Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

РОЗРОБКА ВЕБПОМІЧНИКА «SMARTLIST» З ВИКОРИСТАННЯМ PHP, LARAVEL ТА MYSQL

DEVELOPMENT OF THE «SMARTLIST» WEB ASSISTANT USING PHP, LARAVEL AND MYSQL

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗ-41
Подолянчук Назар Борисович

Керівник:
Гульчук Юрій Миколайович

Кваліфікаційну роботу
допущено до захисту
«10» 06 2025 р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Наталія Миколаївна

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти *бакалавр*
Галузь знань: *12 «Інформаційні технології»*
Спеціальність: *121 «Інженерія програмного забезпечення»*
Освітня програма: «*Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____

«*20*» *12* 20*24* р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Подолянчуку Назару Борисовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *«Розробка вебпомічника «SmartList» з використанням PHP, Laravel та MySQL»*
Керівник роботи: *асистент Гульчук Ю. М.*
затверджені наказом закладу вищої освіти від *«19» грудня 2024 р.* № *474/01-02*
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи бакалавра *«10» червня 2025 р.*
3. Вихідні дані до роботи: *PHP, Laravel, MySQL методичні вказівки до виконання кваліфікаційної роботи бакалавра*
4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити): *проаналізувати наявні системи для управління завданнями проєкту, розробити структуру бази даних та реалізувати міграції в MySQL, реалізувати основний функціонал: створення, редагування, виконання та сортування завдання створити адаптивний UX/UI з використанням Blade-шаблонів, виконати тестування*
5. Перелік графічного матеріалу: *5 рисунків,1 додаток*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Гульчук Ю. М.*		
Специфікація вимог до розробленої системи	*Гульчук Ю. М.*		
Розробка об'єкта проектування	*Гульчук Ю. М.*		
Нормоконтроль	*Вознюк А. В.*		
Гарант ОП	*Ліщина Н. М.*		
Показник запозичень тексту	2,37 %		
Академічна доброчесність	*Гульчук Ю. М.*		

7. Дата видачі завдання *«20» грудня 2024 р.*

№	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	Огляд літературних джерел по темі кваліфікаційної роботи бакалавра	до 04.02.2025 р.	Виконано
2	Аналіз проблеми розробки та впровадження об`єкту проектування	до 01.03.2025 р.	Виконано
3	Обґрунтування вибору шляхів, технологій і засобів вирішення поставленого завдання	до 15.03.2025 р.	Виконано
4	Розробка функціонально-структурної схеми роботи об'єкта проектування та проектування бази даних	до 29.03.2025 р.	Виконано
5	Практична реалізація об`єкта проектування та розробка бази даних	до 26.04.2025 р.	Виконано
6	Тестування та налагодження об'єкта проектування	до 03.05.2025 р.	Виконано
7	Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру	до 10.06.2025 р.	Виконано

Здобувач вищої освіти _____ Назар ПОДОЛЯНЧУК

Керівник кваліфікаційної роботи _____ Юрій ГУЛЬЧУК

АНОТАЦІЯ

Подянчук Н. Б. Розробка вебпомічника «SmartList» з використанням PHP, Laravel та MySQL. Рукопис. Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення» спеціальності «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатку.

У першому розділі здійснено аналіз предметної області, розглянуто існуючі рішення для управління завданнями, визначено проблеми, які вирішує SmartList, а також сформульовано цілі, завдання, об єкт і предмет дослідження. У другому розділі наведено специфікацію вимог до вебзастосунку, розроблено архітектуру системи, спроєктовано базу даних, визначено функціональні та нефункціональні характеристики, описано користувацькі ролі, структуру інтерфейсу та методи тестування. У третьому розділі описано практичну реалізацію програмного забезпечення. Подано структуру Laravel-проєкту, фрагменти коду контролерів, шаблонів і моделей, продемонстровано реалізацію основного функціоналу вебпомічника, а також приклади сторінок інтерфейсу користувача. У висновках підбито підсумки розробки системи SmartList, оцінено досягнення поставленої мети, визначено перспективи подальшого вдосконалення програмного продукту.

Ключові слова: PHP, MySQL, Laravel, SmartList.

ABSTRACT

Podianchuk N. Development of the "SmartList" Web Assistant Using PHP, Laravel and MySQL. Manuscript. Bachelor's qualification thesis in the Educational Program "Software Engineering" of thespecialty "Software Engineering". Lutsk National Technical University. Lutsk, 2025.

The bachelor's qualification thesis consists of an introduction, three chapters, conclusions, a list of references, and appendis.

The first chapter presents an analysis of the subject area, reviews existing task management solutions, identifies the problems that SmartList aims to solve, and defines the purpose, objectives, object, and subject of the research. The second chapter provides the specification of system requirements, describes the software architecture, designs the database structure, outlines functional and non-functional requirements, user roles, user interface components, and testing methods. The third chapter covers the practical implementation of the software. It presents the structure of the Laravel project, fragments of controller, model, and template code, demonstrates the implementation of key functionality of the web assistant, and includes examples of the user interface pages. The conclusion summarizes the results of the SmartList system development, evaluates the achievement of the stated goals, and identifies prospects for future improvement of the software product.

Keywords: PHP, MySQL, Laravel, SmartList.

ЗМІСТ

ВСТУП

Актуальність теми. Розробка вебпомічника «SmartList» з використанням PHP, Laravel та MySQL для управління особистими завданнями зумовлена стрімким розвитком цифрових технологій, зростанням обсягів інформації та необхідністю підвищення особистої ефективності користувачів у повсякденному житті, навчанні та професійній діяльності. У сучасному світі, де темп виконання завдань постійно зростає, а обсяг справ постійно розширюється, надзвичайно важливо мати надійні, зручні й гнучкі засоби для планування, контролю й управління власними задачами.

Існуючі програмні продукти, зокрема Trello, Todoist, Asana, Notion, хоча й мають широкий функціонал, здебільшого орієнтовані на корпоративний сегмент або англомовного користувача. Більшість із них є закритими хмарними платформами, що вимагає обов’язкової реєстрації, передбачає збереження даних на сторонніх серверах, часто має обмеження у безкоштовній версії, а також не завжди пропонує повну підтримку української мови. Така ситуація створює об'єктивну потребу у створенні автономного, локалізованого та адаптивного рішення, що дозволяло б користувачам самостійно керувати своїми справами без залежності від сторонніх сервісів.

У цьому контексті актуальним є створення власного вебпомічника для управління особистими завданнями, який поєднує простоту використання, функціональність та відкритість до подальшої модифікації. Саме таким проєктом є система «SmartList» – вебзастосунок, що розробляється на основі PHP, фреймворку Laravel та СУБД MySQL. Такий вибір технологій забезпечує масштабованість, стабільність, ефективність та безпеку зберігання даних.

Мета роботи полягає у створенні програмного засобу, який дозволяє користувачам створювати, редагувати, переглядати та організовувати свої завдання в інтуїтивно зрозумілому вебінтерфейсі. У розробці будуть реалізовані ключові функції – реєстрація користувача, авторизація, створення

та редагування завдань, фільтрація за статусом, категоризація, архівація, адаптивність до різних пристроїв.

Об’єктом кваліфікаційної роботи є процес організації та управління особистими завданнями за допомогою веборієнтованих програмних засобів.

Предметом кваліфікаційної роботи є методи, моделі та технології розробки вебзастосунку для керування персональними завданнями, з використанням PHP, Laravel і MySQL.

Для реалізації поставленої мети у межах кваліфікаційної роботи було сформульовано наступні завдання:

– провести аналіз існуючих вебрішень, що виконують функції особистих помічників або систем керування завданнями (Todo List, Task Manager тощо), визначити їх сильні та слабкі сторони;

– обґрунтувати вибір інструментів розробки, зокрема мови програмування PHP, фреймворку Laravel і системи керування базами даних MySQL, враховуючи їх переваги, недоліки та актуальність;

– розробити структуру та логіку функціонування вебпомічника, що включає архітектуру застосунку, структуру бази даних, взаємодію між користувачем і сервером;

– реалізувати функціонал реєстрації та автентифікації користувачів з урахуванням вимог безпеки;

– забезпечити можливість створення, редагування, сортування, видалення та перегляду завдань;

– розробити адаптивний інтерфейс користувача відповідно до принципів UX/UI-дизайну, що дозволяє зручно взаємодіяти з вебпомічником на різних пристроях (десктоп, планшет, смартфон);

– протестувати вебзастосунок на функціональність, стабільність та відповідність заданим вимогам.

Виконання зазначених завдань дало змогу реалізувати повноцінний прототип вебпомічника «SmartList», адаптований до потреб сучасного користувача та побудований на базі відкритих технологій.

Розроблений вебпомічник може бути використаний як у персональних цілях (планування особистих справ), так і в малих командах для базової координації робочих завдань. Його відкритий технологічний стек (PHP, Laravel, MySQL) дозволяє легко модифікувати і масштабувати програмний продукт, додаючи нові модулі (нагадування, категоризація завдань, календарна інтеграція тощо) або інтегрувати його у складніші системи (CRM, ERP). Це відкриває широкі можливості для подальшого вдосконалення застосунку та використання його як освітнього чи комерційного продукту. Крім того, досвід, здобутий під час реалізації цього проєкту, може бути використаний у подальшій професійній діяльності в галузі веб-розробки та програмної інженерії.

РОЗДІЛ 1
АНАЛІЗ ВЕБОРІЄНТОВАНИХ СИСТЕМ УПРАВЛІННЯ ЗАВДАННЯМИ І ПОСТАНОВКА ЗАВДАНЬ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблеми

У XXI столітті темп життя зростає з неймовірною швидкістю, що формує нові виклики перед людиною: необхідність швидко адаптуватися, ефективно планувати час, утримувати баланс між особистим та професійним. В умовах зростання обсягів інформації та обов'язків традиційні методи організації повсякденних справ – щоденники, блокноти, навіть електронні таблиці – дедалі частіше виявляються неефективними. На цьому тлі виникає об'єктивна потреба у використанні цифрових рішень, які можуть забезпечити користувачу централізоване управління завданнями, дедлайнами, пріоритетами та командною взаємодією. Такі інструменти здобули назву вебпомічників.

Ринок цифрових інструментів для планування розвинувся настільки, що охоплює як окремі додатки для приватного використання, так і комплексні корпоративні системи. Серед них – добре відомі Trello, Todoist, Notion, Asana, Microsoft To Do та інші. Вони демонструють приклади вдалого застосування концепцій канбан-дощок, списків завдань, таймлайн-планування, інтеграції з календарями, синхронізації між пристроями та хмарного збереження даних.

Наприклад, платформа Trello широко використовується як у бізнесі, так і серед фрілансерів. Її ключова особливість – візуальна подача задач через канбан-систему, що дозволяє оцінити загальний стан проєкту в межах однієї дошки. Аналогічно, Asana поєднує завдання зі строками виконання, відстеженням відповідальних осіб і дедлайнами. Вони підтримують складні сценарії: від індивідуального використання до корпоративної командної роботи з інтеграцією Slack, Google Drive тощо.

Проте попри свою функціональність, ці платформи мають низку обмежень. Часто вони побудовані на закритому коді, що унеможливлює адаптацію під специфічні потреби користувача, перевантажені

функціональністю, що створює бар’єр входження для нових користувачів, не мають повноцінної підтримки локалізації (зокрема української мови), обмежені в можливостях зміни інтерфейсу або логіки без платної підписки не завжди забезпечують збереження конфіденційності даних (наприклад, у безкоштовних тарифах дані можуть оброблятися сторонніми аналітичними сервісами) [1].

Ці недоліки стали особливо помітними під час активного переходу до віддаленої роботи, де особиста цифрова організованість має вирішальне значення. Потреба у простому, зрозумілому та адаптивному рішенні набула особливої актуальності в освітньому середовищі, серед студентів, викладачів, малого бізнесу та приватних осіб.

У відповідь на це дедалі більше уваги звертається на веборієнтовані рішення, що базуються на відкритому стеку технологій: PHP, Laravel, MySQL, JavaScript, HTML та CSS. Вони дозволяють створювати модульні системи, що легко масштабуються, адаптуються до конкретних вимог і є придатними для інтеграції в інші цифрові середовища.

Фреймворк Laravel, зокрема, пропонує гнучку архітектуру MVC, вбудовані механізми автентифікації, захисту від CSRF, ORM-модель Eloquent для взаємодії з базами даних та систему маршрутів, що дозволяє чітко структурувати логіку додатку [2]. MySQL, як класична реляційна СУБД, демонструє високу стабільність, продуктивність та гнучкість структурування даних, що підтверджено не лише спільнотою розробників, а й численними кейсами корпоративного використання [3].

У дослідженні [4] порівнюються підходи до розробки таск-менеджерів, де Laravel і Django виділяються як найбільш ефективні фреймворки для побудови продуктивних, безпечних та масштабованих рішень. Саме Laravel, з огляду на простоту реалізації авторизації, валідаторів, обробки запитів та REST API, визнано оптимальним вибором для створення CRUD-застосунків середнього рівня складності показано на рисунку 1.1

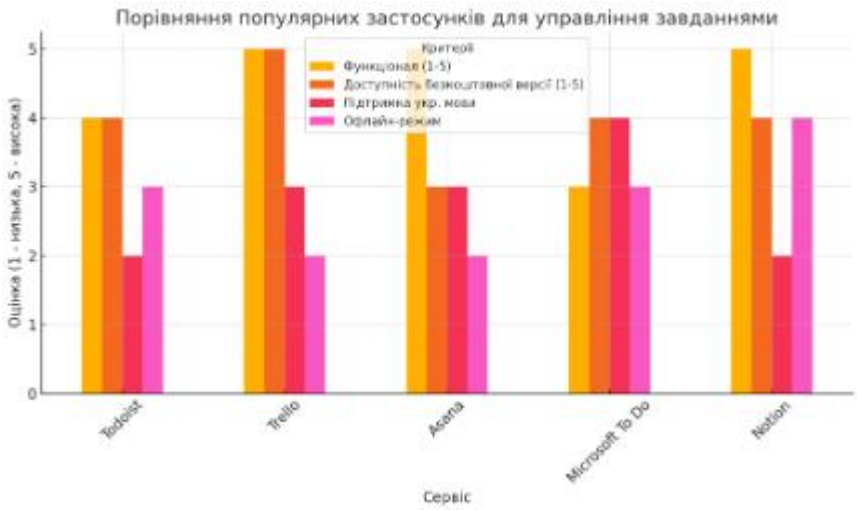

Рисунок 1.1 – Порівняння популярних застосунків для управління завданнями за основними критеріями [4]

У межах українського цифрового простору ситуація ще більше акцентує потребу у створенні локалізованих, доступних рішень. Згідно з аналітичним дослідженням [5], понад 70 % користувачів в Україні віддають перевагу українській мові в інтерфейсі, проте лише 30 % популярних сервісів її підтримують. Це є ще одним аргументом на користь власної розробки.

Більше того, у фахових публікаціях з цифрової грамотності вказується, що користувачі вимагають інтуїтивно зрозумілого інтерфейсу, гнучкої фільтрації завдань, автоматизації повторюваних дій, що реалізуються через cron-джоби, логіку сповіщень, категоризацію та тегування задач [6]. Відомо, що користувачам важливо мати можливість не лише створювати й редагувати завдання, а й мати ефективну візуалізацію прогресу, наприклад, через стовпчасті або кругові діаграми (charts.js, ApexCharts).

Окремої уваги заслуговує UX. У дослідженнях [7] [8] наголошується, що успішні таск-менеджери повинні не лише мати логічну структуру, а й забезпечувати емоційний комфорт користувача: зручне розташування елементів, мінімалістичний дизайн, швидкість реакції, адаптивність на мобільних пристроях – усе це формує лояльність до продукту.

Зрештою, створення власного вебпомічника не є лише вправою з програмування – це відповідь на виклики, які ставить сучасний стиль життя,

адаптований до умов локального користувача, з урахуванням мовних, культурних, функціональних та технічних особливостей. Проєкт «SmartList» має на меті реалізувати такий застосунок.

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

Метою кваліфікаційної роботи є розробка вебпомічника «SmartList» – інтерактивного вебзастосунку для управління списками завдань, що забезпечує зручний інтерфейс користувача, зберігання даних у базі даних та можливість керування задачами через браузер з будь-якого пристрою.

Для досягнення поставленої мети необхідно вирішити низку конкретних завдань:

– провести огляд і аналіз існуючих програмних рішень у сфері управління особистими та робочими завданнями;

– обґрунтувати вибір інструментальних засобів та технологій, серед яких: мова програмування PHP, фреймворк Laravel, система управління базами даних MySQL;

– спроєктувати структуру бази даних, яка дозволяє ефективно зберігати та обробляти інформацію про користувачів, категорії, завдання, терміни виконання, пріоритети тощо;

– реалізувати функціонал вебпомічника, що охоплює створення, редагування, перегляд, видалення та фільтрацію завдань;

– впровадити механізми реєстрації, автентифікації та авторизації користувачів для захисту персональних даних;

– розробити адаптивний вебінтерфейс, зручний для використання на ПК, планшетах і смартфонах;

– провести тестування роботи системи на предмет працездатності, продуктивності та зручності використання.

Таким чином, кваліфікаційна робота спрямована на розробку практично значущого вебзастосунку, що відповідає сучасним вимогам.

РОЗДІЛ 2
СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення

У сучасних умовах цифрової трансформації все більше організацій та приватних користувачів потребують ефективних інструментів для організації повсякденної діяльності, планування, контролю виконання задач та координації командної роботи. Існуючі рішення, такі як Trello, Notion, Todoist або Google Tasks, надають різноманітний функціонал, однак далеко не завжди повністю задовольняють потреби українських користувачів через складність інтерфейсу, обмежену кастомізацію або надлишкову функціональність. У цьому контексті актуальною є розробка власного, локалізованого, адаптованого до конкретних потреб вебпомічника, який забезпечить зручний спосіб управління завданнями та робочими процесами. Таким проєктом є система «SmartList».

Вебпомічник SmartList створюється як універсальний засіб організації особистих і командних справ, який забезпечує централізований доступ до списків завдань, можливість відстеження прогресу, призначення пріоритетів, ведення нагадувань і аналітики. Його основна ідея – спростити роботу користувача та дозволити фокусуватися на результаті, а не на складній системі керування. У зв’язку з цим постає завдання: створити вебсистему, яка поєднує простоту інтерфейсу, стабільність роботи та можливість масштабування.

Розробка такої системи потребує попереднього аналізу предметної області, визначення вимог і обмежень, а також формування архітектури, що дозволить у подальшому ефективно реалізувати функціонал. Аналіз показав, що цільовою аудиторією є малі команди, окремі користувачі, фрілансери, студенти, викладачі та представники малого бізнесу, які шукають простий інструмент для планування та моніторингу завдань. Серед типових процесів, які має підтримувати система, створення завдань, групування за списками,

призначення дедлайнів, зміна статусу (виконано/не виконано), можливість нагадувань, ведення історії та аналітика.

Система має підтримувати базову модель CRUD (створення, читання, оновлення, видалення) для завдань і списків завдань. Користувач повинен мати змогу авторизуватися, створити кілька списків, наприклад: «Навчання», «Покупки», «Робота», та додавати до них завдання з описом, терміном, пріоритетом. Інтерфейс повинен бути адаптивним, зручним як для перегляду з ПК, так і зі смартфону. Для підвищення юзабіліті реалізується механізм візуального позначення статусу (наприклад, кольорові індикатори або піктограми виконання).

До нефункціональних вимог відносяться вимоги до продуктивності, безпеки, доступності та масштабованості. Система повинна мати мінімальний час відгуку (менше 1 секунди при завантаженні списків), використовувати шифрування (HTTPS, bcrypt), підтримувати адаптивний інтерфейс та бути розширюваною – тобто такою, що дозволяє у майбутньому додавати нові модулі (наприклад, спільні завдання або інтеграції з Google Calendar).

На основі цих вимог сформовано загальну архітектуру системи. Обрана модель розробки – MVC (Model-View-Controller), реалізована за допомогою фреймворку Laravel. Контролери відповідають за обробку HTTP-запитів, взаємодію з моделями (Eloquent ORM) та повернення даних у вигляді HTML (Blade) або JSON (у випадку API). Моделі відображають структуру таблиць у базі даних і реалізують бізнес-логіку. Представлення (views) генерують HTML-сторінки для користувача. Така архітектура забезпечує модульність, гнучкість і спрощує тестування та підтримку.

Схема використання системи може бути представлена у вигляді класичного CRUD-процесу: користувач взаємодіє з вебінтерфейсом, надсилає запити на створення, редагування, перегляд або видалення завдань, а серверна частина обробляє ці запити, працює з базою даних та повертає відповідь. Це дає змогу зручно реалізувати як вебінтерфейс, так і потенційно – мобільний застосунок або API для сторонніх систем.

База даних системи побудована на трьох основних сутностях: users (користувачі), task_lists (списки завдань), tasks (окремі завдання). Таблиця users містить дані про облікові записи, task_lists зберігає назви списків, які належать користувачеві, а tasks – це конкретні завдання з назвою, описом, терміном, статусом. Зв’язки між таблицями реалізовано через зовнішні ключі (user_id, task_list_id), що відповідає принципу нормалізації даних та дозволяє ефективно масштабувати систему.

Загалом, аналіз предметної області та вимог до системи дозволив сформувати чітке бачення майбутнього вебзастосунку SmartList. У розділі були обґрунтовані принципи архітектури, визначені основні функціональні та нефункціональні вимоги, проєктована структура бази даних. Усе це створює надійне підґрунтя для подальшої розробки, реалізації та тестування системи.

Для досягнення цих цілей було обрано стек: PHP (Laravel) – як серверна технологія, MySQL – для зберігання даних, HTML/CSS/JavaScript (Bootstrap) – для реалізації інтерфейсу [9].

Базова логіка реалізується через шаблон MVC (Model-View-Controller), який дозволяє чітко розділити логіку, представлення та зберігання даних. Контролери Laravel обробляють HTTP-запити, моделі Eloquent відповідають за взаємодію з базою даних, а Blade-шаблони забезпечують генерацію HTML.

Для реалізації логіки зберігання даних використовується реляційна база даних MySQL [10]. Її структура була сформована з урахуванням мінімальної надмірності та забезпечення зв’язків типу «один до багатьох»:

- таблиця users: зберігає облікові записи;
- таблиця task_lists: списки, прив’язані до користувача;
- таблиця tasks: завдання, прив’язані до списку.

Таке проєктування дозволяє кожному користувачу мати кілька незалежних списків, а в кожному списку – довільну кількість завдань. Всі таблиці містять поля created_at і updated_at, що полегшує облік змін (рис. 2.1).

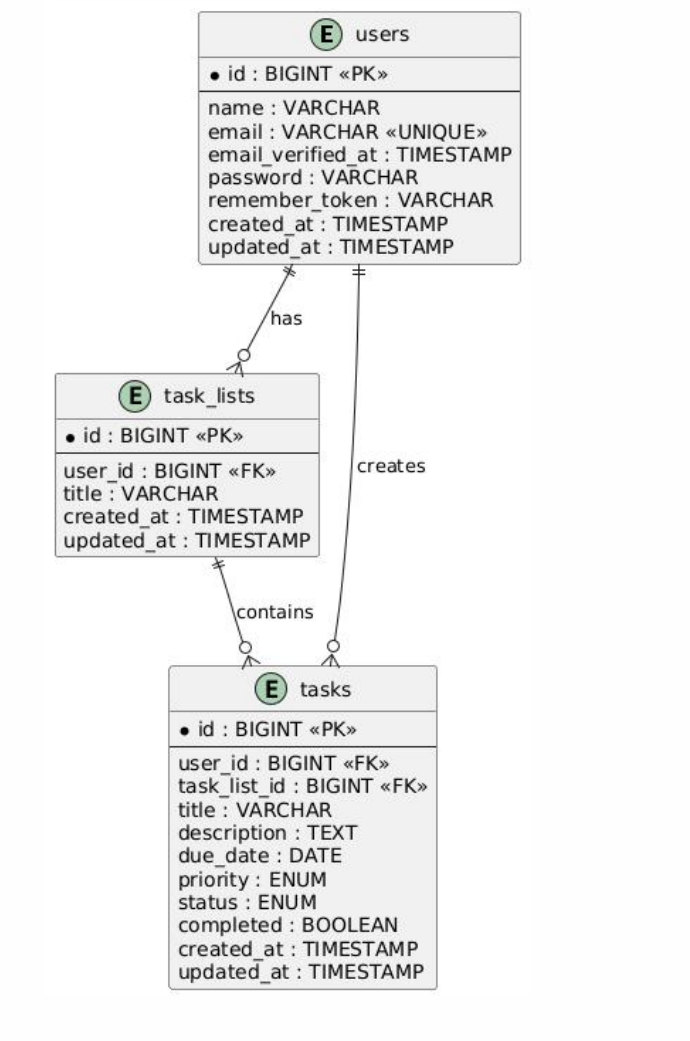

Рисунок 2.1 – Схема бази данних в PlantUML web server

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

Ефективне вирішення завдань у процесі створення вебзастосунку «SmartList» передбачає використання сучасних технологічних засобів, мов програмування, фреймворків, систем управління базами даних, архітектурних шаблонів і алгоритмів. Вибір кожного компонента обґрунтований з точки зору продуктивності, зручності реалізації, підтримки безпеки та масштабованості. У цьому розділі подано огляд обраних технологій, принципів побудови програмного забезпечення та логіки взаємодії його компонентів.

Першим кроком стало визначення мови програмування та платформи для реалізації серверної частини. Для цього обрано мову PHP, яка широко використовується для веброзробки завдяки відкритості, активній спільноті та широкому набору бібліотек. PHP забезпечує високу сумісність із більшістю вебсерверів (Apache, Nginx), має розвинену інфраструктуру для роботи з базами даних, підтримує об'єктно-орієнтоване програмування та інтегрується з сучасними фреймворками [3].

У якості фреймворку обрано Laravel – один із найпотужніших і найбільш використовуваних PHP-фреймворків, що базується на архітектурному шаблоні MVC (Model-View-Controller) [1]. Laravel надає вбудовану систему маршрутизації, ORM Eloquent для роботи з базами даних, систему автентифікації, валідації форм, підтримку шаблонізатора Blade, API-контролерів та багато інших інструментів. Використання Laravel дозволяє значно скоротити час розробки, покращити якість коду та забезпечити його модульність.

Клієнтська частина вебзастосунку реалізується з використанням HTML5, CSS3, JavaScript та бібліотеки Bootstrap. HTML використовується для створення структури вебсторінок, CSS – для оформлення та адаптивності, JavaScript – для інтерактивності (наприклад, обробки подій без перезавантаження сторінки), а Bootstrap дозволяє швидко створювати адаптивні макети, що коректно відображаються на різних пристроях. Для деяких динамічних компонентів використовується Vue.js (опційно), який може бути легко інтегрований у Blade-шаблони [7].

Збереження даних реалізовано з використанням MySQL – надійної реляційної системи управління базами даних із відкритим кодом. MySQL забезпечує високу швидкість обробки запитів, підтримує індексацію, транзакції, зв’язки між таблицями, забезпечує резервне копіювання та масштабування. Структура бази даних передбачає три основні таблиці: users, task_lists та tasks, між якими встановлено зв’язки один-до-багатьох. Така структура забезпечує гнучкість, простоту запитів і розширюваність у майбутньому [4].

Наступним важливим етапом є вибір алгоритмів взаємодії з користувачем. Кожна взаємодія базується на обробці HTTP-запитів. Наприклад, при створенні нового завдання користувач заповнює форму, яка передає дані методом POST до контролера. Контролер перевіряє валідність даних, звертається до моделі, яка зберігає інформацію в базі даних, після чого повертається відповідь у вигляді HTML або JSON. Таким чином, реалізується шаблон «запит – обробка – відповідь», що є стандартом у RESTful архітектурі.

Для забезпечення безпеки вебзастосунку застосовуються такі методи та інструменти CSRF-захист – всі форми мають токени перевірки достовірності; Хешування паролів – за допомогою алгоритму bcrypt, Авторизація через middleware – перевірка прав доступу до маршрутів, валідація форм – як на боці клієнта, так і на боці сервера, фільтрація введених даних – для запобігання SQL-ін'єкціям та XSS-атакам.

Для збереження стабільності застосунку в майбутньому реалізується резервне копіювання бази даних, автоматичне через Cron або планувальник Laravel (Scheduler). Це дозволяє щоденно створювати резервні SQL-дампи, які можуть бути відновлені в разі втрати даних [5].

У процесі розробки застосовується система контролю версій Git, що дозволяє вести історію змін, створювати гілки для експериментів, інтегрувати код командою. Репозиторій розміщено на GitHub, де також ведеться документація (README.md), issue-трекінг та версіонування релізів [11].

Тестування програмного забезпечення виконується кількома методами. Для юніт-тестування використовується PHPUnit, яке дозволяє перевірити роботу окремих моделей, функцій та класів. Наприклад, перевірка того, чи зберігається нове завдання в базі даних після виклику методу Task::create(). Для інтеграційного тестування застосовується Laravel Dusk, що дозволяє симулювати дії реального користувача в браузері, перевірити навігацію, форми, повідомлення тощо. Крім того, REST API перевіряється вручну за допомогою Postman.

На рівні бізнес-логіки застосовуються прості алгоритми фільтрації та сортування. Наприклад, завдання можна фільтрувати за списком, статусом, пріоритетом або терміном. Усі ці фільтри реалізовано через SQL-запити з параметрами (WHERE, ORDER BY). У подальшому можливе розширення до повнотекстового пошуку або індексування задач.

Загалом, обрані засоби та методи реалізації дозволяють ефективно реалізувати поставлене завдання створення зручного, масштабованого, безпечного та продуктивного вебпомічника. Laravel забезпечує потужну серверну основу, MySQL – надійне сховище даних, а фронтенд на HTML/JS/Bootstrap – сучасний інтерфейс, зручний для користувача [6]. Усі компоненти є взаємодоповнюваними та утворюють єдину архітектурну систему.

РОЗДІЛ 3
РОЗРОБКА ВЕБПОМІЧНИКА «SMARTLIST» З ВИКОРИСТАННЯМ PHP, LARAVEL ТА MYSQL

3.1 Практична реалізація об’єкта проектування

Розробка прикладного веб-застосунку здійснювалася з використанням сучасного PHP-фреймворку Laravel, що побудований за архітектурною моделлю MVC – «Model-View-Controller» (Модель-Представлення-Контролер). Це дозволяє чітко відокремити логіку роботи програми, користувацький інтерфейс та доступ до даних. У рамках практичної частини було реалізовано інформаційну систему для управління завданнями, що дозволяє зареєстрованим користувачам створювати списки завдань, додавати до них підзавдання, редагувати їх або видаляти у зручному веб-інтерфейсі (рис. 3.1).

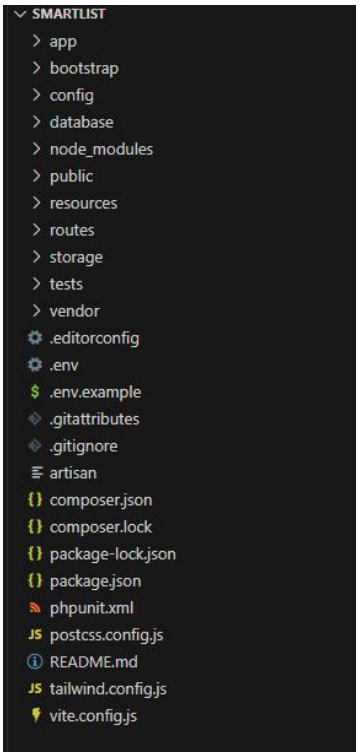

Рисунок 3.1 – Структура проєкту

На етапі проєктування було визначено, що кожен користувач системи повинен мати можливість керувати лише своїми власними списками завдань.

Усі сутності додатку пов’язані між собою через відповідні таблиці бази даних, з якими працюють моделі Laravel через ORM – Eloquent. У результаті було створено три головні сутності: користувачі (User), списки завдань (TaskList) та завдання (Task).

Файл маршрутів web.php визначає взаємозв’язок між URL-адресами веб-сторінок і відповідними методами контролерів [9]. Завдяки методам Route::resource можна автоматично створити RESTful-маршрути для обробки всіх базових операцій показано на лістингу 3.1.

Лістинг 3.1 – Метод Route::resource

```
Route::resource('tasks', TaskController::class);
Route::resource('task-lists', TaskListController::class);
```

кінець лістингу 3.1

Ці маршрути автоматично підключають методи (ліст. 3.2):

– index (перегляд списку);
– create (відображення форми створення);
– store (обробка форми створення);
– edit, update, destroy (редагування, оновлення та видалення).

Лістинг 3.2 – Побудова навігаційної структури

```
public function store(Request $request)
{

.     $validated = $request->validate([
          'title' => 'required|string|max:255',
          'description' => 'nullable|string',
          'task_list_id' => 'required|exists:task_lists,id',
      ]);
      Task::create($validated);
      return redirect()->route('tasks.index')->with('success',
'Завдання
створено');
```

кінець лістингу 3.2

Щоб забезпечити збереження цілісності даних, усі перевірки здійснюються за допомогою вбудованого механізму Laravel validate(). У випадку невідповідності вимогам (наприклад, відсутнє обов'язкове поле або неправильне посилання), користувач буде автоматично повернутий на форму з повідомленнями про помилки (рис. 3.2).

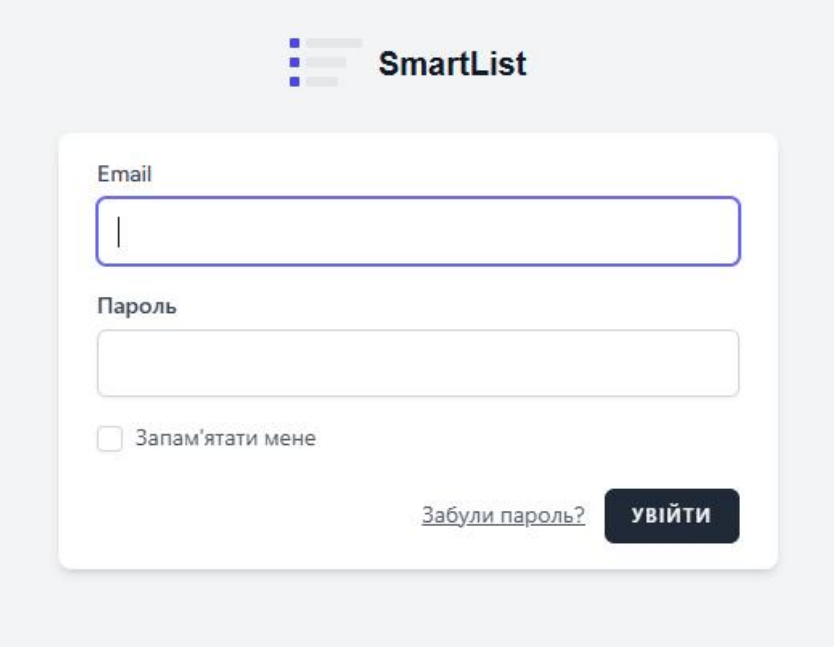

Рисунок 3.2 – Автозація SmartList

Додатково до базових маршрутів Laravel дозволяє створювати спеціальні маршрути з авторизацією (ліст. 3.3). У файлі RegisteredUserController.php обробляється реєстрація користувача.

Лістинг 3.3 – Автозація додаткку

```
public function store(Request $request)
{
    $request->validate([
        'name' => 'required|string|max:255',
        'email' => 'required|string|email|max:255|unique:users',
        'password' => 'required|string|confirmed|min:8',
    ]);
    $user = User::create([
        'name' => $request->name,
        'email' => $request->email,
        'password' => Hash::make($request->password),
```

```
    ]);
    Auth::login($user);
    return redirect(RouteServiceProvider::HOME);
}
```

кінець лістингу 3.3

Цей код створює нового користувача, шифрує його пароль та автоматично виконує вхід до системи.

3.2 Тестування та налагодження інформаційно-комп'ютерної системи

Тестування функціональності є заключним і надзвичайно важливим етапом у процесі розробки вебзастосунку. Воно дозволяє переконатися, що реалізований функціонал відповідає поставленим вимогам, система працює стабільно, а користувач отримує очікуваний результат під час взаємодії з інтерфейсом [10]. У проєкті «SmartList» було застосовано як ручне функціональне тестування, так і елементи автоматизованого тестування на рівні модулів і компонентів (рис. 3.3).

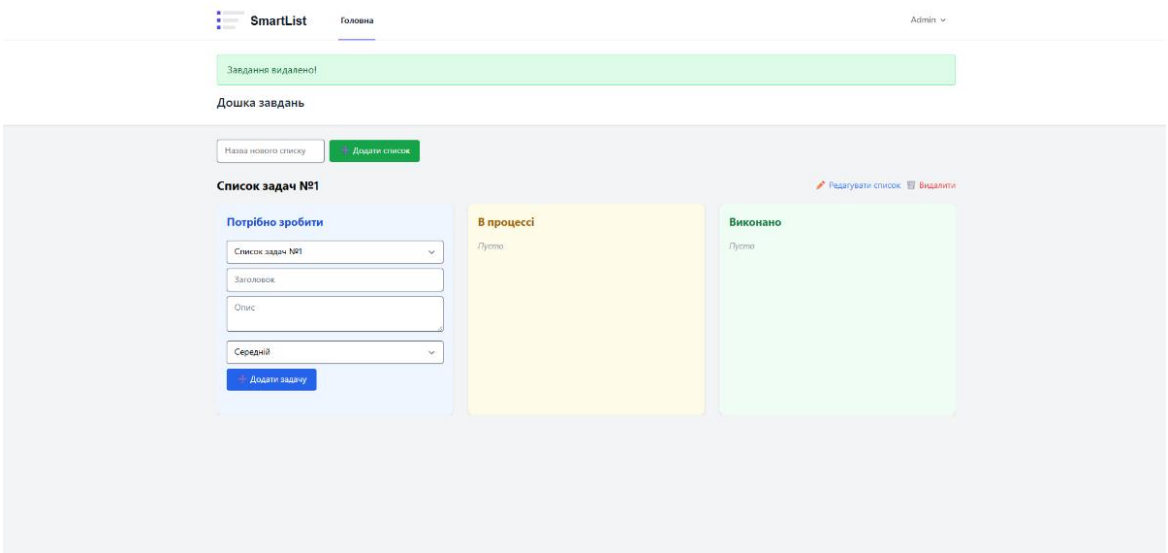

Рисунок 3.3 – Вигляд Веб-додатку в браузері Google Chrome

Перший етап передбачав проведення ручного тестування через послідовну перевірку основних сценаріїв роботи користувача. Зокрема,

перевірялася реєстрація та вхід до системи, створення та редагування списків завдань, додавання, зміна та видалення завдань, відмітка про виконання, перегляд завдань у списку, а також реакція інтерфейсу на помилки введення. Було протестовано коректність валідації форм, зокрема обов'язковість полів назви та дати, недопущення порожніх або некоректних значень, збереження введених даних після виведення помилки, а також правильність виводу повідомлень про успішні чи неуспішні дії. Перевірку здійснено як у десктопному, так і мобільному браузерах, що дозволило переконатися в адаптивності інтерфейсу та стабільності функціонування на різних розширеннях екрана.

Другий етап включав елементи автоматизованого тестування з використанням вбудованого механізму Laravel, який ґрунтується на фреймворку PHPUnit. Laravel надає зручний інтерфейс для створення так званих feature-тестів, які дозволяють перевірити роботу конкретного функціоналу у вигляді HTTP-запитів, і unit-тестів – для перевірки окремих класів, методів або моделей. Такі тести розташовуються в директорії tests і виконуються командою php artisan test. У проєкті «SmartList» було створено базовий тест доступності головної сторінки, а також тест авторизованого доступу до маршруту зі списками завдань. Це дозволило переконатися, що неавторизовані користувачі автоматично перенаправляються на сторінку входу, а авторизовані – мають доступ до функціоналу згідно з роллю.

Наприклад, типовий feature-тест для перевірки відображення списків завдань для авторизованого користувача може виглядати так: створюється користувач за допомогою фабрики, виконується HTTP-запит до маршруту /task_lists, після чого перевіряється, що відповідь має статус 200 і містить назву хоча б одного списку. Подібний підхід дозволяє швидко виявляти помилки при зміні коду, особливо якщо проєкт масштабуватиметься.

Окрім цього, тестуванню підлягала логіка взаємодії моделей. Було перевірено, що зв'язки між користувачами, списками та завданнями коректно працюють відповідно до визначених відношень Eloquent. Зокрема, для кожного

користувача має бути доступ до списків через метод taskLists(), а у списку – до завдань через метод tasks(). При створенні нового завдання через контролер воно повинно автоматично зберігатися у відповідному списку користувача та відображатися у подальших запитах.

Загалом, результати тестування показали, що система працює стабільно у межах заявленого функціоналу, не містить критичних помилок, підтримує правильну авторизацію, валідацію форм, взаємодію між моделями та коректно реагує на введення даних. Під час перевірки не було виявлено збоїв у роботі маршрутизатора, контролерів чи шаблонів. Отже, проєкт «SmartList» може вважатися готовим до подальшого розгортання та використання в реальному середовищі. У разі масштабування або впровадження додаткових функцій автоматизовані тести можна буде розширити для покриття нових сценаріїв, що значно спростить підтримку та розвиток системи.

3.3 Інтерфейс користувача вебпомічника

Інтерфейс користувача є одним із ключових елементів будь-якого вебзастосунку, особливо у випадку персональних помічників, де зручність взаємодії напряму впливає на ефективність використання. У вебпомічнику «SmartList» інтерфейс було реалізовано з урахуванням сучасних вимог до зручності, логіки взаємодії, адаптивності та доступності для різних типів користувачів. Основне завдання інтерфейсу – надати користувачу швидкий і зрозумілий доступ до основних функцій: перегляду, створення, редагування та завершення завдань.

Весь інтерфейс базується на шаблонізаторі Blade, який входить до складу фреймворку Laravel. Blade дозволяє компонувати HTML-код із вбудованими шаблонними інструкціями, забезпечуючи гнучкість та повторне використання елементів [11]. Наприклад, усі сторінки наслідують основний макет, який містить спільні частини, такі як меню навігації, шапка сайту, повідомлення про помилки або успішні дії, а також зону для виведення контенту кожної

конкретної сторінки. Це дозволяє уникнути дублювання коду, централізовано змінювати дизайн та швидко додавати нові сторінки в рамках однієї структури.

Структура інтерфейсу була спроєктована так, щоб основна увага користувача зосереджувалася саме на його завданнях і списках справ. Головна сторінка після авторизації містить перелік доступних списків, кожен з яких представлено у вигляді окремої картки з кнопкою переходу до перегляду завдань. У межах конкретного списку користувач бачить всі пов’язані з ним завдання, відмічені статусом (виконано, у процесі, заплановано), датою виконання та рівнем пріоритету. Всі дії з завданнями (редагування, видалення, відмітка про виконання) винесені у вигляді зручних іконок, що розміщуються праворуч від кожного елементу [13].

Інтерфейс адаптивний – тобто автоматично підлаштовується під ширину екрана, що забезпечує комфортне використання як на комп’ютерах, так і на мобільних пристроях. Для реалізації адаптивності було використано фреймворк Bootstrap 5, який містить готові компоненти сіток, кнопок, форм, випадаючих списків, панелей та інших елементів. Завдяки цьому зменшилась кількість коду, необхідного для розмітки, а також покращилась сумісність інтерфейсу з різними браузерами.

Навігаційне меню реалізовано окремим шаблоном, який підключається до всіх сторінок. Меню автоматично змінюється в залежності від авторизації користувача: якщо користувач увійшов у систему, йому доступна навігація до списків, профілю, а також кнопка виходу. У випадку, коли користувач не авторизований, відображаються кнопки реєстрації та входу. Такий підхід забезпечує безпечний контроль доступу до функціоналу та спрощує логіку навігації.

Форми створення й редагування завдань побудовані так, щоб мінімізувати час на введення даних. Користувач має змогу швидко ввести назву завдання, опис, обрати пріоритет і дедлайн – без перенаправлень на інші сторінки. Всі форми містять обробку помилок і виведення повідомлень у випадку невірного заповнення (наприклад, якщо не вказано назву завдання). Це

реалізовано за допомогою серверної валідації у Laravel, яка автоматично пов’язана з шаблонними повідомленнями у Blade [5].

Крім форм, у проєкті використовуються компоненти повторного використання, наприклад шаблони полів введення (text-input.blade.php). Вони забезпечують уніфікований вигляд усіх елементів форм, а також дозволяють централізовано змінювати логіку відображення або стилізацію. Це спрощує масштабування інтерфейсу у разі додавання нових форм або розділів.

Особливу увагу приділено зворотному зв’язку. Після будь-якої дії користувач бачить відповідне повідомлення – наприклад, «Завдання створено успішно» або «Поле Назва є обов’язковим». Ці повідомлення реалізовано через механізм flash-сесій Laravel і виводяться безпосередньо у шаблонах. Це дозволяє підвищити довіру до системи та зменшити кількість помилок при роботі [2].

У результаті реалізовано простий, інтуїтивно зрозумілий, гнучкий інтерфейс, який дає змогу користувачеві зручно керувати власними завданнями і списками у будь-якому браузері. Його архітектура побудована так, щоб забезпечити максимальну масштабованість – додавання нових сторінок або компонентів не потребує переписування існуючого коду, а лише включення їх у вже наявну шаблонну систему.

3.4 Підключення до бази даних MySQL та міграції

У процесі реалізації вебпомічника «SmartList» важливу роль відіграє робота з базою даних, оскільки саме вона забезпечує зберігання, структурування та цілісність інформації про користувачів, списки завдань і самі завдання. Як сервер бази даних обрано MySQL, що є однією з найпоширеніших, безкоштовних та ефективних реляційних систем керування базами даних. Її використання у поєднанні з Laravel дозволяє реалізувати повний життєвий цикл даних – від створення таблиць до їх обслуговування через ORM [13].

Підключення до бази даних реалізовано через конфігураційний файл .env, де вказано основні параметри: хост (звичайно 127.0.0.1), порт (3306), ім’я бази даних, ім’я користувача та пароль. Ці налаштування автоматично використовуються усіма компонентами Laravel, що працюють із базою – зокрема, при запуску міграцій, використанні моделей або виконанні запитів.

У структурі проєкту передбачено три основні сутності, які безпосередньо реалізують предметну область: користувачі, списки завдань і самі завдання. У файлі smartlist.sql представлено повну структуру таблиць і приклади даних. Таблиця users містить стандартні для Laravel поля: ім’я, email, зашифрований пароль, токен запам’ятовування та мітки часу. Усі email-и унікальні, що забезпечується відповідним обмеженням. Таблиця task_lists прив’язана до користувача зовнішнім ключем user_id, а її структура дозволяє зберігати назву кожного списку. Видалення користувача автоматично призводить до видалення всіх його списків, завдяки використанню обмеження ON DELETE CASCADE.

Найбільш насиченою за структурою є таблиця tasks, яка зберігає самі завдання. Вона містить поля title, description, due_date, priority, status, completed, а також зовнішні ключі до користувача і списку. Варто відзначити використання ENUM-полів для статусу (todo, in_progress, done) і пріоритету (low, medium, high), що дозволяє чітко контролювати допустимі значення без необхідності створення окремих таблиць. Крім того, використано ON DELETE SET NULL для зовнішніх ключів, що забезпечує збереження завдання навіть у разі видалення пов’язаного списку або користувача.

Відповідно до філософії Laravel, структура таблиць не прописується вручну в SQL, а формується за допомогою системи міграцій. У проєкті передбачено три основні міграції: create_users_table, create_task_lists_table, create_tasks_table. Кожна з них є PHP-класом, у якому визначено логіку створення таблиці через метод Schema::create(). Такий підхід дозволяє застосовувати контроль версій до структури бази, швидко розгортати її в новому середовищі, а також забезпечувати зворотну сумісність через функцію відкату міграцій (migrate:rollback) [13].

Особливістю роботи з базою даних є також наповнення її початковими даними. У наданому дампі присутні демонстраційні записи: користувач із email admin@admin.com, приклад списку завдань з назвою «Список задач №1», а також кілька прикладних завдань. Це дозволяє вже після першого запуску одразу оцінити функціонал системи без необхідності вручну створювати записи. Подібне сидування є корисним при тестуванні та демонстрації робот [9].

Таким чином, у «SmartList» реалізовано повноцінну структуру бази даних із чіткими зв’язками між таблицями, підтримкою обмежень цілісності, логікою каскадного видалення, чітким розділенням статусів і пріоритетів. Завдяки системі міграцій Laravel цей підхід дозволяє не лише спростити розробку, а й забезпечити надійність і передбачуваність поведінки бази у будь-якому середовищі розгортання. Уся логіка взаємодії з базою повністю узгоджується з моделями Eloquent ORM, що забезпечує зручну роботу з даними на рівні об’єктів без написання SQL-запитів вручну.

3.5 Налаштування середовища та збірка проєкту

Розгортання вебпомічника «SmartList» потребує попереднього налаштування програмного середовища. Laravel як фреймворк має чітко структуровану систему залежностей і конфігурації, що дозволяє швидко підготувати систему до розробки або запуску в продакшн. Передусім необхідно створити файл .env на основі шаблону .env.example, де вказуються всі основні параметри: ім’я застосунку, базовий URL, дані доступу до бази даних, ключ шифрування тощо. Фрагмент налаштувань підключення до MySQL у .env.example зображено на лістингу 3.4.

Лістинг 3.4 Налаштування підключення MySQL

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=smartlist
DB_USERNAME=root
```

```
DB_PASSWORD=
```

кінець лістингу 3.4

Ці значення забезпечують взаємодію Laravel із локальною інсталяцією MySQL. Після заповнення конфігурації виконується команда php artisan key:generate, яка створює унікальний ключ шифрування для забезпечення безпеки сесій та токенів.

На наступному етапі встановлюються всі необхідні PHP-залежності. Laravel використовує менеджер пакетів Composer вказано в додатку а, який читає файл composer.json [3]. У цьому файлі визначено, які бібліотеки будуть використані в проєкті. Зокрема, у випадку SmartList, тут зазначено (ліст. 3.5).

Лістинг 3.5 Пакети Composer

```
"require": {
  "php": "^8.2",
  "laravel/framework": "^11.0",
  "laravel/breeze": "^2.0",
  "laravel/sanctum": "^3.0"
}
```

кінець лістингу 3.5

Breeze додає базову реалізацію автентифікації, а Sanctum забезпечує підтримку токенів для API. Після виконання composer install усі залежності автоматично завантажуються до директорії vendor/.

Для фронтенд-збірки використовується Vite – сучасна альтернатива Webpack, що має вищу продуктивність за лістингом 3.6. Вміст файлу vite.config.js визначає вихідні точки CSS і JS.

Лістинг 3.6 – Доступ до збірки JS та CSS

```
export default defineConfig({
    plugins: [laravel({
        input: ['resources/css/app.css', 'resources/js/app.js'],
        refresh: true,
    })],
});
```

кінець лістингу 3.6

Це означає, що при запуску Vite автоматично оброблятиме ці файли, підключаючи зміни у реальному часі. Щоб зібрати фронтенд, спочатку потрібно виконати команду npm install, яка встановить залежності, зазначені в package.json. вказано на лістингу 3.7.

Лістинг 3.7 – Обробка файлів

```
"devDependencies": {
  "tailwindcss": "^3.3.2",
  "vite": "^5.0.0",
  "autoprefixer": "^10.4.14",
  "axios": "^1.5.0"
}
```

кінець лістингу 3.7

Після інсталяції залежностей за допомогою npm run dev запускається середовище розробки, що оновлює стилі та скрипти при кожному збереженні файлу. Для фінальної збірки у продакшн середовищі виконується npm run build, яка оптимізує ресурси – мінімізує JavaScript, очищує CSS та готує файли для деплою.

Завершальним етапом є запуск Laravel Artisan-команд для створення таблиць у базі даних.

Ця команда зчитує всі визначені міграції у каталозі database/migrations і створює відповідні таблиці. Якщо у проєкті передбачено сидування (наприклад, початковий користувач або демонстраційні дані), також виконується: php artisan db:seed.

У результаті всі дані підготовлені для роботи, і вебзастосунок може бути запущений у браузері за допомогою вбудованого сервера: php artisan serve

Таким чином, завдяки продуманій екосистемі Laravel, встановлення й збірка проєкту «SmartList» не потребує складних налаштувань. Всі дії логічно впорядковані, автоматизовані та легко повторювані в інших середовищах

(наприклад, у Docker або на сервері), що дозволяє зосередитися на функціональності замість інфраструктурних складностей.

ВИСНОВКИ

У межах виконання кваліфікаційної роботи було успішно реалізовано повний цикл створення вебпомічника «SmartList» для управління особистими та робочими завданнями. Розробка системи ґрунтувалася на сучасних підходах до вебпрограмування та відповідала актуальним потребам користувачів у сфері цифрового планування.

Перш за все було проведено ґрунтовний огляд та аналіз веборієнтованих систем управління завданнями у сфері управління завданнями, таких як Trello, Asana, Todoist, Microsoft To Do та інші. Основна увага приділялася дослідженню функціональних можливостей, архітектурних підходів, інтерфейсних рішень, моделі взаємодії з користувачем, а також наявним обмеженням у контексті персоналізації, масштабування та безпеки. Аналіз дозволив ідентифікувати ключові функції, які користувачі очікують від сучасного вебзастосунку для керування завданнями: зручний інтерфейс, гнучку систему фільтрації, підтримку категорій та пріоритетів, можливість спільного доступу до завдань, адаптивність до різних пристроїв, синхронізацію даних і стабільну роботу без затримок. Крім того, у ході дослідження було виявлено низку недоліків у деяких популярних рішеннях, зокрема: перевантаженість функціоналу, що ускладнює використання новачками; обмежена можливість кастомізації; залежність від підключення до зовнішніх API; недостатній рівень локалізації інтерфейсу українською мовою; а також потреба у підписці для розблокування базових можливостей. Всі ці аспекти стали підґрунтям для формулювання вимог до власного програмного продукту. Таким чином, аналіз конкурентних систем дав змогу не лише виявити найкращі практики розробки та впровадження таких сервісів, а й сформувати чітке бачення цільової аудиторії, її очікувань та реальних потреб. Це стало визначальним кроком для створення вебпомічника «SmartList», який поєднує простоту використання, інтуїтивно зрозумілий інтерфейс і функціональну гнучкість.

Обґрунтовано вибір технологічного стеку, до складу якого увійшли сучасні та перевірені інструменти веброзробки. Основою серверної частини став мова програмування PHP, що відзначається широкою популярністю, активною спільнотою розробників та підтримкою великої кількості фреймворків. Зокрема, було використано фреймворк Laravel, який надає потужний набір засобів для створення безпечних, продуктивних та масштабованих вебзастосунків. Завдяки вбудованим можливостям, таким як маршрутизація, ORM Eloquent, система шаблонів Blade, підтримка міграцій баз даних та засобів тестування, розробка проєкту значно спростилася й пришвидшилася. Для зберігання й опрацювання даних була обрана реляційна система управління базами даних MySQL. Ця СУБД поєднує високу швидкість роботи, надійність і простоту адміністрування, що робить її ідеальним рішенням для невеликих та середніх за масштабом вебпроєктів. Завдяки інтеграції з Laravel, MySQL забезпечує ефективну роботу з базою даних через ORM-механізми, що дозволяє легко маніпулювати даними без написання складних SQL-запитів.

У процесі проєктування інформаційної системи особлива увага була приділена побудові логічної структури бази даних, яка є основою для зберігання, обробки та ефективного керування інформацією в межах вебпомічника «SmartList». Створена модель бази даних охоплює всі ключові сутності, необхідні для повноцінного функціонування системи: користувачі, категорії завдань, завдання, статуси виконання, пріоритети, терміни виконання, а також додаткові атрибути, що забезпечують гнучкість і масштабованість роботи додатку. Зокрема, сутність «користувач» містить усю необхідну інформацію для автентифікації, авторизації та персоналізації досвіду взаємодії з системою. Кожен користувач має змогу створювати власні категорії, які дозволяють групувати завдання за напрямками діяльності або типами задач (наприклад: робота, навчання, особисте тощо). Сутність «завдання» є центральною у структурі, містить атрибути назви, опису, дедлайну, статусу

виконання, пріоритету та пов’язана з конкретним користувачем і категорією, до якої належить.

У межах реалізації вебпомічника «SmartList» було розроблено широкий спектр функціональних можливостей, що охоплюють основні потреби користувачів у процесі керування особистими й робочими завданнями. Ключовим завданням стало забезпечення зручного та інтуїтивно зрозумілого інтерфейсу для взаємодії з елементами системи, що дозволяє мінімізувати час навчання нового користувача та зробити роботу із застосунком максимально ефективною. Функціонал вебзастосунку охоплює повний життєвий цикл завдань: створення, перегляд, редагування, видалення та фільтрація. Користувач має змогу швидко створювати нові задачі, визначаючи для них назву, опис, категорію, статус виконання, пріоритет і дедлайн. Усі ці параметри дозволяють гнучко керувати поточними справами та довгостроковими цілями.

Функція редагування забезпечує можливість внесення змін у вже створені завдання, що особливо корисно у випадках зміни обставин або уточнення деталей. Для зручності використання реалізовано фільтрацію та сортування завдань за різними критеріями: категорією, датою виконання, статусом або пріоритетом. Це дозволяє користувачеві легко орієнтуватися в потоці завдань та фокусуватися на найбільш важливих. Передбачена також можливість перегляду історії виконаних завдань або списків майбутніх задач, що підтримує принципи ефективного тайм-менеджменту.

Особливу увагу під час реалізації вебпомічника «SmartList» було приділено створенню адаптивного інтерфейсу, який забезпечує комфортну та ефективну взаємодію користувача з системою незалежно від типу пристрою – стаціонарного комп’ютера, ноутбука, планшета чи смартфона. З огляду на сучасні тенденції, згідно з якими дедалі більше користувачів взаємодіють із вебзастосунками саме через мобільні пристрої, адаптивність стала критично важливою характеристикою для досягнення зручності та широкого охоплення аудиторії.

Під час розробки інтерфейсу було використано принципи mobile-first дизайну, згідно з якими спочатку створюється інтерфейс для малих екранів, а потім поступово адаптується до більших. Це дозволяє уникнути перевантаження елементами інтерфейсу на смартфонах і водночас забезпечити повну функціональність на десктопах. У розмітці та стилях активно використовувалися flexbox і grid-сітки, що надали змогу формувати гнучке компонування елементів і забезпечити правильне відображення контенту на різних розширеннях екрана.

Окрему увагу приділено юзабіліті: усі основні функції доступні через мінімальну кількість дій, кнопки та поля введення мають достатні розміри для зручної роботи на сенсорних екранах, використовуються контрастні кольори для кращої візуалізації елементів, а навігація адаптована для вертикальної орієнтації екрана. Крім того, були протестовані різні сценарії взаємодії на популярних мобільних браузерах, таких як Google Chrome, Safari та Firefox, що забезпечило стабільність роботи вебзастосунку в більшості реальних умов користування..

На завершальному етапі було проведено тестування функціональності системи, її продуктивності та зручності використання. Результати тестування підтвердили відповідність реалізованого програмного продукту поставленим вимогам і його готовність до практичного використання.

Таким чином, поставлену мету кваліфікаційної роботи досягнуто. Розроблений вебпомічник «SmartList» може бути використаний як особистими користувачами, так і невеликими командами для ефективного планування, організації та контролю виконання завдань у щоденній діяльності.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Laravel. The PHP Framework For Web Artisans. URL: https://laravel.com/ (дата звернення: 03.04.2025).

2. Власюк І. В. Веб-програмування з використанням PHP і MySQL: навч. посіб. Львів: Видавництво Львівської політехніки. 2021. 212 с.

3. PHP Manual. PHP Documentation. URL: https://www.php.net/manual/en/ (дата звернення: 17.03.2025).

4. MySQL 8.0 Reference Manual. URL: https://dev.mysql.com/doc/ (дата звернення: 10.06.2025).

5. Composer – Dependency Manager for PHP. URL: https://getcomposer.org/ (дата звернення: 28.02.2025).

6. Tailwind CSS Documentation. URL: https://tailwindcss.com/docs (дата звернення: 22.04.2025).

7. Bootstrap5. Getting started. URL: https://getbootstrap.com/docs/5.3/getting-started/introduction/ (дата звернення: 14.05.2025).

8. Вітюк О. С. Архітектура інформаційних систем: навч. посіб. К.: Ліра-К. 2020. 196 с.

9. PHPUnit – Testing framework for PHP. URL: https://phpunit.de/ (дата звернення: 11.06.2025).

10. Доценко С. І. Організація та системи керування базами даних: Навч. посіб. Харків: УкрДУЗТ. 2023. 117 с.

11. Git – Free and open source distributed version control system. URL: https://git-scm.com/ (дата звернення: 07.02.2025).

12. Trello – Organize anything with anyone. URL: https://trello.com/ (дата звернення: 25.05.2025).

13. Notion – The connected workspace. URL: https://www.notion.so/ (дата звернення: 06.06.2025).

14. Google Tasks. URL: https://www.google.com/tasks (дата звернення: 19.03.2025).

15. ClubManager – Управління спортивним клубом. URL: https://xn--clubmanagr-6wi.it/ (дата звернення: 12.04.2025).

16. Stauffer M. Laravel: Up & Running. A Framework for Building Modern PHP Apps. 3rd ed. O'Reilly Media. 2023. 569 с.

17. Welling L., Thomson L. PHP and MySQL Web Development. 6th Edition. Addison-Wesley Professional. 2023. 1152 с.

18. Hoffman A. Web Application Security: Exploitation and Countermeasures for Modern Web Applications. 2nd ed. O'Reilly Media. 2024. 406 с.

19. Rubin J., Chisnell D. Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests. 3rd Edition. Wiley. 2024. 432 с.

20. Rubin K.S., Vodde B. Essential SAFe: An Introduction to the Scaled Agile Framework. Addison-Wesley Professional. 2021. 320 с.

ДОДАТКИ

Додаток А

Фрагмент коду TaskListController – відповідає за логіку та керування Https запитів

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class TaskListController extends Controller
{
   /**
     * Display a listing of the resource.
     */
    public function index()
    {
        //
    }

    /**
     * Show the form for creating a new resource.
     */
    public function create()
    {
        //
    }

    /**
     * Store a newly created resource in storage.
     */
    public function store(Request $request)
    {
        $validated = $request->validate([
            'title' => 'required|string|max:255',
        ]);

        $validated['user_id'] = auth()->id();

        \App\Models\TaskList::create($validated);

                return  redirect()->back()->with('success',  'Список
створено!');
    }

    /**
     * Display the specified resource.
     */
    public function show(string $id)
    {
        //
    }
```

```
    /**
     * Show the form for editing the specified resource.
     */
    public function edit(string $id)
    {
        //
    }

    /**
     * Update the specified resource in storage.
     */
     public function update(Request $request, \App\Models\TaskList
$taskList)
    {
        $request->validate([
            'title' => 'required|string|max:255',
        ]);

        $taskList->update(['title' => $request->title]);

            return  redirect()->route('dashboard')->with('success',
'Список оновлено!');
    }

    /**
     * Remove the specified resource from storage.
     */
    public function destroy(\App\Models\TaskList $taskList)
    {

        $taskList->delete();

            return  redirect()->route('dashboard')->with('success',
'Список видалено!');
    }
}
```