МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК ТА ТЕХНОЛОГІЙ

Кафедра Комп'ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри
Аліна САВЧЕНКО
«_____» _____ 2023р.

КВАЛІФІКАЦІЙНА РОБОТА

(ДИПЛОМНИЙ ПРОЄКТ, ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ **"БАКАЛАВР"**
ЗА ОСВІТНЬО-ПРОФЕСІЙНОЮ ПРОГРАМОЮ
"ІНФОРМАЦІЙНІ УПРАВЛЯЮЧІ СИСТЕМИ ТА ТЕХНОЛОГІЇ"

Тема: "Система управління проектами на Java з використанням Agile методологій**"**

Виконавець: студентка групи УС-411Б Волошина Юлія Валеріївна

Керівник: к.т.н., доцент Віноградов Микола Анатолійович

Нормоконтролер: Олександр ШЕВЧЕНКО

Київ – 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних наук та технологій*

Кафедра *Комп'ютерних інформаційних технологій*

Галузь знань, спеціальність, освітньо-професійна програма: 12 "Інформаційні технології", 122 "Комп'ютерні науки", "Інформаційні управляючі системи та технології"

ЗАТВЕРДЖУЮ
Завідувач випускової кафедри
_____ Аліна САВЧЕНКО
«____»_____2023р.

ЗАВДАННЯ

на виконання дипломного проекту студента

Волошиної Юлії Валеріївни

(прізвище, ім'я, по батькові)

1. **Тема роботи:** «Система управління проектами на Java з використанням Agile методологій» затверджена наказом ректора № 623/ст. від 01.05.2023р.
2. **Термін виконання роботи:** з 15.05.2023р. по 25.06.2023р.
3. **Вихідні дані до роботи:** інтегрована середа розробки InteliJ IDEA, база даних MySQL, інструмент керування HTTP-запитами Postman.
4. **Зміст пояснювальної записки (перелік питань, що підлягають розробці):** дослідження Agile методологій, аналіз існуючих систем управління проектами, та оцінка іх переваг та недоліків. Реалізація власної системи управління проектами з представленням ідей покращення застосунку в майбутньому.
5. **Перелік обов'язкового ілюстративного матеріалу:** рисунки етапів життєвого циклу методологій, рисунок UML діаграми класів.

6. Календарний план-графік

№ з/п	Завдання	Термін виконання	Підпис керівника
1.	Аналіз предметної області дослідження	15.05.2023–16.05.2023	
2	Збір та аналіз інформації за темою дипломного проекту	17.05.2023–19.05.2023	
3	Створення проекту та реалізація основних класів та інтерфейсів.	20.05.2023–23.05.2023	
4	Визначення основної функціональності та її впровадження	24.05.2023–28.05.2023	
5	Перевірка на правильне функціонування програми	29.05.2023	
6	Написання пояснювальної записки дипломного проекту	30.05.2023–31.05.2023	
7	Підготовка доповіді та демонстраційного матеріалу	01.06.2023 – 18.06.2023	

Дата видачі завдання: «15» травня 2023 р.

Керівник дипломного проекту _____ Микола ВІНОГРАДОВ

(підпис керівника)

Завдання прийняв до виконання _____ Юлія ВОЛОШИНА

(підпис випускника)

РЕФЕРАТ

Пояснювальна записка до дипломного проекту «Система управління проектами на Java з використанням Agile методологій» викладена на 79 сторінках, містить 19 рисунків та 16 літературно-наукових джерел.

Об’єкт дослідження. Вивчення та розробка системи управління проектами з використанням гнучких методологій.

Мета і завдання. Метою роботи є розробка системи управління проектами на Java з використанням Agile методологій на основі аналізу вже існуючих систем.

Методи дослідження. В роботі досліджені наступні методи: аналіз наукової літератури з питань Agile методологій та управління проектами, дослідження сучасних систем управління та розробка власної системи управління проектами на Java.

Результати. У роботі буде проведено дослідження Agile методологій, таких як Scrum, Kanban, Lean, та Extreme Programming, і визначено їх переваги та особливості. Розробка системи буде проведена на мові програмування Java, яка відома своєю надійністю та розширюваністю.

JAVA, AGILE МЕТОДОЛОГІЇ, AGILE, СИСТЕМА УПРАВЛІННЯ ПРОЕКТАМИ, УПРАВЛІННЯ ПРОЕТАМИ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ.

ЗМІСТ

ВСТУП

Системи управління проектами відіграють важливу роль у сучасному бізнесі та інформаційних технологіях. Швидкий темп розвитку технологій та зростання складності проектів ставлять нові виклики перед управлінцями та командами проектів. У цьому контексті Agile методології та системи управління проектами на Java набувають все більшої актуальності та значущості.

Актуальність теми полягає у зростаючій потребі управління проектами в умовах невизначеності, швидкого змінного середовища та постійно зростаючих очікувань з боку замовників. Agile методології стають необхідним інструментом для досягнення гнучкості та швидкості в процесі розробки та управління проектами. Вивчення та аналіз існуючих систем управління проектами на Java дозволить зрозуміти переваги та обмеження кожної з них, а також ідентифікувати можливі напрямки подальшого вдосконалення.

Мета даної дипломної роботи полягає в описі та аналізі Agile методологій та існуючих систем управління проектами, а також створення власної системи управління проектами з використанням власних ідей на основі проведеного аналізу. Використання Java в реалізації систем управління проектами дозволяє отримати потужні та гнучкі інструменти для розробки та розширення функціональності. Для досягнення поставленої мети необхідно вирішити такі завдання:

- Вивчити найбільш популярні різновиди методологій гнучкої розробки;
- Дослідити ринок систем управління проектами: їх переваги та недоліки;
- На основі аналізу існуючих систем, зробити висновок щодо їх функціональних можливостей та обмежень, та розробити власну систему керування проектами на мові Java з впровадженням власних ідей для полегшення роботи командам розробників;

- Спроектувати систему з орієнтацією на легку розширюваність функціональності;
- Задля високої якості та ефективності системи, використати сучасні технології та бібліотеки.

У майбутньому покращення систем управління проектами стануть ще більш важливими. Зростаюча складність проектів та необхідність швидкого реагування на зміни вимагають постійного розвитку та впровадження нових інструментів та підходів. Впровадження автоматизованих систем управління проектами дозволить забезпечити ефективну комунікацію, керування завданнями та ресурсами, а також контроль над процесом розробки.

Системи управління проектами на Java, які використовують Agile методології, є цінними інструментами для ефективного керування проектами. Вони допомагають забезпечити гнучкість та прозорість у процесі розробки, покращують комунікацію між учасниками проекту, сприяють вчасному виявленню проблем та ризиків. Крім того, такі системи дозволяють покращити планування, контроль за витратами та виконанням завдань, що сприяє досягненню більшої ефективності та якості проектів.

РОЗДІЛ 1

AGILE МЕТОДОЛОГІЇ ТА ЇХ ВИКОРИСТАННЯ В УПРАВЛІННІ ПРОЕТКАМИ

1.1. Огляд Agile методологій

Перші згадки про Agile розробку відносяться до 2001 року, коли спільнота розробників, які втомилися від використання методів розробки, що вважалися "важкими", - а саме, водоспадного моделю - вирішила створити маніфест: Agile Manifesto.

З'явлення Agile Manifesto у 2001 році відзначає народження Agile як методології. З того часу з'явилося безліч Agile-фреймворків, таких як Scrum, Kanban, Lean та Extreme Programming (XP), кожен з яких втілює основні принципи частої ітерації, постійного навчання та високої якості на свій спосіб. Scrum та XP найбільш поширені серед команд розробників програмного забезпечення, тоді як Kanban користується популярністю серед команд, що надають сервіси, таких як IT чи людські ресурси.

Сьогодні багато Agile-команд поєднують практики з кількох різних фреймворків, додавши унікальні для своєї команди практики. Деякі команди використовують деякі Agile-ритуали (такі як щоденні збори, ретроспективи, беклоги тощо), тоді як інші створюють нові Agile-практики.

Agile методології стверджують, що головна мета полягає в задоволенні клієнта завдяки постійній поставці програмного забезпечення, яке додає цінності, за допомогою постійного зв'язку з клієнтом і фокусуванням на комунікації між членами команди.

На відміну від попередніх практик, Agile методологія не характеризується повною визначеністю продукту, а скоріше "крок за кроком" - повним аналізом або визначенням усіх категорій/вимог, динамічною взаємодією, що дозволяє постійну поставку - фокусуючись на "навколо-бережних" можливостях, але ніколи не втрачаючи довгострокової мети продукту [1].

Основною перевагою використання методологій Agile є не тільки швидка доставка програмного забезпечення, але й постійна доставка "вартості" клієнту, оскільки доставки є інкрементальними. Agile методи адаптивні, що дозволяє швидко приймати рішення та миттєво впливати на розвиток бізнесу.

Вважається, що Agile-команди майбутнього будуть оцінювати свою ефективність більше, ніж дотримання доктрини. Відкритість, довіра та автономія стають культурною валютою для компаній, які хочуть привернути найкращих працівників та отримати від них максимальну користь.

1.2. Основні Agile методології: переваги та недоліки

Існує безліч методологій, що дотримуються цієї концепції Agile. У цьому підрозділі ми виділяємо чотири основні методології Agile і їх переваги та недоліки в сфері розробки програмного забезпечення.

1.2.1. Scrum-методологія

Scrum є найбільш популярною рамкою, що лежить в основі методології Agile, і характеризується циклами або етапами розробки, відомими як спринти. Основною метою Scrum є максимізація часу розробки програмного продукту для досягнення більшої цільової мети - Продуктової Мети, досягнення якої забезпечується під час кожного спринту [2].

Спринт – період часу, зазвичай тривалістю від 1 до 4 тижнів, протягом якого, команда розробляє функціонально відповідні та готові до використання шматки

програмного забезпечення, що можуть бути випущені виробником програмного забезпечення або включені до продукту [3].

Візуалізація циклу спринту представлена на рис. 1.1.

Планування спринту – на цьому етапі команда визначає мету, та завдання спринту, розглядає потенційні ризики та складнощі, визначає обсяг роботи, який вона здатна виконати за час спринту, та створює план дій на наступний спринт.

Щоденна зустріч команди – щоденний етап(зазвичай займає 15 хвилин), під час якого команда обговорює свій прогрес, плани та можливі проблеми, які виникли під час розробки продукту.

Огляд спринту – щотижневий етап, під час якого команда презентує результати своєї роботи за останній спринт замовнику, а також обговорює які завдання були виконані та які не були виконані.

Ретроспектива спринту – етап, що відбувається по завершенню кожного спринту і призначений для огляду та аналізу роботи команди під час спринту [3].

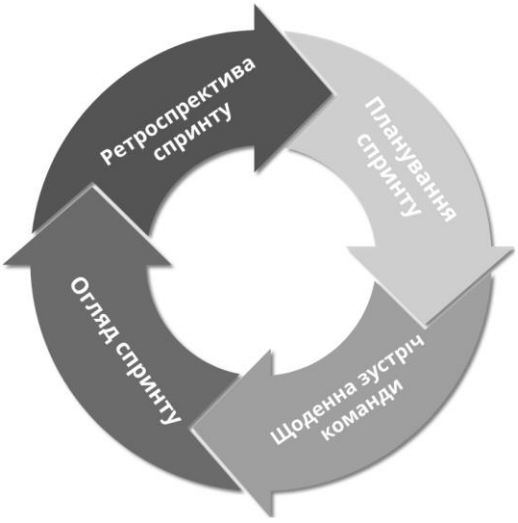

Рис. 1.1. Цикл спринту.

Перевагами Скраму є:

- Висока мотивація команди, оскільки програмісти прагнуть дотримуватись термінів кожного спринту;

- Прозорість, що дозволяє всім членам команди або організації стежити за проектом; використання простого "визначення готовності" для перевірки вимог;
- Постійний фокус на якості забезпечує менше помилок; динаміка методу дозволяє розробникам перепланувати пріоритети, забезпечуючи більше уваги незавершеним спринтам;

Недоліки:

- Сегментація проекту та пошук гнучкості розробки може привести до втрати командою загального уявлення про проект фокусуючись на одній частині;
- Роль кожного розробника може бути не чітко визначеною, що призводить до деякої плутанини серед членів команди.

1.2.2. Kanban-методологія

Слово "Канбан" має японське походження і пов'язане з ідеєю "точно вчасно". На практиці, метод Канбан використовує дошку або таблицю (Канбан-дошка див. рис. 1.2), розділену на стовпці, які відображають кожен потік у проекті розробки програмного забезпечення. Під час розвитку проекту інформація в таблиці змінюється, і кожного разу, коли з'являється нове завдання, створюється нова "картка" [2].

Рис. 1.2. Kanban-дошка.

Переваги:

- Можливість перегляду всіх завдань в рамках одного проекту (за статусами «Завершено», «В процесі» або «В тестуванні», наприклад), використовуючи просту концепцію «Карток»;
- Можливість обмеження кількості одночасно виконуваних завдань (тобто, обсягу роботи з урахуванням його вирішення або доставки);
- Фокусування на тривалості циклу - скільки часу займає завданню перейти від Backlog до заключної стадії;
- Дозволяє здійснювати безперервну доставку;
- Ймовірно, одна з найпростіших методологій для впровадження поза «IT-світом».

Недоліки:

- Є можливість того, що члени команди неправильно трактують інформацію, що відображається на Канбан дошці, особливо, якщо вона застаріла;
- Оскільки в Канбан немає термінових рамок, можна стикнутися з проблемами, пов'язаними з часом, такими як затримки, на кожному етапі.

1.2.3. Lean розробка

Lean розробка є методологією, яка безпосередньо випливає з Lean виробництва, створеного компанією Toyota, і застосовується до розробки програмного забезпечення. Цей метод пропонує концептуальну рамку і дотримується цінностей, принципів та добрих практик розробки, які можна застосовувати до Agile методології розробки. На рис. 1.3. представлена схема методології Lean розробки.

Lean розробка змушує команду безжалісно видаляти будь-яку діяльність, яка не приносить остаточної цінності продукту.

Є сім основних принципів:

- видалення непотрібних речей (все, що не приносить ефективної цінності проекту для клієнта);

- розробка якості (створення якості у розробці потребує дисципліни та контролю кількості залишків);
- створення знань (команда мотивується документувати всю інфраструктуру для подальшого збереження цінності);
- відмінні зобов'язання (ця точка підтримує команду, щоб не фокусуватись занадто сильно на плануванні та передбаченні ідей без повного розуміння вимог бізнесу);
- швидка доставка (отримання цінності для клієнта якомога швидше);
- повага до команди (комунікація та управління конфліктами є двома важливими пунктами);
- оптимізація всього (послідовність розробки повинна бути досконалою, щоб можна було видаляти помилки у коді, створюючи потік справжньої цінності) [2].

Рис. 1.3. Схема Lean розробки.

Переваги:

- Дозволяє команді видаляти зайві дії, що зберігає час та гроші;

- Зменшує час, необхідний для впровадження функціональностей, оскільки під час процесу прийняття рішень підготовлює команду розробників, що збільшує загальний рівень мотивації;
- Легко масштабована методологія, яка легко адаптується до проектів будь-якого розміру;
- Не надмірно розробляє рішення або бізнес-вимоги.

Недоліки:

- Залежить від здатності розробницької команди і дотримання "Принципів Lean", що вимагає надзвичайно затребуваних та талановитих розробників;
- Легше втратити фокус, оскільки різні задачі розбиваються на кілька елементів;
- Вимагає деяких документів, зокрема щодо характеристик бізнесу, що є предметом роботи. В іншому випадку існує ризик того, що розробка буде проведена неправильно та міститиме помилки.

1.2.4. Extreme Programming (XP)

Extreme Programming - це Agile-фреймворк розробки, придуманий Кентом Беком, який може бути адаптований до компаній різного розміру. Його методологія базується на концепції виявлення "найпростішої речі, яка працює" та надає перевагу задоволенню клієнта. У XP великий акцент зроблений на цінностях комунікації, простоти, зворотного зв'язку, відваги та поваги. Ця методологія сприяє довірі та мотивує розробників приймати зміни в вимогах клієнта. Важливість командної роботи в XP полягає в тому, що коли виникає проблема, вона вирішується всією командою. В XP програмне забезпечення тестується з першого дня, збираючи зворотний зв'язок для поліпшення розробки. XP сприяє таким активностям, як парне програмування, і є чудовою інженерною методологією з сильною складовою тестування [2].

Візуалізація роботи Extreme Programming представлена на рис 1.4.

Рис. 1.4. Extreme Programming.

Переваги:

- Написаний код в XP є простим і легко піддається вдосконаленню, дозволяючи здійснювати постійну вдосконалення програмного продукту.
- Розробний цикл XP видимий, забезпечуючи чіткі цілі для розробників та дозволяючи досягати досить швидких результатів.
- Постійне тестування в XP робить процес розробки програмного забезпечення більш гнучким.
- XP сприяє енергійному способу роботи.
- XP сприяє підвищенню та підтримці талановитості команди.

Недоліки:

- Екстремальна увага на код у XP може призвести до меншої уваги до дизайну, що може потребувати додаткової уваги пізніше.
- XP може не працювати оптимально, якщо члени команди не знаходяться в одному географічному регіоні.
- У проектах XP не завжди зберігається реєстр можливих помилок, що може призвести до появи подібних помилок у майбутньому через відсутність моніторингу.

1.3. Ролі в Agile методологіях

Agile-команда - це самостійно керована команда, яка може швидко адаптуватися до змін. Команда складається з осіб з різноманітними навичками, які працюють разом для виконання історій користувачів протягом спринту.

Product owner (Власник продукту) є представником користувачів та відповідає за створення візії продукту та написання історій користувачів. Він також оновлює список пріоритетів історій користувачів та спілкується з командою, щоб забезпечити зрозуміння візії та визначеної вартості для користувачів. На зустрічах планування спринту він допомагає команді зрозуміти, які історії користувачів потрібно виконувати на наступному спринті та яку бізнес-вартість вони мають.

Scrum Master (Scrum майстер) допомагає команді у досягненні успіху в доставці вартості, визначеної користувачем. Замість того, щоб бути керівником і контролювати команду, Scrum майстер повинен бути лідером-слугою. Вони не вказують команді, що робити, а допомагають команді зберігати фокус, постійно дотримуючись кращих Agile практик, та проводити багато зустрічей, на яких команда бере участь. Крім того, Scrum майстер співпрацює з Власником продукту, щоб зрозуміти потреби користувачів, що можуть змінюватися. Хоча Власник продукту безпосередньо фокусується на потребах користувачів, Scrum майстер зосереджується більше на задоволенні потреб команди.

Team members (Члени команди) виконують фактичну роботу зі створення цінності. У команді враховуються всі технічні навички, необхідні для розробки рішення. Високо бажана комбінація глибокої технічної експертизи спеціалістів разом з людьми з більш широким спектром навичок. По-перше, це дозволяє команді уникнути одної точки відмови (коли тільки одна людина знає, як виконати необхідну задачу). По-друге, команди з крос-функціональним складом можуть більш готово адаптуватися до швидкого розвитку та тестування, необхідних для кількох обмежених історій користувача, які команда повинна завершити протягом 2-4-х тижневого спринту.

Agile Coach (Agile тренер) - остання роль, яка технічно не належить до команди, але може тимчасово підтримувати інші ролі. Agile тренер зазвичай долучається до менш досвідченої Agile команди (або до тієї, яка має проблеми з дотриманням найкращих Agile практик), щоб підвищити розуміння та результативність команди. Agile тренер не тільки повинен мати дуже глибоке розуміння принципів Agile, але також широкий досвід роботи з Agile буде дуже корисним, щоб він міг швидко ідентифікувати наявні проблеми на основі міцного минулого досвіду, а потім допомагати Scrum майстру та команді реалізувати життєздатні рішення. Важливо зазначити, що Agile тренер не очолює команду або не замінює Scrum майстра. Однак він повинен бути здатен звернутися до Scrum майстра і професійно, але стверджувально, направити його, якщо той починає вести себе як керівник, який видає команди, а не дотримується настанов про слугування, які передбачає Agile [4].

1.4. Висновок до першого розділу

У цьому розділі дипломного проекту було розглянуто Agile методології та їх використання в управлінні проектами. Agile є ефективним підходом до розробки програмного забезпечення, а його застосування розповсюджене в різних галузях.

Один з провідних фреймворків Agile - Scrum - пропонує ітеративний підхід до розробки проектів через короткі проміжки часу, відомі як спринти. Спринти дозволяють командам швидко реагувати на зміни та зосередитись на важливих завданнях, забезпечуючи прогрес та результативність.

Крім Scrum, також розглянуті Kanban та Extreme Programming (XP). Kanban використовує візуальну дошку для відстеження робочого потоку та оптимізації продуктивності. XP, з іншого боку, покладає акцент на колективну роботу, тестування та постійний зворотний зв'язок.

Застосування Agile методологій в управлінні проектами дозволяє командам працювати більш гнучко та ефективно. Вони забезпечують зниження ризиків,

прискорення розробки, поліпшення комунікації та досягнення високої якості продукту.

Наприклад, Scrum надає рамки для організації роботи команди, зазначаючи ролі та встановлюючи короткі цикли розробки. Це дозволяє забезпечити гнучкість та швидкість виконання завдань, а також сприяє вчасному виявленню й вирішенню проблем.

Kanban, у свою чергу, допомагає візуалізувати потік роботи та контролювати його. Завдяки візуальній дошці команда може легко відстежувати прогрес та ідентифікувати можливі блокування чи затримки. Це дозволяє вчасно реагувати та впроваджувати корективи для підтримки продуктивності.

Extreme Programming (XP) покладає акцент на співпрацю, комунікацію та взаємодію між учасниками команди. Процес програмування базується на постійному зворотному зв'язку та тестуванні, що сприяє якості продукту та забезпечує вчасне виявлення та виправлення помилок.

Застосування Agile методологій управління проектами дозволяє організаціям працювати більш гнучко та ефективно. Вони дозволяють швидко адаптуватися до змінних умов, визначати пріоритети та прискорювати процес розробки. Крім того, Agile підходи полегшують комунікацію в команді, сприяють взаєморозумінню та підвищують якість продукту.

Загалом, Agile методології є цінним інструментом для сучасного управління проектами, який допомагає організаціям досягати успіху в конкурентному середовищі. Використання Agile підходів сприяє забезпеченню результатів та задоволення потреб учасників проекту, сприяє ефективному використанню ресурсів та підвищує якість розробки програмного забезпечення.

РОЗДІЛ 2
АНАЛІЗ СИСТЕМ УПРАВЛІННЯ, ЩО ВИКОРИСТОВУЮТЬ AGILE МЕТОДОЛОГІЇ

2.1. Ключові аспекти систем управління проектами

Система управління проектами - це програмне забезпечення, яке використовується в різноманітних галузях для планування проектів, розподілу ресурсів та розкладу. Воно дозволяє керівникам проектів та всім командам контролювати свій бюджет, управління якістю та всі документи, які обмінюються під час проекту. Це програмне забезпечення також слугує платформою для полегшення співпраці між учасниками проекту [5].

Від відстеження досягнення результатів до управління ресурсами та від управління бюджетом до співпраці з членами команди - є багато аспектів, які необхідно враховувати при запуску та управлінні проектами.

Автоматизація

Лідери все більше зацікавлені в використанні автоматизації або хоча б відкриті до того, що це може зробити для покращення бізнес-операцій та результативності. Навіть якщо автоматизовані функції працюють трохи по-іншому на різних платформах, вони зазвичай дозволяють проектним командам зменшити або усунути ручну роботу.

Звітність

Інструменти управління проектами Agile потребують функцій звітності, щоб користувачі могли докладно ознайомитися з деталями щодо робочого навантаження членів команди, проблем або успіху проекту, або з того, скільки часу компанія

відводить на роботу з певним клієнтом протягом середнього місяця.

Ця інформація особливо корисна, коли члени керівництва хочуть перевірити, наскільки вигідний бізнес клієнта для підсумкових показників або виявити випадки, коли клієнт може запитувати проектну роботу, яка не відповідає тарифам, які організація збирається з нього.

Можливості планування

Оскільки багато Agile-інструментів для управління проектами дозволяють створювати графіки роботи команди та контролювати прогрес, вони відповідають бізнес-потребам, незалежно від того, чи працюють співробітники на одному сайті, чи на десятках. Наявність мобільних додатків дозволяє вам змінювати або перевіряти графіки навіть якщо ви не в офісі. Після створення графіків, їх можна переглядати в кількох форматах, щоб задовольнити потреби користувачів.

2.2. Огляд систем управління проектами

2.2.1. Jira

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. Платформа використовує всі види навичок управління проектами, включаючи розробку програмного забезпечення, гнучке управління проектами, відстеження помилок, управління Scrum, управління контентом, маркетинг, управління професійними послугами та багато іншого [6].

Переваги Jira

Гарна видимість робочого процесу: Програмне забезпечення Jira забезпечує кращу видимість робочого процесу шляхом автоматизації процесу та відстеження прогресу в режимі реального часу. Це економить час і ресурси для підприємств і підвищує їх загальну ефективність. Він також пропонує різні функції, такі як дошки

Kanban, дошки Scrum, спеціальні робочі процеси, гнучке маркування проблем тощо, що полегшує керування роботою.

Відстеження часу: Програмне забезпечення Jira також пропонує кращу функцію відстеження часу, щоб допомогти компаніям і окремим особам зрозуміти графіки та бюджети проектів. Автоматично реєструючи час, витрачений на кожне завдання, Jira надає точне уявлення про те, де витрачається час і скільки роботи включено в кожен проект.

Глибокі звіти та аналітика: Спеціальна функція штучного інтелекту Jira, яка може створювати нові поглиблені звіти та статистику, економить час у різних сценаріях. Це дозволяє користувачам відстежувати прогрес проекту, виявляти проблеми та створювати індивідуальні звіти.

Підвищення продуктивності: Гнучка система відстеження проблем Jira дає змогу розробникам ефективніше відстежувати та керувати завданнями, а її надійні функції звітування дають уявлення про хід проекту та визначають області, які потрібно вдосконалити. Це, загалом, підвищує продуктивність команди для пошуку нових рішень; натомість вони заплутуються у пошуку та вирішенні поточних проблем.

Безпечність при розробці програмного забезпечення: Налаштування безпеки програмного забезпечення в Jira обмежує доступ до певної помилки лише тим людям, яким дозволено працювати над помилкою, або членам команди з певним рівнем безпеки. Можна встановити рівень безпеки конкретної помилки під час її створення або редагування. Так само існує функція безпеки, як-от Default Permission Scheme (Схема дозволів за замовчуванням).

Недоліки Jira

Складність для початківців: Jira є популярним інструментом управління послугами, яким користуються багато компаній, але він може бути складним для початківців. Інтерфейс користувача не є особливо інтуїтивно зрозумілим, а розмаїття функцій та опцій можуть на початку спричиняти плутанину.

Обмеження розміру файлу: Обмеження розміру файлу Jira є ще одним недоліком, який може впливати на прогрес. Можна завантажувати лише до 10 МБ на

файл. Це обмеження дуже обмежує, якщо користувач працює з великими файлами або керуєте багатьма проектами. Якщо в проекті потрібно регулярно обмінюватися великими файлами або керувати великою кількістю даних, це програмне забезпечення може бути не зручним для використання.

Відсутність засобів комунікації: Цей недолік ускладнює координацію команди, бо немає можливості надсилати повідомлення або спілкуватися з іншими учасниками команди, що може призвести до проблем. Одним зі способів вирішення цієї проблеми є використання зовнішніх інструментів чату.

2.2.2. Wrike

Wrike - це хмарний інструмент співпраці та управління проектами, що допомагає користувачам керувати проектами від початку до кінця, забезпечуючи повну видимість та контроль над завданнями. Кінцеве рішення бере проекти від початкового запиту до відстеження прогресу робіт та звітів про результати [7].

Менеджери проектів та керівники команд отримують ефективний інструмент для збору та організації вимог проекту, створення планів проекту та візуалізації графіків. При роботі можна легко коригувати плани проекту, а всі залучені команди отримуватимуть повідомлення про зміни.

Переваги Wrike

All-In-One співпраця: Wrike спрощує управління проектами, надаючи командам комплексне рішення для співпраці. Замість телефонних дзвінків, відправлення текстових повідомлень та листування по електронній пошті, користувачі Wrike можуть співпрацювати безпосередньо всередині рішення.

Швидкий та зручний інтерфейс: Інтерфейс Wrike складається з трьох панелей, що дозволяють побачити всю робочий процес одним поглядом. Все необхідне відображається на головному екрані. У першій панелі інформаційної панелі можна легко управляти ресурсами, отримати доступ до фінансів, команд та проектів. У другій панелі можна створювати задачі, встановлювати терміни виконання та

призначати їх відповідним членам команди. А в останній панелі можна переглянути поточні задачі та підзадачі.

Аналітика в реальному часі: Wrike забезпечує свіжу інформацію, автоматично оновлюючи звіти, аналітичні панелі, діаграми та інфографіки кожні 15 хвилин. Завдяки цим постійним оновленням, керівники проектів завжди можуть отримувати актуальну інформацію про прогрес завдань, продуктивність команди, статус проекту та більше.

Автоматизація робочого процесу: Члени команди можуть не витрачати багато часу на рутинні завдання, що може допомогти збільшити ефективність роботи в цілому. В Wrike можна автоматизувати навіть процес затвердження завдання. Також, ще існує понад 400 сторонніх інструментів, що можуть бути впровадженні в робочий процес для автоматизації роботи керівників та команди.

Наявність інструментів рівня підприємства: Керівники можуть відстежувати час всього проекту та окремих учасників команди, публікувати активи та ділитися файлами. А також, з усього, все це підсилюється безпекою на рівні підприємства.

Недоліки Wrike

Високий рівень навчання: Великий набір функцій на рівні підприємства може ускладнити роботу початківцям, які тільки ознайомлюються з програмним забезпеченням управління проектами. Окрім самого набору функцій, настройка та впровадження Wrike може бути складним процесом.

2.2.3. ClickUp

ClickUp — це хмарний інструмент для співпраці та керування проектами, який підходить для компаній будь-якого розміру та галузей. Функції включають інструменти спілкування та співпраці, призначення завдань і статуси, сповіщення та панель інструментів завдань [8].

Користувачі можуть призначати коментарі та завдання окремим членам команди або групам членів команди. Коментарі та завдання можна позначати як

вирішені або в процесі виконання, або користувачі можуть створювати власні статуси [9].

ClickUp - це універсальний інструмент можна налаштувати відповідно до будь-якого типу проекту, а також робочого процесу, потреб і вимог будь-якої команди.

Переваги ClickUp

Командна співпраця: Це чудовий інструмент для командної співпраці, оскільки він дозволяє командам працювати разом над проектами, призначати завдання, встановлювати дедлайни та спілкуватися в режимі реального часу без необхідності переходити з однієї програми в іншу.

Управління проектами: Функції управління проектами ClickUp є надійними та гнучкими. Це дозволяє користувачам створювати завдання, встановлювати крайні терміни, призначати завдання членам команди та відстежувати прогрес. Користувачі також можуть використовувати діаграми Ґанта та спеціальні робочі процеси для керування проектами.

Планування заходів: Функції ClickUp можуть допомогти організаторам заходів ефективно керувати своїми завданнями та графіком. Користувачі можуть створювати завдання для кожного аспекту події, призначати членів команди та відстежувати прогрес.

Особиста продуктивність: ClickUp також можна використовувати для особистого управління, зростання та цілей. Користувачі можуть створити особистий робочий простір і використовувати його для керування списками справ, встановлення нагадувань і відстеження прогресу.

Недоліки ClickUp

Не для початківців: Для членів команди, які тільки починають працювати з системами управління проектами, ClickUP може бути важко опанувати з самого початку. У ньому є безліч функцій, пошук яких може зайняти деякий час, і, незважаючи на те, що це добре для досвідчених користувачів, можливості налаштування для деяких можуть здатися приголомшливими.

Необхідність навчання: Команда, не може перейти до ClickUP і одразу приступити до роботи. Натомість, знадобиться деякий час для навчання та ознайомлення з програмним забезпеченням. Потрібно виділити багато часу на процес налаштування, оскільки він передбачає введення безлічі деталей.

2.3. Оцінка ефективності Agile методологій в системах управління проектами

Ефективність використання Agile методології у системах управління 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.

Хоча Agile методології можуть бути ефективними, варто зауважити, що вони не є універсальним рішенням для всіх проектів. Деякі проекти, особливо ті, що мають жорсткі вимоги до графіка, бюджету або безпеки, можуть краще підходити для інших методологій управління проектами. Істотно важливо адаптувати методологію до конкретного контексту і потреб проекту, а також враховувати особливості команди та організаційну культуру з метою досягнення максимальної ефективності.

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.

Важливо розуміти, що 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.

2.4. Висновок до другого розділу

В даному розділі дипломного проекту було проведено аналіз систем управління проектами, що використовують Agile методології. Розглянуті системи включають Jira, Wrike та ClickUp. Кожна з цих систем має свої переваги та особливості, але також має певні недоліки, які варто врахувати при виборі підходящої системи для конкретного проекту.

Jira є потужною системою управління проектами, розробленою спеціально для Agile розробки програмного забезпечення. Вона надає широкий спектр функціональності для планування, відстеження та співпраці команди. Jira дозволяє створювати і керувати беклогом завдань, стежити за прогресом спринтів, відстежувати баги та проблеми, здійснювати звітність і багато іншого. Однак, вона може бути складною для новачків, оскільки має велику кількість функцій і конфігураційних параметрів. Інтерфейс Jira також може здаватися перевантаженим і незручним для деяких користувачів.

Wrike є інтегрованою системою управління проектами, яка пропонує простий і зручний інтерфейс. Вона дозволяє командам спільно працювати над завданнями, планувати та відстежувати прогрес. Wrike має широкий спектр функціональних можливостей, включаючи відстеження часу, спільний доступ до документів, коментарі та зворотній зв'язок. Однак, у порівнянні з іншими системами, Wrike може бути менш гнучким у плануванні та управлінні проектами, а також має обмежені можливості звітності.

ClickUp є високофункціональною системою управління проектами, яка надає широкий набір інструментів для організації та керування проектами. Вона підтримує різні Agile методології і пропонує зручне візуальне середовище для створення, відстеження та співпраці над завданнями. ClickUp має багато корисних функцій, включаючи календар, графіки Ганта, звіти, шаблони проектів та інтеграцію з іншими інструментами. Однак, через свою велику кількість функцій, ClickUp може бути завантаженим та складним для навігації, особливо для новачків.

Порівнюючи ці системи, Jira відзначається широким функціоналом та розширеними можливостями звітності, але може бути складною у використанні. Wrike привертає простотою та зручністю інтерфейсу, але може бути менш гнучким у плануванні та управлінні. ClickUp забезпечує високий рівень функціональності, але може бути складним для новачків та вимагати більше часу на освоєння.

Остаточний вибір системи управління проектами залежить від потреб конкретного проекту та команди. Важливо врахувати специфіку проекту, розмір команди, потреби відстеження прогресу, співпраці та інтеграції з іншими інструментами. Також слід звернути увагу на вартість, підтримку та наявність необхідних функцій.

Висновок підкреслює, що вибір системи управління проектами є важливим етапом при впровадженні Agile методологій. Жодна з розглянутих систем не є універсальною, і кожна має свої переваги та обмеження.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ AGILE-ОРІЄНТОВАНОЇ СИСТЕМИ УПРАВЛІННЯ ПРОЕКТАМИ НА JAVA

3.1. Проектування архітектури системи

Архітектура програмного забезпечення 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 [10].

3.1.1. Опис базової функціональності системи

Базовий функціонал нашого програмного забезпечення буде включати такі модулі:

Клас Project (Проект). Цей модуль є один з ключових, бо він дозволяє користувачам почати нові проекти та налаштовувати їх параметри. При створенні 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). Епік виступає в ролі великого блоку 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. Зазвичай спринт 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.

3.1.2. Створення UML-діаграми класів

Щоб описати архітектуру програми, ми будемо використовувати UML діаграму класового типу.

UML (Unified Modeling Language) означає уніфіковану мову моделювання [11].

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. Назва вгорі, атрибути посередині та операції або методи внизу. У великій системі з багатьма пов’язаними класами класи групуються разом для створення діаграм класів. Різні відносини між класами показані різними типами стрілок [11].

Надана діаграма класів (рис. 3.1) представляє систему для управління проектами, спрінтами, епіками, заявками, користувачами та коментарями.

Розшифровка класів та їх взаємозв'язків надана в додатку А.

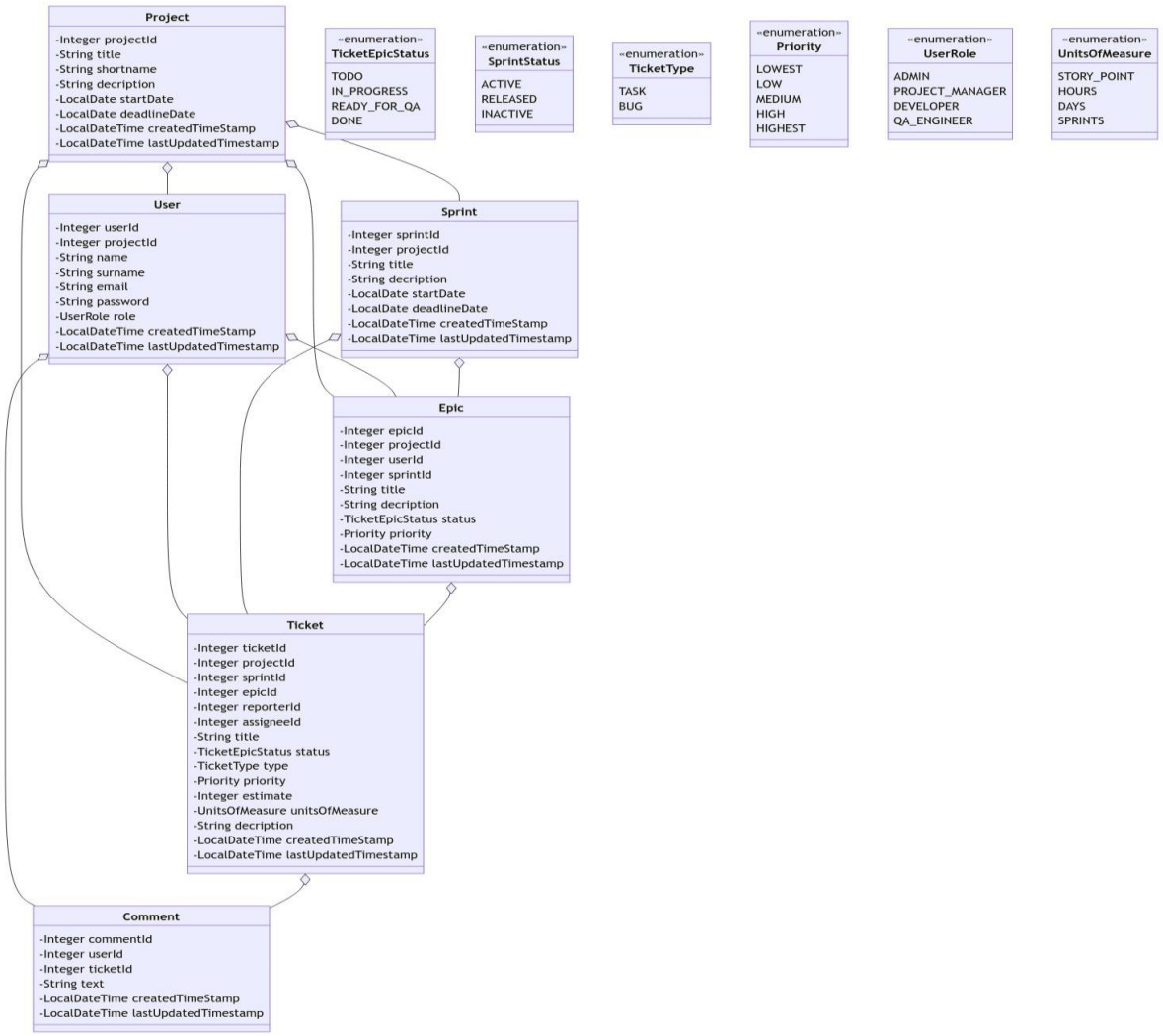

Рис. 3.1. Реалізація UML-діаграми класів.

3.2. Створення початкової структури проекту

Реалізація системи управління проектами буде з використанням засобу автоматизації Maven та Java 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 [12].

Maven — це інструмент управління проектами та розуміння, який надає розробникам повну структуру життєвого циклу збірки [13].

Для легкого створення Spring Boot проекту використовуємо spring initializr.

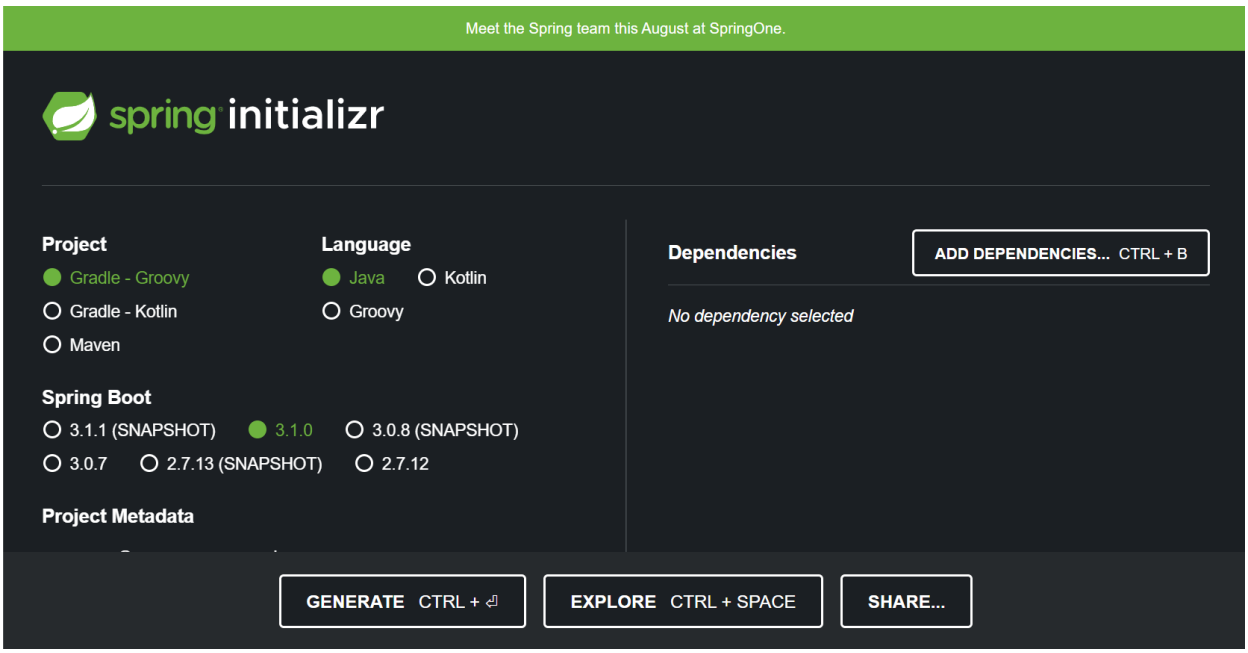

Рис 3.2. Сайт для створення Spring Boot проекту https://start.spring.io/.

У файлі конфігурації pom.xml описуємо всі необхідні залежності для роботи застосунку. Приклад pom.xml файлу наведено в додатку Б.

Створюємо Aplication клас, для запуску програми:

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

3.3. Реалізація контролерів для обробки HTTP-запитів

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
}
```

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

```
@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. Приклад:

```
@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 внести зміни лише до певних полів або властивостей ресурсу, не торкаючись інших полей. Для обробки PATCH запитів у Spring Boot потрібно використовувати анотацію *@PatchMapping*. Приклад:

```
@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. Приклад:

```
@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 неправильні або некоректні запити.

В прикладі вище в методі *createUser(@Valid @RequestBody UserDTO user)* анотація *@Valid* вказує на те, що деякі поля можуть бути 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 представлені в додатку В

Код створення контролерів наведений в додатку Г.

3.4. Створення та підключення бази даних до програмного продукту

3.4.1. Система керування базами даних MySQL

Для реалізації нашої системи управління проектами, була використана *реляційна база даних MySQL*.

Реляційна база даних зберігає дані в окремих таблицях, а не в одному великому сховищі. Структура бази даних організована у фізичні файли, оптимізовані для швидкості. 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 [14].

Щоб створити базу даних в MySQL Workbench, потрібно підключитися до сервера баз даних. Там створюємо нову базу і вписуємо скрипт, щоб створити таблиці. Лістинг скрипту наданий в додатку Ґ.

3.4.2. Бібліотека Hibernate

Робота як з об’єктно-орієнтованим програмним забезпеченням, так і з реляційними базами даних може бути громіздкою та займати багато часу. Hibernate — це рішення для об’єктного/реляційного відображення (ORM – Object Relational Mapping) для середовищ 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.

Hibernate є бібліотекою для роботи з базами даних, яка не тільки дозволяє відображати класи Java у таблиці бази даних і типи даних Java у типи даних SQL, але також надає потужні засоби для запиту та пошуку даних. Головна мета Hibernate - звільнити розробника від більшості рутинних задач, пов'язаних з збереженням даних [15].

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?serverTimezone=UTC

spring.datasource.username=root

spring.datasource.password=root

server.port=8081

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

Далі необхідно налаштувати Hibernate-анотації. Для цього створюються *Entity* класи (ще називаються *Model*), 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. Приклад оформлення Entity класу:

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

Entity-класи нашого програмного забезпечення представленні в додатку Д.

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.

Реалізація всіх сервісів представлена в додатку Е.

3.5. Функціональні можливості та результати виконання програми

Для перевірки роботи програми та представлення результатів ми використовуємо програмний застосунок *Postman*.

Створення даних на сервері

Щоб створити новий запит 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" (Рис. 3.3).

Рис. 3.3. Приклад HTTP-запиту в Postman.

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 (див. Рис 3.4).

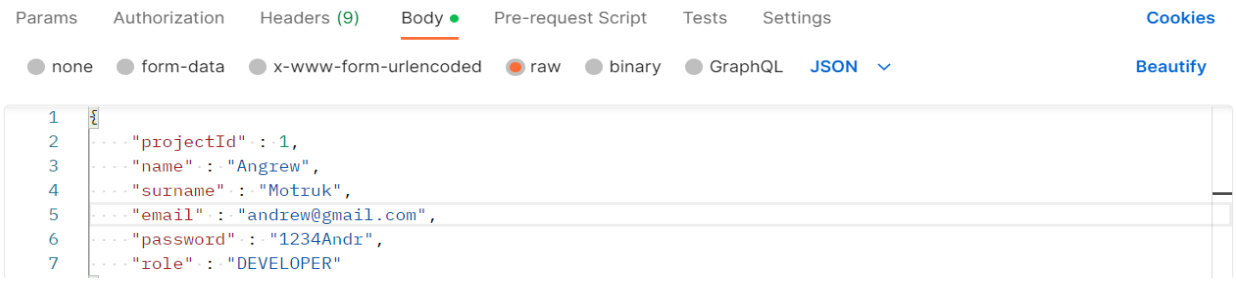

Рис. 3.4. Тіло запиту.

Після відправки запиту Postman відображуємо отриману відповідь. Можна переглянути статус запиту, заголовки, тіло відповіді та іншу інформацію (див. Рис 3.5).

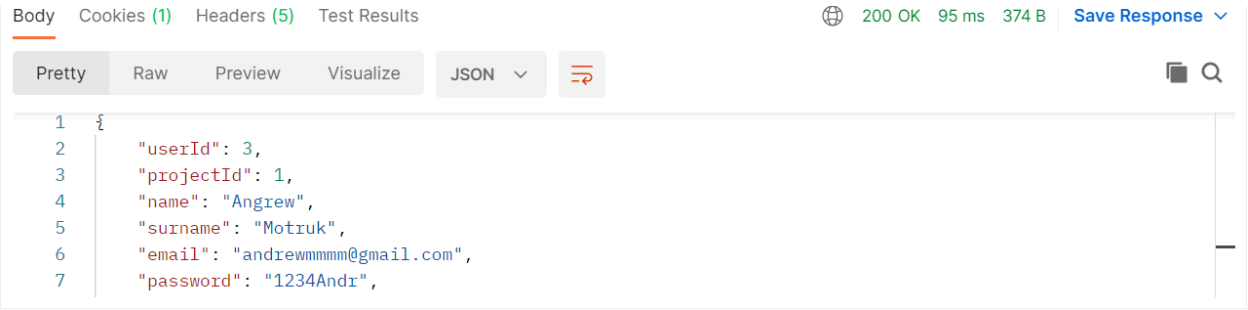

Рис 3.5. Відповідь на запит.

На рис. 3.6 зображений результат створення декількох варіантів 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. 3.6. Таблиця користувачів.

Оновлення даних на сервері

На рис. 3.7 зображено HTTP-запит на часткове оновлення даних на прикладі об’єкту проекта. Для цього використовуємо запит типу PATCH, а також змінюємо URL-адресу. В тілі змінюємо поля, що підлягають під оновлення, та отримуємо результат змінених даних проекту.

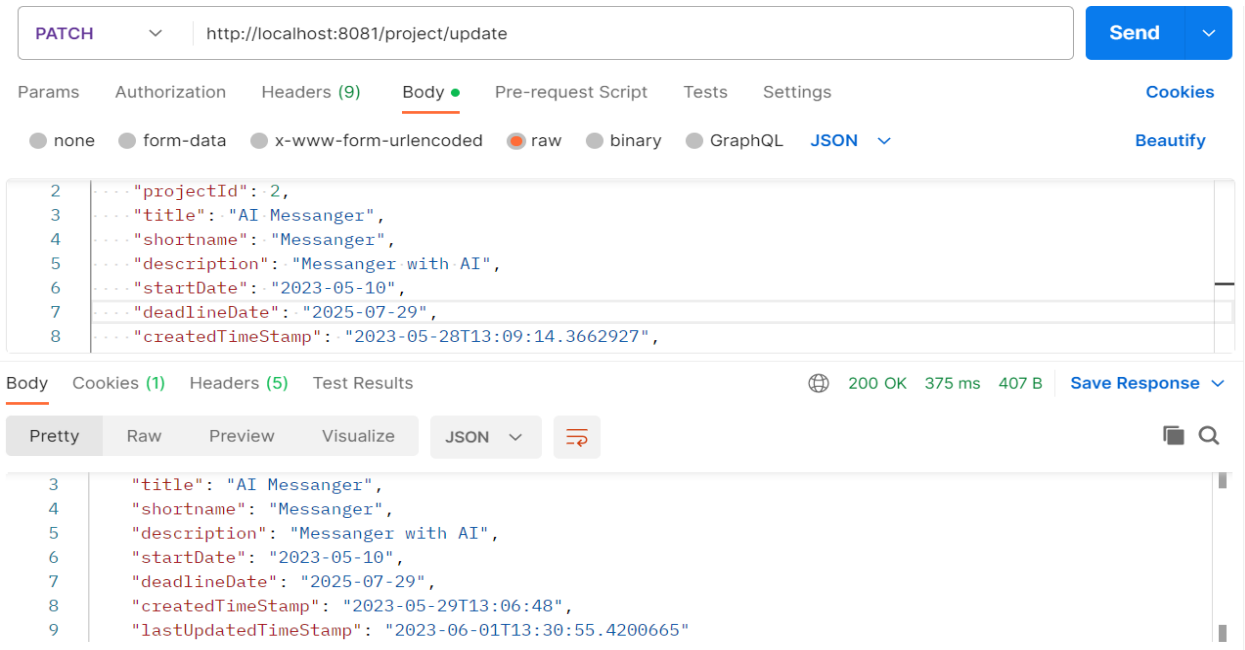

Рис. 3.7. HTTP-запит на оновлення даних проекту.

На рисунках 3.8 і 3.9 зображено дані в таблиці проектів до і після оновлення ресурсу під індексом 2.

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

Рис. 3.8. Таблиця проектів до оновлення даних.

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

Рис. 3.9. Таблиця проектів після оновлення даних.

Видалення даних на сервері

Видалення ресурсу буде представлено на прикладі об’єкта спринта. На рис. 3.10. зображено новий HTTP-запит, а також змінений тип запиту DELETE. В тіло передаємо об’єкт, який ми хочемо видалити.

Тіло результату пусте, і статус запиту 200ОК. Це означає, що видалення пройшло успішно.

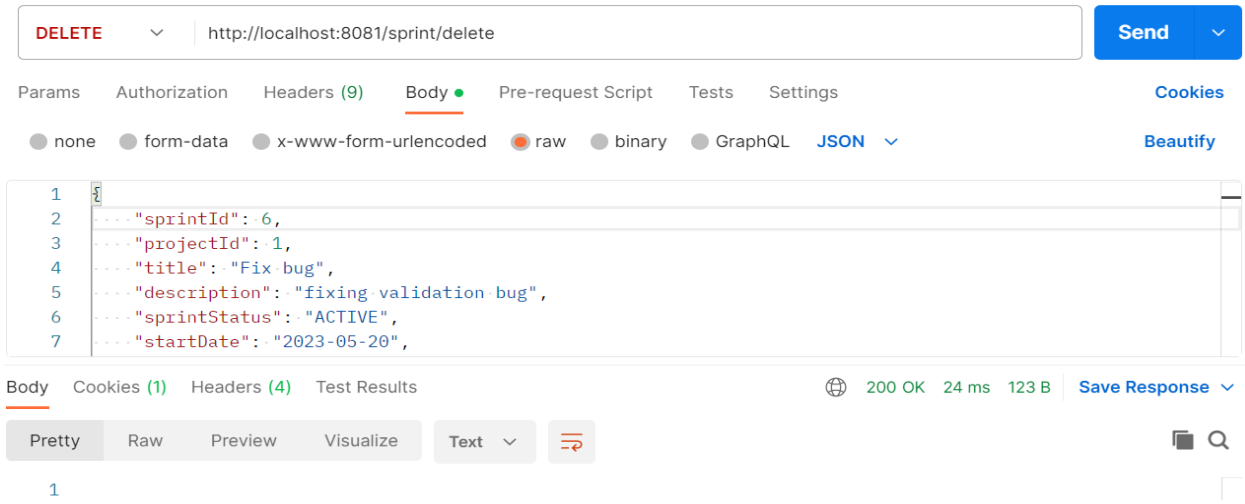

Рис. 3.10. HTTP-запит на видалення даних спрінта.

Отримання даних із серверу (з використанням фільтрів)

На рис. 3.11. зображено пошук користувачів по фільтру. Пошук користувача відбувається за його роллю. Тип запиту змінюємо на GET, вписуємо нову URL-адресу, і в тілі запиту вказуємо роль, за якою хочемо знайти існуючих користувачів.

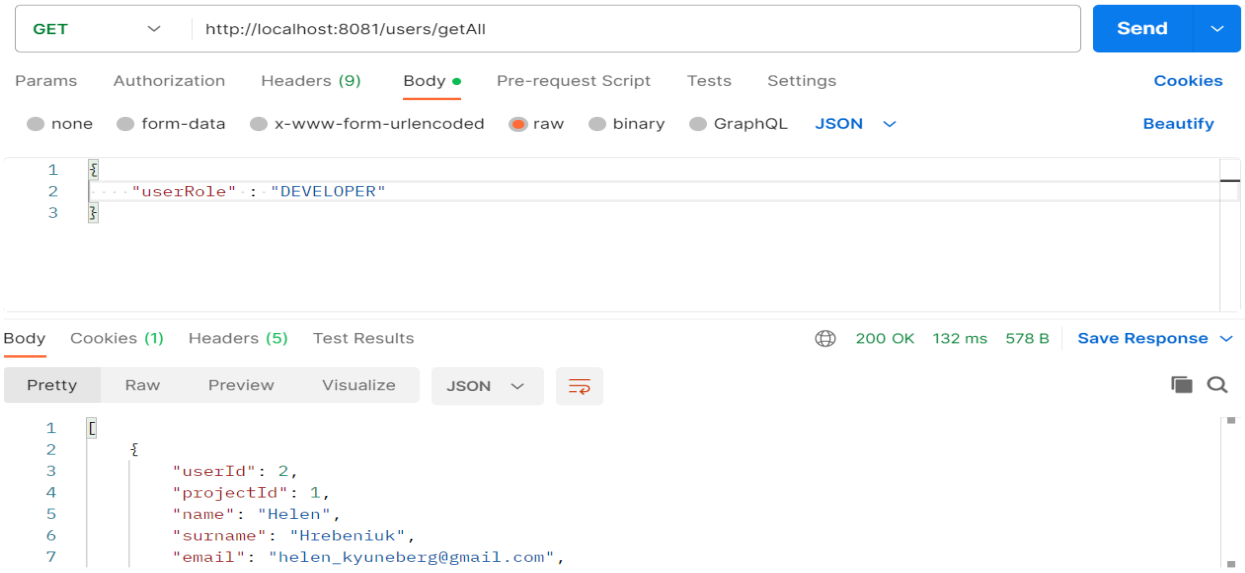

Рис. 3.11. HTTP-запит на отримання даних користувача за фільтром.

Ось повне тіло результату запиту. Як ми бачимо, всі 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
}
```


Отримання даних із сервера (з використанням фільтрів та унікального ідентифікатора)

На рис. 3.12. зображено запит із зазначенням ID і фільтра. На прикладі об’єкта спринта, ми 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.

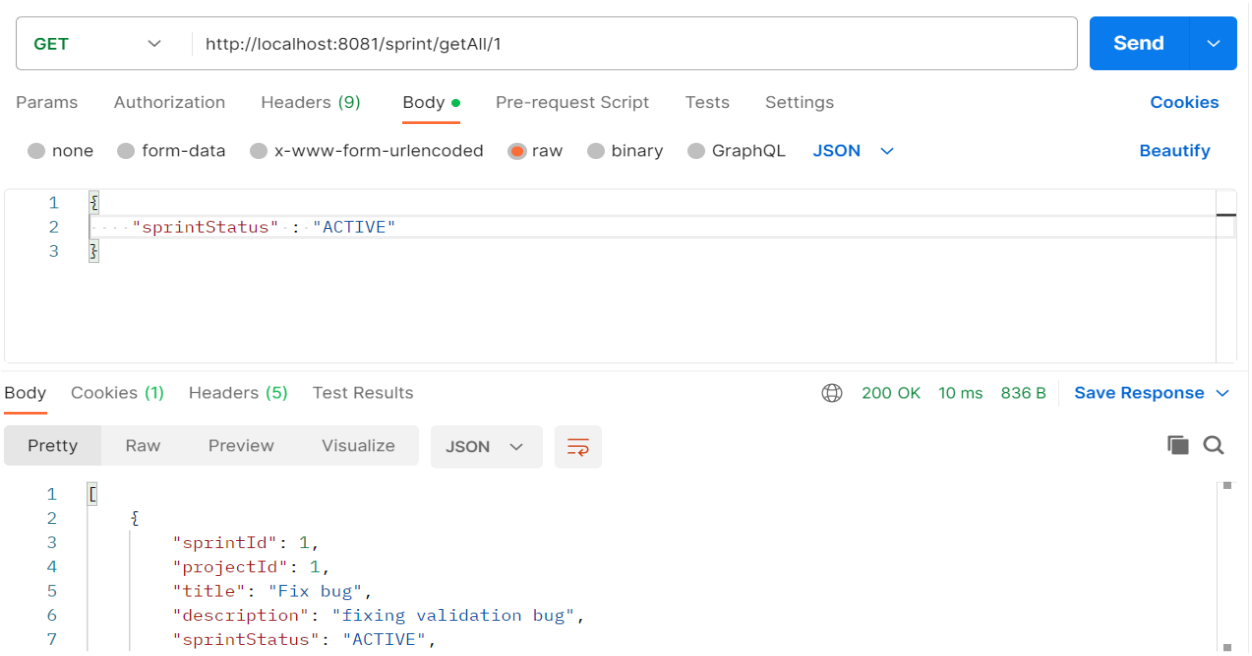

Рис. 3.12. HTTP-запит на отримання даних користувача за фільтром і ID.

Ось повне тіло результату запиту. Як ми бачимо, всі 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
  }
```


3.6. Обробка виключень

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. Він буде 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 вищевказаними 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
}
```

У цьому прикладі, якщо виникає виключення типу UserNotFoundException, буде відправлена відповідь зі статусом 404 Not Found і текстом "User is not found ".

Лістинг обробників всіх виключень нашого програмного забезпечення дивитися в Додатку Є.

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

1. На рис. 3.13 зображений варіант HTTP-запиту, де дата початку розробки проекту (“startDate”) пізніше ніж дата закінчення (“deadlineDate”), що не є логічним і правильним.

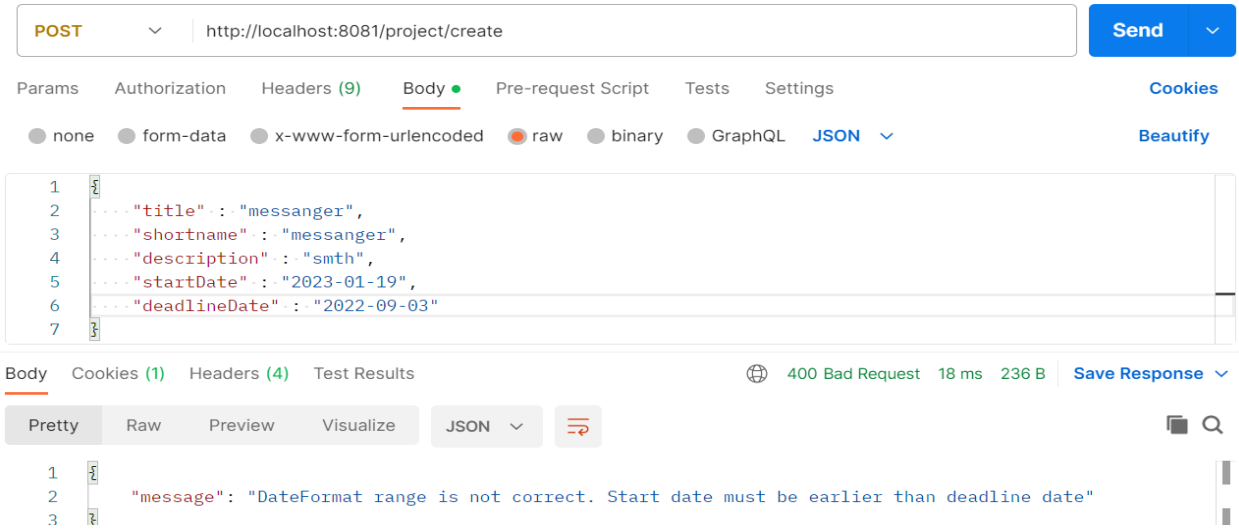

Рис. 3.13. HTTP-запит з неправильним проміжком часу розробки програми.

2. На рис. 3.14 зображений варіант HTTP-запиту, в якому вказується електрона пошта, що вже існує в базі системи. Це означає, що такий користувач вже існує, і йому потрібно просто авторизуватися.

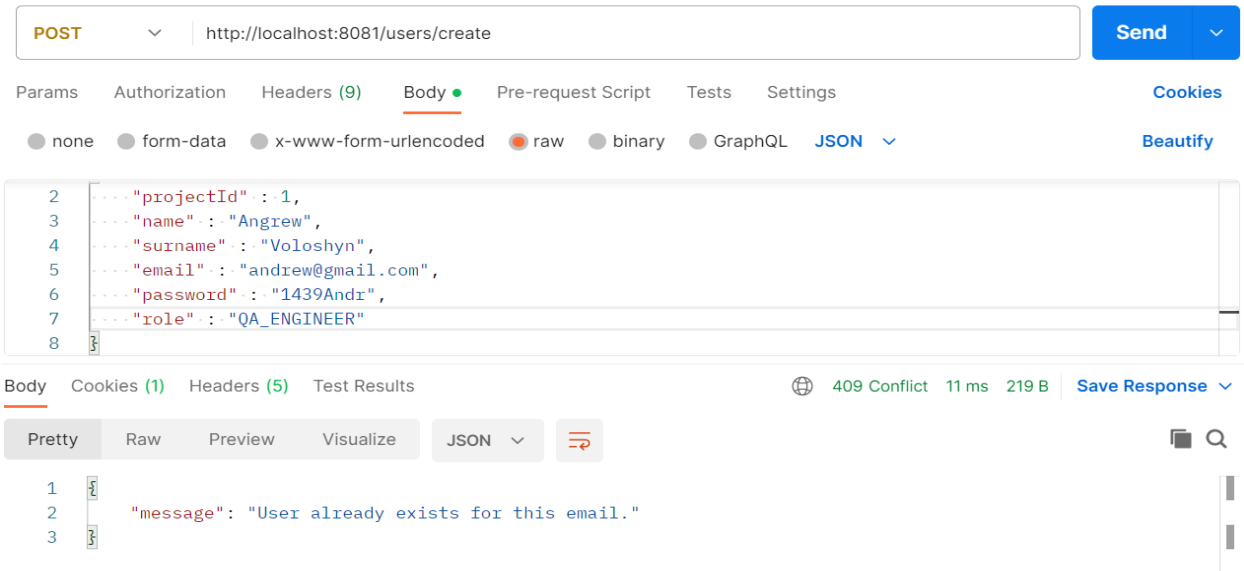

Рис. 3.15. HTTP-запит із вже існуючою на сервері електронною поштою.

3. На рис. 3.15 зображений варіант HTTP-запиту, в якому викликається спринт за унікальним ідентифікатором проекту, якого не існує на сервері.

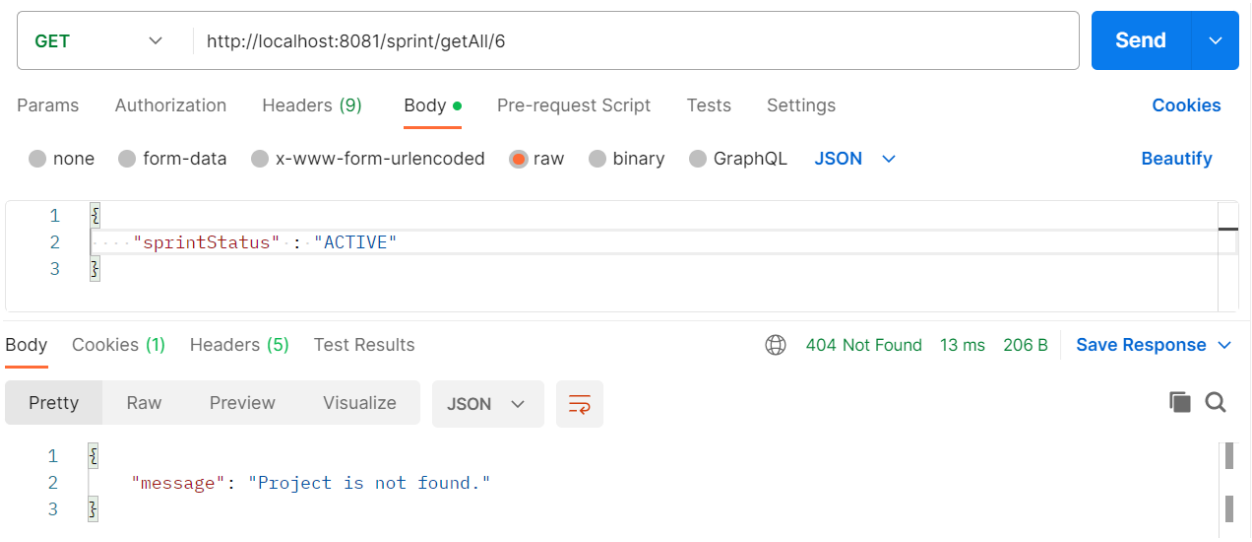

Рис. 3.15. HTTP-запит із неіснуючим проектом.

3.7. Основна ідея та майбутні покращення.

Основна ідея та унікальність системи управління проектами, яку ми розробляли, заключається в тому, що вона буде предствлена у вигляді окремого API. Це надасть 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 і ефективно.

При використанні цього API розробники отримують гнучкість у виборі того, які функції вони хочуть використовувати. Вони можуть вибрати лише ті API-методи, які їм потрібні для їх конкретних потреб і ігнорувати решту. Це дозволяє оптимізувати використання ресурсів і забезпечити ефективну роботу з системою управління 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 даній системі управління проектами передбачаються такі роширення функціональності, що в майбутньому зможуть надати більше можливостей якісно керувати проектом:

Spring Security. Структура, надана Spring, яка допомагає налаштувати доступ і процес автентифікації. Він відіграє дуже важливу роль у забезпеченні безпеки програм [16].

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'язкок 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 додатковим запитом або задачею, яка пов'язана з основним квитком і потребує окремого виконання.

Коментарі усюди. У майбутньому 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.

Фільрація 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.

3.8. Висновок до третього розділу

У третьому розділі дипломного проекту, присвяченому реалізації agile-орієнтованої системи управління проектами на Java, було проведено глибоке дослідження та розробка ключових аспектів системи. Зокрема, була створена UML-діаграма класів, що відображає структуру системи та зв'язки між її компонентами.

Для реалізації системи було обрано фреймворк Spring Boot, який забезпечує швидку розробку та простоту в налаштуванні. Були розроблені контролери, які взаємодіють з 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 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.

При створенні системи була врахована основна функціональність системи управління 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.

У майбутньму в програмне забезпечення буде додана можливість аутентифікації користувачів, розширяться можливості коментарів в усіх компонентах, для якісної співпраці команди, а також звітність для ефективного прийняття рішень.

Реалізація системи управління 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.

ВИСНОВОК

В результаті виконання даної дипломної роботи було досягнуто поставленої мети - розробки системи управління проектами на мові програмування Java з використанням Agile методологій, що зможе полегшити та покращити роботу команд розробників. Таким чином, в результаті роботи:

- Досліджено різні Agile методології.
- Проведений аналіз існуючих систем управління проектами: їх переваги та недолгіки.
- Створено систему орієнтовану на API, для легкої інтеграції з іншими інструментами, що надасть розробникам легку співпрацю з функціональністю системи.
- Надано системі можливість на легку розширюваність функціональності, шляхом розподілу програми на логічні модулі, які виконують конкретні функції.
- Були використані сучасні технології та бібліотеки, такі як Spring Framework, Hibernate, JpaRepository, що дозволило забезпечити високу якість та ефективність системи.

Наша робота включала аналіз найпопулярніших Agile методологій, таких як Scrum і Kanban, та їх застосування для ефективного керування проектами та прискорення їх розробки.

Під час аналізу вже існуючих систем управління проектами було виявлено, що багато з них мають складний користувацький інтерфейс та обмежену функціональність, що ускладнює їх використання для розробників. Враховуючи це, ми прийняли рішення реалізувати систему управління 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.

Після розширення нашого програмного забезпечення до API, система зможе надати розробникам можливість легко підключати та використовувати необхідні функції системи у своїх проектах через API. Це спроситить 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.

Наша розроблена система стане інноваційним інструментом, що дозволить розробникам працювати з функціоналом управління проектами без зайвих труднощів та витрат часу. Вона забезпечить ефективне управління проектами з використанням Agile підходу, дозволяючи командам розробників працювати швидше, гнучкіше та результативніше.

У цьому дипломному проекті ми також розробили докладну UML-діаграму класів, що відображає структуру системи управління проектами. Крім того, ми розглянули процес створення Spring Boot проекту, реалізацію контролерів, репозиторіїв, сервісів та обробників помилок. Всі ці компоненти взаємодіють між собою та забезпечують високу якість та ефективність системи управління проектами.

Загалом, дипломний проект по реалізації системи управління проектами на Java з використанням Agile методологій є важливим внеском у галузь розробки програмного забезпечення. Він демонструє, що шлях до успішного управління проектами може бути спрощений та оптимізований за допомогою сучасних інструментів та технологій.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Стаття «The Agile Coach. Atlassian's no-nonsense guide to agile development» [Electronic resource]. Режим доступу: https://www.atlassian.com/agile (дата звернення 15.05.2023). – Назва з екрана.
2. Стаття «Top 5 main Agile methodologies: advantages and disadvantages» [Electronic resource]. Режим доступу: https://www.xpand-it.com/blog/top-5-agile-methodologies/ (дата звернення 15.05.2023). – Назва з екрана.
3. Стаття «What Is a Scrum Sprint Cycle?» [Electronic resource]. Режим доступу: https://www.wrike.com/scrum-guide/faq/what-is-a-scrum-sprint-cycle/ (дата звернення 16.05.2023). – Назва з екрана.
4. Стаття «Agile Team Roles: What They Are and What They Do» [Electronic resource]. Режим доступу: https://projectmanagementacademy.net/resources/blog/agile-team-roles what-they-are-and-what-they-do/ (дата звернення 16.05.2023). – Назва з екрана.
5. Стаття «Agile Project Management - What is it and how to get started? How agile methodologies can work for your software team» [Electronic resource]. Режим доступу: https://www.atlassian.com/agile/project-management#:~:text=Agile%20project%20management%20is%20an,customer%20feedback%20with%20every%20iteration (дата звернення 20.05.2023). – Назва з екрана.
6. Стаття «What is JIRA? Overview and Full Guide» [Electronic resource]. Режим доступу до ресурсу: https://appmaster.io/blog/what-is-jira (дата звернення 20.05.2023). – Назва з екрана.
7. Стаття «What is Wrike? Overview of Wrike benefits» [Electronic resource]. Режим доступу: https://elearningindustry.com/directory/elearning-software/wrike (дата звернення 20.05.2023). – Назва з екрана.

8. Стаття «About ClickUp» [Electronic resource]. Режим доступу: https://www.softwareadvice.com/project-management/clickup-profile/ (дата звернення 20.05.2023). – Назва з екрана.

9. Стаття «ClickUp Overview: How It Works & Why Use It for Project Management» [Electronic resource]. Режим доступу: https://friday.app/p/what-is-clickup (дата звернення 22.05.2023). – Назва з екрана.

10. Стаття «Software architecture diagramming and patterns» [Electronic resource]. Режим доступу: https://www.educative.io/blog/software-architecture-diagramming-and-patterns (дата звернення 23.05.2023). – Назва з екрана.

11. Стаття «UML Diagram Types Guide: Learn About All Types of UML Diagrams with Examples» [Electronic resource]. Режим доступу: https://creately.com/blog/diagrams/uml-diagram-types-examples/#ClassDiagram (дата звернення 28.05.2023). – Назва з екрана.

12. Стаття «Spring Boot» [Electronic resource]. Режим доступу: https://spring.io/projects/spring-boot (дата звернення 28.05.2023). – Назва з екрана.

13. Стаття «What is Maven: Here's What You Need to Know» [Electronic resource]. Режим доступу до ресурсу: https://www.simplilearn.com/tutorials/maven-tutorial/what-is-maven (дата звернення 30.05.2023). – Назва з екрана.

14. Стаття «What is MySQL?» [Electronic resource]. Режим доступу: https://www.oracle.com/mysql/what-is-mysql/ (дата звернення 30.05.2023). – Назва з екрана.

15. Стаття «Hibernate Tutorial» [Electronic resource]. Режим доступу: https://www.javatpoint.com/hibernate-tutorial (дата звернення 30.05.2023). – Назва з екрана.

16. Стаття «Spring Security: How it works internally» [Electronic resource]. Режим доступу: https://blog.knoldus.com/spring-security-internal/ (дата звернення 30.05.2023). – Назва з екрана.

ДОДАТКИ

Додаток А

Розшифровка UML-діаграми класів

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

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

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

- *Epic (Епік):* Представляє епік в проекті. Має зв'язок багато-до-одного з класами Project (Проект), User (Користувач) та Sprint (Спрінт). Містить атрибути, такі як epicId (ідентифікатор епіка), projectId (ідентифікатор проекту), userId (ідентифікатор користувача), sprintId (ідентифікатор спрінту), title (назва), description (опис), status (статус), priority (пріоритет), createdTimeStamp (час створення) та

lastUpdatedTimestamp (останній час оновлення). Перерахування TicketEpicStatus (Статус епіка та завдання) представляє різні статуси епіка, а перерахування Priority (Пріоритет) представляє різні рівні пріоритету.

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

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

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

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

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

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

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

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

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

Лістинг файлу pom.xml

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

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

```
    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));
    }
}
```

Лістинг SQL запиту на створення таблиць в базі даних

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

```
sprint_status varchar(255) NOT NULL,
start_date datetime,
deadline_date datetime,
created_time_stamp datetime NOT NULL,
last_updated_time_stamp datetime,
FOREIGN KEY (project_id) REFERENCES project(project_id)
);
CREATE TABLE IF NOT EXISTS epic(
epic_id int NOT NULL auto_increment PRIMARY KEY,
project_id int NOT NULL,
user_id int NOT NULL,
sprint_id int NOT NULL,
title varchar(255) NOT NULL,
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)
)
```

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 TicketService.java

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

Лістинг файлу TicketServiceImpl.java

```
@Service
public class TicketServiceImpl implements TicketService {

    ModelMapper mapper = new ModelMapper();

   TicketRepository ticketRepository;
    public TicketServiceImpl(TicketRepository ticketRepository) {
        this.ticketRepository = ticketRepository;
    }
    @Override
    public TicketDTO create(TicketDTO ticket) {
        TicketModel ticketModel = mapper.map(ticket, TicketModel.class);
        return mapper.map(
```

```
            ticketRepository.save(ticketModel), TicketDTO.class
        );
    }
    @Override
    public List<TicketDTO> getAllByProject(Integer projectId, Filter filter) {
        List<TicketModel> tickets = ticketRepository.getAllByProjectId(projectId);
        if(tickets.isEmpty()) {
            throw new ProjectNotFoundException();
        }

        return tickets.stream()
                .filter(ticket -> ticket.getTicketType() == filter.getTicketType())
                .filter(ticket -> ticket.getPriority() == filter.getTicketPriority())
                .filter(ticket -> ticket.getStatus() == filter.getTicketStatus())
                .map(ticket -> mapper.map(ticket,
TicketDTO.class)).collect(Collectors.toList());
    }
    @Override
    public List<TicketDTO> getAllBySprint(Integer sprintId, Filter filter) {
        List<TicketModel> tickets = ticketRepository.getAllBySprintId(sprintId);
        if(tickets.isEmpty()) {
            throw new SprintNotFoundException();
        }
        return tickets.stream()
                .filter(ticket -> ticket.getTicketType() == filter.getTicketType())
                .filter(ticket -> ticket.getPriority() == filter.getTicketPriority())
                .filter(ticket -> ticket.getStatus() == filter.getTicketStatus())
```

```
            .map(ticket -> mapper.map(ticket,
TicketDTO.class)).collect(Collectors.toList());
    }

    @Override
    public List<TicketDTO> getAllByEpic(Integer epicId, Filter filter) {
        List<TicketModel> tickets = ticketRepository.getAllByEpicId(epicId);
        if(tickets.isEmpty()) {
            throw new EpicNotFoundException();
        }

        return tickets.stream()
                .filter(ticket -> ticket.getTicketType() == filter.getTicketType())
                .filter(ticket -> ticket.getPriority() == filter.getTicketPriority())
                .filter(ticket -> ticket.getStatus() == filter.getTicketStatus())
                .map(ticket -> mapper.map(ticket,
TicketDTO.class)).collect(Collectors.toList());
    }

    @Override
    public List<TicketDTO> getAllByReporter(Integer reporterId, Filter filter) {
        List<TicketModel> tickets = ticketRepository.getAllByReporterId(reporterId);
        if(tickets.isEmpty()) {
            throw new UserNotFoundException();
        }
        return tickets.stream()
```

```
            .filter(ticket -> ticket.getTicketType() == filter.getTicketType())
            .filter(ticket -> ticket.getPriority() == filter.getTicketPriority())
            .filter(ticket -> ticket.getStatus() == filter.getTicketStatus())
            .map(ticket -> mapper.map(ticket,
TicketDTO.class)).collect(Collectors.toList());
    }
    @Override
    public List<TicketDTO> getAllByAssignee(Integer assigneeId, Filter filter) {
        List<TicketModel> tickets = ticketRepository.getAllByAssigneeId(assigneeId);
        if(tickets.isEmpty()) {
            throw new UserNotFoundException();
        }
        return tickets.stream()
            .filter(ticket -> ticket.getTicketType() == filter.getTicketType())
            .filter(ticket -> ticket.getPriority() == filter.getTicketPriority())
            .filter(ticket -> ticket.getStatus() == filter.getTicketStatus())
            .map(ticket -> mapper.map(ticket,
TicketDTO.class)).collect(Collectors.toList());
    }
    @Override
    public TicketDTO getTicket(Integer id) {
        TicketModel ticketModel = ticketRepository.getTicketById(id);
        if (ticketModel == null) {
            throw new TicketNotFoundException();
        }
        return mapper.map(ticketModel, TicketDTO.class);
```

```
    }
    @Override
    public void deleteTicket(TicketDTO ticket) {
        TicketModel ticketModel =
mapper.map(ticketRepository.getTicketById(ticket.getTicketId()), TicketModel.class);
        if (ticketModel == null) {
            throw new TicketNotFoundException();
        } else {
            ticketRepository.delete(ticketModel);
        }
    }
    @Override
    public TicketDTO updateTicket(TicketDTO ticket) {
        Integer ticketId = ticket.getEpicId();
        TicketDTO updateTicket = mapper.map(ticketRepository.getTicketById(ticketId),
TicketDTO.class);
        if (updateTicket == null) {
            throw new TicketNotFoundException();
        }
        if(ticket.getAssigneeId() != 0) {
            updateTicket.setAssigneeId(ticket.getAssigneeId());
        }
        if (ticket.getTitle() != null) {
            updateTicket.setTitle(ticket.getTitle());
        }
        if (ticket.getDescription() != null) {
```

```
            updateTicket.setDescription(ticket.getDescription());
        }
        if (ticket.getStatus() != null) {
            updateTicket.setStatus(ticket.getStatus());
        }
        if (ticket.getTicketType() != null) {
            updateTicket.setTicketType(ticket.getTicketType());
        }
        if (ticket.getPriority() != null) {
            updateTicket.setPriority(ticket.getPriority());
        }
        if (ticket.getEstimate() != 0) {
            updateTicket.setEstimate(ticket.getEstimate());
        }
        if (ticket.getUnitsOfMeasure() != null) {
            updateTicket.setUnitsOfMeasure(ticket.getUnitsOfMeasure());
        }

        updateTicket.setLastUpdatedTimeStamp(LocalDateTime.now());
        TicketModel ticketModel = mapper.map(updateTicket, TicketModel.class);

        return mapper.map(
                ticketRepository.save(ticketModel), TicketDTO.class
        );
    }
}
```

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);
    }
}
```