Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка програмного забезпечення управління проєктами за методологією Scrum з використанням PHP

Виконав: студент IV курсу, групи СП-42
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва спеціальності)

_____ Рубльов А.І.
(підпис) (прізвище та ініціали)

Керівник _____ Михалик Д.М.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Стоянов Ю.М.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Петрик М.Р.
(підпис) (прізвище та ініціали)

Рецензент _____ _____
(підпис) (прізвище та ініціали)

Тернопіль 2026

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Петрик М.Р.
(підпис) (прізвище та ініціали)
«_____» _____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня _____ бакалавр
(назва освітнього ступеня)

за спеціальністю 121 «Інженерія програмного забезпечення»
(шифр і назва спеціальності)

студенту Рубльов Андрій Ігорович
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка програмного забезпечення управління проєктами за методологією Scrum з використанням PHP

Керівник роботи Михалик Д.М. к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «__» _____ 2026 року № _____

2. Термін подання студентом завершеної роботи _____ 2026 р.

3. Вихідні дані до роботи Технічне завдання на розробку програмного забезпечення управління проєктами.

4. Зміст роботи (перелік питань, які потрібно розробити)
Сформулювати, погодити та затвердити тему кваліфікаційної роботи; розробити та затвердити завдання;
Аналіз предметної області: огляд принципів Scrum, аналіз аналогів, обґрунтування вибору.
Проєктування програмної системи (життєвий цикл розробки, проєктування архітектури MVC, проєктування бази даних, UML-діаграма класів, розробка інтерфейсу користувача).
Реалізація та тестування системи (модульне та інтеграційне тестування, розгортання системи, навантажувальне тестування, автоматизація процесів та резервне копіювання).
Безпека життєдіяльності, основи охорони праці.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Ілюстративна частина кваліфікаційної роботи оформляється у вигляді слайдів презентації. Графічний матеріал висвітлює основні етапи роботи та містить: тему, мету, предмет, об'єкт дослідження та сформульовані завдання; короткий аналіз актуальності розробки Scrum-системи; архітектуру програмної системи; UML-діаграми; логічну модель бази даних; опис реалізованого інтерфейсу та результати тестування програмного забезпечення.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання

_____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Вибір та погодження теми з керівником. Створення та затвердження технічного завдання	10.02.26 - 14.02.26	виконано
2.	Обґрунтування вибору технологічного стека	16.02.26 -	
	. та написання Розділу 1.	28.02.26	виконано
3.	Проєктування архітектури системи (MVC), структури	01.03.26 -	
	бази даних та розробка UML-діаграм.	20.03.26	виконано
4.	Оформлення Розділу 2 (Проєктування	21.03.26 -	
	програмної системи).	31.03.26	виконано
5.	Програмна реалізація серверної (Laravel) та клієнтської	01.04.26 -	
	частини, налаштування Drag-and-Drop функціоналу.	30.04.26	виконано
6.	Проведення тестування, налаштування CI/CD	01.05.26 -	
	та бекапів. Написання Розділу 3.	15.05.26	виконано
7.	Виконання та узгодження Розділу 4 (Безпека	16.05.26 -	
	життєдіяльності, основи охорони праці).	25.05.26	виконано
8.	Формування загальних висновків, вступу, анотації	26.05.26-	
	та списку використаних джерел.	02.06.26	виконано
9.	Оформлення пояснювальної записки згідно з вимогами	03.06.26 -	
	нормоконтролю, підготовка графічного матеріалу.	08.06.26	виконано
10.	Перевірка роботи на плагіат, підписання роботи у керівника.	10.06.26 - 20.06.26	виконано
11.	Підготовка презентації, доповіді та захист		
	кваліфікаційної роботи.		виконано

Студент _____ Рубльов А.І.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Михалик Д.М
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Рубльов Андрій Ігорович. Розробка програмного забезпечення для управління проєктами за методологією Scrum. Кваліфікаційна робота бакалавра, кафедра програмної інженерії, група СП-42, спец. 121 Інженерія програмного забезпечення. ТНТУ ім. І. Пулюя, Тернопіль, 2026.

Робота: 63 с., 19 рис., додатків 2 , таблиць 8, 30 джерел.

Ключові слова: управління проєктами, Scrum, вебдодаток, PHP, Laravel, бази даних, клієнт-серверна архітектура, трекінг завдань.

Мета – розробка власного вебдодатка для управління проєктами на основі гнучкої методології Scrum для забезпечення ефективної взаємодії в командах розробників, з використанням фреймворку Laravel.

Об’єкт: програмно-алгоритмічні засоби та процес управління проєктами за методологією Scrum.

Предмет: архітектура, методи та інструменти реалізації клієнт-серверної системи для трекінгу завдань та ведення проєктів на базі PHP та Laravel.

У роботі проаналізовано принципи гнучких методологій розробки програмного забезпечення, зокрема Scrum, розглянуто існуючі аналоги на ринку систем управління проєктами. Спроєктовано та розроблено клієнт-серверний вебдодаток із використанням мови програмування PHP та фреймворку Laravel. Створено реляційну базу даних, реалізовано ключові модулі: управління завданнями (спринти, беклог, Scrum-дошки), розмежування прав користувачів та управління проєктами. Проведено тестування розробленого функціоналу.

Удосконалено підхід до інтеграції інструментів Scrum в єдине легковагове вебсередовище на базі Laravel, що знижує поріг входження для нових команд та оптимізує процеси планування спринтів.

Практичне значення: Розроблено функціональний вебдодаток, готовий до впровадження, який дозволяє ефективно управляти життєвим циклом розробки програмного продукту, відстежувати задачі та організовувати командну роботу.

ABSTRACT

Rublov Andriy Igorovich. Development of project management software using the Scrum methodology. Bachelor's qualification work, Department of Software Engineering, Group SP-42, Specialization 121 Software Engineering. I. Pulyuy National Technical University of Ternopil, 2026.

Thesis: 63 p., 19 fig., 2 appendices, table 8, 30 sources.

Keywords: project management, Scrum, web application, PHP, Laravel, databases, client-server architecture, task tracking.

The purpose is to develop a proprietary project management web application based on the agile Scrum methodology to ensure effective collaboration in development teams, using the Laravel framework.

Object: software-algorithmic tools and the process of project management using the Scrum methodology.

Subject: architecture, methods, and tools for implementing a client-server system for task tracking and project management based on PHP and Laravel.

The thesis analyzes the principles of agile software development methodologies, specifically Scrum, and reviews existing project management systems on the market. A client-server web application was designed and developed using the PHP programming language and the Laravel framework. A relational database architecture was created, and key modules were implemented: task management (sprints, backlog, Scrum boards), user role separation, and project management. The developed functionality was tested.

The approach to integrating Scrum tools into a single lightweight web environment based on Laravel has been improved, lowering the entry barrier for new teams and optimizing sprint planning processes.

Practical significance: A functional, ready-to-deploy web application has been developed that allows for effective management of the software development life cycle, task tracking, and team collaboration organization.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД – База даних

КРБ – Кваліфікаційна робота бакалавра

ПЗ – Програмне забезпечення

СУБД – Система управління базами даних

API (Application Programming Interface) – Інтерфейс прикладного програмування

CRUD (Create, Read, Update, Delete) – Базові функції управління даними (створення, читання, оновлення, видалення)

CSS (Cascading Style Sheets) – Каскадні таблиці стилів

ER-діаграма (Entity-Relationship diagram) – Діаграма «сутність-зв'язок», логічна модель бази даних

HTTP (HyperText Transfer Protocol) – Протокол передавання гіпертексту

JSON (JavaScript Object Notation) – Текстовий формат обміну даними

MVC (Model-View-Controller) – Архітектурний шаблон розробки програмного забезпечення (Модель-Представлення-Контролер)

MVP (Minimum Viable Product) – Мінімально життєздатний продукт

ORM (Object-Relational Mapping) – Об'єктно-реляційне відображення для зв'язування баз даних з об'єктно-орієнтованими мовами програмування

PHP (Hypertext Preprocessor) – Скриптова мова програмування загального призначення

SaaS (Software as a Service) – Модель поширення програмного забезпечення (програмне забезпечення як послуга)

SDLC (Software Development Life Cycle) – Життєвий цикл розробки програмного забезпечення

UI (User Interface) – Інтерфейс користувача

UML (Unified Modeling Language) – Уніфікована мова моделювання

UX (User Experience) – Користувацький досвід

ЗМІСТ

ВСТУП

У сучасній індустрії розробки програмного забезпечення ефективна організація командної роботи є критичним фактором успіху. Більшість IT-компаній використовують гнучкі методології розробки (Agile), серед яких найпопулярнішим є Scrum. Він дозволяє командам швидко адаптуватися до змін, постачати продукт ітеративно та оптимізувати процеси управління вимогами (Requirements management). Незважаючи на велику кількість існуючих систем управління проєктами (Jira, Asana, Trello), багато з них мають суттєві недоліки для певних категорій користувачів. Великі корпоративні системи часто є перевантаженими зайвим функціоналом, вимагають тривалого навчання персоналу та мають високу вартість ліцензій. З іншого боку, прості інструменти не завжди підтримують специфічні артефакти Scrum (беклог продукту, спринти, оцінку завдань).

Додатковим фактором, що зумовлює актуальність теми, є зростаюча потреба IT-компаній у суверенітеті та безпеці даних. Використання публічних хмарних сервісів часто несе ризики витоку інтелектуальної власності компаній, які прагнуть зберігати конфіденційні дані на власних серверах. З огляду на це, розробка власної, легковагової та інтуїтивно зрозумілої вебсистеми управління проєктами за методологією Scrum є актуальною задачею. Використання сучасної мови програмування PHP та фреймворку Laravel дозволяє створити надійний, безпечний та легко масштабований продукт із можливістю швидкого розгортання на власних (On-Premise) потужностях, що відповідає сучасним тенденціям оптимізації та автоматизації програмних рішень.

Метою кваліфікаційної роботи є розробка вебдодатка для управління проєктами на основі методології Scrum, що забезпечить ефективне планування спринтів, відслідковування завдань та координацію роботи команди. Для досягнення поставленої мети потрібно вирішити такі завдання: проаналізувати предметну область, принципи методології Scrum та існуючі аналоги систем управління проєктами, спроєктувати загальну архітектуру клієнт-серверного вебдодатка, розробити структуру реляційної бази даних для зберігання інформації

про користувачів, проєкти, спринти та завдання, реалізувати серверну частину з використанням фреймворку Laravel, включаючи бізнес-логіку та систему автентифікації, розробити зручний інтерфейс користувача (фронт-енд) для взаємодії з системою (робота з дошками, беклогом), провести тестування розробленого програмного забезпечення та підготувати його до впровадження.

Об’єкт дослідження - процес управління проєктами та командною взаємодією за методологією Scrum.

Предмет дослідження - архітектура, моделі даних, методи та інструменти розробки вебсистеми для трекінгу завдань та ведення проєктів на базі PHP та фреймворку Laravel.

Цінність отриманих результатів полягає в удосконаленні архітектурного підходу до побудови систем управління проєктами. На відміну від існуючих монолітних рішень, запропоновано оптимізовану клієнт-серверну модель взаємодії на базі фреймворку Laravel, яка за рахунок використання технології асинхронних запитів (Fetch API) та мінімізації надлишкового рендерингу сторінок на стороні сервера дозволяє суттєво знизити навантаження на апаратні ресурси при колективній роботі зі Scrum-дошкою. В результаті знижено поріг входження для нових команд та оптимізовано процеси планування спринтів.

Розроблено функціональний вебдодаток, який може бути впроваджений у реальні процеси розробки невеликих та середніх IT-команд. Система готова до розгортання на серверах компанії, забезпечуючи потреби в управлінні життєвим циклом розробки програмного продукту, від створення беклогу до завершення спринту.

Методи дослідження. Для вирішення поставлених завдань і досягнення мети в роботі використано комплекс наукових та інженерних методів:

- методи системного та порівняльного аналізу - для дослідження предметної області, вивчення існуючих програмних рішень та формулювання функціональних вимог до системи;

- методи об'єктно-орієнтованого проєктування та UML-моделювання - для розробки архітектури вебдодатка, побудови логіки взаємодії компонентів та моделювання бізнес-процесів;
- методи теорії реляційних баз даних - для проєктування структури бази даних, нормалізації таблиць та забезпечення цілісності інформації;
- методи програмної інженерії та тестування - для безпосередньої реалізації серверної та клієнтської частин додатка (написання коду, рефакторинг, відлагодження) та перевірки коректності роботи функціоналу.

Апробація результатів роботи.

Основні положення, технічні рішення та результати кваліфікаційної роботи доповідалися й обговорювалися на IX міжнародній студентсько науковій - технічній конференції "ПРИРОДНИЧІ ТА ГУМАНІТАРНІ НАУКИ. АКТУАЛЬНІ ПИТАННЯ", 24-25 квітня 2026 р.

За матеріалами кваліфікаційної роботи опубліковано тези доповідей: РОЗРОБКА ІНТЕРАКТИВНОЇ SCRUM-ДОШКИ У ВЕБСИСТЕМІ УПРАВЛІННЯ ПРОЄКТАМИ / Рубльов А.І., Михалик Д.М. // Матеріали збірника тез конференції IX МІЖНАРОДНА студентська науково - технічна конференція "ПРИРОДНИЧІ ТА ГУМАНІТАРНІ НАУКИ. АКТУАЛЬНІ ПИТАННЯ" - Тернопіль: ТНТУ, 2026. - с. 227-228.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

У цьому розділі проводиться комплексний аналіз предметної області управління IT-проєктами. Розглядаються базові принципи та артефакти методології Scrum, здійснюється порівняльний огляд існуючих систем трекінгу завдань, а також наводиться обґрунтування вибору технологічного стека для програмної реалізації розроблюваного продукту.

1.1 Огляд методології Scrum та специфіка управління IT-проєктами

У сучасній індустрії розробки програмного забезпечення класичні каскадні моделі управління проєктами, такі як Waterfall, поступово поступаються місцем гнучким методологіям (Agile). Головною причиною цього є високий ступінь невизначеності на старті проєктів, часті зміни вимог з боку замовників та необхідність швидкого випуску робочого продукту на ринок.

Серед сімейства Agile-фреймворків найбільш поширеним та ефективним є Scrum. Згідно з офіційним посібником Scrum Guide, Scrum — це легковаговий фреймворк, який допомагає людям, командам та організаціям створювати цінність за допомогою адаптивних рішень для комплексних проблем [8,18]. Він базується на принципах емпіризму, знання ґрунтуються на досвіді, та ощадливого мислення (Lean thinking).

Основою Scrum є ітеративний та інкрементальний підхід. Весь процес розробки розбивається на короткі фіксовані проміжки часу, які називаються спринтами (Sprints). Зазвичай тривалість спринту становить від одного до чотирьох тижнів. Результатом кожного спринту має бути потенційно готовий до випуску інкремент продукту.

Емпіричний підхід, на якому ґрунтується Scrum, реалізується через три ключові стовпи: прозорість, інспекцію та адаптацію [18]. Прозорість гарантує, що всі аспекти процесу розробки є видимими та зрозумілими для тих, хто несе відповідальність за результат. Інспекція вимагає регулярної перевірки артефактів

Scrum та прогресу команди для своєчасного виявлення небажаних відхилень. Адаптація дозволяє команді негайно коригувати робочий процес, якщо інспекція показує, що певні аспекти виходять за межі допустимих норм. Впровадження таких методологій також суттєво оптимізує процеси управління вимогами (Requirements management) в умовах гнучкої розробки [24].

Особливість управління IT-проєктами полягає у високому рівні абстракції програмного продукту та постійній еволюції технологій. На відміну від класичного виробництва, де вимоги можна чітко зафіксувати ще на стадії проєктування, розробка програмного забезпечення часто стикається з тим, що замовник усвідомлює свої реальні потреби лише після тестування перших робочих прототипів. Використання Scrum дозволяє ефективно зменшити ці ризики, оскільки кожен спринт фактично є самостійним мікропроєктом. Це дає змогу регулярно отримувати зворотний зв'язок від стейкхолдерів і гнучко змінювати пріоритети розробки без шкоди для бюджету та архітектури.

Рисунок 1.1 – Життєвий цикл методології Scrum

Для ефективного функціонування фреймворку, Scrum чітко визначає три основні ролі в команді:

- Product Owner (Власник продукту) — особа, яка відповідає за максимізацію цінності продукту. Він формує вимоги, пріоритезує їх та управляє беклогом продукту.
- Scrum Master (Скрам-майстер) — спеціаліст, який відповідає за те, щоб Scrum розуміли і застосовували правильно. Він допомагає команді усувати перешкоди та підвищувати ефективність.
- Developers (Команда розробників) — крос-функціональна група спеціалістів, які безпосередньо виконують роботу зі створення інкременту під час спринту.

Процес управління проєктом за методологією Scrum спирається на три ключові артефакти, які забезпечують прозорість та контроль:

- Product Backlog (Беклог продукту) — впорядкований список усього, що може знадобитися в продукті. Це єдине джерело вимог для будь-яких змін.
- Sprint Backlog (Беклог спринту) — набір завдань з Product Backlog, вибраних для виконання у поточному спринті, а також план їх реалізації.
- Increment (Інкремент) — сума всіх елементів беклогу продукту, завершених під час поточного та всіх попередніх спринтів.

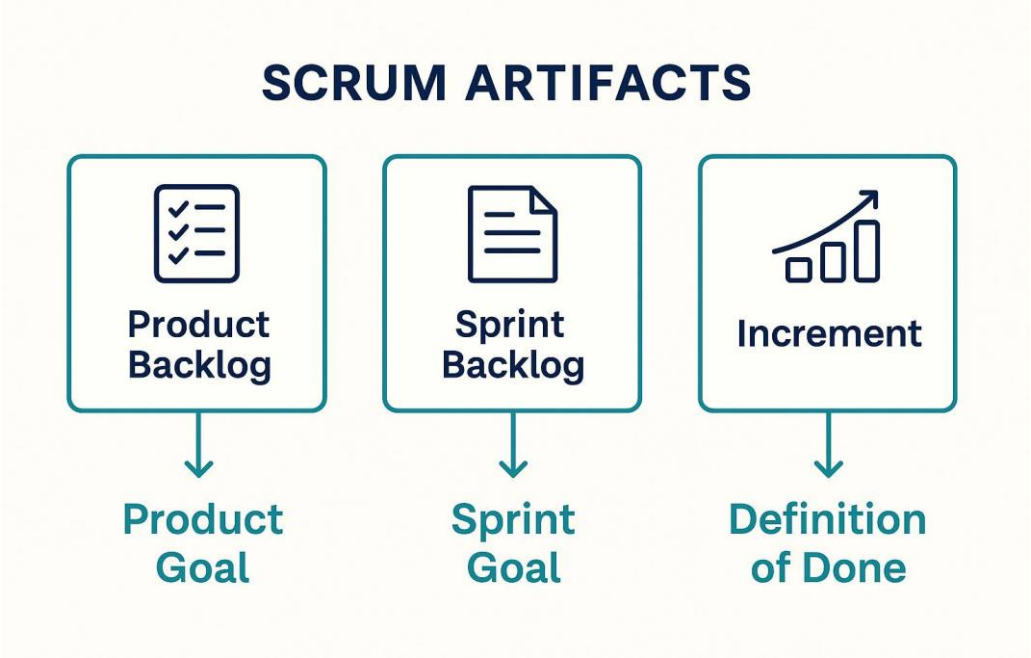


Рисунок 1.2 – Візуалізація ключових артефактів методології Scrum

Всі ці процеси вимагають суворої дисципліни та регулярної синхронізації команди через систему подій: планування спринту (Sprint Planning), щоденні зустрічі (Daily Scrum), огляд спринту (Sprint Review) та ретроспективу (Sprint Retrospective).

Зважаючи на велику кількість артефактів, ролей та постійну зміну статусів завдань, ручне управління процесом Scrum є неефективним і призводить до втрати інформації. Виникає об'єктивна необхідність у використанні спеціалізованого програмного забезпечення — систем управління проєктами (Project Management Systems, або Issue Trackers). Такі системи повинні забезпечувати створення ієрархії завдань, ведення беклогів, візуалізацію робочого процесу через дошки (Scrum/Kanban boards) та збір статистики для аналізу продуктивності команди (наприклад, графік згоряння завдань — Burndown chart).

Впровадження таких інформаційних систем стає особливо критичним в умовах сучасних реалій розробки, де переважають розподілені (віддалені) команди і фізична взаємодія біля традиційної офісної дошки є неможливою.

1.2 Аналіз існуючих систем управління проєктами та їхні недоліки

На сучасному ринку програмного забезпечення представлено десятки інструментів для управління проєктами (Issue Trackers), які підтримують гнучкі методології [5]. Для визначення функціональних вимог до власної системи було проведено порівняльний аналіз трьох найпопулярніших рішень: Jira (від компанії Atlassian), Trello та Asana.

Таблиця 1.1 – Порівняльна характеристика систем управління проєктами

Критерій порівняння	**Jira Software**	**Trello**	**Asana**	**Розроблювана система**
Підтримка Scrum (спринти, беклог)	Повна підтримка	Відсутня (потребує сторонніх плагінів)	Часткова (через списки завдань)	Повна підтримка
Простота інтерфейсу	Складний, потребує тривалого навчання	Інтуїтивно зрозумілий	Зручний, структурований	Максимально простий та інтуїтивний
Можливість локального розгортання (On-Premise)	Відсутня (доступні лише хмарні Cloud-версії)	Відсутня	Відсутня	Підтримується (розгортання на власному сервері)
Наявність візуальної дошки завдань	Наявна	Наявна	Наявна	Наявна
Функціональна перевантаженість	Висока (багато надлишкового функціоналу)	Відсутня	Помірна	Відсутня (реалізовано лише необхідні інструменти)
Вартість використання	Висока вартість комерційних ліцензій	Базова версія безкоштовна	Базова версія безкоштовна	Безкоштовна у використанні

Jira Software

Jira є фактичним корпоративним стандартом для управління розробкою програмного забезпечення за методологією Scrum. Система пропонує вичерпний набір інструментів: ведення беклогів, дошки спринтів, розширену аналітику (Burndown charts, Velocity charts), гнучке налаштування прав доступу та інтеграцію з системами контролю версій (Git). Недоліки:

- Високий поріг входження: інтерфейс Jira є перевантаженим, а налаштування робочих процесів (workflows) вимагає значних витрат часу та, часто, наявності окремого адміністратора системи.
- Надмірність функціоналу: для невеликих або новостворених команд більшість функцій залишаються невикористаними, проте ускладнюють навігацію.
- Вартість та ресурсоємність: комерційні ліцензії є дорогими, а локальне (Self-hosted) розгортання вимагає значних серверних потужностей (використовує стек Java).

Trello — це популярний інструмент, який базується на парадигмі Kanban-дошок. Його головна перевага полягає в максимальній простоті: користувач створює колонки та переміщує між ними картки завдань. Інтерфейс є інтуїтивно зрозумілим і не потребує навчання. Недоліки:

- Відсутність нативної підтримки Scrum: у базовій версії Trello немає вбудованих понять "спринт", "беклог продукту" чи "оцінка складності (Story Points)".
- Масштабованість: система чудово працює для простих списків завдань, але коли кількість карток та учасників зростає, дошка перетворюється на неструктурований хаос.
- Залежність від плагінів: для реалізації Scrum-артефактів необхідно підключати сторонні модулі (Power-Ups), які часто є платними і порушують цілісність екосистеми.

Asana позиціонується як універсальний менеджер завдань для бізнесу, не обов'язково пов'язаного з IT-розробкою. Система пропонує зручне відображення завдань у вигляді списків, дошок, календарів та діаграм Ганта. Недоліки:

- Брак специфічних IT-інструментів: Asana менш адаптована саме під процес розробки програмного забезпечення. У ній складніше реалізувати типовий життєвий цикл коду (наприклад, зв'язок гілок коду з конкретними завданнями).
- Обмеження безкоштовної версії: ключові функції для гнучкого управління (наприклад, кастомні поля для оцінки завдань) доступні лише у преміум-підписках.
- Хмарна закритість (SaaS): система не надає можливості розгортання на власних серверах компанії (On-Premises), що є критичним недоліком для проєктів із високими вимогами до конфіденційності даних.

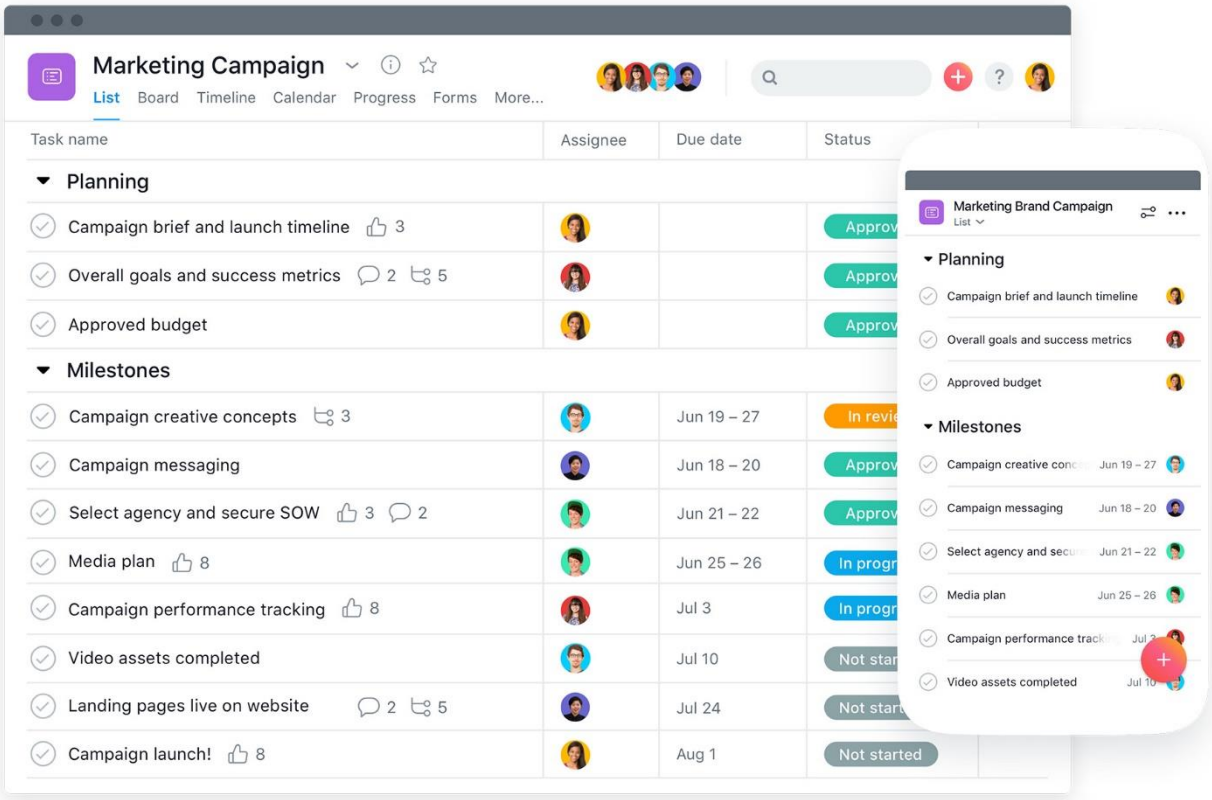


Рисунок 1.3 – Інтерфейс та структуризація завдань в Asana

Проведений аналіз показує, що на ринку існує певна поляризація. З одного боку, існують потужні, але перевантажені і складні системи (Jira). З іншого — прості, але недостатньо функціональні для повноцінного Scrum інструменти (Trello).

Для багатьох команд оптимальним рішенням є "золота середина": система, яка має чітку спеціалізацію під Scrum, але при цьому залишається легковаговою, має простий інтерфейс і може бути розгорнута на власному сервері. Саме це обґрунтовує доцільність розробки власного вебдодатка, архітектура якого буде позбавлена надмірності, але повною мірою покриватиме потреби розробників.

1.3 Обґрунтування вибору технологічного стека розробки

Вибір правильного технологічного стека є критично важливим етапом проєктування програмного забезпечення, оскільки він визначає архітектуру системи. Було обрано клієнт-серверний підхід із використанням сучасних вебтехнологій.

Серверна частина (Back-end)

В якості основної мови програмування серверної логіки обрано PHP. На сьогоднішній день PHP є однією з найпопулярніших мов для веброзробки [1, 9], яка відзначається високою швидкістю роботи та широкою підтримкою хостинг-провайдерів, що спрощує процес розгортання додатку.

Для забезпечення надійної архітектури було обрано фреймворк Laravel. Використання сучасних вебтехнологій дозволяє автоматизувати багато рутинних процесів розробки програмного забезпечення [25]. Цей вибір обґрунтовується такими перевагами:

- Архітектурний шаблон MVC (Model-View-Controller): Laravel змушує розробника розділяти бізнес-логіку, роботу з даними та відображення інтерфейсу, що робить код структурованим та легким для підтримки [4,7].
- Eloquent ORM: Вбудована система об'єктно-реляційного відображення дозволяє зручно працювати з базою даних за допомогою об'єктно-орієнтованого

синтаксису замість написання сирих SQL-запитів [10]. Це ідеально підходить для управління складними зв'язками (наприклад, між проєктами, спринтами, завданнями та користувачами).

- Вбудовані механізми безпеки: Фреймворк "з коробки" надає захист від поширених вебвразливостей, таких як SQL-ін'єкції, підробка міжсайтових запитів (CSRF) та міжсайтовий скриптинг (XSS).
- Маршрутизація та автентифікація: Laravel має потужну систему роутингу та готові інструменти для реєстрації, авторизації і управління сесіями користувачів.

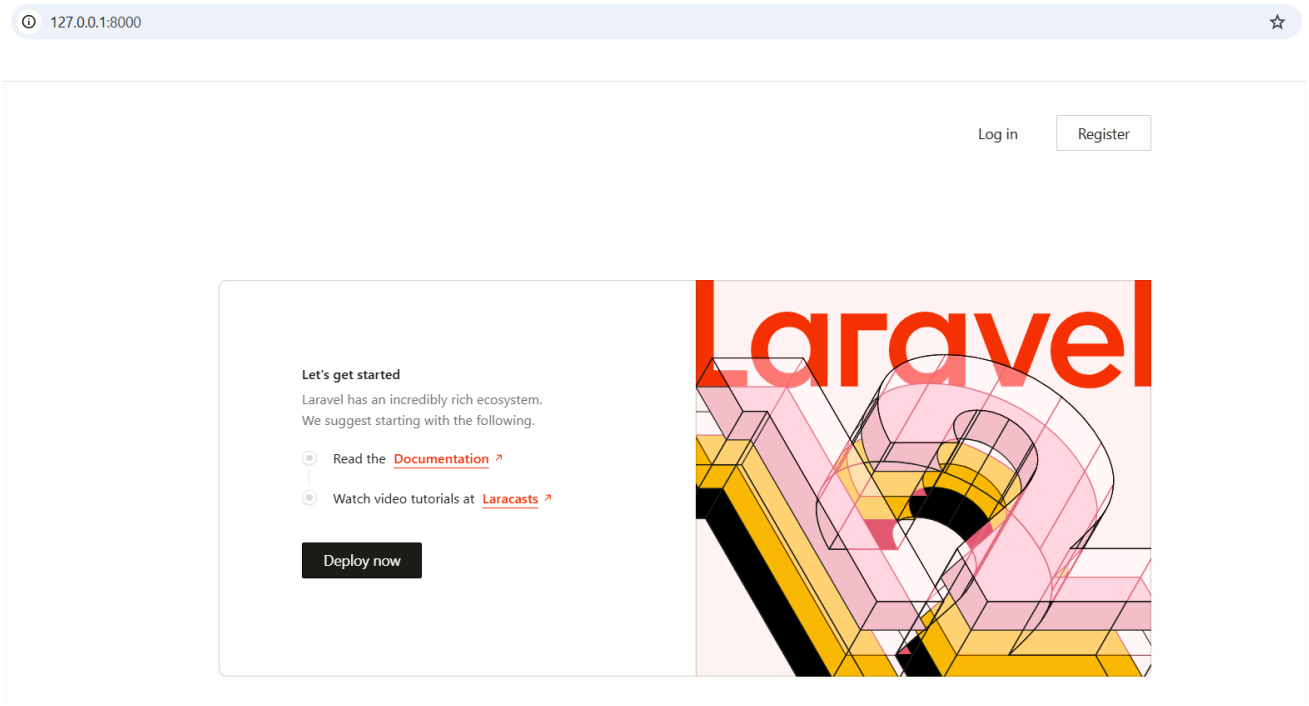


Рисунок 1.4 – Початкова сторінка створюваного додатку при використанні фреймворку Laravel

Оскільки для реалізації серверної логіки раніше розглядалися три популярні технологічні стеки: Python (Django/FastAPI), Node.js (Express/NestJS) та PHP (Laravel/Symfony), для кращого розуміння особливостей розглянемо порівняльну таблицю цих стеків.

Таблиця 1.2 – Порівняльний аналіз технологій серверної частини

Критерій	**Node.js (Express)**	**Python (Django)**	**PHP (Laravel)**
Парадигма	Асинхронна (Event Loop)	Синхронна / Асинхронна	Синхронна (з підтримкою черг)
Швидкість розробки	Середня (потребує збирання багатьох пакетів)	Висока ("Батарейки в комплекті")	Висока (Багатий вбудований функціонал)
Робота з реляційними БД	Sequelize, TypeORM (складна конфігурація)	Django ORM (потужна, але специфічна)	Eloquent ORM (інтуїтивна та дуже потужна)
Архітектурний підхід	Мікросервіси, REST API	Моноліт (MVC)	Моноліт (MVC), модульна структура
Простота розгортання	Потребує налаштування процес-менеджерів (PM2, Docker)	Потребує WSGI/ASGI серверів (Gunicorn)	Максимально проста (широка підтримка хостингів)

База даних

Враховуючи, що система управління проєктами працює з чітко структурованими даними, між якими існують жорсткі зв'язки (наприклад, одне завдання належить лише одному проєкту та одному спринту), найкращим вибором є використання реляційної системи управління базами даних MySQL. Вона забезпечує високу продуктивність, підтримує транзакції для збереження цілісності даних та легко інтегрується з Laravel [16, 21].

Клієнтська частина (Front-end)

Для реалізації інтерфейсу користувача було обрано стандартний стек вебтехнологій: HTML5, CSS3 та JavaScript.

Відображення даних генерується за допомогою вбудованого в Laravel шаблонізатора Blade, що дозволяє динамічно формувати сторінки на стороні сервера. Для стилізації інтерфейсу використовується CSS-фреймворк Tailwind CSS, що дозволяє створити адаптивний та сучасний дизайн, який коректно відображається на різних пристроях [20].

2 ПРОЄКТУВАННЯ СИСТЕМИ

Даний розділ присвячено етапам проєктування архітектури системи управління проєктами. Описано обраний ітеративний процес розробки та змодельовано ключові алгоритми взаємодії компонентів вебдодатка.

2.1 Вибір процесу розробки

Процес створення будь-якого програмного забезпечення вимагає чіткого дотримання обраної методології управління життєвим циклом (SDLC) [12]. Враховуючи специфіку розроблюваного забезпечення, було проведено порівняльний аналіз каскадної та гнучкої методологій. У таблиці 2.1 наведено порівняння основних характеристик у контексті створення веб-застосунку.

Таблиця 2.1 – Порівняльний аналіз методологій Waterfall та Agile

Критерій	**Waterfall (Каскадна)**	**Agile (Гнучка)**
Гнучкість до змін	Низька. Зміни важко вносити після завершення етапу проєктування	Висока. Вимоги можуть адаптуватися на кожній ітерації
Отримання результату	На фінальній стадії проєкту (монолітний реліз)	Поступове (інкрементне) отримання працюючого продукту
Взаємодія з користувачем	На етапах збору вимог та фінального тестування	Постійна протягом усього процесу розробки
Ризики	Високі (неправильне розуміння вимог виявляється занадто пізно)	Низькі (коригування вектора розробки відбувається регулярно)

Як свідчать результати порівняння (табл. 2.1), гнучка методологія Agile є значно ефективнішою для розробки сучасних SaaS-рішень. Зважаючи на те, що архітектура та бізнес-логіка розроблюваної системи базуються на принципах

Scrum, застосування саме цього фреймворку для її безпосереднього створення є концептуально обґрунтованим.

Відповідно до обраної методології та сучасних стандартів конструювання ПЗ [26], процес реалізації програмного забезпечення не був монолітним. Замість цього весь обсяг інженерних робіт було декомпозивано та розподілено на серію послідовних робочих ітерацій, кожна з яких завершувалася випуском робочого інкременту продукту. Структурна схема застосованого життєвого циклу розробки представлена на рисунку 2.1.

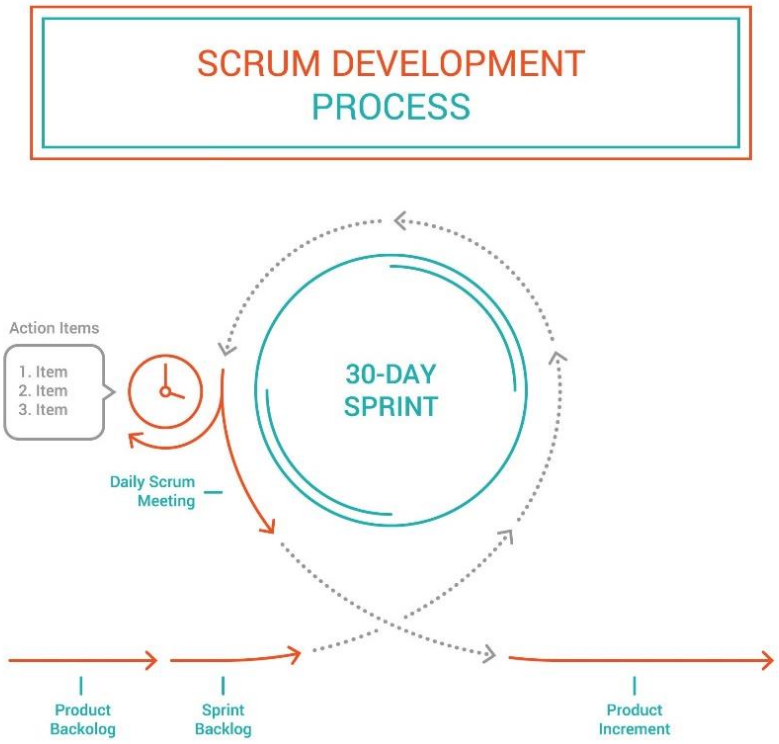


Рисунок 2.1 – Схема процесу розробки програмного забезпечення за методологією Scrum

Відповідно до рис. 2.1, увесь життєвий цикл розробки був побудований навколо артефактів Scrum. Роль Product Owner (Власника продукту) та Development Team (Команди розробки) виконував безпосередньо розробник програмного забезпечення, що дозволило оптимізувати процес прийняття архітектурних рішень.

Весь процес імплементації програмного коду було поділено на чотири основні спринти (тривалістю від одного до двох тижнів кожен). Детальний розподіл завдань та інкрементів за спринтами наведено в таблиці 2.2.

Таблиця 2.2 – Етапи реалізації програмної системи (Спринти)

Ітерація	**Основна мета спринту**	**Склад робіт (Backlog Items)**	**Отриманий інкремент**
Sprint 1	Фундамент системи та безпека	1. Налаштування середовища Laravel. 2. Проєктування базових міграцій БД. 3. Реалізація реєстрації та автентифікації.	Базовий каркас програми, готова система логіну/реєстрації користувачів.
Sprint 2	Ізоляція даних (Workspaces)	1. Створення таблиць просторів та ролей. 2. Реалізація зв'язку Many-to-Many. 3. Написання Middleware для захисту маршрутів.	Можливість створювати власні простори та переключатися між ними. Ролі Admin/Member.
Sprint 3	Логіка беклогу та проєктів	1. Реалізація архітектури проєктів. 2. Створення CRUD для завдань (Tasks). 3. Категоризація (Story/Task/Bug).	Робоча сторінка беклогу. Користувач може створювати та оцінювати завдання.
Sprint 4	Scrum-дошка та інтерактивність	1. Проєктування сутності "Спринт". 2. Реалізація Канбан-дошки (UI). 3. Підключення Drag-and-Drop зміни статусів.	Повністю робочий MVP системи. Можливість візуально керувати завданнями у спринті.

Використання ітеративного підходу, показаного в табл. 2.2, гарантувало, що після завершення кожного спринту система залишалася в працездатному стані. Це також дозволило адаптувати структуру бази даних "на льоту". Наприклад, необхідність створення поля assignee_id була виявлена на етапі реалізації Scrum-дошки у четвертому спринті, що було легко реалізовано завдяки гнучкості механізму Laravel Migrations, не порушуючи логіку попередніх етапів.

2.2 Проєктування архітектури системи

Розроблене програмне забезпечення побудоване за класичною клієнт-серверною архітектурою типу «Тонкий клієнт». Уся складна бізнес-логіка, обробка даних, управління сесіями та взаємодія з базою даних виконуються на стороні сервера (Backend). Клієнтська частина (Frontend) відповідає виключно за рендеринг графічного інтерфейсу та передачу команд користувача через протокол HTTP/HTTPS.

Основою програмної архітектури серверної частини є патерн MVC (Model-View-Controller) [11]. Цей архітектурний шаблон дозволяє чітко розділити дані, інтерфейс користувача та керуючу логіку на три незалежні рівні, що суттєво полегшує подальше масштабування, тестування та підтримку програмного коду системи.

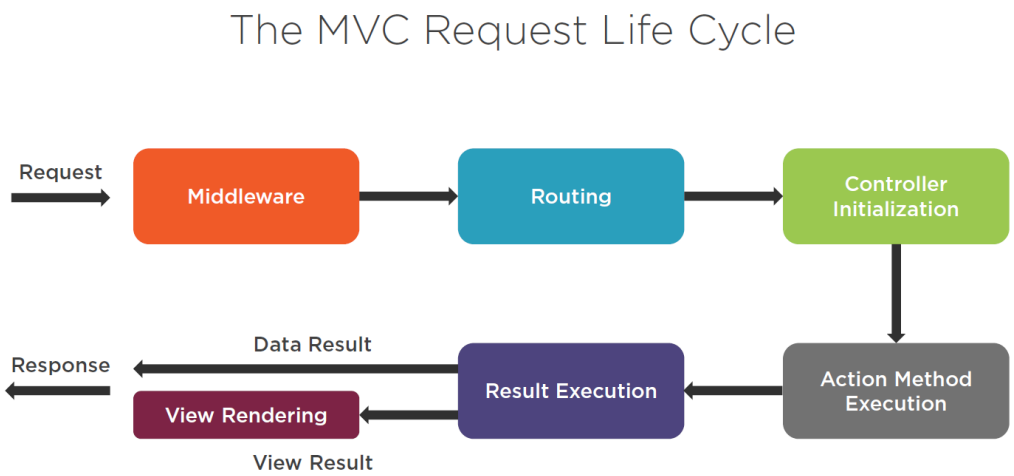


Рисунок 2.2 – Архітектура взаємодії компонентів системи за патерном MVC

Відповідно до рис. 2.2, взаємодія компонентів у розробленій системі відбувається за наступним життєвим циклом (Request Lifecycle):

1. Маршрутизація (Routing) та Middleware: Усі вхідні запити від користувача спочатку потрапляють до єдиної точки входу (Front Controller), після чого обробляються маршрутизатором (Router). Перед передачею запиту до основного контролера він проходить через шар проміжних фільтрів (Middleware). У розробленій системі активно використовуються фільтри auth (для перевірки наявності активної сесії користувача) та спеціалізовані перевірки прав доступу, що гарантує безпеку даних у робочих просторах.

2. Контролер (Controller): Отримавши валідований запит, відповідний контролер (наприклад, TaskController або SprintController) перебирає на себе управління бізнес-логікою. Він обробляє вхідні параметри, визначає необхідні дії та звертається до рівня абстракції даних (Моделей) для виконання операцій запису або читання.

3. Модель (Model): Забезпечує об'єктно-орієнтовану абстракцію над базою даних за допомогою патерну Active Record (реалізованого через ORM Eloquent). Моделі (Workspace, Project, Task) автоматично перетворюють PHP-об'єкти та їхні методи на безпечні параметризовані SQL-запити до бази даних MySQL, що повністю нівелює ризики SQL-ін'єкцій.

4. Представлення (View): Отримавши масив оброблених даних від контролера, система передає їх у Представлення. Шаблонізатор Blade компілює фінальний HTML-документ, динамічно інтегруючи отримані дані (наприклад, рендерить картки завдань у циклах), і відправляє сформовану сторінку назад у браузер користувача.

Крім класичного синхронного MVC-циклу, архітектура розробленої системи підтримує гібридний підхід. Для реалізації технології Drag-and-Drop на Канбан-дошці були спроєктовані окремі API-маршрути. У цьому випадку контролер працює без рівня View, повертаючи клієнту лише серіалізовану структуру даних у форматі JSON, що дозволяє JavaScript-клієнту оновлювати стан інтерфейсу асинхронно, без повного перезавантаження сторінки.

Враховуючи те, що розроблювана система функціонує за моделлю SaaS (Software as a Service), критично важливою вимогою є забезпечення суворої ізоляції даних між різними робочими просторами (Tenant Isolation). Для вирішення цієї задачі на архітектурному рівні було впроваджено модель управління доступом на основі ролей (Role-Based Access Control – RBAC).

Архітектура безпеки застосунку реалізована через використання двох ключових механізмів фреймворку Laravel:

1. Middleware (Проміжне програмне забезпечення): Забезпечує фільтрацію HTTP-запитів на рівні маршрутизатора. Застосовується для глобальної перевірки автентифікації сесії користувача перед наданням доступу до будь-яких захищених маршрутів системи.

2. Policies (Політики авторизації): Інкапсулюють логіку перевірки прав доступу до конкретних ресурсів (моделей). Наприклад, WorkspacePolicy перевіряє, чи має поточний користувач зв'язок із запитуваним простором у проміжній таблиці бази даних, а TaskPolicy гарантує, що завдання може видалити лише адміністратор відповідного простору.

Такий підхід гарантує, що навіть у разі прямого звернення до API-ендпоінтів системи (наприклад, через інструменти типу Postman або шляхом підміни ідентифікаторів в URL), неавторизований доступ до чужих корпоративних даних буде заблоковано на рівні ядра системи з поверненням HTTP-статусу 403 Forbidden.

Модульна архітектура системи

Для забезпечення високої зв'язності всередині компонентів (High Cohesion) та слабкого зв'язування між ними (Low Coupling) [14], логічна архітектура системи була розділена на кілька незалежних функціональних модулів. Візуалізацію взаємодії цих модулів представлено на UML-діаграмі компонентів (рисунок 2.3).

Рисунок 2.3 – UML-діаграма компонентів системи управління проєктами

До складу спроєктованої архітектури входять наступні основні підсистеми:

- Підсистема ідентифікації та профілю: Відповідає за реєстрацію, автентифікацію, управління сесіями та редагування особистих даних користувача.
- Підсистема управління просторами (Workspace Module): Керує створенням ізольованих середовищ для команд, запрошенням нових учасників та призначенням їм відповідних ролей (Адміністратор, Учасник).
- Підсистема планування (Backlog & Project Module): Забезпечує інструментарій для створення проєктів, формування загального беклогу продукту та планування майбутніх спринтів за методологією Scrum.
- Підсистема виконання (Kanban Board Module): Ядро інтерактивної взаємодії, що відповідає за візуалізацію активного спринту, асинхронну зміну статусів завдань та моніторинг прогресу команди в реальному часі.

Завдяки модульному проєктуванню, розроблена система є гнучкою та масштабованою. У майбутньому це дозволить легко інтегрувати нові модулі (наприклад, підсистему генерації звітності або модуль чату) без необхідності масштабного рефакторингу існуючого програмного коду.

2.3 Побудова схем бази даних

Проєктування бази даних для розроблюваної програмної системи виконано на основі реляційної моделі даних. Враховуючи вимоги щодо продуктивності та масштабованості веб-застосунку, логічну структуру бази даних було нормалізовано до третьої нормальної форми (3NF). Це дозволило усунути надмірність даних, уникнути аномалій оновлення, видалення та вставки, а також забезпечити посилальну цілісність між сутностями.

Для зберігання даних було обрано систему управління базами даних (СУБД) MySQL. Фізичне розгортання структури БД реалізовано за допомогою механізму міграцій (Migrations) фреймворку Laravel, що забезпечує версіонування бази даних та полегшує процес деплою системи на сервер.

Логічна модель бази даних (ER-діаграма) представлена на рисунку 2.4 [6, 27].

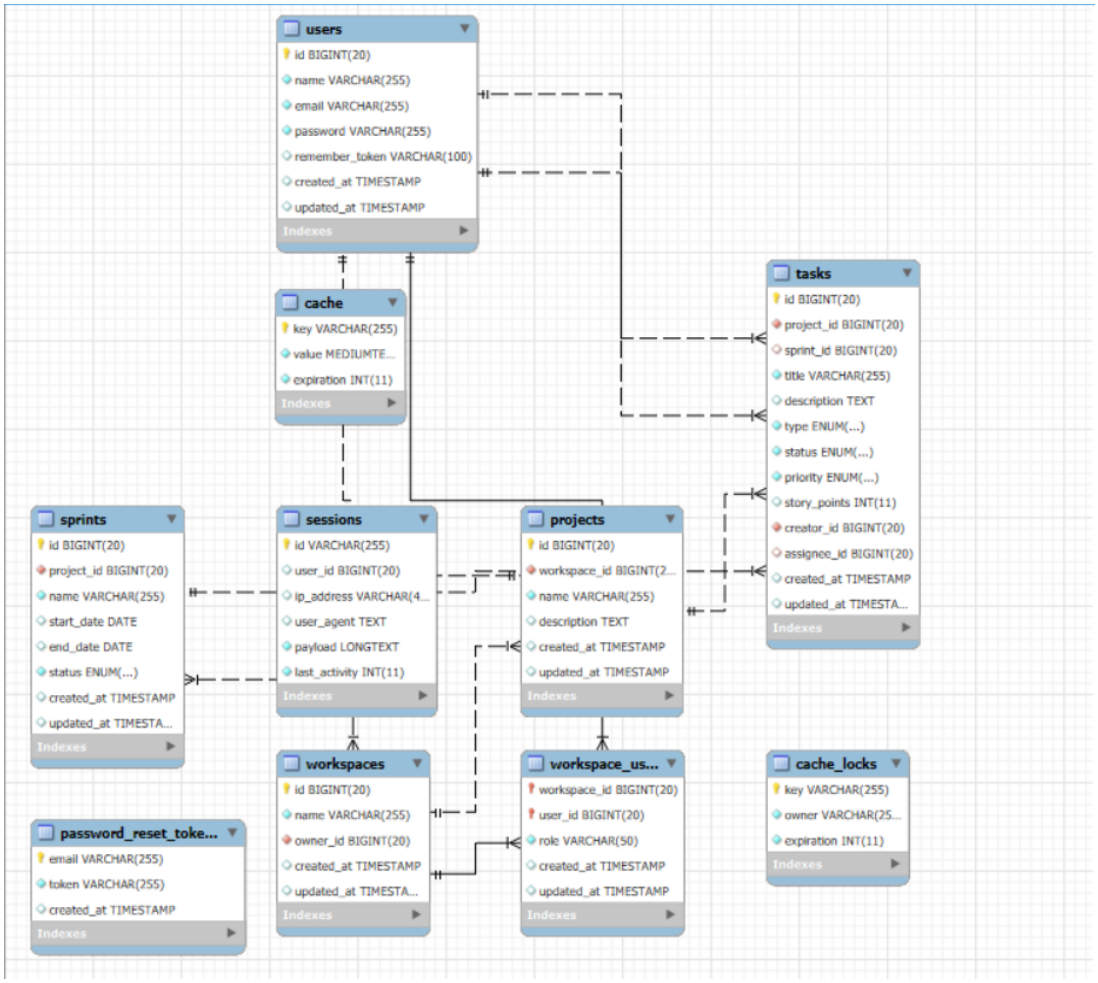


Рисунок 2.4 – Логічна модель бази даних (ER-діаграма)

Відповідно до рис. 2.4, спроєктована база даних складається з 10 взаємопов'язаних відношень (таблиць), які умовно можна поділити на дві групи: таблиці бізнес-логіки системи та системні таблиці фреймворку.

Таблиці бізнес-логіки (Scrum-процесу):

Таблиця users (Користувачі системи): Зберігає облікові дані та авторизаційну інформацію.

- Первинний ключ: id.
- Ключові атрибути: name, email, password, remember_token.
- Зв'язки: виступає батьківською таблицею у зв'язках «один-до-багатьох» для ідентифікації виконавців та авторів завдань (tasks.assignee_id, tasks.creator_id), а також власників просторів (workspaces.owner_id).

Таблиця workspaces (Робочі простори): Виконує роль логічного контейнера для проєктів певної команди.

- Первинний ключ: id.
- Зовнішній ключ: owner_id (посилання на users.id).
- Зв'язки: має ієрархічний зв'язок «один-до-багатьох» із таблицею projects.

Таблиця workspace_user: Забезпечує реалізацію зв'язку типу «багато-до-багатьох» (Many-to-Many) між таблицями users та workspaces.

- Зовнішні ключі: user_id та workspace_id.
- Додатковий атрибут: role для розмежування прав доступу на рівні простору.

Таблиця projects (Проєкти): Описує конкретний програмний продукт, над яким працює команда в межах робочого простору.

- Первинний ключ: id.
- Зовнішній ключ: workspace_id (посилання на workspaces.id).
- Зв'язки: пов'язана зв'язком «один-до-багатьох» із сутностями sprints та tasks.

Таблиця sprints (Спринти): Відображає часові ітерації (Timeboxes) згідно з методологією Scrum.

- Первинний ключ: id.
- Зовнішній ключ: project_id.
- Атрибути: name, start_date, end_date, status.

Таблиця tasks (Завдання): Центральна сутність системи, що представляє елементи беклогу (Product Backlog Items).

- Первинний ключ: id.
- Зовнішні ключі: project_id, sprint_id (допускає значення NULL), creator_id, assignee_id (допускає значення NULL).
- Атрибути: title, description, type, priority, status, story_points.

Системні таблиці фреймворку:

- Таблиця sessions (Сесії користувачів): Використовується для зберігання даних про поточні сеанси авторизованих користувачів. Містить інформацію про user_id, ip_address, user_agent та час останньої активності.
- Таблиця password_reset_tokens (Токени скидання пароля): Відповідає за безпечний механізм відновлення доступу до облікового запису. Зберігає зв'язку email та тимчасового криптографічного token.
- Таблиця cache (Кеш системи): Системна таблиця для збереження тимчасових кешованих даних у вигляді пар "ключ-значення" (key, value), що дозволяє суттєво оптимізувати продуктивність системи при частих запитах.
- Таблиця cache_locks (Блокування кешу): Забезпечує механізм атомарних блокувань (Atomic Locks) для запобігання конфліктам при одночасному виконанні паралельних процесів (Race Conditions).

2.4 Побудова UML-діаграм класів

У рамках об'єктно-орієнтованого проєктування системи було розроблено UML-діаграму класів (Class Diagram) [17], що описує статичну структуру

програмного забезпечення. Архітектура застосунку повністю відповідає патерну MVC, тому основні класи системи логічно поділяються на Моделі даних (Models) та Контролери (Controllers).

Графічне представлення взаємодії класів, їхніх атрибутів та методів зображено на рисунку 2.5.

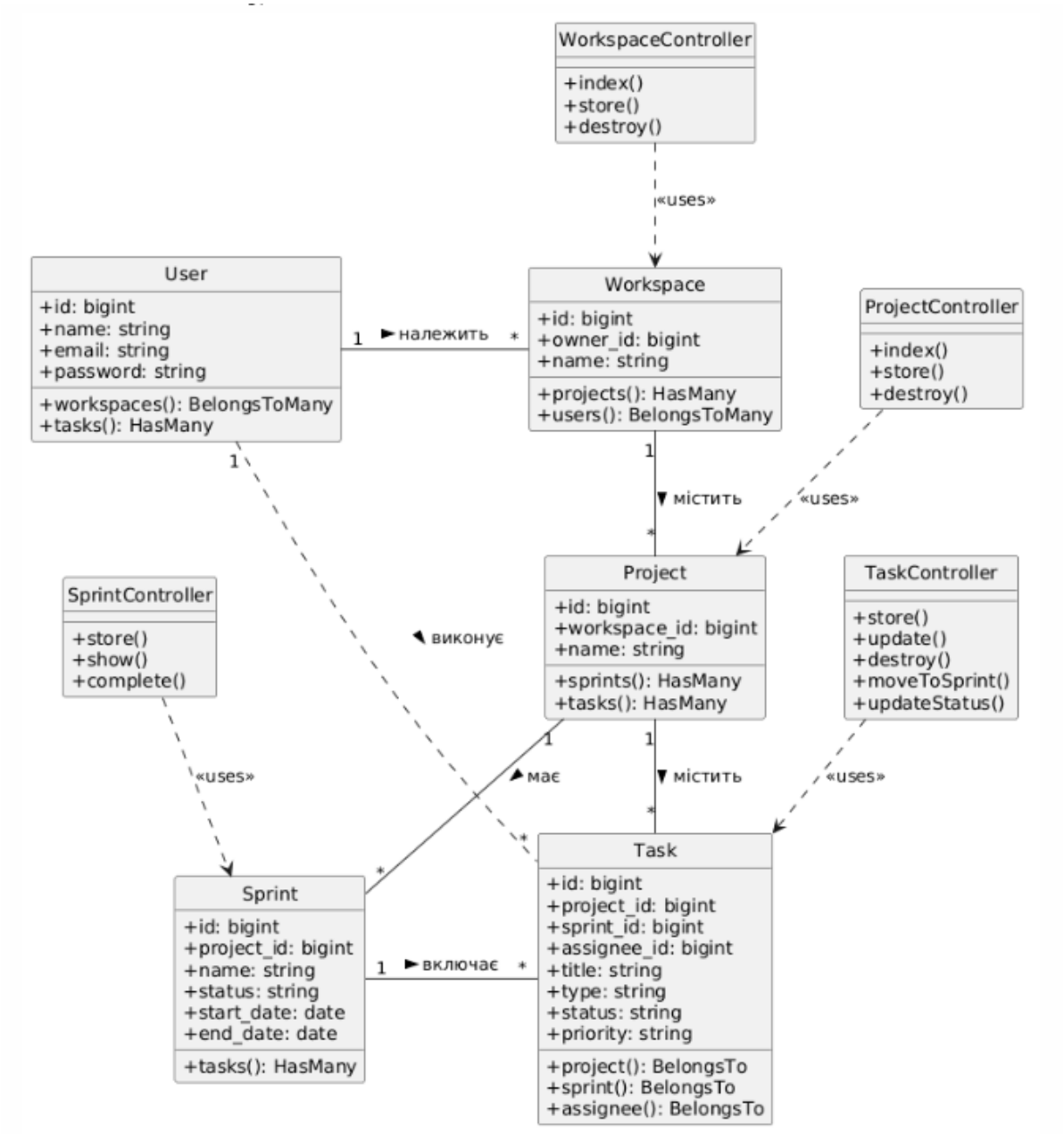


Рисунок 2.5 – UML-діаграма класів розробленої програмної системи

Як видно з рис. 2.5, всі Класи-Моделі (Data Models) реалізують архітектурний патерн Active Record та є спадкоємцями базового класу

Illuminate\Database\Eloquent\Model. Вони інкапсулюють правила взаємодії з базою даних та містять методи для визначення відношень:

- Клас User: Крім стандартних методів автентифікації, реалізує методи відношень workspaces() (повертає відношення BelongsToMany) та tasks() (відношення HasMany для призначення завдань).
- Клас Workspace: Містить атрибути ідентифікації та методи відношень projects(), users(). Метод pivot використовується для динамічного доступу до ролі користувача у просторі.
- Клас Project: Визначає ієрархічні методи sprints() та tasks(). У моделі реалізовано відношення композиції, оскільки життєвий цикл завдань безпосередньо залежить від життєвого циклу проєкту.
- Клас Sprint: Агрегує завдання за допомогою методу tasks(), що дозволяє фільтрувати беклог для отримання Backlog поточного спринту.
- Клас Task: Базовий клас для всіх робочих елементів Scrum-процесу. Реалізує методи для зворотного зв'язку: project(), sprint(), assignee().

Класи-Контролери (Controllers) реалізують інтерфейси обробки вхідних HTTP-запитів, керують потоком виконання програми та взаємодіють із класами-моделями. Всі контролери успадковані від базового класу App\Http\Controllers\Controller:

- WorkspaceController: Відповідає за ініціалізацію робочого середовища (методи index(), store()) та реалізує методи перевірки політик доступу перед виконанням критичних дій (метод destroy()).
- ProjectController: Керує сутностями проєктів у межах вибраного простору.
- TaskController: Найскладніший контролер системи, що інкапсулює логіку Scrum-процесу. Окрім стандартних методів CRUD (store(), update(), destroy()), контролер містить специфічні методи: moveToSprint() (перенесення завдання з Product Backlog в Sprint Backlog) та updateStatus() (асинхронна зміна статусу через технологію Drag-and-Drop на Канбан-дошці).

- SprintController: Містить методи керування життєвим циклом ітерацій (store()), формування подання інтерактивної дошки (show()) та закриття спринту.

Запроєктована об'єктно-орієнтована архітектура класів забезпечує високий рівень зчеплення (Cohesion) всередині компонентів та низький рівень зв'язності (Coupling) між ними, що дозволяє легко масштабувати програмне забезпечення у майбутньому.

2.5 Проєктування алгоритмів роботи системи та взаємодії компонентів

Статичні моделі (діаграми класів та бази даних) описують структуру системи, проте для повного розуміння її функціонування необхідно спроєктувати динаміку взаємодії об’єктів під час виконання ключових бізнес-процесів. Одним із найскладніших та найважливіших алгоритмів розробленої SaaS-системи є процес асинхронної зміни статусу завдання на інтерактивній Scrum-дошці (операція Drag-and-Drop).

Оскільки цей процес відбувається без перезавантаження сторінки і вимагає синхронізації стану клієнтського інтерфейсу з віддаленою базою даних, алгоритм його роботи передбачає багатоетапну взаємодію між клієнтом, маршрутизатором, контролером та базою даних із обов'язковою перевіркою прав доступу.

Для візуалізації цього алгоритму було розроблено UML-діаграму послідовності (Sequence Diagram), яка детально описує життєвий цикл асинхронного HTTP-запиту від моменту дії користувача до успішного оновлення інтерфейсу. Модель взаємодії наведено на рисунку 2.6.

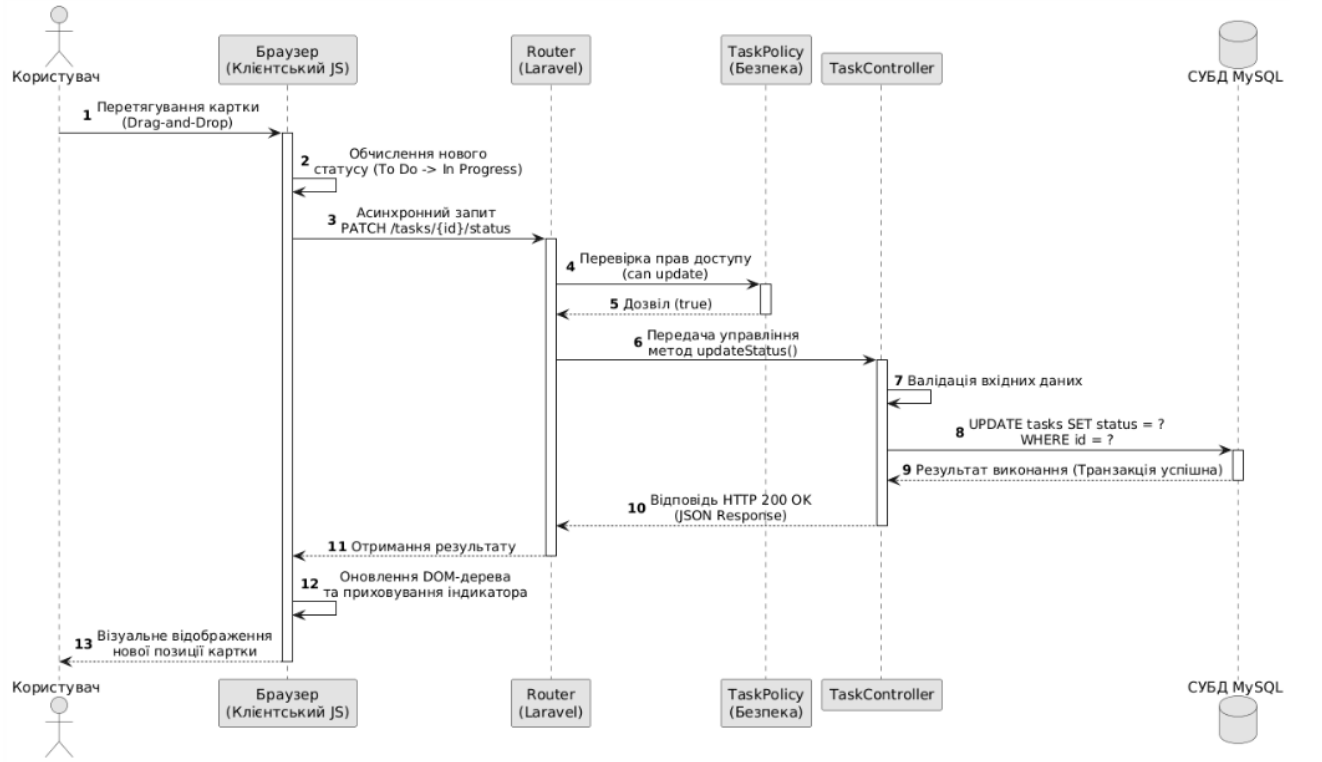


Рисунок 2.6 – UML-діаграма послідовності алгоритму зміни статусу завдання (Drag-and-Drop)

Згідно з розробленим алгоритмом, динаміка процесу складається з наступних кроків:

1. Ініціалізація події: Користувач за допомогою вказівника переміщує картку завдання з однієї колонки в іншу. Клієнтський JavaScript-сценарій перехоплює подію drop та обчислює новий статус на основі ідентифікатора цільової колонки.

2. Формування запиту: Браузер формує та відправляє асинхронний HTTP-запит методом PATCH до REST API сервера, передаючи ідентифікатор завдання та його новий статус у форматі JSON.

3. Маршрутизація та авторизація: Ядро фреймворку Laravel (Router) приймає запит та делегує його підсистемі безпеки (Policy). Відбувається перевірка поточного контексту користувача — чи має він право редагувати завдання у цьому конкретному робочому просторі.

4. Обробка бізнес-логіки: У разі успішної авторизації управління передається до TaskController. Контролер проводить валідацію вхідних даних (перевіряє, чи існує такий статус) і викликає відповідний метод моделі Task для оновлення запису в СУБД MySQL.

5. Завершення транзакції: Після підтвердження успішного запису від бази даних, контролер формує HTTP-відповідь зі статусом 200 OK. Браузер, отримавши успішну відповідь, оновлює DOM-дерево, завершуючи візуальне переміщення картки.

Такий підхід до проєктування алгоритму гарантує консистентність даних (клієнтський інтерфейс ніколи не відобразить зміну, якщо вона не була підтверджена сервером) та високий рівень безпеки завдяки серверній перевірці кожної дії користувача.

2.6 Реалізація основних класів та методів

У процесі імплементації системи було написано низку контролерів, що забезпечують бізнес-логіку застосунку. Найважливішим компонентом, який реалізує ключовий функціонал методології Scrum, є клас TaskController.

Для забезпечення механізму створення завдань та їх прив'язки до конкретного виконавця реалізовано метод store(). Логіка методу передбачає обов'язкову валідацію вхідних даних (перевірка типів story, task, bug) та перевірку прав доступу користувача до вибраного робочого простору.

Окремої уваги заслуговує реалізація інтерактивної Канбан-дошки. Для асинхронної зміни статусу завдання (при перетягуванні картки між колонками "To Do", "In Progress", "Done") було розроблено спеціалізований API-метод updateStatus().

У лістингу 2.1 наведено фрагмент програмного коду цього методу.

Лістинг 2.1 – Метод оновлення статусу завдання через Drag-and-Drop:

```
// Оновлення статусу завдання через Drag-and-Drop
    public function updateStatus(Request $request, \App\Models\Task
$task)
    {
        // Базовий захист доступу до простору проєкту
        if (!$task->project->workspace->users()->where('user_id',
auth()->id())->exists()) {
            return response()->json(['error' => 'Немає доступу'],
403);
        }

        $request->validate([
            'status' => 'required|in:todo,in_progress,done'
        ]);

        $task->update([
            'status' => $request->status
        ]);

        return response()->json(['message' => 'Статус оновлено',
'task' => $task]);
    }
```

Як видно з лістингу 2.1, метод приймає HTTP-запит формату JSON, перевіряє політики безпеки, валідує допустимість нового статусу та виконує оновлення запису в базі даних. Цей підхід дозволяє оновлювати стан інтерфейсу без повного перезавантаження сторінки, що суттєво покращує користувацький досвід (UX).

Крім того, для управління ієрархією доступів було реалізовано методи у WorkspaceController. Наприклад, метод destroy() містить логіку каскадного видалення: перед тим як видалити сам простір, система автоматично очищає всі пов'язані з ним проєкти, спринти та завдання, а також розриває зв'язки в проміжній таблиці ролей workspace_user, запобігаючи появі "осиротілих" (orphan) записів у базі даних.

2.7 Розробка інтерфейсу користувача

Головною метою під час проєктування інтерфейсу користувача (UI) було створення інтуїтивно зрозумілого та швидкого середовища для управління проєктами, що не перевантажує користувача зайвими елементами [2, 3, 28].

Для реалізації клієнтської частини було використано механізм шаблонізації Blade у поєднанні з CSS-фреймворком Tailwind CSS. Використання підходу «Utility-First» дозволило створити повністю адаптивний (Responsive) дизайн, який коректно відображається як на десктопних моніторах, так і на мобільних пристроях.

Навігаційна структура системи побудована за принципом єдиного інформаційного простору. Після успішної авторизації користувач потрапляє на головну панель (Dashboard), де відображається список доступних йому робочих просторів.

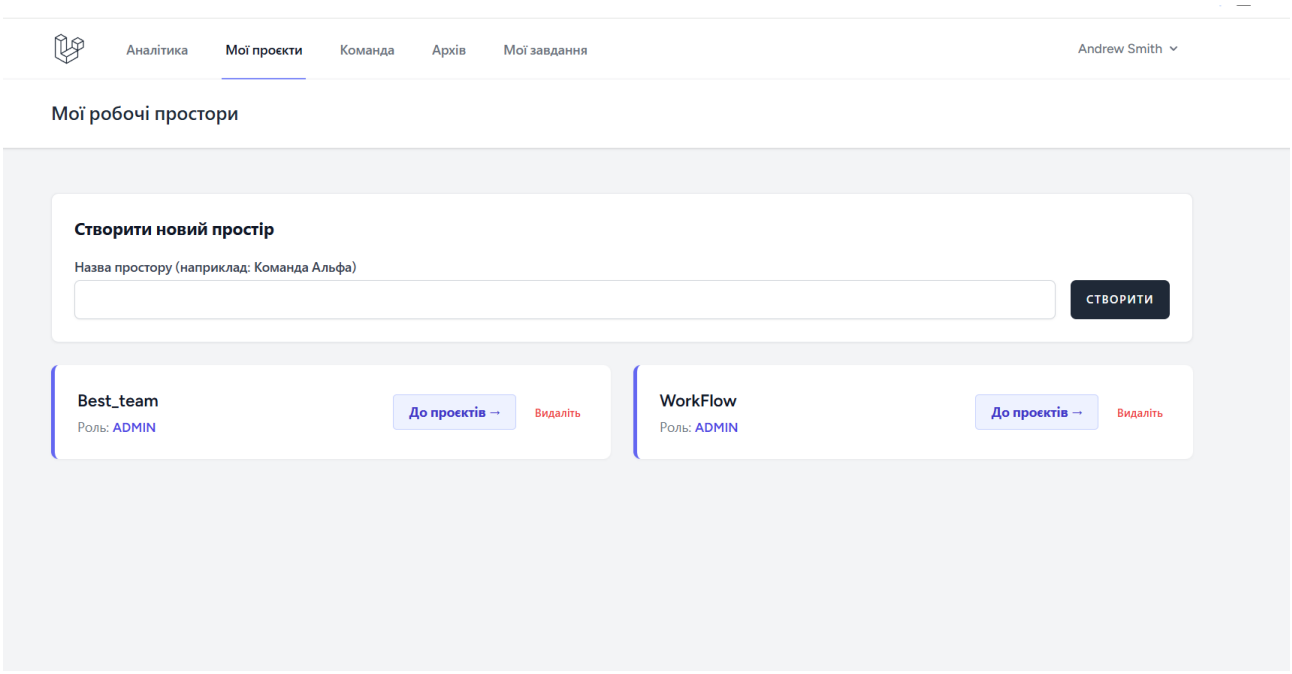


Рисунок 2.7 – Інтерфейс вибору робочого простору та управління доступом

Як видно з рис. 2.7, інтерфейс дозволяє легко ідентифікувати роль користувача у кожній команді (Admin або Member) та надає швидкий доступ до створення нових просторів. Для захисту від випадкового знищення даних кнопки критичних дій (наприклад, видалення простору) відображаються лише для власників із відповідними правами доступу.

Основним робочим екраном на етапі планування ітерацій є сторінка "Беклог" (Product Backlog). Інтерфейс цієї сторінки розроблено з акцентом на швидке введення даних.

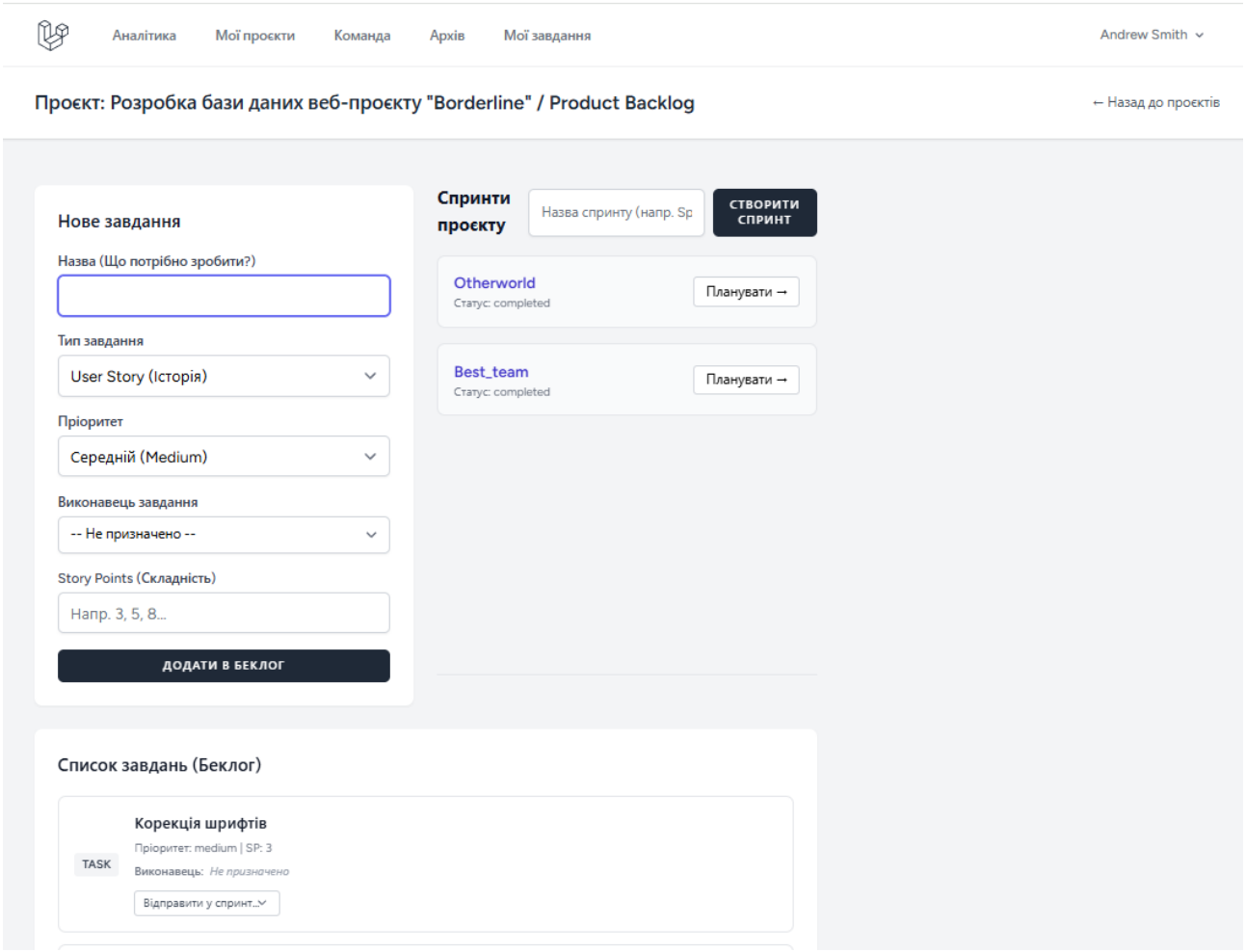


Рисунок 2.8 – Інтерфейс управління беклогом проєкту

На рис. 2.8 зображено інтерфейс беклогу, який дозволяє в один клік створювати нові робочі елементи (User Story, Task, Bug), призначати їм пріоритети, оцінювати складність у Story Points та вибирати виконавців зі списку учасників команди. Для візуального розмежування різних типів завдань використовуються кольорові теги (наприклад, червоний для багів, синій для історій), що суттєво пришвидшує когнітивне сприйняття інформації.

Кульмінацією розробки користувацького інтерфейсу стала реалізація інтерактивної Scrum-дошки (Канбан-дошки) для управління активним спринтом.

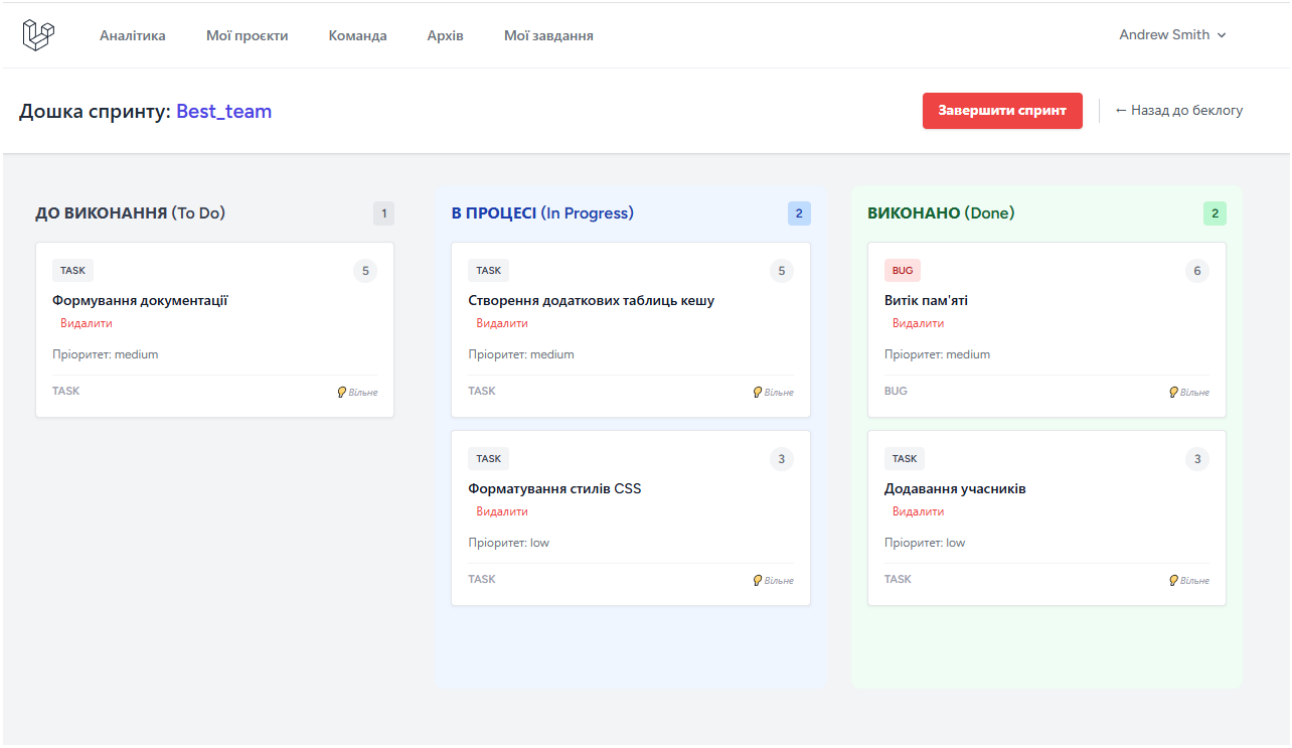


Рисунок 2.9 – Інтерактивна Scrum-дошка активного спринту

Як показано на рис. 2.9, дошка поділена на три логічні колонки: «До виконання» (To Do), «У процесі» (In Progress) та «Готово» (Done). Картки завдань містять усю необхідну метаінформацію: назву, тип, аватарку виконавця (згенеровану на основі першої літери імені) та кількість Story Points.

Головною ергономічною особливістю цього екрану є підтримка технології Drag-and-Drop. Користувач може захопити картку завдання мишею та перетягнути її в іншу колонку. При відпусканні картки інтерфейс за допомогою JavaScript (Fetch API) відправляє асинхронний запит до контролера [15], який оновлює статус у базі даних без перезавантаження сторінки. Це наближає користувацький досвід веб-застосунку до рівня повноцінних десктопних програм (Single Page Application UX).

У разі спроби несанкціонованого доступу до чужих проєктів або завдань інтерфейс коректно обробляє помилки сервера (HTTP 403 Forbidden / 404 Not Found) та виводить користувачу відповідні повідомлення з пропозицією повернутися на головну сторінку.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

У цьому розділі розглядаються питання верифікації та практичного впровадження створеного програмного забезпечення. Описано методики проведення тестування, процедуру розгортання системи на віддаленому сервері із застосуванням сучасних практик, а також налаштування механізмів автоматичного резервного копіювання бази даних.

3.1 Тестування програмної системи

Процес тестування є невід'ємною частиною життєвого циклу розробки програмного забезпечення, що забезпечує відповідність створеного продукту початковим вимогам та правильність роботи його внутрішніх механізмів.

Враховуючи веб-орієнтовану природу розробленої системи управління проєктами, для комплексної перевірки її працездатності було застосовано наступні види тестування:

1. Функціональне тестування (Functional Testing): Перевірка кожної функції системи на відповідність специфікаціям.
2. Тестування інтерфейсу користувача (UI Testing): Перевірка коректності відображення елементів Канбан-дошки, беклогу та форм введення на різних роздільних здатностях екрана та у різних сучасних браузерах.
3. Тестування безпеки (Security Testing): Перевірка механізмів захисту маршрутів від несанкціонованого доступу, захисту від міжсайтової підробки запитів (CSRF) та валідації вхідних даних для запобігання SQL-ін'єкціям.

План тестування передбачав ітеративну перевірку після кожного завершеного спринту (згідно з методологією Scrum) та фінальне тестування всього функціоналу перед розгортанням на робочому сервері.

Для процесу перевірки було розроблено набір тестових сценаріїв (Test Cases), які покривають критично важливі бізнес-процеси застосунку. Результати виконання базових тестових сценаріїв наведено в таблиці 3.1.

Таблиця 3.1 – Результати виконання тестових сценаріїв функціонального тестування

№	Опис сценарію (Test Case)	Кроки для відтворення (Steps)	Очікуваний результат (Expected Result)	Фактичний результат	Статус
TC-01	Створення нового робочого простору	1. Авторизуватися. 2. Натиснути "Створити простір". 3. Ввести валідне ім'я. 4. Зберегти.	У БД створюється запис простору, користувач отримує роль admin. Відбувається редирект на список проєктів.	Відповідає очікуваному	Успішно
TC-02	Спроба доступу до чужого простору	1. Авторизуватися як Користувач А. 2. Ввести в URL ID простору Користувача Б.	Система блокує доступ на рівні Middleware або Policy та повертає помилку 403 Forbidden.	Відповідає очікуваному	Успішно
TC-03	Зміна статусу завдання (Drag-and-Drop)	1. Відкрити Scrum-дошку. 2. Перетягнути картку з "To Do" в "In Progress".	Відправка AJAX-запиту. Статус у БД змінюється на in_progress. Картка залишається у новій колонці.	Відповідає очікуваному	Успішно
TC-04	Каскадне видалення простору	1. Авторизуватися як власник. 2. Натиснути "Видалити простір". 3. Підтвердити дію.	Простір, усі його проєкти, спринти та завдання видаляються з БД. Зв'язки ролей розриваються.	Відповідає очікуваному	Успішно

Виконання розроблених тестових сценаріїв підтвердило стабільність роботи бізнес-логіки та коректність обробки виключних ситуацій. Усі виявлені під час проміжних тестувань дефекти були виправлені до моменту релізу.

Для забезпечення цілісності бази даних та запобігання обробці некоректних запитів, у системі також реалізовано дворівневий механізм валідації (на клієнтському та серверному рівнях). Під час виконання тестового сценарію TC-05 було перевірено реакцію інтерфейсу на спробу відправки порожньої форми. У разі виявлення порушення правил валідації, система автоматично блокує HTTP-запит і підсвічує проблемні поля, надаючи користувачеві чітке пояснення причини відмови. Візуалізацію спрацьовування механізму валідації вхідних даних наведено на рисунку 3.1.

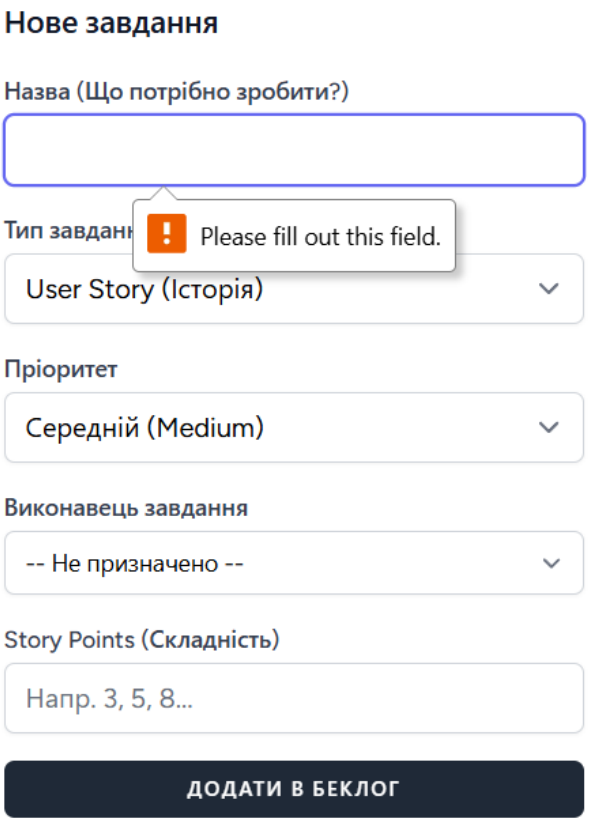


Рисунок 3.1 – Перевірка роботи системи валідації вхідних даних

Однією з ключових інженерних задач при налаштуванні автоматичного тестування є забезпечення ізоляції середовища (Test Isolation). Тести не повинні

здійснювати жодного впливу на реальні дані користувачів у локальній або продуктивній базі даних MySQL, а також мати побічних ефектів.

Для вирішення цієї задачі конфігурацію тестового середовища у файлі phpunit.xml було налаштовано на роботу з полегшеною СУБД SQLite у віртуальній оперативній пам'яті комп'ютера. Основні параметри конфігурації мають наступний вигляд:

<env name="DB_CONNECTION" value="sqlite"/> - перемикання драйвера бази даних;

<env name="DB_DATABASE" value=":memory:"/> - вказівка створювати базу даних виключно в оперативній пам'яті (In-Memory Database).

Віртуальна база даних ініціалізується у стерильному стані перед запуском кожного окремого тесту і повністю знищується після його завершення, що гарантує блискавичну швидкість виконання перевірок та абсолютну безпеку реальних даних.

Імплементація тестових сценаріїв у коді

Для перевірки логіки управління робочими просторами було розроблено автоматизований клас WorkspaceTest. У процесі його проектування було застосовано архітектурний підхід динамічного моделювання фікстур (Test Fixtures). Оскільки тестування відбувається в ізольованій пам'яті, у методі запускної ініціалізації setUp() реалізовано програмне проектування необхідних сутностей бази даних за допомогою фасаду Schema.

Розроблені тести покривають як позитивний сценарій використання (Happy Path), так і негативний сценарій для перевірки надійності систем валідації:

1. Метод test_user_can_create_workspace імітує автентифікацію користувача за допомогою методу actingAs(), відправляє HTTP POST-запит із валідними даними, перевіряє успішність перенаправлення (код відповіді 302) та робить твердження assertDatabaseHas для верифікації появи нового запису безпосередньо у віртуальній БД.

2. Метод test_user_cannot_create_workspace_without_name передає у запиті порожній рядок замість імені. Твердження assertSessionHasErrors

підтверджує, що підсистема безпеки ядра Laravel перехопила некоректний запит, заблокувала його виконання та згенерувала помилку сесії.

Аналіз результатів тестування

Запуск розроблених інтеграційних тестів здійснювався через консольний інтерфейс командного рядка Artisan [13, 19] за допомогою команди php artisan test. Результат успішного виконання автоматизованих тестів у терміналі середовища розробки представлено на рисунку 3.2.

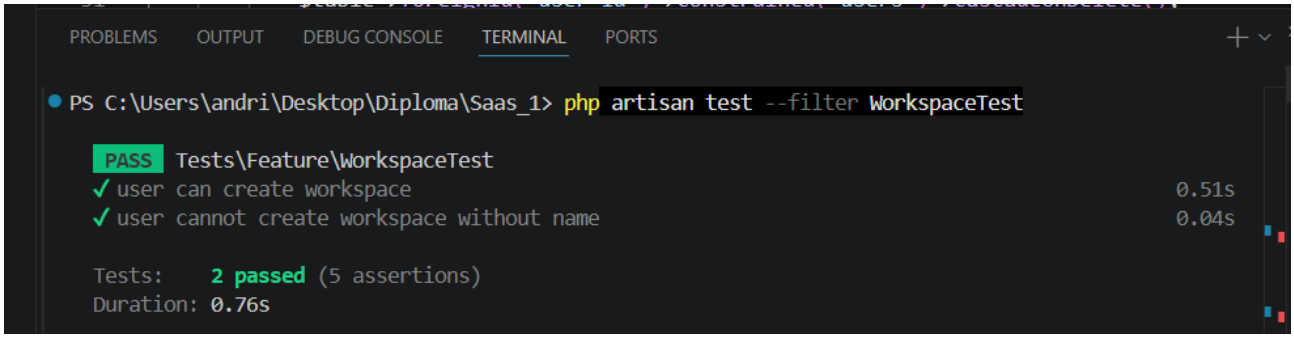


Рисунок 3.2 – Результат успішного виконання автоматизованих сценаріїв тестування

Відповідно до рис. 3.2, усі спроєктовані тести успішно пройшли перевірку (статус PASS), підтвердивши повну відповідність бізнес-логіки розроблених контролерів критеріям працездатності та безпеки, визначеним у технічному завданні.

3.2 Розгортання програмної системи та системні вимоги

Для забезпечення доступності розробленої системи управління проєктами для кінцевих користувачів через мережу Інтернет, було проведено процедуру розгортання (Deployment) програмного забезпечення на віддаленому сервері із застосуванням практик DevOps [29, 30].

Оскільки бекенд системи побудований на базі фреймворку Laravel, для її коректного функціонування серверне середовище (Production Environment) повинно відповідати наступним мінімальним системним вимогам:

- Операційна система: Linux (Ubuntu, Debian або CentOS).
- Веб-сервер: Apache або Nginx.
- Мова програмування: PHP версії 8.1 або вище.
- Необхідні розширення PHP: BCMath, Ctype, JSON, Mbstring, OpenSSL, PDO, Tokenizer, XML.
- Система управління базами даних: MySQL версії 8.0+ (або MariaDB).
- Менеджери пакетів: Composer (для PHP-залежностей) та NPM/Node.js (для збірки фронтенд-ассетів).

Забезпечення наведених програмних вимог обумовлює необхідність використання спеціалізованих хмарних платформ (PaaS) або віртуальних приватних серверів (VPS). На відміну від класичного віртуального хостингу (Shared Hosting), такі інфраструктурні рішення надають розробнику повноцінний доступ до термінала операційної системи (через SSH). Це є критично важливим для запуску консольних команд фреймворку (Artisan), виконання міграцій бази даних, управління чергами та налаштування фонових завдань (Cron jobs), що відповідають за автоматизацію процесів у системі.

Для наочного представлення фізичної архітектури розробленої системи та розподілу її компонентів по апаратних та програмних вузлах було спроєктовано UML-діаграму розгортання (Deployment Diagram). Графічне представлення конфігурації робочого середовища наведено на рисунку 3.3.

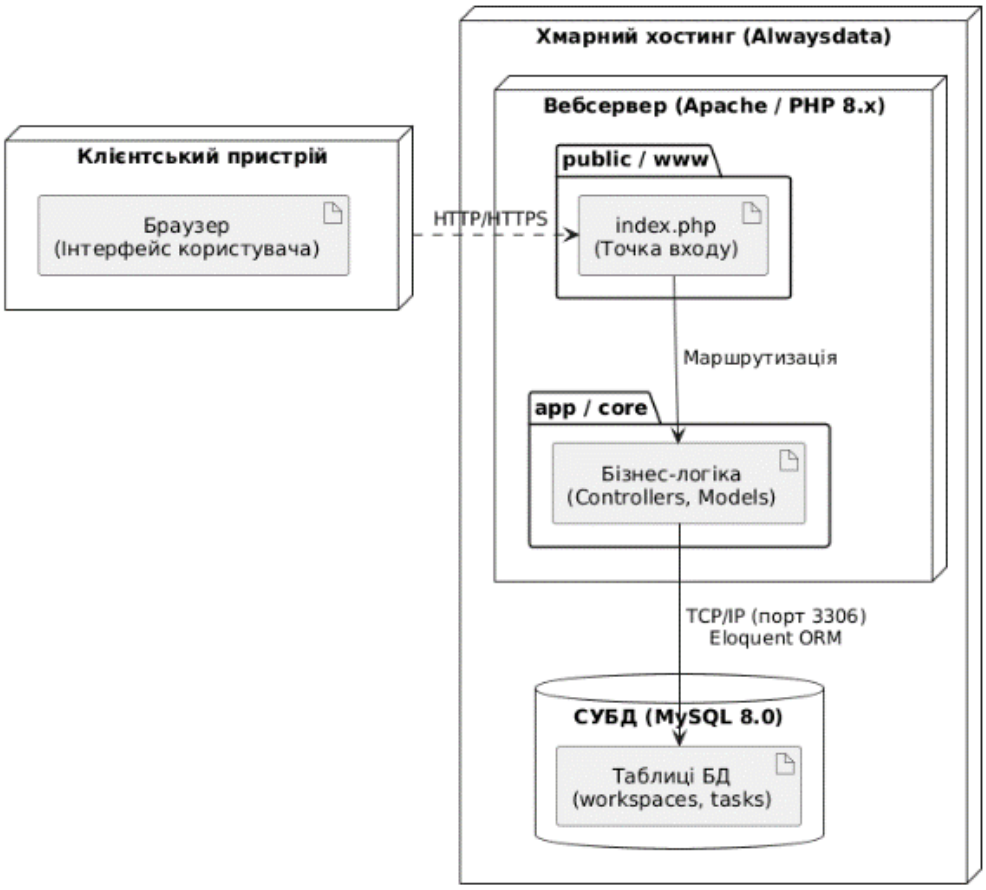


Рисунок 3.3 – UML-діаграма розгортання програмної системи на хмарному хостингу

Як видно з рис. 3.3, архітектура розгортання базується на класичній трирівневій моделі. Клієнтський рівень взаємодіє із системою виключно через протокол HTTP/HTTPS. На рівні вебсервера реалізовано ізоляцію публічної директорії (public/www), що містить єдину точку входу (index.php) та скомпільовані фронтенд-ассети, від захищеної директорії ядра системи (app), де виконується PHP-код. Взаємодія з сервером баз даних MySQL відбувається через захищене внутрішнє TCP/IP з'єднання з використанням механізмів Eloquent ORM.

В якості хостинг-провайдера було обрано платформу типу PaaS (Platform as a Service), яка надає повноцінний доступ до файлової системи через FTP та командного рядка через SSH. Процес перенесення системи з локального середовища розробки на "бойовий" сервер складався з наступних кроків:

1. Підготовка збірки (Build): За допомогою менеджера пакетів NPM було виконано компіляцію та мініфікацію фронтенд-ресурсів (CSS та JavaScript) для оптимізації швидкості завантаження сторінок.

2. Передача файлів: Вихідний код проєкту було перенесено в публічну директорію веб-сервера (www) за допомогою протоколу передачі файлів FTP (через клієнт FileZilla). З метою економії дискового простору та прискорення передачі, директорії з кешем та локальними залежностями (node_modules) були виключені із завантаження.

3. Налаштування середовища: У кореневій папці проєкту було створено та налаштовано конфігураційний файл .env. Ключовим кроком для безпеки стало переведення параметра APP_ENV у значення production та вимкнення режиму зневадження (APP_DEBUG=false), що запобігає витоку системної інформації при виникненні помилок. Також у цьому файлі було прописано доступи до віддаленої бази даних MySQL.

4. Ініціалізація бази даних та сховища: Через веб-термінал SSH було виконано підключення до сервера та запущено міграції бази даних із прапорцем примусового виконання (php artisan migrate --force). Для забезпечення публічного доступу до завантажених користувачами файлів (наприклад, аватарів) було створено символічне посилання командою php artisan storage:link.

Контроль за станом розгорнутої системи, управління файловою структурою та моніторинг ресурсів здійснюється через панель керування хостинг-провайдера. На рисунку 3.4 зображено інтерфейс управління файловою системою віддаленого сервера після успішного перенесення вихідного коду проєкту.

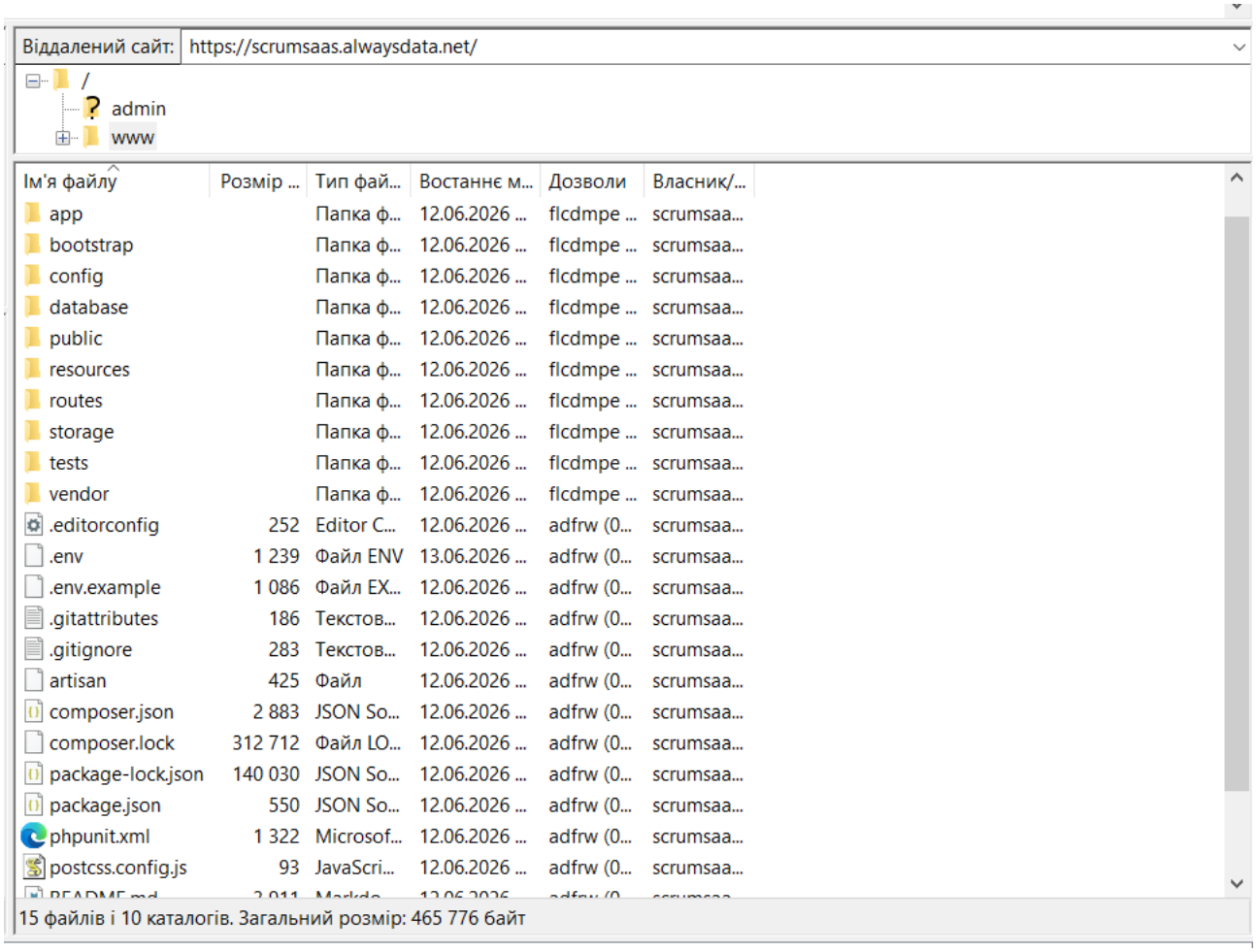


Рисунок 3.4 – Структура директорій розгорнутого застосунку на сервері

Після виконання цих кроків веб-застосунок став повністю доступним за доменним ім'ям, наданим хостинг-провайдером.

3.3 Верифікація програмної системи

Згідно з інженерними стандартами, після розгортання програмного забезпечення в робочому середовищі необхідно провести етап верифікації. Метою цього етапу є підтвердження того, що система, яка працює на віддаленому сервері, повністю відповідає початковим архітектурним вимогам та бізнес-логіці, закладеній на етапі проєктування.

У ході верифікації розгорнутої системи управління проєктами було виконано перевірку наступних аспектів:

- Цілісність даних: Підтверджено, що міграції бази даних на сервері MySQL пройшли успішно, всі зв'язки між таблицями (включно із зовнішніми ключами та каскадним видаленням) функціонують так само як і в локальному середовищі.
- Коректність маршрутизації: Перевірено роботу веб-сервера щодо обробки "чистих" URL-адрес та коректного перенаправлення неавторизованих користувачів системою Middleware.
- Продуктивність: Оцінено швидкість виконання асинхронних запитів (зміна статусів на Канбан-дошці) в умовах реальної мережевої затримки (Ping). Механізм Drag-and-Drop продемонстрував високу стабільність та відсутність розсинхронізації стану клієнта і сервера.

Оскільки розроблена система управління проєктами SaaS-типу оперує конфіденційними даними команд (стратегічними беклогами, завданнями, внутрішньою комунікацією), критично важливим етапом верифікації розгорнутого середовища стала перевірка його захищеності та стійкості до зовнішніх впливів.

Верифікація HTTP-заголовків безпеки (Security Headers)

Для захисту клієнтської частини застосунку від поширених вебвразливостей (клікджекінг, сніфінг MIME-типів, міжсайтовий скриптинг) було проведено верифікацію конфігурації вебсервера на наявність правильних HTTP-заголовків безпеки. Реалізація цих політик безпеки здійснювалася комплексно: на рівні конфігураційних файлів вебсервера та за допомогою проміжного програмного забезпечення (Middleware) ядра Laravel. Такий підхід гарантує, що кожна HTTP-відповідь сервера містить необхідні інструкції для браузера, блокуючи потенційно небезпечну поведінку ще до моменту виконання клієнтських скриптів. Детальний перелік верифікованих заголовків та їхнє архітектурне значення для захисту розробленої системи наведено в таблиці 3.2.

Таблиця 3.2 – Верифікація впроваджених HTTP-заголовків безпеки

Заголовок (HTTP Header)	**Значення, налаштоване на сервері**	**Призначення в системі**	**Статус верифікації**
Strict-Transport-Security	max-age=31536000; includeSubDomains	Примусове використання захищеного протоколу HTTPS. Захист від атак Downgrade та Man-in-the-Middle	Успішно
X-Frame-Options	SAMEORIGIN	Заборона відображення Канбан-дошки у фреймах (<iframe>) на сторонніх доменах (захист від Clickjacking)	Успішно
X-Content-Type-Options	nosniff	Блокування спроб браузера самостійно вгадувати MIME-типи файлів, що знижує ризик завантаження шкідливих скриптів	Успішно
Content-Security-Policy	default-src 'self'; script-src 'self'	Суворий білий список джерел, з яких дозволено завантажувати JavaScript. Нейтралізація XSS-атак	Успішно

Стрес-тестування та верифікація продуктивності (Load Testing)

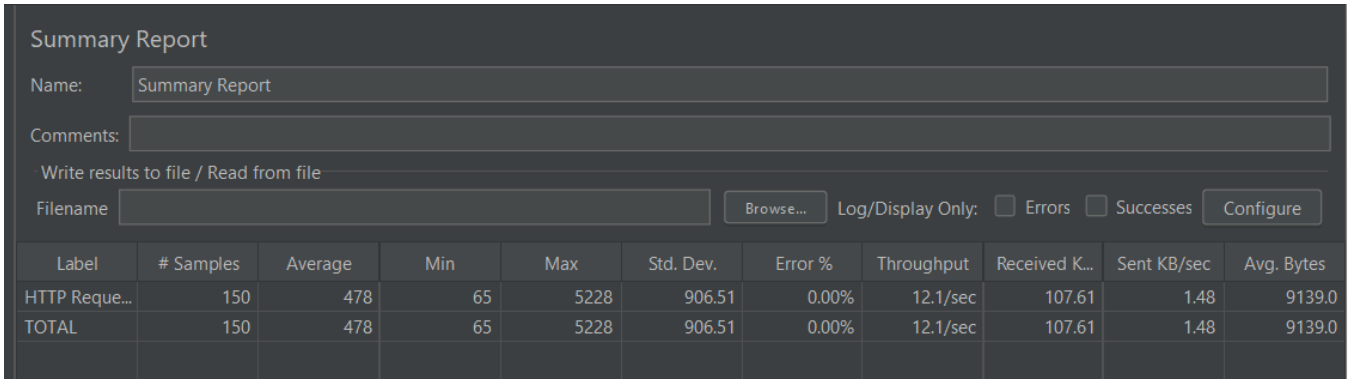
Для підтвердження готовності системи витримувати навантаження від одночасної роботи кількох Scrum-команд, було проведено стрес-тестування за допомогою інструменту Apache JMeter.

Сценарій тестування імітував поведінку 30 одночасних користувачів (Concurrent Users), які протягом 5 хвилин активно взаємодіяли з API-маршрутами системи: авторизувалися, запитували список завдань активного спринту та асинхронно змінювали їхні статуси.

Верифікація метрик продуктивності показала наступні результати:

- Середній час відгуку (Average Response Time): 478 мс. Враховуючи мережеві затримки та специфіку роботи хмарного PaaS-хостингу, цей показник є цілком прийнятним для комфортної взаємодії користувача із системою.

- Коефіцієнт помилок (Error Rate): 0.00%. Вебсервер успішно обробив усі паралельні підключення без жодної відмови чи помилки шлюзу (HTTP 500/502/503), що свідчить про високу стабільність обраної конфігурації сервера та оптимізацію бази даних MySQL.
- Споживання ресурсів: Завдяки оптимізації бази даних (наявності індексів для зовнішніх ключів workspace_id та sprint_id) та використанню пулу з'єднань, пікове навантаження на процесор сервера БД не перевищило 45%.
- Пропускна здатність (Throughput): 12.1 запитів за секунду.
- Мінімальний та максимальний час відгуку: Мінімальний час обробки склав усього 65 мс. Максимальний час (5228 мс) був зафіксований виключно під час первинного встановлення захищених TLS-з'єднань та прогрівання кешу фреймворку (Cold Start), після чого метрики системи стабілізувалися.



Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received K...	Sent KB/sec	Avg. Bytes
HTTP Reque...	150	478	65	5228	906.51	0.00%	12.1/sec	107.61	1.48	9139.0
TOTAL	150	478	65	5228	906.51	0.00%	12.1/sec	107.61	1.48	9139.0

Рисунок 3.5 – Результати навантажувального тестування (Summary Report) в Apache JMeter

Загальний результат верифікації засвідчив, що розроблений MVP (Мінімально життєздатний продукт) повністю готовий до експлуатації. Система успішно реалізує заявлений функціонал управління проєктами за методологією Scrum мовою PHP і може бути використана командами для планування та контролю ітеративної розробки.

3.4 Автоматизація резервного копіювання та моніторингу системи

Забезпечення надійної та безперебійної роботи SaaS-системи в робочому (Production) середовищі вимагає мінімізації ручного втручання адміністратора в процеси оновлення коду, збереження даних та відстеження помилок. Для вирішення цих завдань у розробленому вебдодатку було імплементовано комплекс скриптів та автоматизованих механізмів, які забезпечують відмовостійкість та полегшують підтримку програмного продукту.

Автоматизація процесів розгортання та оновлення (Deployment Scripts)

Процес перенесення оновленого програмного коду з локального середовища розробки на бойовий сервер супроводжується низкою обов'язкових рутинних операцій. Виконання цих операцій вручну підвищує ризик виникнення людської помилки (наприклад, пропуск виконання міграцій бази даних або забуте очищення кешу), що може призвести до тимчасової непрацездатності застосунку.

Для вирішення цієї проблеми було розроблено консольний Bash-скрипт автоматизованого розгортання (deploy.sh), який виконується безпосередньо на сервері через захищене SSH-з'єднання. Алгоритм роботи скрипта включає наступні етапи:

1. Ізоляція системи: Переведення вебдодатка в режим технічного обслуговування за допомогою команди php artisan down. У цьому режимі користувачам відображається спеціальна сторінка-заглушка (HTTP 503 Service Unavailable), що запобігає втраті чи пошкодженню даних під час транзакцій, які можуть збігтися з моментом оновлення бази даних.

2. Оновлення кодової бази та залежностей: Після завантаження нових файлів на сервер ініціюється команда composer install --optimize-autoloader --no-dev. Вона гарантує, що будуть встановлені виключно необхідні для роботи системи пакети (без бібліотек для тестування), а класи будуть оптимізовані для максимальної швидкості завантаження.

3. Оновлення структури БД: Виконання міграцій (php artisan migrate --force). Прапорець --force є обов'язковим для Production-середовища, оскільки він

дозволяє застосувати зміни до схеми бази даних без додаткових підтверджень з боку користувача.

4. Оптимізація та кешування: Скидання старого та генерація нового кешу для конфігураційних файлів, маршрутів та шаблонів (php artisan optimize). Це суттєво зменшує навантаження на файлову систему під час обробки HTTP-запитів.

5. Відновлення доступу: Повернення системи в робочий стан командою php artisan up.

Такий підхід дозволяє проводити оновлення SaaS-платформи за лічені секунди з мінімальним часом простою (Zero-Downtime Deployment концепція).

Стратегія та автоматизація резервного копіювання (Backup Strategy)

Враховуючи, що система управління проєктами містить критично важливу корпоративну інформацію користувачів (стратегічні беклоги, оцінки завдань, історію спринтів), забезпечення надійного резервного копіювання є одним із найголовніших завдань архітектури безпеки.

Для автоматизації створення резервних копій було використано вбудований планувальник завдань фреймворку (Laravel Task Scheduler). Планувальник працює у зв'язці з системним демоном cron на сервері Linux, який щохвилини викликає команду php artisan schedule:run, передаючи управління ядру застосунку.

Безпосередня логіка створення копій налаштована у файлі app/Console/Kernel.php наступним чином:

- Частота та розклад: Резервне копіювання бази даних MySQL ініціюється автоматично щодня о 03:00 за місцевим часом. Цей час обрано як період найменшої активності користувачів, щоб уникнути зниження продуктивності системи (Table Locks) під час читання великих об'ємів даних.
- Механізм архівації: Система використовує утиліту mysqldump для формування повного SQL-дампа структури та даних усіх таблиць. Створений файл автоматично стискається за допомогою алгоритму GZIP (.sql.gz), що дозволяє зменшити розмір архіву в середньому на 70-80% і зекономити дисковий простір сервера.

- Політика утримання (Retention Policy): Для уникнення переповнення пам'яті сервера реалізовано механізм ротації (очищення) старих архівів. Скрипт перевіряє директорію сховища і автоматично видаляє файли резервних копій, термін створення яких перевищує 14 днів.
- Резервування: Згенерований архів не лише зберігається в локальній директорії storage/app/backups, але й копіюється на віддалений FTP-сервер. Це є частковою імплементацією правила резервного копіювання "3-2-1", що захищає дані у випадку критичного апаратного збою (виходу з ладу жорсткого диска) на основному сервері.

Для наочної демонстрації взаємодії програмних компонентів під час збереження даних, було розроблено структурну схему процесу резервного копіювання, яку наведено на рисунку 3.6.

Рисунок 3.6 – Структурна схема процесу автоматичного резервного копіювання та ротації даних

Логування та моніторинг помилок у робочому середовищі

З метою забезпечення безпеки та приховання архітектурних деталей від потенційних зловмисників, у конфігураційному файлі .env робочого сервера параметр APP_DEBUG встановлено у значення false. Це означає, що у випадку

виникнення критичної помилки на сервері, користувач побачить лише стандартну сторінку "500 Server Error" без виведення трасування стека викликів (Stack Trace).

Проте для ефективного обслуговування системи адміністратору необхідний доступ до детальної інформації про помилки. Тому в проєкті налаштовано підсистему логування на базі бібліотеки Monolog (вбудованої в Laravel).

У файлі config/logging.php активовано канал логування типу daily (щоденний). Замість запису всіх подій в один суцільний масивний файл (який з часом стає непридатним для читання та аналізу), система генерує окремий файл журналу для кожного дня (наприклад, laravel-2026-06-15.log).

У системні журнали записуються:

- Внутрішні виключення системи (Exceptions) та помилки виконання SQL-запитів.
- Спроби несанкціонованого доступу (наприклад, коли користувач намагається звернутися до маршруту чужого робочого простору, отримуючи помилку 403 Forbidden). Це дозволяє адміністратору ідентифікувати підозрілу активність та можливі спроби сканування вразливостей.
- Помилки валідації критичних форм.

Налаштована інфраструктура розгортання, моніторингу та бекапування перетворює розроблену систему управління проєктами на стійкий до збоїв SaaS-продукт, повністю готовий до комерційної або внутрішньокорпоративної експлуатації.

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

Процес розробки SaaS-системи управління проєктами за методологією Scrum передбачає тривалу розумову працю та постійну взаємодію розробника з персональним комп'ютером (ПК) і відеодисплейними терміналами (ВДТ). Специфіка такої діяльності пов'язана з гіподинамією, значним зоровим та нервово-емоційним напруженням, а також впливом електромагнітного випромінювання. Тому забезпечення безпечних і здорових умов праці є невіддільною складовою організації робочого процесу.

4.1 Безпека життєдіяльності

Головним завданням безпеки життєдіяльності під час створення програмного продукту є мінімізація шкідливого впливу виробничого середовища на організм програміста [22]. Основними небезпечними факторами є: нераціональне освітлення, підвищений рівень електромагнітного випромінювання від моніторів та ергономічні проблеми, пов'язані з тривалим перебуванням у сидячому положенні.

Ергономічні вимоги до організації робочого місця Для забезпечення високої працездатності та збереження здоров’я розроблено раціональне компонування робочого місця. Робочий стіл має висоту 720 мм, що дозволяє вільно розмістити ноги. Для роботи використовується крісло з регульованою висотою сидіння (в межах 400-500 мм) та кутом нахилу спинки, що забезпечує підтримку поперекового відділу хребта і знижує статичне навантаження на м'язи спини.

Монітор розташований на відстані 600 мм від очей розробника, при цьому верхній край екрана знаходиться на рівні очей, що запобігає перенапруженню шийного відділу. Клавіатура та миша розміщені на спеціальній висувній полиці, що дозволяє підтримувати кут згину в ліктьовому суглобі близько 90 градусів.

Санітарно-гігієнічні вимоги та освітлення Мікроклімат у приміщенні підтримується на оптимальному рівні. Оптимальна температура повітря становить

22-24 °С, відносна вологість — 40-60%. Для досягнення таких параметрів приміщення обладнане системою кондиціонування та регулярно провітрюється. Зведені оптимальні показники мікроклімату наведено у таблиці 4.1.

Таблиця 4.1 – Оптимальні параметри мікроклімату на робочому місці

Період року	**Температура повітря, °С**	**Відносна вологість, %**	**Швидкість руху повітря, м/с**
Холодний	22 – 24	40 – 60	не більше 0,1
Теплий	23 – 25	40 – 60	не більше 0,1

Особлива увага приділена освітленню, оскільки робота з кодом (PHP, JavaScript) вимагає високої концентрації зору. Рівень освітленості на робочому столі становить 500 лк. Використовується комбіноване освітлення: природне світло (через вікна, обладнані жалюзі для уникнення відблисків на екрані) та штучне освітлення за допомогою світлодіодних ламп із колірною температурою 4000К, що максимально наближена до природного спектра і не викликає втоми очей.

4.2 Основи охорони праці

До основних аспектів охорони праці при розробці та розгортанні серверної частини програмного забезпечення належать електробезпека та захист від пожеж, оскільки комп'ютерна техніка і серверне обладнання підключені до електричної мережі [23].

Електробезпека на робочому місці Персональний комп'ютер та периферійні пристрої живляться від мережі змінного струму напругою 220 В, що створює ризик ураження електричним струмом у разі пошкодження ізоляції. Робоче приміщення належить до приміщень без підвищеної небезпеки (сухе, з нормальною температурою та струмонепровідною підлогою).

Для захисту від ураження електричним струмом реалізовано такі інженерно-технічні заходи:

- Усі розетки робочого місця обладнані заземлюючим контактом, який надійно з'єднаний із контуром захисного заземлення будівлі.
- Лінія живлення комп'ютерної техніки оснащена пристроєм захисного відключення (ПЗВ) зі струмом спрацьовування 30 мА, що гарантує миттєве відключення напруги при витоку струму на корпус.
- Кабелі живлення та мережеві дроти впорядковані за допомогою пластикових коробів (кабель-каналів), що виключає їх механічне пошкодження або випадкове заплутування під ногами.

Протипожежні заходи Комп'ютерна техніка містить горючі матеріали (пластикові корпуси, ізоляція кабелів) і може стати джерелом займання внаслідок короткого замикання або перегріву блоку живлення. Приміщення розробника належить до категорії В (пожежонебезпечне).

Для забезпечення пожежної безпеки впроваджено такі рішення:

- Приміщення обладнане автоматичною системою пожежної сигналізації з димовими сповіщувачами, які реагують на початкові стадії тління ізоляції.
- Робоче місце укомплектоване вуглекислотним вогнегасником (ВВК-2). Вибір саме вуглекислотного вогнегасника обґрунтований тим, що він призначений для гасіння електроустановок під напругою до 1000 В (клас пожежі Е) і не пошкоджує електронні компоненти серверів та ПК, на відміну від порошкових чи пінних аналогів.
- Електрична мережа захищена автоматичними вимикачами, які запобігають перегріву проводки у разі короткого замикання або перевищення допустимого навантаження.
- На видному місці розміщено план евакуації, а шляхи евакуації звільнені від сторонніх предметів.

Типовий алгоритм дій персоналу, передбачений затвердженим планом евакуації у разі виникнення нештатної ситуації (пожежі), наведено у таблиці 4.2.

Таблиця 4.2 – Порядок дій згідно з планом евакуації у разі пожежі

Етап евакуації	**Послідовність дій персоналу**	**Відповідальні особи / Засоби**
1. Виявлення пожежі	Негайне повідомлення пожежної охорони (за номером 101), вказавши точну адресу та місце займання.	Кожен працівник, який першим виявив ознаки пожежі
2. Оповіщення	Активація ручного пожежного сповіщувача, голосове оповіщення колег у робочому приміщенні.	Система пожежної сигналізації / Персонал
3. Знеструмлення	Відключення електроживлення серверного та комп'ютерного обладнання на загальному автоматичному щитку.	Адміністратор приміщення / Відповідальний за електробезпеку
4. Евакуація	Організований вихід із приміщення згідно зі стрілками-покажчиками на плані евакуації. Сувора заборона користування ліфтами.	Усі працівники, які перебувають у приміщенні
5. Первинне гасіння	Спроба локалізувати займання за допомогою вуглекислотних вогнегасників (ВВК-2) (виконується виключно за умови відсутності загрози життю та здоров'ю).	Добровільна пожежна дружина / Навчений персонал

Впроваджені інженерні та організаційні рішення з охорони праці та безпеки життєдіяльності повністю створюють безпечне, комфортне і продуктивне середовище для розробки програмного забезпечення.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи бакалавра було розв'язано актуальну науково-технічну задачу розробки інформаційної системи (SaaS-додатка) для управління проєктами за гнучкою методологією Scrum. Створений програмний продукт дозволяє автоматизувати процеси планування, моніторингу та контролю за виконанням завдань у командах розробників.

Під час виконання роботи було отримано наступні основні результати:

1. Проведено системний аналіз предметної області**.** Досліджено існуючі програмні рішення для управління проєктами (Jira, Trello, Asana) та виявлено їхні переваги і недоліки. Сформульовано функціональні та нефункціональні вимоги до розроблюваної системи з урахуванням специфіки методології Scrum.

2. Спроєктовано архітектуру програмного забезпечення. Використання архітектурного шаблону MVC (Model-View-Controller) та фреймворку Laravel забезпечило чітке розділення бізнес-логіки, даних та інтерфейсу користувача. Розроблено UML-діаграму класів, що описує об'єктно-орієнтовану структуру системи та взаємодію контролерів з моделями даних.

3. Розроблено оптимізовану реляційну базу даних. Спроєктовано та нормалізовано до третьої нормальної форми (3NF) базу даних MySQL, яка складається з 10 взаємопов'язаних відношень. Реалізовано складні ієрархічні зв'язки (один-до-багатьох, багато-до-багатьох) для сутностей робочих просторів, проєктів, спринтів та завдань із забезпеченням посилальної цілісності через механізм міграцій.

4. Програмно реалізовано ключовий функціонал системи. Створено механізми ізоляції даних на рівні робочих просторів (Workspaces) із системою розмежування прав доступу (Role-based access control). Розроблено інтерфейс управління беклогом продукту (Product Backlog) та реалізовано інтерактивну Канбан-дошку активного спринту з підтримкою асинхронної зміни статусів завдань (технологія Drag-and-Drop) за допомогою Fetch API.

5. Проведено тестування та розгортання системи. Виконано функціональне тестування бізнес-логіки системи. Програмний продукт успішно верифіковано та розгорнуто на віддаленому сервері (хмарний хостинг), що підтверджує його готовність до експлуатації в реальних умовах як MVP (мінімально життєздатний продукт).

Окрім функціональної частини, під час верифікації розгорнутої інфраструктури особливу увагу було приділено питанням інформаційної безпеки та стійкості. У системі налаштовано базовий захист від поширених вебвразливостей (зокрема XSS та CSRF) шляхом конфігурації відповідних HTTP-заголовків безпеки та використання механізмів ядра Laravel. Проведене навантажувальне тестування підтвердило високу стійкість архітектури до одночасних конкурентних запитів від багатьох користувачів, що є критичною вимогою для додатків SaaS-типу.

У рамках виконання кваліфікаційної роботи також було розв'язано питання охорони праці та безпеки життєдіяльності. Розроблено комплекс інженерно-технічних та організаційних заходів для забезпечення безпечних і комфортних умов розумової праці IT-фахівців під час створення програмного продукту. Зокрема, обґрунтовано ергономічні й санітарно-гігієнічні вимоги до робочого місця розробника, запропоновано заходи з електробезпеки при роботі з комп'ютерною технікою та описано алгоритм дій на випадок виникнення пожежонебезпечних ситуацій.

Практичне значення одержаних результатів полягає у створенні готового до використання програмного забезпечення, яке може бути впроваджене в невеликих IT-командах (стартапах) для оптимізації процесів управління життєвим циклом розробки (SDLC) без необхідності використання перевантажених або дорогих корпоративних аналогів. Застосований стек технологій (PHP, Laravel, MySQL, Tailwind CSS) забезпечує високу продуктивність та простоту подальшого масштабування системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Грицюк Ю. І., Рак Т. Є. Програмування мовою PHP: навч. посіб. Львів: Вид-во ЛДУ БЖД, 2018. 321 с.

2. Костик П. В., Тиш Є. В. Фактори впливу на ефективність проектування програмних інтерфейсів комп'ютерних систем // Інформаційні моделі, системи та технології: Матеріали VI наук.-техн. конф. ТНТУ ім. І.Пулюя. Тернопіль, 2018. С. 85.

3. Кравченко О. В. Сучасні підходи до розробки інтерактивних вебзастосунків // Вісник Національного технічного університету "ХПІ". 2022. № 1. С. 15–22.

4. Мельник А. О. Архітектура програмних систем: підручник. Львів: Видавництво Львівської політехніки, 2020. 344 с.

5. Недашківський О. М. Планування та проектування інформаційних систем. Київ, 2014. 215 с.

6. Петрик М. Р., Петрик О. Ю. Моделювання програмного забезпечення. Тернопіль: Вид-во ТНТУ, 2015. 200 с.

7. Савченко В. В. Порівняльний аналіз фреймворків для розробки вебзастосунків на базі мови PHP // Сучасні інформаційні системи. 2021. Т. 5, № 3. С. 78–84.

8. Синєокий О. В. Особливості застосування гнучких методологій розробки програмного забезпечення // Інформаційні технології та комп'ютерна інженерія. 2021. № 2. С. 45–52.

9. Duckett J. PHP & MySQL: Server-side Web Development. Wiley, 2022. 672 p.

10. Eloquent ORM Documentation. Laravel, 2024. URL: https://laravel.com/docs/eloquent (дата звернення: 14.06.2026).

11. Freeman E., Robson E. Head First Design Patterns: Building Extensible and Maintainable Object-Oriented Software. 2nd ed. O'Reilly Media, 2020. 672 p.

12. Guide to the Software Engineering Body of Knowledge (SWEBOK Guide). Version 4.0 / ed. H. Washizaki. IEEE Computer Society, 2024. 411 p.

13. Laravel Documentation. Release 11.x. 2024. URL: https://laravel.com/docs/11.x (дата звернення: 14.06.2026).

14. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Prentice Hall, 2017. 432 p.

15. MDN Web Docs: Fetch API. Mozilla, 2023. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API (дата звернення: 14.06.2026).

16. MySQL 8.0 Reference Manual. Oracle Corporation, 2023. URL: https://dev.mysql.com/doc/refman/8.0/en/ (дата звернення: 14.06.2026).

17. PlantUML Language Reference Guide. 2024. URL: https://plantuml.com/guide (дата звернення: 14.06.2026).

18. Schwaber K., Sutherland J. The Scrum Guide. The Definitive Guide to Scrum: The Rules of the Game. 2020. URL: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf (дата звернення: 14.06.2026).

19. Stauffer M. Laravel: Up & Running: A Framework for Building Modern PHP Apps. 3rd ed. O'Reilly Media, 2023. 582 p.

20. Tailwind CSS Documentation. Tailwind Labs, 2024. URL: https://tailwindcss.com/docs (дата звернення: 14.06.2026).

21. Welling L., Thomson L. PHP and MySQL Web Development. 5th ed. Addison-Wesley Professional, 2016. 1008 p.

22. Желібо Є.П. Безпека життєдіяльності : підручник / В. В. Зацарний. Київ : Каравела, 2023. 344 с.

23. Жидецький В.Ц. Охорона праці користувачів комп’ютерів : підручник. Львів : Афіша, 2020. 176 с.

24. Oleh Zaiats; Dmytro Mykhalyk; Vasyl Yatsyshyn; Oleh Pastukh; Dmytro Uhryn Methods for integrating large language models into requirements management in agile methodologies / ITTAP-2025: 5th International Workshop on Information

Technologies: Theoretical and Applied Problems| (2025), CEUR Workshop Proceedings Volume 4146. P.379-397

25. M.R. Petryk, A.Yu. Doroshenko, D.M. Mykhalyk, O.A. Yatsenko Automated Parallelization of Software for Identifying Parameters of Intraparticle Diffusion and Adsorption in Heterogeneous Nanoporous Media. In: , et al. Mathematical Modeling and Simulation of Systems. MODS 2022. Lecture Notes in Networks and Systems, vol 667. Springer, Cham.(2023)

26. Буров Є. В. Конструювання програмного забезпечення: навчальний посібник. Львів: Видавництво Львівської політехніки, 2021. 316 с.

27. Пасічник В. В., Резніченко В. А. Організація баз даних та знань. Київ: Видавнича група BHV, 2018. 384 с.

28. Кухарєва О. М., Радченко А. О. Сучасні підходи до проєктування користувацьких інтерфейсів вебдодатків // Комп'ютерно-інтегровані технології: освіта, наука, виробництво. 2022. № 47. С. 112–118.

29. Kim G., Humble J., Debois P., Willis J. The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations. 2nd ed. IT Revolution Press, 2021. 480 p.

30. Бойко В. І., Ткаченко О. М. Аналіз та застосування інструментів безперервної інтеграції (CI/CD) у процесі розробки програмного забезпечення // Системи управління, навігації та зв'язку. 2023. Вип. 2 (72). С. 85–91.

ДОДАТКИ

ДОДАТОК А

Тези наукової конференції

Міністерство освіти і науки України
Тернопільський національний технічний університет
імені Івана Пулюя
Маріборський університет (Словенія)
Технічний університет в Кошице (Словаччина)
Каунаський технологічний університет (Литва)
Львівський національний університет
імені Івана Франка
Гірничо-металургійна академія ім. Станіслава Сташиця (Польща)
Луцький національний технічний університет
Чернівецький національний університет
імені Юрія Федьковича
Вроцлавський економічний університет (Польща)
Університет технологій та економіки
імені Хелени Ходковської (Польща)
Донбаська державна машинобудівна академія

Студентське наукове товариство

IX МІЖНАРОДНА

студентська науково - технічна конференція

"ПРИРОДНИЧІ ТА ГУМАНІТАРНІ НАУКИ. АКТУАЛЬНІ ПИТАННЯ"

24-25 квітня 2026 р.

(збірник тез конференції)

Тернопіль 2026

УДК 004.42
Рубльов А. - ст. гр. СП-42
Тернопільський національний технічний університет імені Івана Пулюя

РОЗРОБКА ІНТЕРАКТИВНОЇ SCRUM-ДОШКИ У ВЕБСИСТЕМІ УПРАВЛІННЯ ПРОЄКТАМИ

Науковий керівник: Михалик Д.М. - к.т.н., доцент

Rublov A.
Ternopil Ivan Puluj National Technical University

DEVELOPMENT OF AN INTERACTIVE SCRUM BOARD IN A WEB-BASED PROJECT MANAGEMENT SYSTEM

Supervisor: Mykhalyk D. M., Ph.D., Assoc. Prof.

Ключові слова: Scrum-дошка, PHP, управління проєктами, Agile, реляційна БД.
Key words: Scrum board, PHP, project management, Agile, relational DB.

В сучасному світі ефективність управління командами розробників напряму залежить від якості інструментів візуалізації процесів та правильної організації життєвого циклу програмного забезпечення [2]. Методологія Scrum є одним із найпопулярніших підходів до управління проєктами завдяки своїй гнучкості та ітеративності, а ключовим інструментом тут виступає інтерактивна дошка завдань.

Пропонується підхід до розробки вебмодуля «Scrum-дошка», який забезпечує динамічне відображення та зміну статусів завдань у режимі реального часу. Для реалізації системи було спроєктовано структуру реляційної бази даних з ключовими сутностями для збереження завдань, довідника статусів (To Do, In Progress, Review, Done) та інформації про спринти, використовуючи можливості СУБД MySQL [4].

Для розробки бекенд-частини було обрано мову PHP [3] із застосуванням архітектурного патерну MVC (Model-View-Controller). Використання цього патерну дозволяє побудувати чисту архітектуру з чітким розділенням бізнес-логіки та представлення даних.

Основною інженерною перевагою є впровадження асинхронної взаємодії між клієнтом та сервером через технологію AJAX. Це дозволяє створювати динамічний веб-інтерфейс та синхронізувати стани без перезавантаження сторінки [1].

Програмну архітектуру розробленого модуля на базі патерну MVC проілюстровано на рисунку 1. Взаємодія між клієнтськими запитами та базою даних відбувається через клас BoardController, який звертається до моделі TaskModel для виконання транзакцій оновлення статусів (наприклад, метод updateTaskStatus).

Рисунок 1 – UML-діаграма класів модуля Scrum-дошки

При перетягуванні картки завдання (Drag-and-Drop) на клієнтській стороні генерується запит до PHP-контролера, який валідує дані та оновлює відповідні записи в базі даних. Такий підхід мінімізує мережевий трафік, покращує користувацький досвід та дозволяє легко масштабувати систему або додавати нові типи завдань у майбутньому. Подальші дослідження мають бути спрямовані на інтеграцію розробленого модуля у повноцінну систему управління проєктами та автоматизацію формування беклогу.

Окрему увагу під час проєктування приділено питанням надійності та оптимізації клієнт-серверної взаємодії. Оскільки зміна статусів на Scrum-дошці відбувається асинхронно, система використовує механізм багаторівневої валідації. Зі свого боку, на рівні серверного PHP-контролера здійснюється фінальна перевірка цілісності даних та авторизації сесії користувача перед виконанням транзакції у базі даних. Дане рішення дозволяє суттєво знизити кількість холостих звернень до сервера, відсіюючи некоректні дії ще на етапі фронтенду, та гарантує структурну цілісність проєкту.

Такий підхід мінімізує мережевий трафік, покращує користувацький досвід та дозволяє легко масштабувати систему або додавати нові типи завдань у майбутньому. Подальші дослідження мають бути спрямовані на інтеграцію розробленого модуля у повноцінну систему управління проєктами та автоматизацію формування беклогу.

Список використаних джерел:

1. Ніксон Р. Створюємо динамічні веб-сайти з допомогою PHP, MySQL, JavaScript, CSS і HTML5 / Р. Ніксон. — Київ : Діалектика, 2019. — 816 с.
2. Соммервілл І. Інженерія програмного забезпечення / І. Соммервілл. — 10-те вид. — Київ : Центр навчальної літератури, 2020. — 728 с.
3. Офіційний посібник з мови програмування PHP. — [Електронний ресурс]. — Режим доступу: https://www.php.net/manual/uk/
4. Довідкове керівництво з MySQL (MySQL Reference Manual). — [Електронний ресурс]. — Режим доступу: https://dev.mysql.com/doc/

ДОДАТОК Б

ЛІСТИНГ ПРОГРАМНОГО КОДУ ОСНОВНИХ МОДУЛІВ СИСТЕМИ

Контролер для управління завданнями та обробки логіки Scrum-дошки:

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\Project;
use Illuminate\Support\Facades\Gate;

class TaskController extends Controller
{
    // Відображення беклогу
    public function index(Project $project)
    {
        // Захист: перевіряємо, чи має користувач доступ до простору 
цього проєкту
        if (!$project->workspace->users()->where('user_id', auth()-
>id())->exists()) {
            abort(403, 'У вас немає доступу до цього проєкту.');
        }

        // Дістаємо завдання проєкту, які ще не прив'язані до 
спринту (sprint_id IS NULL)
        $backlogTasks = $project->tasks()->whereNull('sprint_id')-
>latest()->get();

        $sprints = $project->sprints()->latest()->get();

        $members = $project->workspace->users;

        return view('projects.backlog', compact('project', 
'backlogTasks', 'sprints', 'members'));
    }

    // Збереження нового завдання
    public function store(Request $request, Project $project)
    {
        if (!$project->workspace->users()->where('user_id', auth()-
>id())->exists()) {
            abort(403, 'У вас немає доступу до цього проєкту.');
        }

        // Валідація даних з форми (типи та пріоритети мають 
збігатися з ENUM у базі)
        $request->validate([
            'title' => 'required|string|max:255',
            'type' => 'required|in:story,task,bug',
```

```
            'priority' => 'required|in:low,medium,high',
            'story_points' => 'nullable|integer|min:1|max:100',
            'assignee_id' => 'nullable|exists:users,id',
        ]);

        // Створення завдання
        $project->tasks()->create([
            'title' => $request->title,
            'type' => $request->type,
            'priority' => $request->priority,
            'story_points' => $request->story_points,
            'assignee_id' => $request->assignee_id,
            'status' => 'todo', // за замовчуванням
            'creator_id' => auth()->id(),
            // sprint_id залишається null, тому таска потрапляє в
беклог
        ]);

        return back()->with('status', 'Завдання додано до
беклогу!');
    }
    // Метод для переміщення завдання у спринт
    public function moveToSprint(Request $request, \App\Models\Task
$task)
    {
        // Базовий захист доступу
        if (!$task->project->workspace->users()->where('user_id',
auth()->id())->exists()) {
            abort(403, 'У вас немає доступу до цього завдання.');
        }

        $request->validate([
            'sprint_id' => 'required|exists:sprints,id',
        ]);

        // Оновлюємо sprint_id у таски
        $task->update([
            'sprint_id' => $request->sprint_id
        ]);

        return back()->with('status', 'Завдання переміщено у
спринт!');
    }
    // Оновлення статусу завдання через Drag-and-Drop
    public function updateStatus(Request $request, \App\Models\Task
$task)
    {
        // Базовий захист
        if (!$task->project->workspace->users()->where('user_id',
auth()->id())->exists()) {
            return response()->json(['error' => 'Немає доступу'],
403);
```

```
        }

        $request->validate([
            'status' => 'required|in:todo,in_progress,done'
        ]);

        $task->update([
            'status' => $request->status
        ]);

        return response()->json(['message' => 'Статус оновлено',
'task' => $task]);
    }

    // Відображення сторінки деталей завдання
    public function show(\App\Models\Task $task)
    {
        // Перевірка доступу через TaskPolicy
        Gate::authorize('view', $task);

        return view('tasks.show', compact('task'));
    }

    // Оновлення даних завдання (опис, пріоритет тощо)
    public function update(Request $request, \App\Models\Task $task)
    {
        Gate::authorize('update', $task);

        $request->validate([
            'title' => 'required|string|max:255',
            'description' => 'nullable|string',
            'type' => 'required|in:story,task,bug',
            'priority' => 'required|in:low,medium,high',
            'story_points' => 'nullable|integer|min:1|max:100',
            'assignee_id' => 'nullable|exists:users,id',
        ]);

        // Оновлюємо лише дозволені поля
        $task->update($request->only(['title', 'description',
'type', 'priority', 'story_points']));

        $task->update($request->only(['title', 'description',
'type', 'priority', 'story_points', 'assignee_id']));

        return back()->with('status', 'Деталі завдання успішно
оновлено!');
    }

    // Видалення завдання
    public function destroy(\App\Models\Task $task)
    {
```

```
        // Перевірка доступу (можна через Gate або існуючу логіку
простору)
        if (!$task->project->workspace->users()->where('user_id',
auth()->id())->exists()) {
            abort(403, 'У вас немає доступу до цього завдання.');
        }

        $task->delete();

        return back()->with('status', 'Завдання успішно видалено!');
    }

}
```

Клієнтський скрипт для реалізації Drag-and-Drop інтерфейсу без перезавантаження сторінки:

```
document.addEventListener('DOMContentLoaded', () => {
    const draggables = document.querySelectorAll('.task-card');
    const dropzones = document.querySelectorAll('.scrum-column');

    draggables.forEach(draggable => {
        draggable.addEventListener('dragstart', () => {
            draggable.classList.add('dragging');
        });

        draggable.addEventListener('dragend', () => {
            draggable.classList.remove('dragging');
        });
    });

    dropzones.forEach(zone => {
        zone.addEventListener('dragover', e => {
            e.preventDefault();
            const afterElement = getDragAfterElement(zone,
e.clientY);
            const draggable = document.querySelector('.dragging');
            if (afterElement == null) {
                zone.appendChild(draggable);
            } else {
                zone.insertBefore(draggable, afterElement);
            }
        });

        zone.addEventListener('drop', e => {
            const draggable = document.querySelector('.dragging');
            const taskId = draggable.dataset.taskId;
            const newStatus = zone.dataset.status;

            // Відправка асинхронного запиту на сервер
            updateTaskStatusOnServer(taskId, newStatus);
        });
```

```
    });

    function updateTaskStatusOnServer(taskId, status) {
        const csrfToken = document.querySelector('meta[name="csrf-
token"]').content;

        fetch(`/tasks/${taskId}/status`, {
            method: 'PATCH',
            headers: {
                'Content-Type': 'application/json',
                'X-CSRF-TOKEN': csrfToken,
                'Accept': 'application/json'
            },
            body: JSON.stringify({ status: status })
        })
        .then(response => {
            if (!response.ok) {
                console.error('Помилка оновлення статусу на
сервері');
            }
        });
    }
});
```