorController` та зареєструвати визначення біну цього типу, або додати бін типу `ErrorAttributes`, щоб використовувати існуючий механізм, але замінити вміст. Окрім цього, можна визначити клас, анотований `@ControllerAdvice`, щоб налаштувати JSON-документ, що повертається, під певний контролер і/або тип виключення.

Отже, обробка виключень в ***Java*** – це процес обробки і відповіді на виключення (помилки), які можуть виникати під час виконання програми. Виключення в Java використовуються для представлення непередбачуваних ситуацій або помилок, які виникають під час виконання програми. Для обробки виключень в Spring Boot потрібно створити клас, який буде являтися глобальним обробником виключень в додатку.

Клас для обробки виключень в Spring Boot відмічений анотацією `@RestControllerAdvice`. Над методами обробки виключень використовуються анотації `@ResponseStatus` і `@ExceptionHandler`.

`@ResponseStatus` – ця анотація використовується для вказання HTTP-статусу відповіді, який повинен бути відправлений клієнту в разі виникнення певного виключення.

@ExceptionHandler – ця анотація використовується для позначення метода, який буде викликатися для обробки певного типу виключення.

Ось невеликий приклад коду, із цими анотаціями:

```
@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
}
```

Якщо виникає виключення типу *UserNotFoundException*, буде відправлена відповідь зі статусом 404 Not Found і текстом "*User is not found* ". Лістинг обробників всіх виключень нашого ПЗ наведений в Додатку В.

Ось декілька варіантів результату виконання програми з використанням не коректних даних:

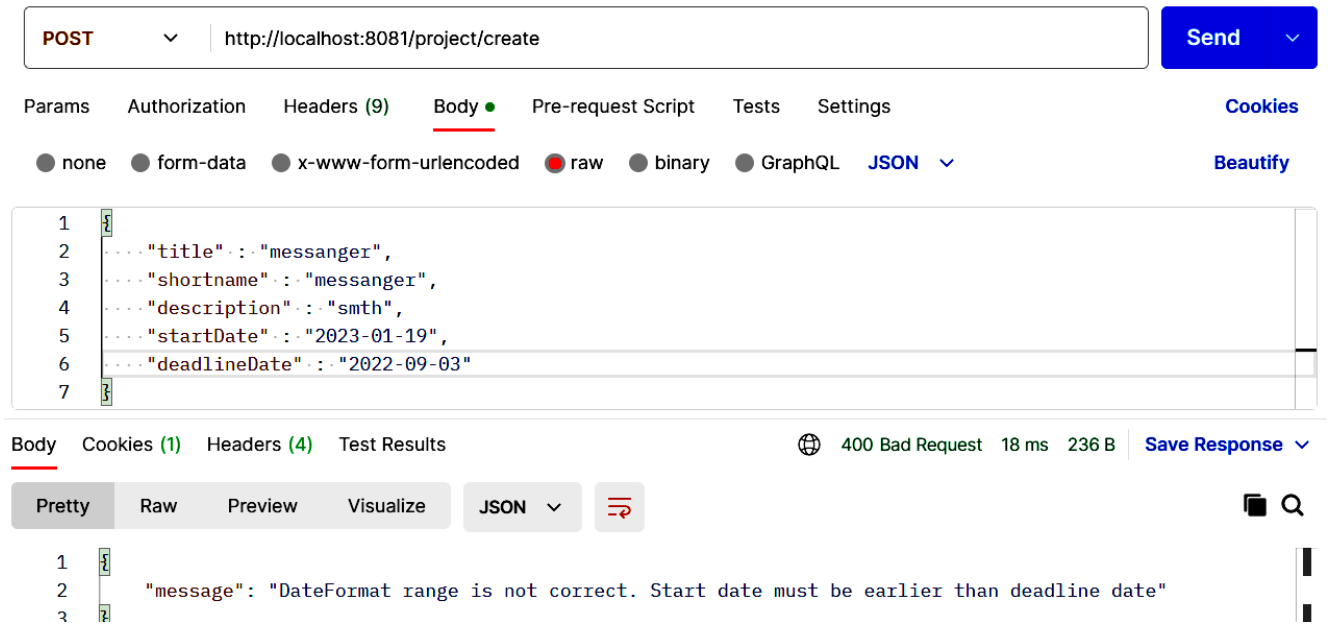

Рис. 4.3 – HTTP-запит та відповідь сервера щодо неправильного проміжку часу розробки програми

Тут зображений варіант HTTP-запиту, де дата початку розробки проекту (“startDate”) пізніше аніж дата закінчення (“deadlineDate”), що є неправильно.

Наступний варіант виключення у HTTP-запиті демонструє помилку із вже існуючою на сервері електронною поштою (рис. 4.4).

Інший варіант помилки в якому наведений HTTP-запит, що викликає Спринт за унікальним ідентифікатором проекту, якого не існує на сервері (рис. 4.5).

Рис. 4.4 – HTTP-запит та відповідь сервера щодо помилки дублювання електронної пошти

Рис. 4.5 – HTTP-запит та відповідь сервера щодо неіснуючого проекту

Окрім того, запропоноване ПЗ можна розширити аутентифікацією користувачів, можливість надавати коментарі в усіх компонентах, для якісної співпраці команди, а також звітність для ефективного прийняття рішень. Реалізація системи колективного управління IT-проектами у вигляді окремого API забезпечить розробникам зручний спосіб використання всіх необхідних функцій системи, спростить інтеграцію з існуючими проектами, забезпечить гнучкість у використанні та прискорює процес розробки тощо.

Слід також додати, що реалізація системи на базі Spring Boot дозволяє отримати багато переваг, таких як швидкий розгортання, гнучкість налаштувань, підтримка інтеграції з іншими технологіями та простота розробки. Використання контролерів, сервісів та репозиторіїв дозволяє досягти розподіленої архітектури та чіткої розділеності відповідальностей між компонентами системи.

Також у запропонованій системі реалізована основна функціональність системи управління IT-проектами, така як створення та відстеження Тікетів, Спринтів та Епіків, а також управління користувачами. Описані компоненти системи дозволяють зручно та ефективно використовувати її функціонал у реальних проектах.

РОЗДІЛ 5
ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

5.1. Розробка логіко-імітаційної моделі виникнення травм і аварій

Методикою оцінки рівня небезпеки робочих місць, машин, виробничих процесів та окремих виробництв передбачено пошук об'єктивного критерію рівня небезпеки для конкретного об'єкта [7]. Таким показником вибрана ймовірність виникнення аварії, травми залежно від явища, що досліджується.

Для побудови логіко-імітаційної моделі процесу, формування і виникнення аварії та травми в процесі створення мікрокліматичних умов у приміщенні оцінюють відповідні небезпечні події. Кожній із них присвоїмо ймовірність виникнення:

Шифр	Назва події	Ймовірність
P_1	Відсутність захисного заземлення	0,02
P_2	Пошкодження захисного заземлення	0,04
P_3	Спрацювання складових захисту	0,1
P_4	Неправильна експлуатація захисту	0,02
P_5	Відсутність профілактичних заходів	0,2
P_6	Відсутність захисного щита	0,12
P_7	Недотримання правил вибору взуття	0,15
P_8	Незнання правил техніки безпеки	0,1
P_9	Відсутність засобів індивідуального захисту	0,2
P_{10}	Легковажність	0,08

На основі наведених подій будуємо матрицю логічних взаємозв'язків між окремими пунктами, графічна інтерпретація якої зображено на рис. 5.1.

Розрахуємо ймовірності виникнення подій, що формують логіко-імітаційну модель процесів створення мікрокліматичних умов. Розглянемо травмонебезпечну ситуацію, що виникає за умови роботи працівників із

електронебезпекою.

Підставивши дані ймовірностей базових подій у формулу, отримаємо ймовірність події 13: $P_{13} = 0{,}2 + 0{,}4 - 0{,}2 \cdot 0{,}4 = 0{,}0592$.

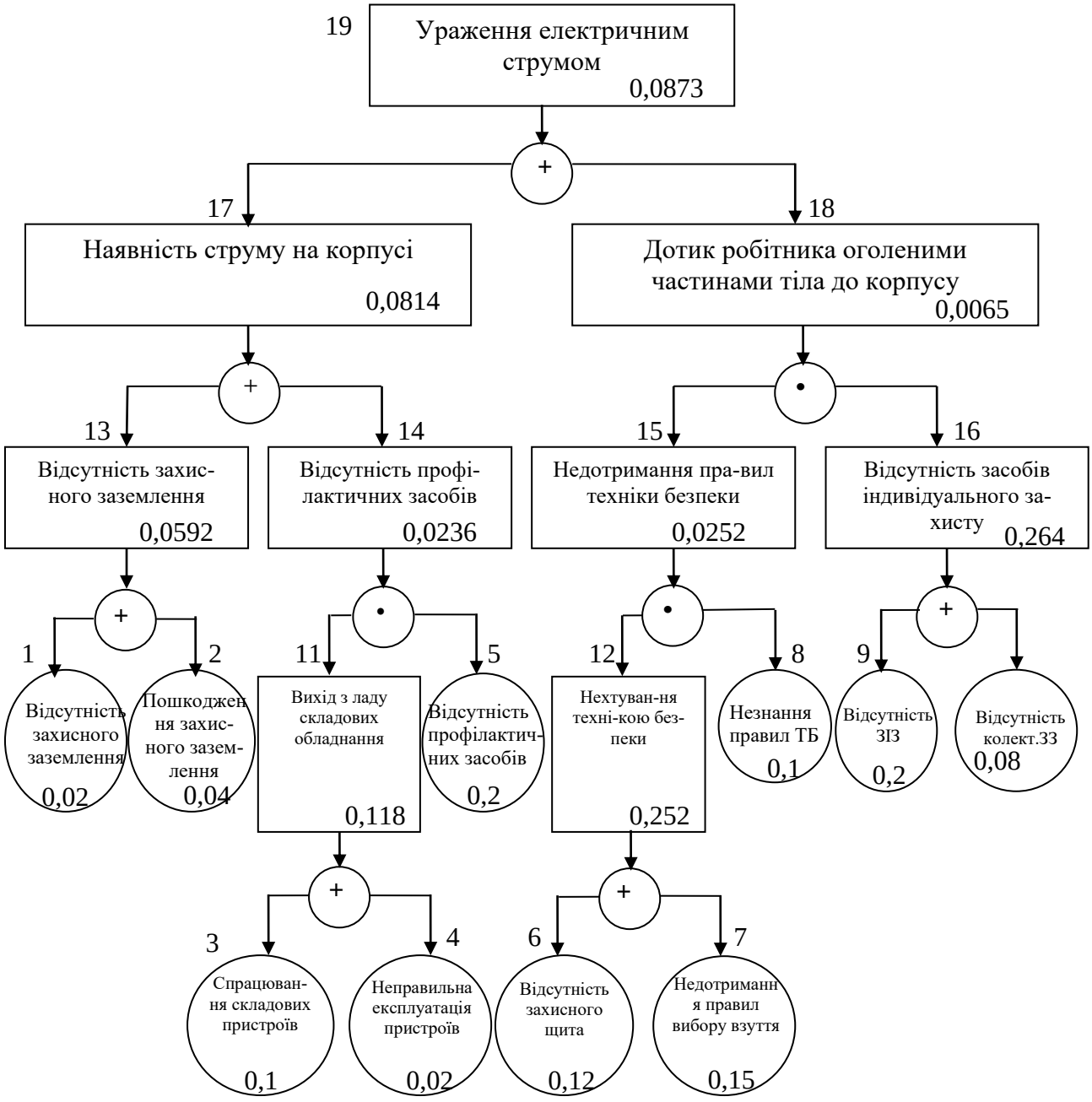

Рис. 5.1. Матриця логічних взаємозв’язків між окремими подіями травмонебезпечної ситуації [7]

Аналогічно визначаємо ймовірність інших подій:

$$P_{11} = P_4 + P_5 - P_4P_5 = 0{,}3 + 0{,}4 - 0{,}3 \cdot 0{,}4 = 0{,}118.$$
$$P_{12} = P_6 + P_7 - P_6P_7 = 0{,}3 + 0{,}5 - 0{,}3 \cdot 0{,}5 = 0{,}252.$$
$$P_{16} = P_9 + P_{10} - P_9P_{10} = 0{,}2 + 0{,}15 - 0{,}2 \cdot 0{,}15 = 0{,}264.$$

$$P_{14} = P_{11} \cdot P_5 = 0{,}118 \cdot 0{,}2 = 0{,}0236.$$
$$P_{15} = P_{12} \cdot P_8 = 0{,}252 \cdot 0{,}1 = 0{,}0252.$$
$$P_{17} = P_{13} + P_{14} - P_{13} \cdot P_{14} = 0{,}592 + 0{,}0236 - 0{,}0592 \cdot 0{,}0236 = 0{,}0814.$$
$$P_{18} = P_{15} \cdot P_{16} = 0{,}264 \cdot 0{,}0252 = 0{,}0065.$$
$$P_{19} = P_{17} + P_{18} - P_{17} \cdot P_{18} = 0{,}0065 + 0{,}0814 - 0{,}0065 \cdot 0{,}0814 = 0{,}0873.$$

Таким чином, ймовірність перекидання машини та наслідкового виникнення травми працівника є досить мала і становить – $P_{19} = 0,0873$.

5.2. Планування заходів із покращення умов праці

До заходів щодо покращення умов праці належать всі види діяльності, спрямовані на попередження, нейтралізацію або зменшення негативної дії шкідливих і небезпечних виробничих факторів на працівників.

Рівень умов праці оцінюють порівнянням за фактичними і нормативними значеннями узагальнених (групових) показників.

Заходи щодо поліпшення умов праці здійснюють з метою створення безпечних умов праці шляхом:

- доведення до нормативного рівня показників виробничого середовища за елементами умов праці;
- захисту працівників від дії небезпечних і шкідливих виробничих факторів.

До показників ефективності заходів щодо поліпшення умов праці належать:

а) зміни стану умов праці:

- зміна кількості засобів виробництва, приведених у відповідність до вимог стандартів безпеки праці;
- покращання санітарно-гігієнічних показників;
- покращання психофізичних показників, зменшення фізичних і нервово-психічних навантажень, в т.ч. монотонних умов праці;

б) соціальні результати заходів:

- збільшення кількості робочих місць, що відповідають нормативним

вимогам;

- зниження рівня виробничого травматизму;
- престиж та задоволення працею.

Отже, на покращення охорони праці потрібно виділити кошти на відновлення вентиляційних систем у ремонтних майстернях, естетично оформити приміщення офісу, відновити кабінет з охорони праці, поновити протипожежний інвентар.

5.3. Безпека в надзвичайних ситуаціях

Актуальність проблеми природно-техногенної безпеки для населення і території, зумовлена зростанням втрат людей, що спричиняється небезпечними природними явищами, промисловими аваріями та катастрофами [7]. У системі цивільної оборони окремого господарства необхідно забезпечити захист населення таким чином:

Укриття в захисних спорудах, якому підлягає усе населення відповідно до приналежності, досягається створенням фонду захисних споруд.

Евакуаційні заходи, які проводяться в містах та інших населених пунктах, які мають об’єкти підвищеної небезпеки, а також у воєнний час, основним способом захисту населення є евакуація і розміщення його у позаміській зоні.

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

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 І РЕКОМЕНДАЦІЇ

Для реалізації системи командної розробки та управління ІТ-проектами обрано фреймворк Spring Boot, який забезпечує швидку розробку та простоту в налаштуванні. Для коректного функціонування ПЗ розроблено контролери, які взаємодіють із клієнтськими додатками та обробляють HTTP-запити, репозиторії для взаємодії з базою даних, сервіси для логіки бізнес-процесів та обробники помилок для ефективного вирішення виняткових ситуацій.

Реалізація системи на базі Spring Boot дозволяє отримати переваги – швидкий розгортання, гнучкість налаштувань, підтримка інтеграції з іншими технологіями та простота розробки. Використання контролерів, сервісів та репозиторіїв дозволяє досягти розподіленої архітектури та чіткої розділеності відповідальностей між компонентами системи.

Врахована основна функціональність системи управління ІТ-проектами, така як створення та відстеження Тікетів, Спринтів та Епіків, а також управління користувачами. Описані компоненти системи дозволяють зручно та ефективно використовувати її функціонал у реальних ІТ-проектах.

Запропоноване ПЗ має свої перспективи до розвитку, зокрема, в плані додавання можливості аутентифікації користувачів, коментарів в усіх компонентах, для якісної співпраці команди, а також звітність для ефективного прийняття рішень тощо.

Реалізація системи управління ІТ-проектами у вигляді окремого API забезпечить розробникам зручний спосіб використання всіх необхідних функцій системи, спростить інтеграцію з існуючими проектами, забезпечить гнучкість у використанні та прискорює процес розробки.

Аналіз існуючих систем управління ІТ-проектами вказує на те, що багато із них мають складний користувацький інтерфейс та обмежену функціональність, що ускладнює їх використання для розробників. Враховуючи це, нами прийнято рішення реалізувати систему управління ІТ-проектами у вигляді API, яке забезпечує просту і зручну співпрацю розробників з функціоналом системи без

необхідності працювати зі складним інтерфейсом та обмеженою функціональністю існуючих систем.

Розширення функціоналу запропонованого ПЗ до API, дасть змогу надавати розробникам можливість легко підключати та використовувати необхідні функції системи у своїх проектах через API. Це спростить процес розробки, оскільки розробники можуть вибрати лише необхідні компоненти API і використовувати їх для реалізації своїх проектів, уникнувши непотрібного навантаження та зайвого коду. Крім того, такий підхід полегшує обмін даними між різними системами управління проектами та забезпечує їх взаємодію без проблем.

Запропонована система має ознаки інноваційності – дозволяє розробникам працювати із функціоналом управління IT-проектами без зайвих труднощів та витрат часу. Вона забезпечуватиме ефективне управління IT-проектами з використанням Agile підходу, дозволяючи командам розробників працювати швидше, гнучкіше та результативніше.

Розроблена UML-діаграма класів відображає структуру системи управління IT-проектами. Її реалізація передує процесу створення Spring Boot проекту, реалізації контролерів, репозиторіїв, сервісів та обробників помилок. Всі ці компоненти взаємодіють між собою та забезпечують високу якість та ефективність системи командного управління IT-проектами.

Загалом, реалізація системи командного управління IT-проектами на Java з використанням Agile методологій є важливим внеском у галузь розробки ПЗ. Вона демонструє, що шлях до успішного управління IT-проектами може бути спрощений та оптимізований за допомогою сучасних інструментів та технологій.

БІБЛІОГРАФІЧНИЙ СПИСОК

1. Застосування архітектурного проектування в гнучких методах розробки програмних продуктів / Харченко О.Г., Боднарчук І.О., Галай І.О., Лісовий В. // Інженерія програмного забезпечення. 2012. № 3-4 (11-12). С. 5-11.

2. Комплексна система IT-рішень для управління агробізнесом. URL: https://agrichain.com.ua/ (дата звернення: 20.11.2024).

3. Кузьміних В.О., Коваль О.В., Воронько М.П. Оцінка часу виконання типових задач проектів на підприємствах з функціональною організаційною структурою // Реєстрація, зберігання і обробка даних, ISSN 1560-9189, 2012 т. 14, № 3, с.77-82.

4. Лавріщева К.М. Програмна інженерія. / К.М. Лавріщева. К.: Видавничий дім "Академперіодика", 2008. 319 с.

5. Лехман С.Д. та ін. Запобігання аварійності і травматизму у сільському господарстві / С.Д. Лехман, В.І. Рубльов, Б.І. Рябцев. К.: Урожай, 1993. 272 с.

6. Основи управління IT проектами: Навч. посіб. для студ. спеціальності 122 «Комп’ютерні науки» / В. О. Кузьміних, Р. А. Тараненко. Київ: КПІ ім. Ігоря Сікорського, 2019. 75 с.

7. Підходи до управління програмними проектами у SWEBOK V3. URL: https://www.researchgate.net/publication/316493834 (дата звернення: 10.11.2024).

8. Agile Development at Scale: The Next Frontier. URL: https://www.researchgate.net/Agile_Development_at_Scale (дата звернення 05.10.2024)

9. Agile manifesto. URL: http://agilemanifesto.org (дата звернення 10.10.2024)

10. Classical Waterfall Model. URL: https://www.geeksforgeeks.org/software-engineering*classical-waterfall-model (дата звернення 29.10.2024)

11. Extreme programming. URL: http://www.xprogramming.com/xpmag/whatisxp.htm (дата звернення 17.10.2024).

12. Henrik Kniberg. Scrum and XP from the trenches. — C4Media, 2007. — C. 140. — ISBN 978-1-4303-2264-1.

13. Iterative Model: What Is It And When Should You Use It? URL: https://airbrake.io/iterative-model (дата звернення 25.10.2024)

14. Kanban. URL: https://www.agilealliance.org/kanban (дата звернення 18.10.2024)

15. Key Lessons From Tailoring Agile Methods for Large-Scale Software Development. URL: https://www.researchgate.net/publication/Key_Lessons_Agile_Methods_for_Large-Scale_Software_Development (дата звернення 05.10.2024)

16. Managing Successful Projects with PRINCE2 (2009 Edition). The Stationery Office/Tso; 2009th edition, 2009. 327.

17. Postman – що це за інструмент і які його основні функції? 2024. URL: https://foxminded.ua/postman/ (дата звернення 25.11.2024)

18. Saaty T. Decision Making with the Analytic Network Process./ Saaty T. Vargas L.// N.Y.: Springer, 2006. 278 p.

19. Scrum. URL: http://www.mountaingoatsoftware.com/scrum (дата звернення 15.10.2024)

20. SDLC - Waterfall Model. URL: https://www.tutorialspoint.com/waterfall_model (дата звернення 28.10.2024)

21. Spring Boot. URL: https://spring.io/projects/spring-boot (дата звернення 28.11.2024)

22. Spring Security: How it works internally. URL: https://blog.knoldus.com/spring-security-internal/ (дата звернення 28.11.2024)

23. Software Engineering. SDLC V-Model. URL: https://www.geeksforgeeks.org/software-engineering-sdlc-v- model (дата звернення 30.10.2024)

24. Software Extension to the PMBOK® Guide Fifth Edition. Project Management Institute. Publ., 2013. 240 p.

25. Spring Initializr – веб-інструмент для генерування базової структури проекту в Spring Boot. URL: https://start.spring.io/. (дата звернення 30.10.2024)

26. UML Diagram Types Guide: Learn About All Types of UML Diagrams with Examples. 2023. URL: https://creately.com/blog/diagrams/uml-diagram-types-examples/#ClassDiagram (дата звернення 28.11.2024)

27. V-Model. URL: https://www.tutorialspoint.com/sdlc/v_model (дата звернення 30.10.2024)

28. What is Agile Kanban Methodology? URL: https://www.inflectra.com/methodologies/kanban (дата звернення 26.10.2024)

29. What is MySQL? URL: https://www.oracle.com/mysql/what-is-mysql/ (дата звернення 28.11.2024)

ДОДАТКИ

Додаток А

Фрагмент коду головних файлів програми - 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>
      <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
      </dependency>
      <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.33</version>
      </dependency>
      <dependency>
            <groupId>javax.validation</groupId>
            <artifactId>validation-api</artifactId>
            <version>2.0.1.Final</version>
      </dependency>
      <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-validator</artifactId>
```

```
            <version>6.0.10.Final</version>
      </dependency>
      <dependency>
            <groupId>com.google.guava</groupId>
            <artifactId>guava</artifactId>
            <version>12.0</version>
      </dependency>
      <dependency>
            <groupId>org.modelmapper</groupId>
            <artifactId>modelmapper</artifactId>
            <version>3.1.1</version>
      </dependency>
      </dependencies>
      <build>
      <plugins>
      <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
      </plugins>
      </build>
</project>
```

Лістинг файлу ProjectModel.java

```
@AllArgsConstructor
@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;
}
```

Лістинг файлу SprintModel.java

```
@Getter
@Setter
@Entity
@Table(name = "sprint")
public class SprintModel {
@Id
@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;
}
```

Лістинг файлу EpicComtroller.java

```
@RestController
@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}") 65
```

Продовження додатку Г

```
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 epic){
return ResponseEntity.ok(epicService.update(epic));
}
}
```

Додаток Б.
SQL запит на створення таблиць в БД

```
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,
epic_description varchar(400),
epic_status varchar(255) NOT NULL,
priority varchar(255) NOT NULL,
created_time_stamp datetime NOT NULL,
last_updated_time_stamp datetime,
FOREIGN KEY (project_id) REFERENCES project(project_id),
FOREIGN KEY (user_id) REFERENCES users(user_id),
FOREIGN KEY (sprint_id) REFERENCES sprint(sprint_id)
);

CREATE TABLE IF NOT EXISTS ticket(
ticket_id int NOT NULL auto_increment PRIMARY KEY,
project_id int NOT NULL,
sprint_id int NOT NULL,
epic_id int NOT NULL,
reporter_id int NOT NULL,
assignee_id int,
```

```
title varchar(255) NOT NULL,
ticket_description varchar(400),
ticket_status varchar(255) NOT NULL,
ticket_type varchar(255) NOT NULL,
priority varchar(255) NOT NULL,
estimate int,
units_of_measure varchar(255),
created_time_stamp datetime NOT NULL,
last_updated_time_stamp datetime,
FOREIGN KEY (project_id) REFERENCES project(project_id),
FOREIGN KEY (assignee_id) REFERENCES users(user_id),
FOREIGN KEY (reporter_id) REFERENCES users(user_id),
FOREIGN KEY (sprint_id) REFERENCES sprint(sprint_id),
FOREIGN KEY (epic_id) REFERENCES epic(epic_id)
);

CREATE TABLE IF NOT EXISTS comments(
comment_id int NOT NULL auto_increment PRIMARY KEY,
user_id int NOT NULL,
ticket_id int NOT NULL,
comment_text varchar(400) NOT NULL,
created_time_stamp datetime NOT NULL,
FOREIGN KEY (user_id) REFERENCES users(user_id),
FOREIGN KEY (ticket_id) REFERENCES ticket(ticket_id)
)
```

Додаток В.
Обробка виключень *Exception*

Лістинг 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.class)
public ResponseEntity<Object> showUserExistence() {
ExceptionResponse response = new ExceptionResponse();
response.setMessage("User already exists for this email.");
return new ResponseEntity<>(response, HttpStatus.CONFLICT);
}

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

@ExceptionHandler(ProjectNotFoundException.class)
public ResponseEntity<Object> showProjectAbsence() {
ExceptionResponse response = new ExceptionResponse();
response.setMessage("Project is not found.");
return new ResponseEntity<>(response, HttpStatus.NOT_FOUND);
}

@ExceptionHandler(SprintNotFoundException.class)
public ResponseEntity<Object> showSprintAbsence() {
ExceptionResponse response = new ExceptionResponse();
response.setMessage("Sprint is not found.");
return new ResponseEntity<>(response, HttpStatus.NOT_FOUND);
}

@ExceptionHandler(EpicNotFoundException.class)
public ResponseEntity<Object> showEpicAbsence() {
ExceptionResponse response = new ExceptionResponse();
response.setMessage("Epic is not found.");
return new ResponseEntity<>(response, HttpStatus.NOT_FOUND);
}

@ExceptionHandler(TicketNotFoundException.class)
public ResponseEntity<Object> showTicketAbsence() {
ExceptionResponse response = new ExceptionResponse();
response.setMessage("Ticket is not found.");
return new ResponseEntity<>(response, HttpStatus.NOT_FOUND);
}
```

```
@ExceptionHandler(CommentNotFoundException.class)
public ResponseEntity<Object> showCommentAbsence() {
ExceptionResponse response = new ExceptionResponse();
response.setMessage("Comment is not found.");
return new ResponseEntity<>(response, HttpStatus.NOT_FOUND);
}

@ExceptionHandler(DateRangeNotCorrectException.class)
public ResponseEntity<Object> showBadRange() {
ExceptionResponse response = new ExceptionResponse();
response.setMessage("DateFormat range is not correct. Start date must be earlier
than deadline date");
return new ResponseEntity<>(response, HttpStatus.BAD_REQUEST);
}

@ExceptionHandler(UserRoleNotFoundException.class)
public ResponseEntity<Object> showBadUserRole() {
ExceptionResponse response = new ExceptionResponse();
response.setMessage("User role is not correct. There are such roles as
'DEVELOPER', 'QA_ENGINEER', 'ADMIN', 'PROJECT_MANAGER'");
return new ResponseEntity<>(response, HttpStatus.NOT_FOUND);
}

@ExceptionHandler(SprintPeriodIsNotCorrectException.class)
public ResponseEntity<Object> showBadSprintPeriod() {
ExceptionResponse response = new ExceptionResponse();
response.setMessage("Sprint period is not correct. It has to last from one to
four weeks.");
return new ResponseEntity<>(response, HttpStatus.BAD_REQUEST);
}
}
```