Міністерство освіти і науки України

Луцький національний технічний університет
(повне найменування закладу вищої освіти)

Факультет комп'ютерних та інформаційних технологій
(повне найменування факультету)

Кафедра комп'ютерної інженерії та кібербезпеки
(повне найменування кафедри)

КВАЛІФІКАЦІЙНА РОБОТА ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

ПІДСИСТЕМА ЗБЕРІГАННЯ МУЛЬТИМЕДІЙНОГО КОНТЕНТУ

SUBSYSTEM FOR MULTIMEDIA CONTENT STORAGE

спеціальність 123 Комп'ютерна інженерія
(шифр і назва спеціальності)

освітня програма Комп'ютерна інженерія
(назва освітньої програми)

Виконав: здобувач вищої освіти
групи КІ-41
Євсюк Володимир Миколайович

(підпис)

Керівник:
к.т.н., доцент
Христинець Наталія Анатоліївна

(підпис)

Кваліфікаційну роботу
допущено до захисту
« _____ » червня 2023 р.
Гарант освітньої програми:
к.т.н., доцент
Лавренчук Світлана Василівна

(підпис)

Луцьк – 2023 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *комп'ютерної інженерії та кібербезпеки*
Ступінь вищої освіти: *бакалавр*
Галузь знань: *12 Інформаційні технології*
Спеціальність: *123 Комп'ютерна інженерія*
Освітня програма: *«Комп'ютерна інженерія»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ проф. Н.Чернящук
«_____» _____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Євсюку Володимиру Миколайовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *Підсистема зберігання мультимедійного контенту*

Керівник роботи *к.т.н., доцент Христинець Наталія Анатоліївна*

затверджені наказом закладу вищої освіти від «28» грудня 2022 року № 982/01-02
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи 01.06.2023р.

3. Вихідні дані до роботи *методичні та літературні джерела з програмування, наукові статті в області web-розробки, публікації про архітектурні концепції та шаблони програмування, інтернет-ресурси з різних джерел на тему мультимедійних систем, документація з мови програмування PHP та фреймворки Laravel.*

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):
Вступ
Аналіз предметної області з питань розробки мультимедійних систем
Огляд інструментів та технологій для реалізації підсистеми
Порівняння найпопулярніших платформ зберігання мультимедійного контенту
Розробка підсистеми зберігання мультимедійного контенту
Висновки

5. Перелік графічного (ілюстративного) матеріалу:

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Христинець Н.А.*		
Засоби та інструменти фреймворку Laravel	*Христинець Н.А.*		
Розробка та реалізація підсистеми збереження мультимедійного контенту	*Христинець Н.А.*		
Висновки	*Христинець Н.А.*		

7. Дата видачі завдання 01.11.2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	*Обгрунтування теми*	До 15.11.2022 р.	Виконано
2.	*Огляд літературних джерел на тему мультимедійних платформ*	До 15.12.2022 р.	Виконано
3.	*Аналіз існуючої проблеми та актуальності теми про збереження мультимедійних даних*	До 02.02.2023 р.	Виконано
4.	*Дослідження інструментів розробки додатку*	До 02.03.2023 р.	Виконано
5.	*Розгортання та налаштування проекту для розробки*	До 02.04.2023 р.	Виконано
6.	*Розробка підсистеми збереження мультимедійного контенту*	До 15.04.2023 р.	Виконано
7.	*Оформлення матеріалів роботи*	До 15.05.2023 р.	Виконано
8.	*Оформлення ілюстративного матеріалу*	До 15.05.2023 р.	Виконано
9.	*Нормоконтроль*	До 25.05.2023 р.	Виконано
10.	*Інструментальна перевірка на академічний плагіат*	До 01.06.2023 р	Виконано
10.	*Представлення кваліфікаційної роботи бакалавра до захисту*	До 07.06.2023 р.	Виконано

Здобувач вищої освіти _____ (Євсюк В.М.)
(підпис) (прізвище, ініціали)

Керівник кваліфікаційної роботи _____ (Христинець Н.А.)
(підпис) (прізвище, ініціали)

АНОТАЦІЯ

Євсюк В. М. Підсистема зберігання мультимедійного контенту. Рукопис.

Кваліфікаційна робота бакалавра ОП «Комп'ютерна інженерія» спеціальності 123 Комп'ютерна інженерія. Луцький національний технічний університет. Луцьк, 2023.

Кваліфікаційна робота обсягом 85 сторінок містить 30 ілюстрацій, 1 таблицю, 7 додатків та 13 джерел за переліком посилань.

У першому розділі проведено аналіз предметної області, а саме: аналіз існуючої проблеми, складено порівняльну характеристику інструментів для розробки підсистеми та здійснено обґрунтований вибір технологій. Також проаналізовано найпопулярніші платформи для зберігання мультимедійного контенту (YouTube, Vimeo, Google Photos, Dropbox).

У другому розділі розглянуто загальні методики розробки додатків на основі використання фреймворку Laravel. Розглянуто архітектурний шаблон MVC.

Третій розділ присвячено опису розробки та реалізації підсистеми зберігання мультимедійного контенту. Спроектовано архітектуру бази даних системи, наведено зображення програмних інтерфейсів платформи та додано приклади написання коду для функціоналу збереження мультимедійного контенту.

Об'єкт дослідження – технології та інструменти для розробки підсистеми зберігання мультимедійного контенту.

Предмет дослідження – розробка платформи для зберігання мультимедійного контенту на основі фреймворку Laravel.

Метою роботи є створення та розробка підсистеми для зберігання мультимедійного контенту з використанням сучасних технологій та інструментів.

Ключові слова: підсистема, мультимедійний контент, фреймворк, база даних, веб-розробка, програмний інтерфейс.

ANNOTATION

Yevsiuk V. M. Subsystem for Multimedia Content Storage. Manuscript.

Qualification work of a Bachelor's degree in the field of "Computer Engineering," specialization 123 Computer Engineering. Lutsk National Technical University. Lutsk, 2023.

The qualification work consists of 85 pages and includes 30 illustrations, 1 table, 7 appendices, and 13 references.

The first chapter provides an analysis of the subject area, namely: analysis of the existing problem, comparative characteristics of tools for subsystem development, and a justified selection of technologies. It also analyzes the most popular platforms for storing multimedia content (YouTube, Vimeo, Google Photos, Dropbox).

The second chapter discusses general methodologies for developing applications based on the Laravel framework. The MVC architectural pattern is considered.

The third chapter is dedicated to the description of the development and implementation of the subsystem for multimedia content storage. The database architecture of the system is designed, images of the platform's software interfaces are provided, and examples of code writing for multimedia content storage functionality are added.

The research object is the technologies and tools for developing a subsystem for multimedia content storage.

The research subject is the development of a platform for storing multimedia content based on the Laravel framework.

The aim of this work is to create and develop a subsystem for storing multimedia content using modern technologies and tools.

Keywords: subsystem, multimedia content, framework, database, web development, software interface.

ЗМІСТ

ВСТУП

Актуальність теми. У сучасному світі мультимедійний контент є невід'ємною частиною життя багатьох людей. За останні роки зростання об'єму мультимедійного контенту значно перевищило зростання обсягу текстового контенту, і тому зберігання, керування та розповсюдження мультимедійного контенту стає все більш складним завданням для підприємств та організацій. Тому розробка функціональної підсистеми зберігання мультимедійного контенту є досить актуальною темою.

Стан вивченості проблеми. Проблема збереження мультимедійного контенту та розробки відповідних підсистем є актуальною і досить добре дослідженою. Існують різноманітні платформи та технології для зберігання мультимедійного контенту, такі як YouTube, Vimeo, Google Photos, Dropbox та інші. Однак, враховуючи постійний розвиток технологій та зростання обсягу даних, існує потреба в подальшому вдосконаленні та розробці нових підходів до збереження мультимедійного контенту з урахуванням сучасних вимог ефективності, надійності та безпеки.

Метою кваліфікаційної роботи є розробка підсистеми для зберігання мультимедійного контенту з використанням сучасних технологій та інструментів. Основним завданням є аналіз предметної області, вибір оптимальних технологій та архітектури, розробка функціональності для зберігання, управління та доступу до мультимедійного контенту.

Об'єктом дослідження є технології та інструменти для розробки підсистеми зберігання мультимедійного контенту. В рамках роботи будуть розглянуті різні методики розробки додатків на основі фреймворку Laravel, а також архітектурний шаблон MVC, що дозволить створити модульну та здатну до розширення систему збереження мультимедійного контенту.

Предметом дослідження є розробка платформи для зберігання мультимедійного контенту на основі фреймворку Laravel. У цій роботі буде зосереджено увагу на проектуванні та реалізації підсистеми, яка забезпечує

збереження та управління різними типами мультимедійних даних, таких як текстові дані, зображення, відео тощо. Розробка цієї підсистеми дозволить створити ефективний та надійний інструмент для збереження мультимедійного контенту з врахуванням потреб користувачів.

Завдання, які необхідно виконати під час виконання кваліфікаційної роботи:

– Порівняти та обрати інструменти та технології для розробки;
– Проаналізувати найпопулярніші платформи у галузі мультимедії;
– Дослідити роботу та інструменти фреймворку Laravel;
– Розглянути архітектурний шаблон програмування MVC;
– Спроектувати базу даних для підсистеми;
– Розробити функціональну підсистему зберігання мультимедійного контенту.

Апробація роботи: Кваліфікаційна робота виконувалась в межах Договору про співпрацю згідно Міжнародного Проекту «Готський шлях: спільна історична реконструкція та віртуальна подорож у минуле» (укладений між ЛНТУ та Виконавчим комітетом Володимир-Волинської міської ради від 07 лютого 2022 року). Подано до публікації статті: Христинець Н.А., Лавренчук С.В., Євсюк В.М., Крулік Ю.О. Функціональні адаптивні інтерфейси з динамічними компонентами для підсистем зберігання мультимедійного контенту. Науковий журнал «Комп’ютерно-інтегровані технології: освіта, наука, виробництво». Луцьк, 2023. № 51. С. 87-93; Nataliia Khrystynets, Kateryna Melnyk, Svitlana Lavrenchuk, Oksana Miskevych, Volodymyr Yevsiuk. Multiprocessing as a Way to Optimize Query Execution Time. Луцьк, 2023.

Для досягнення поставленої мети та вирішення завдань використовуватиметься комплексний підхід, що включатиме: аналіз літературних джерел, порівняльний аналіз існуючих платформ для зберігання мультимедійного контенту, а також проектування та реалізацію підсистеми з використанням практичних прикладів і програмного коду.

Для підтримки дослідження використовувалися різноманітні джерела інформації, зокрема книги, онлайн-ресурси, документація з використовуваних технологій, а також досвід практичної реалізації подібних підсистем. Наукова література та актуальні джерела забезпечили теоретичну базу для розуміння основних концепцій та принципів зберігання мультимедійного контенту, а також розробки веб-додатків з використанням фреймворку Laravel.

Крім того, використання практичних прикладів, програмного коду та документації забезпечило практичну базу для реалізації підсистеми зберігання мультимедійного контенту.

Завдяки поєднанню теоретичних та практичних джерел інформації, ця робота пропонує комплексний підхід до розробки підсистеми зберігання мультимедійного контенту, що враховує сучасні вимоги та технології.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз існуючої проблеми

У сучасному інформаційному суспільстві мультимедійний контент займає значне місце в нашому житті. Насамперед потрібно дати визначення мультимедійному контенту, щоб при подальшій роботу розуміти з якими даними потрібно працювати. Отже, мультимедійний контент – це будь-який тип інформації, що представлений у вигляді аудіо, відео, тексту, зображень, анімацій або будь-якої комбінації зазначених форматів. За типом форматів даних, які в себе включає мультимедійний контент, його можна поділити на такі групи: статичний (зображення, текст), динамічний (анімації, відео) та інтерактивний (інтерфейси взаємодії із користувачем). Він використовується в багатьох провідних галузях, таких як освіта, наукові дослідження, реклама, розваги та інші. Відповідно, виникає потреба в розробці ефективної підсистеми зберігання мультимедійного контенту, яка забезпечить виконання всіх необхідних функцій користувача.

Для розробки підсистеми зберігання мультимедійного контенту необхідно врахувати такі системні вимоги: швидкість, доступність, масштабованість, безпека, надійність, а також підтримка різних форматів контенту.

На основі аналізу існуючих рішень у сфері зберігання мультимедійного контенту, можна визначити, що для створення оптимальної підсистеми потрібно використовувати як апаратне, так і програмне забезпечення. До апаратного забезпечення можна віднести сервери, системи зберігання, мережеве обладнання, а до програмного забезпечення – операційні системи, бази даних, мультимедійні кодеки та інше.

Обсяг мультимедійного контенту постійно зростає, а формати файлів стають все більш різноманітними. Це вимагає від підсистеми зберігання підтримки великої кількості форматів та кодеків для мультимедійного контенту. Інформаційні потоки можуть бути організовані за різними критеріями, зокрема:

за видом контенту (аудіо, відео, зображення), за призначенням (освітній, розважальний, науковий), за користувачами (індивідуальні, корпоративні) тощо.

Підсистема зберігання мультимедійного контенту має забезпечувати такі функції: зберігання, організацію, пошук, доступ до мультимедійного контенту; підтримку різних форматів даних; розподілене зберігання з масштабованістю та надійністю; захист від несанкціонованого доступу; можливість інтеграції з іншими системами.

Створення підсистеми зберігання мультимедійного контенту є доцільним з огляду на широке застосування мультимедійного контенту в різних сферах життя суспільства. Така система допоможе оптимізувати роботу з мультимедійним контентом, підвищити продуктивність роботи користувачів та забезпечити захист даних. Крім того, вона може стати основою для розвитку нових інформаційних сервісів, що будуть забезпечувати швидкий та зручний доступ до мультимедійного контенту.

1.2 Обґрунтування вибору технологій та інструментів для розробки веб-платформи

Для написання підсистеми для зберігання мультимедійного контенту потрібно визначити стек технологій, які будуть використовуватися. До нього можна віднести : мову програмування, текстовий редактор для написання коду, для зручності та легкості написання коду можна обрати фреймворк на основі вибраної мови програмування, віртуальну машина для запуску проекту в локальному середовищі та систему для керування базами даних. Проаналізувавши окремо кожну технологію, це допоможе нам зібрати список потрібних інструментів для написання сервісу керування мультимедійним контентом.

1.2.1 Аналіз мов програмування

Спочатку потрібно визначити мову програмування, так як подальші інструменти напряму будуть залежати саме від неї. Для написання веб-додатку

існує безліч мов програмування, наприклад Python, C#, Go, PHP, JavaScript. Це дуже популярні мови програмування, які використовуються для веб-розробки. Кожна мова має свої особливості та переваги, які можна врахувати при виборі мови програмування для проекту.

Python є однією з найпопулярніших мов програмування у сфері веб-розробки завдяки своїй простоті та зручності використання. Python має багато бібліотек та фреймворків, які дозволяють розробникам швидко створювати веб-додатки та веб-сайти. Django та Flask – це два дуже популярні фреймворки для веб-розробки на Python. Django надає широкі можливості для розробки веб-додатків з базами даних та адміністративним інтерфейсом, тоді як Flask – це легкий фреймворк для швидкої розробки веб-додатків.

C# – це мова програмування написана на архітектурі C-подібних мов (C, C++) та має статичну типізацію, що забезпечує більший захист для написання програм. Саме для веб-розробки використовується на платформі .NET. Ця мова має багато вбудованих функцій та бібліотек, які дозволяють створювати швидкі та надійні веб-додатки. ASP.NET – це фреймворк для розробки веб-додатків на платформі .NET, який надає засоби для роботи з базами даних та аутентифікації користувачів.

Go – це мова програмування, яка була розроблена Google та використовується для веб-розробки. Вона дозволяє створювати високопродуктивні веб-додатки з великою кількістю паралельних запитів. Go має вбудовану підтримку мережевих операцій та асинхронного програмування.

JavaScript – це мова програмування, яка використовується для створення веб-додатків та додання взаємодії на веб-сторінки. Вона є найпопулярнішою мовою програмування для фронтенд-розробки, тобто для розробки інтерактивних веб-сторінок та додання взаємодії користувача. Крім того, JavaScript також використовується для розробки серверних додатків за допомогою Node.js. У світі веб-розробки є безліч фреймворків та бібліотек на JavaScript, таких як React, Angular, Vue.js, та інші.

PHP – це мова програмування, яка була розроблена для створення веб-додатків. Вона є однією з найпопулярніших мов програмування для веб-розробки, зокрема для створення динамічних веб-сайтів та веб-додатків. PHP має велику кількість фреймворків, таких як Laravel, Symfony, CodeIgniter та Yii, які дозволяють швидко та ефективно розробляти веб-додатки та веб-сайти з базами даних.

В загальному, вибір мови програмування залежить від конкретних потреб проекту та умов розробки. У нашому випадку Python та PHP є найкращими варіантами. Однак, якщо порівняти ці дві мови, тоді PHP буде більше вдалим вибором для написання такої системи. Щоб довести доцільність вибору мови програмування PHP, було визначено наступні основні переваги використання цієї мови кодування.

Широке поширення: PHP є однією з найбільш поширених мов програмування для розробки веб-сайтів та веб-додатків. Вона підтримується на більшості серверів та інтегрованих з середовищем веб-розробки, таких як Apache і NGINX.

Простота використання: PHP має простий синтаксис та відносно невисокий поріг входження. Вона також має багато документації та допоміжних матеріалів, що робить її добрим вибором для початківців.

Функціональність: PHP має велику кількість функцій та бібліотек, які дозволяють розробникам створювати веб-сервіси з різними функціональними можливостями. Наприклад, PHP має вбудовану підтримку для роботи з базами даних та можливості обробки форм.

Продуктивність: PHP може бути дуже продуктивним для розробки веб-сервісів, оскільки має підтримку для використання кешування, що дозволяє зменшити час відповіді сервера та поліпшити швидкість роботи веб-додатку.

1.2.2 Основи програмного забезпечення Docker для управління контейнерами в мультимедійних системах

Docker – це відкрите програмне забезпечення, яке дозволяє запускати програми в контейнерах.

«Щоб зрозуміти принципи роботи Docker, його часто порівнюють з роботою транспортних перевезень. Колись транспортні компанії стикалися з проблемою перевезення різних типів товарів – разом (наприклад, продукти харчування з побутовою хімією або скло з цеглою). Як одна вантажівка може перевезти товари різних розмірів і типів? Із введенням контейнерів це стало можливим. Вантаж різного розміру розподілений по стандартизованих контейнерах, які перевозяться одним і тим же транспортним засобом» [1]. Контейнер – це стандартизоване середовище, у якому можна запускати програмне забезпечення з усіма необхідними залежностями, не залежно від операційної системи, на якій воно запускається.

Docker складається з декількох компонентів:

– Docker Engine – це основний компонент Docker, який дозволяє створювати та управляти контейнерами;

– Docker Hub – це реєстр, де можна зберігати та ділитися контейнерами з іншими користувачами Docker;

– Docker Compose – це інструмент для управління багатоконтейнерними додатками.

Програмне забезпечення, що працює в контейнері, можна запускати на будь-якій машині, що підтримує Docker, без потреби встановлення всіх залежностей окремо. Використання контейнерів дозволяє запускати програмне забезпечення з усіма необхідними залежностями, незалежно від інших встановлених програм на машині. Docker має підвищену ефективність: контейнери можуть бути запущені та зупинені швидко, що дозволяє швидко масштабувати додатки. Окрім цього використання контейнерів дозволяє економити час на встановлення та конфігурування окремих серверів та інфраструктури.

Для роботи з Docker потрібно мати певні знання та навички. Наприклад, розуміння того, як створювати та налаштовувати Docker образи, як використовувати Docker Compose для запуску додатків з кількох контейнерів, як керувати мережами, як використовувати інструменти моніторингу.

Для створення Docker образу потрібно написати Dockerfile, який містить інструкції для створення образу, у тому числі інструкції для встановлення залежностей та налаштування додатку. Після створення Docker образу, його можна запустити як контейнер на будь-якій машині, що підтримує Docker.

Docker Compose є важливою складовою Docker, який дозволяє запускати та управляти багатоконтейнерними додатками. За допомогою Docker Compose можна описати конфігурацію додатку та його залежностей в файлі docker-compose.yml, а потім запустити всі контейнери додатку за допомогою однієї команди.

Docker має широку екосистему інструментів та служб, які допомагають розробникам та адміністраторам підтримувати та контролювати контейнери. Наприклад, Docker Swarm дозволяє керувати кластером контейнерів, а Docker Hub дозволяє зберігати та ділитися контейнерами з іншими користувачами Docker.

Розробники, які пишуть на фреймворці Laravel, також часто використовують бібліотеку Laradock [2]. Це набір уже багатьох налаштованих Docker контейнерів, якими легко керувати при використанні цієї бібліотеки.

Загалом, Docker є потужним та універсальним інструментом, що дозволяє створювати, розгортати та масштабувати додатки в зручний та ефективний спосіб. Багато компаній та проектів використовують Docker для підвищення ефективності розробки та підтримки додатків, тому вивчення Docker може бути корисним для розробників та адміністраторів, що працюють з сучасними технологіями.

1.2.3 Вибір текстового редактору (IDE)

Текстові редактори є необхідними інструментами для розробки веб-додатків мовою програмування PHP. Серед популярних редакторів можна виділити Sublime Text, Visual Studio Code, Atom, PhpStorm та інші. Однак серед цього списку найбільше підходящим варіантом є PhpStorm.

PhpStorm – це інтегроване середовище розробки (IDE) для мови програмування PHP, розроблене компанією JetBrains. Це платний текстовий

редактор, однак для студентів можна отримати безкоштовну версію програми на час навчання в навчальному закладі. PhpStorm має вбудований редактор PHP з підсвічуванням синтаксису, автодоповненням коду, вбудованим дебагером, зручними інструментами для рефакторингу та багатьма іншими корисними функціями. Це дозволяє зробити розробку на PHP більш ефективною та зручною. PhpStorm не обмежується тільки розробкою на PHP. Він підтримує багато інших мов програмування, таких як HTML, CSS, JavaScript, TypeScript, SQL та багато інших. Це дозволяє розробникам працювати з повним стеком технологій, що забезпечує більш широкі можливості для розробки веб-додатків. PhpStorm інтегрується з багатьма іншими інструментами, такими як системи контролю версій, бази даних, фреймворки та бібліотеки. Це дозволяє розробникам працювати з ними безпосередньо з редактора, що забезпечує більш зручний та швидкий розвиток проекту. Крім цього редактор має високу продуктивність для розробки на PHP. Він дозволяє швидко створювати та редагувати код, що зменшує час, потрібний для розробки та тестування веб-додатків.

Однією з найбільш важливих можливостей цього інтегрованого середовища розробки є вбудована підтримка Docker. Цей функціонал забезпечує можливість розробникам працювати зі складовими веб-додатків, що запущені в контейнерах Docker, без необхідності переходу до зовнішніх інструментів. У PhpStorm є спеціальний Docker панель, яка відображає стан контейнерів та дозволяє розробникам запускати, зупиняти, видаляти та перезапускати контейнери. Крім того, PhpStorm має вбудовану підтримку Docker Compose, що дозволяє легко керувати складними проектами, які мають багато контейнерів.

PhpStorm можна вважати найкращим варіантом для розробки веб-додатків на мові програмування PHP через його високу продуктивність, вбудовану підтримку PHP та інших мов програмування, інтеграцію з іншими інструментами, підтримку Docker, підтримку тестування та інші корисні функції.

1.2.4 Порівняльна характеристика фреймворків

Забезпечення ефективного рішення для створення підсистеми мультимедійного контенту вимагає вибору надійного та гнучкого фреймворку.

В сучасному світі розробки веб-додатків існує безліч фреймворків для мови програмування PHP, кожен з яких має свої переваги та недоліки. У цьому дослідженні ми розглянемо кілька популярних PHP фреймворків, зокрема Symfony, Yii, CodeIgniter та Laravel, з метою визначити найкращий фреймворк для створення підсистеми мультимедійного контенту.

Symfony – це потужний та гнучкий фреймворк, який широко використовується для створення великих та складних веб-додатків. Він пропонує відмінну структуру додатків та сильні інструменти для розробки, однак, може бути важким для новачків та має високу криву навчання. Symfony підходить для більших проектів, але може бути надмірним для підсистеми мультимедійного контенту.

Yii – це швидкий та легкий PHP фреймворк, який надає гарну продуктивність та простоту використання. Він містить ряд корисних функцій та розширень, але може не мати всієї потрібної функціональності для створення складної підсистеми мультимедійного контенту.

CodeIgniter – це легкий та простий у використанні PHP фреймворк, який надає гнучкість та швидкість розробки. Він має низьку криву навчання та може бути відмінним вибором для невеликих проектів, однак, може не мати достатньої функціональності для створення високофункціональної підсистеми мультимедійного контенту.

Laravel – це сучасний PHP фреймворк, який забезпечує широкий спектр функцій та гнучкість для розробки веб-додатків. Laravel має простий та елегантний синтаксис, відмінну документацію та велику спільноту розробників. Він надає потужні інструменти для роботи з мультимедійним контентом, такі як підтримка різних типів файлів, обробка завантаження файлів, керування зображеннями та відео, а також інтеграція з різними системами зберігання файлів. Laravel також пропонує вбудовані рішення для автентифікації, авторизації та безпеки.

Розглянемо порівняння чотирьох описаних вище PHP фреймворків у таблиці 1.1:

Таблиця 1.1 – Порівняльна оцінка фреймворків

Фреймворк	Простота використання	Функціональність	Робота з мультимедійним контентом	Спільнота
Symfony	✦✦✧✧✧	✦✦✦✦✦	✦✦✦✧✧	✦✦✦✦✧
Yii	✦✦✦✦✧	✦✦✦✧✧	✦✦✦✧✧	✦✦✦✧✧
CodeIgniter	✦✦✦✦✧	✦✦✧✧✧	✦✦✧✧✧	✦✦✦✧✧
Laravel	✦✦✦✦✦	✦✦✦✦✦	✦✦✦✦✦	✦✦✦✦✦

Враховуючи аналіз та порівняння різних PHP фреймворків, Laravel виявляється найкращим вибором для створення підсистеми мультимедійного контенту. Завдяки його простоті використання, гнучкості, різноманітності функціональності та сильній підтримці мультимедійного контенту, Laravel дозволяє створювати складні та високопродуктивні веб-додатки. Крім того, велика спільнота розробників та якісна документація забезпечують надійну підтримку та допомогу під час процесу розробки.

Laravel є оптимальним вибором для створення підсистеми мультимедійного контенту через гнучкість, функціональність та простоту в роботі з мультимедійним контентом. Використовуючи Laravel, розробники можуть швидко та ефективно реалізовувати високоякісні та надійні рішення для зберігання, керування та обробки мультимедійного контенту.

1.2.5 Вибір технологій

На підставі проведеного аналізу, для розробки підсистеми зберігання мультимедійного контенту стек технологій, який включає у себе мову програмування PHP, систему для керування локальною віртуальною машиною Docker, текстовий редактор PHPStorm та фреймворк Laravel є обґрунтованим та оптимальним для розробки такої платформи (рисунок 1.1).

Рисунок 1.1 – Обраний стек технологій для розробки мультимедійної системи

Використання мови програмування PHP забезпечує широкі можливості та гнучкість для створення веб-додатків, а також має велику спільноту розробників. Фреймворк Laravel було обрано через його простоту використання, потужність, підтримку мультимедійного контенту та активну спільноту. Docker дозволяє створювати ізольовані середовища розробки, що спрощує розгортання та тестування додатків. Текстовий редактор PHPStorm було обрано через його продуктивність, інтеграцію з різними технологіями та зручний інтерфейс.

Використання такого стеку технологій дозволить забезпечити простоту та легкість роботи розробника, а також надійність, високу продуктивність та гнучкість платформи, що задовольнить вимоги до зберігання, обробки та управління мультимедійним контентом та потреби користувача.

1.3 Інструментарій GitHub

Для написання проекту програмісти часто використовують сервіси на яких можна зберігати вихідний код програми, керувати ним, відстежувати зміни по проекту тощо. Особливо зручно використовувати такі платформи для роботи в команді, коли часто доводиться відслідковувати зміни в коді.

Одним із таких веб-сервісів для роботи з контролем версій є GitHub [3]. Він базується на системі контролю версій Git та розроблений з метою спрощення спільної роботи над проектами та забезпечення стабільності коду. У даному розділі ми розглянемо інструментарій GitHub, його основні компоненти та функціональні можливості.

Ключовим компонентом GitHub є репозиторії. Це такі сховища для проекту, які містять у собі файли та історію їхніх змін. В залежності від налаштувань доступу до проекту, репозиторій може бути публічним або приватним. Користувачі можуть створювати власні репозиторії та внести свій вклад у репозиторії інших користувачів через механізм форкування (forking) та подання запитів на злиття (pull requests).

Для будь-якого збереження коду до репозиторія використовуються коміти (commits). Коміт – це зміна або набір змін у коді, які зберігаються у репозиторії.

Кожен коміт має унікальний ідентифікатор (hash), пов'язаний автор, час створення та повідомлення, яке описує зміни. Коміти відображаються у хронологічному порядку та дозволяють розробникам відстежувати зміни у коді та відновлювати попередні версії, якщо це необхідно. На рисунку 1.2 зображено загальний інтерфейс репозиторію користувача на GitHub.

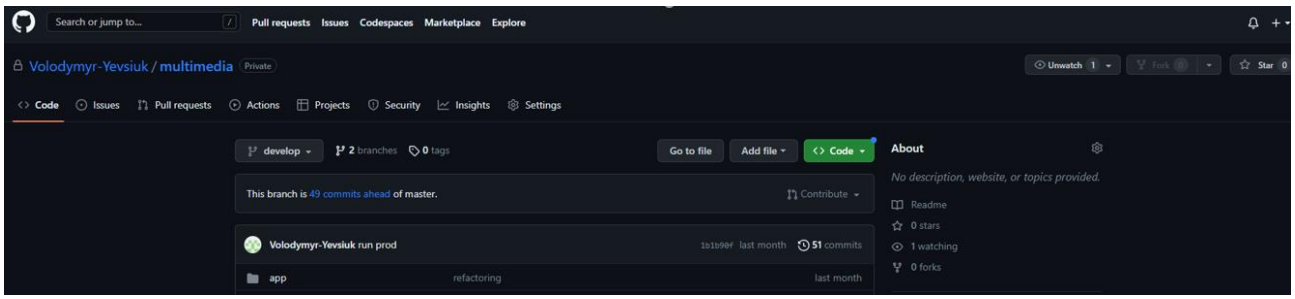


Рисунок 1.2 – Репозиторій проекту на веб-сервісі GitHub

Форкування – це функція, яка довзоляє іншому користувачу зробити копію репозиторія для внесення своїх змін у проект без впливу на оригінальний репозиторій.

Гілка (branch) – це паралельна версія репозиторія, яка дозволяє розробникам працювати над окремими відрізками коду або функціоналом. Після створення репозиторію, зазвичай він має лише одну гілку (main або master). Хорошою практикою вважається створення додаткової гілки develop, на яку програмісти закидають зміни по проекту на стадії розробки. По закінченню проекту робиться через запит на злиття (pull request) гілки develop в основну гілку.

Запит на злиття (pull request) – це процес об'єднання змін з однієї гілки у іншу. Він дозволяє розробникам переглянути, обговорити та внести правки до коду перед злиттям. Цей процес також включає рецензування коду (code review) та перевірку змін на відповідність стандартам та вимогам проекту.

Також GitHub має таку цікаву вкладку як «Проблеми» (Issues). Ця вкладка зазвичай корисна для open-source репозиторіїв, які активно використовуються іншими розробниками. Загалом можна сказати, що це система відстеження помилок та запитів на покращення, яка допомагає розробникам організовувати роботу над проектом. Користувачі можуть створювати проблеми для повідомлення про помилки, ідеї або питання, пов'язані з проектом. Проблеми можуть мати статус (open/closed), мітки (labels), відповідальних (assignees) та посилання на пов'язані запити на злиття або коміти.

Ще одним корисним інструментом веб-сервісу є GitHub Actions. Це набір автоматизованих робочих процесів репозиторію, який забезпечує налаштування таких дій як перевірка коду, тестування, збірка та розгортання програмного забезпечення. Вони активуються на основі певних подій, таких як коміти, запити на злиття, створення проблем або публікація релізів.

Оволодіння цим інструментарієм є важливим для сучасних розробників програмного забезпечення, оскільки GitHub став стандартом для співпраці та розробки відкритого програмного забезпечення.

1.4 Аналіз популярних веб-платформ для зберігання мультимедійного контенту та постановка завдання

У сучасному світі інтернет технологій, зберігання мультимедійного контенту вимагає відповідних веб-платформ, які можуть забезпечити доступність, зручність користування, безпеку та привабливі опції для користувачів. Різні платформи пропонують різні можливості, і вибір платформи для зберігання мультимедійного контенту залежить від потреб користувачів та вимог до функціональності. У даному розділі будуть проаналізовані такі веб-сервіси як YouTube, Vimeo, Google Photos та Dropbox (рисунок 1.3).

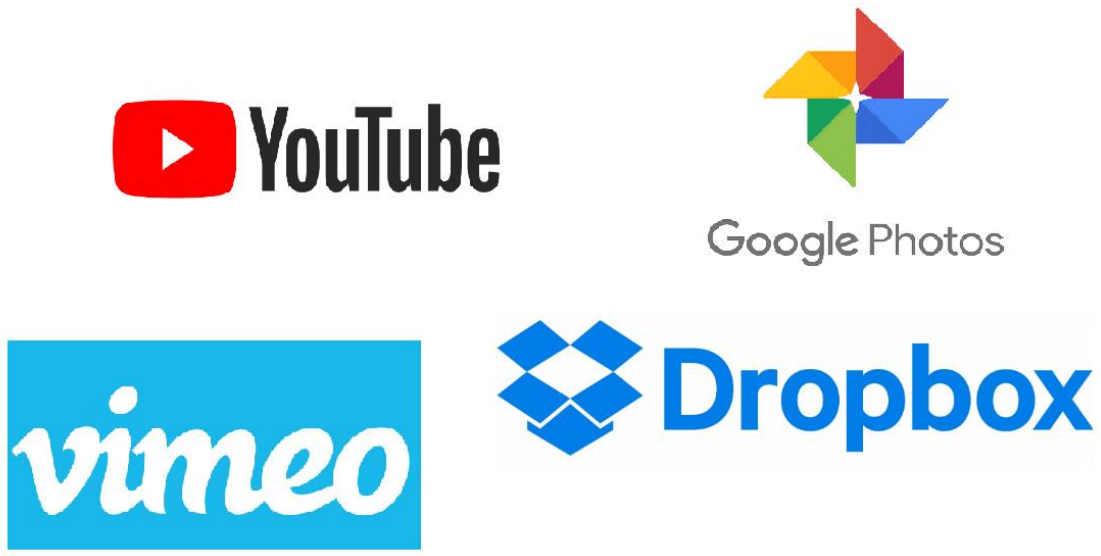


Рисунок 1.3 – Найпопулярніші мультимедійні веб-сервіси

YouTube є найбільш відомою відеохостинговою платформою, що дозволяє користувачам завантажувати, переглядати, коментувати та ділитися відео. Пропонуючи різні плани підписок, YouTube надає можливість вибрати оптимальний обсяг зберігання та функціональність. Однією з головних переваг YouTube є простота використання, висока доступність та інтеграція з іншими

сервісами Google. Водночас, ця платформа має обмеження щодо типів контенту та форматів файлів.

Vimeo відрізняється від YouTube, орієнтуючись на високоякісний контент та професійних відеографів. Відсутність реклами та акцент на якість контенту відрізняють Vimeo на ринку відеохостингу. Незважаючи на більш високі тарифи, Vimeo пропонує різні плани підписок з різними обсягами зберігання та можливостями.

Google Photos – це платформа для зберігання та організації фотографій та відео. Платформа пропонує безкоштовний план з обмеженим обсягом зберігання та платні плани для більшого обсягу зберігання. Основні переваги Google Photos включають автоматичне резервне копіювання, розширені функції пошуку, автоматичну організацію контенту та інтеграцію з іншими сервісами Google. Однак, платформа має певні обмеження щодо форматів файлів та може автоматично стискати фотографії та відео для оптимізації зберігання.

Dropbox є хмарною платформою зберігання, яка дозволяє користувачам зберігати, синхронізувати та обмінюватися різними типами файлів, включаючи мультимедійний контент. Dropbox пропонує безкоштовні та платні плани з різними обсягами зберігання, а також співпрацює з багатьма сторонніми додатками та сервісами. Однак, Dropbox не має вбудованих інструментів для редагування або перегляду мультимедійного контенту, і користувачам може знадобитися встановлювати додаткові програми для цих цілей.

У цілому, аналіз популярних веб-платформ для зберігання мультимедійного контенту показує, що кожна з них має свої переваги та недоліки. Вибір платформи залежить від потреб користувачів та вимог до функціональності. YouTube та Vimeo підходять для зберігання відео, проте мають різний фокус на аудиторію та якість контенту. Google Photos є відмінним варіантом для зберігання фотографій та відео, особливо для користувачів інших сервісів Google. Dropbox пропонує універсальне рішення для зберігання різних типів файлів, включаючи мультимедійний контент, але може вимагати додаткових програм для редагування та перегляду.

Проаналізувавши наявні платформи для роботи з мультимедійним контентом, можна зробити висновок що важливими аспектами при виборі веб-платформи для зберігання мультимедійного контенту є обсяг зберігання, доступність, інтеграція з іншими сервісами, можливості пошуку та організації контенту, а також підтримка різних форматів файлів. Окрім того, варто враховувати потреби користувачів, їхній рівень знань та технічних навичок, а також вимоги до якості контенту та можливостей співпраці. Наприклад, YouTube підходить для користувачів, які хочуть завантажувати, переглядати та ділитися відео, але має обмеження щодо форматів файлів та типів контенту. Vimeo пропонує високоякісний контент та звертає увагу на професійних відеографів, проте має вищі тарифи в порівнянні з іншими платформами. Google Photos ідеально підходить для зберігання та організації фотографій та відео, але може автоматично стискати файли для оптимізації зберігання та має обмеження щодо форматів файлів. Dropbox дозволяє зберігати, синхронізувати та обмінюватися різними типами файлів, але не має вбудованих інструментів для редагування або перегляду мультимедійного контенту. Тому при розробці власної підсистеми зберігання мультимедійного контенту варто виконати такі задачі:

– авторизація користувачів (потрібна для того, щоб редагування контенту на сторінках могли змінювати лише користувачі, які мають такі права);

– сервіс для редагування текстових сторінок платформи (з передбаченими можливостями редагування мета-тегів, тексту, та відображення сторінки);

– сторінка для завантаження та видалення відео контенту;

– сторінка з галереєю (можливість редагування фото).

Для виконання поставлених задач потрібно написати код підсистеми таким чином, щоб далі можна було проаналізувати програмну реалізацію кожного завдання.

РОЗДІЛ 2
ЗАСОБИ ТА ІНСТРУМЕНТИ ФРЕЙМВОРКУ LARAVEL

2.1 Життєвий цикл запиту Laravel

На мою думку, вибравши якусь технологію для використання у роботі, спеціаліст повинен насамперед дослідити яким чином ця технологія працює. Те ж саме стосується й розробки програм, адже коли розробник розуміє як працює інструмент розробки, програміст відчуває себе більш впевнено та комфортно у роботі.

2.1.1 Старт програми

«The entry point for all requests to a Laravel application is the public/index.php file. All requests are directed to this file by your web server (Apache/Nginx) configuration. The index.php file doesn't contain much code. Rather, it is a starting point for loading the rest of the framework.

The index.php file loads the Composer generated autoloader definition, and then retrieves an instance of the Laravel application from bootstrap/app.php. The first action taken by Laravel itself is to create an instance of the application / service container». (Точкою входу для всіх запитів до Laravel-застосунку є файл public/index.php. Усі запити направляються до цього файлу за допомогою конфігурації веб-сервера (Apache/Nginx). Файл index.php не містить багато коду. Натомість, це початкова точка для завантаження решти фреймворку.

Файл index.php завантажує створений Composer автозавантажувач, а потім отримує екземпляр Laravel-застосунку з файлу bootstrap/app.php. Перша дія, яку виконує сам Laravel, – це створення екземпляра застосунку / контейнера служб) [4].

2.1.2 HTTP та консольні ядра

«Next, the incoming request is sent to either the HTTP kernel or the console kernel, depending on the type of request that is entering the application. These two kernels serve as the central location that all requests flow through. For now, let's just focus on the HTTP kernel, which is located in app/Http/Kernel.php.

The HTTP kernel extends the Illuminate\Foundation\Http\Kernel class, which defines an array of bootstrappers that will be run before the request is executed. These bootstrappers configure error handling, configure logging, detect the application environment, and perform other tasks that need to be done before the request is actually handled. Typically, these classes handle internal Laravel configuration that you do not need to worry about.

The HTTP kernel also defines a list of HTTP middleware that all requests must pass through before being handled by the application. These middleware handle reading and writing the HTTP session, determining if the application is in maintenance mode, verifying the CSRF token, and more. We'll talk more about these soon.

The method signature for the HTTP kernel's handle method is quite simple: it receives a Request and returns a Response. Think of the kernel as being a big black box that represents your entire application. Feed it HTTP requests and it will return HTTP responses». (Далі вхідний запит надсилається до ядра HTTP або ядра консолі, в залежності від типу запиту, який надходить до додатку. Ці два ядра служать центральним місцем, через яке проходять усі запити. Наразі зосередимося на ядрі HTTP, яке розташоване за адресою app/Http/Kernel.php.

Ядро HTTP наслідує клас Illuminate\Foundation\Http\Kernel, який визначає масив завантажувачів, які будуть виконані перед виконанням запиту. Ці завантажувачі налаштовують обробку помилок, журналів, виявляють середовище додатку та виконують інші завдання, які потрібно зробити перед тим, як запит буде оброблено. Зазвичай ці класи обробляють внутрішню конфігурацію Laravel.

Ядро HTTP також визначає список посередників HTTP, через які повинні пройти всі запити, перш ніж будуть оброблені додатком. Ці посередники обробляють читання та запис сеансів HTTP, визначають, чи перебуває додаток у режимі технічного обслуговування, перевіряють токен CSRF та більше.

Сигнатура методу handle ядра HTTP досить проста: вона отримує запит Request та повертає відповідь Response. Думайте про ядро як про велику чорну

коробку, яка представляє весь ваш додаток. Подайте йому запити HTTP, і він поверне відповіді HTTP) [4].

2.1.3 Сервіс провайдери

«One of the most important kernel bootstrapping actions is loading the service providers for your application. Service providers are responsible for bootstrapping all of the framework's various components, such as the database, queue, validation, and routing components. All of the service providers for the application are configured in the config/app.php configuration file's providers array.

Laravel will iterate through this list of providers and instantiate each of them. After instantiating the providers, the register method will be called on all of the providers. Then, once all of the providers have been registered, the boot method will be called on each provider. This is so service providers may depend on every container binding being registered and available by the time their boot method is executed.

Essentially every major feature offered by Laravel is bootstrapped and configured by a service provider. Since they bootstrap and configure so many features offered by the framework, service providers are the most important aspect of the entire Laravel bootstrap process». (Однією з найважливіших дій під час завантаження ядра є завантаження сервіс провайдерів для додатка. Сервіс провайдери відповідають за ініціалізацію всіх різноманітних компонентів фреймворку, таких як база даних, черги, перевірка та компоненти маршрутизації. Усі сервіс провайдери налаштовані в масиві providers файлу конфігурації config/app.php.

Laravel буде послідовно проходити через цей список провайдерів та ініціювати кожного з них. Після їхньої ініціалізації буде викликано метод register у всіх провайдерах. Потім, коли всі провайдери будуть зареєстровані, у кожному класі провайдера буде викликано метод boot. Це дозволяє їм покладатися на те, що всі прив'язки контейнерів будуть зареєстровані та доступні на момент виконання їх методу boot.

По суті, кожна основна функція, яку пропонує Laravel, ініціалізується та налаштовується за допомогою сервіс провайдера. Оскільки вони ініціалізують та

налаштовують багато функцій, які пропонує фреймворк, сервіс провайдери є найважливішою частиною всього процесу завантаження Laravel) [4].

2.1.4 Маршрутизація

«One of the most important service providers in your application is the App\Providers\RouteServiceProvider. This service provider loads the route files contained within your application's routes directory. Go ahead, crack open the RouteServiceProvider code and take a look at how it works!

Once the application has been bootstrapped and all service providers have been registered, the Request will be handed off to the router for dispatching. The router will dispatch the request to a route or controller, as well as run any route specific middleware.

Middleware provide a convenient mechanism for filtering or examining HTTP requests entering your application. For example, Laravel includes a middleware that verifies if the user of your application is authenticated. If the user is not authenticated, the middleware will redirect the user to the login screen. However, if the user is authenticated, the middleware will allow the request to proceed further into the application. Some middleware are assigned to all routes within the application, like those defined in the $middleware property of your HTTP kernel, while some are only assigned to specific routes or route groups. You can learn more about middleware by reading the complete middleware documentation.

If the request passes through all of the matched route's assigned middleware, the route or controller method will be executed and the response returned by the route or controller method will be sent back through the route's chain of middleware». (Один з найважливіших сервіс провайдерів у додатку є App\Providers\RouteServiceProvider. Цей провайдер завантажує файли маршрутів, що містяться у каталозі маршрутів додатка.

Після того, як додаток був запущений, і всі постачальники послуг зареєстровані, Request буде переданий маршрутизатору для диспетчеризації. Маршрутизатор надішле запит на маршрут або контролер, а також запустить будь-яке специфічне для маршруту проміжне програмне забезпечення.

Проміжне програмне забезпечення (Middleware) надає зручний механізм для фільтрації або аналізу HTTP-запитів, що надходять у ваш додаток. Наприклад, Laravel включає проміжне програмне забезпечення, яке перевіряє, чи є користувач додатка аутентифікованим. Якщо користувач не аутентифікований, проміжне програмне забезпечення перенаправить користувача на екран входу. Однак, якщо користувач аутентифікований, проміжне програмне забезпечення дозволить запиту пройти далі. Деяке проміжне програмне забезпечення призначено для всіх маршрутів у додатку, наприклад, ті, що визначені у властивості $middleware HTTP-ядра, деякі призначені тільки для конкретних маршрутів або груп маршрутів.

Якщо запит пройде через всі призначені для відповідного маршруту проміжні програми, метод маршруту або контролера буде виконано, і відповідь, що повертається методом маршруту або контролера, буде відправлена назад через зв’язок проміжного програмного забезпечення маршруту) [4].

2.1.5 Завершення життя циклу запиту

«Once the route or controller method returns a response, the response will travel back outward through the route's middleware, giving the application a chance to modify or examine the outgoing response.

Finally, once the response travels back through the middleware, the HTTP kernel's handle method returns the response object and the index.php file calls the send method on the returned response. The send method sends the response content to the user's web browser». (Кожного разу, коли маршрут або метод контролера повертає відповідь, вона проходитиме через проміжне програмне забезпечення маршруту, надаючи додатку можливість змінити або перевірити вихідну відповідь.

Нарешті, коли відповідь проходить назад через проміжне програмне забезпечення, метод handle ядра HTTP повертає об'єкт відповіді, а файл index.php викликає метод send на повернутій відповіді. Метод send відправляє вміст відповіді до веб-переглядача користувача) [4]. Після цього етапу життєвий цикл запиту завершується.

2.2 Міграції, фабрики та посіви даних

Використання міграцій, фабрик та посівів даних в Laravel дозволяє розробникам ефективно керувати базами даних, створювати тестові дані та наповнювати базу даних початковими значеннями. Ці інструменти полегшують роботу з базами даних, забезпечують зручність та послідовність розробки, а також допомагають уникнути конфліктів між розробниками. Вони сприяють створенню відповідного тестового середовища, що підвищує якість та надійність розробленого додатка.

2.2.1 Міграції

Міграції дозволяють ефективно керувати структурою бази даних та слідкувати за її змінами в процесі розробки. Завдяки системі контролю версій, міграції сприяють послідовності та безперебійній роботі команди розробників. Міграції забезпечують створення, модифікацію та видалення таблиць бази даних, дозволяючи відновити структуру бази даних при перенесенні проекту на інші сервери. На рисунку 2.1 зображено приклад міграції таблиці admins.

```php
public function up()
{
    Schema::create( table: 'admins', function (Blueprint $table) {
        $table->id();
        $table->string( column: 'name');
        $table->string( column: 'email')->unique();
        $table->string( column: 'login')->unique()->index();
        $table->string( column: 'password');
        $table->rememberToken();
        $table->timestamps();
    });
}
```

Рисунок 2.1 – Функція міграції для створення структури таблиці БД

2.2.2 Фабрики

Фабрики відіграють важливу роль в автоматичному створенні тестових даних для бази даних. Це дозволяє розробникам швидко та легко наповнити таблиці бази даних даними для тестування функціоналу додатка. Фабрики,

створені за допомогою бібліотеки Faker, генерують випадкові та реалістичні дані, що сприяє ретельному тестуванню додатку перед його розгортанням.

2.2.3 Посіви даних

Посіви даних (Seeders) є доповненням до фабрик, що дозволяють наповнити базу даних початковими або тестовими даними, згенерованими фабриками. Цей інструмент є корисним при створенні демонстраційних версій додатків та під час тестування. Використання посівів даних спрощує процес тестування, забезпечуючи одноразове наповнення бази даних замість ручного введення даних. Приклад звичайно класу посіву даних зображено на рисунку 2.2.

Рисунок 2.2 – Приклад реалізації посіва даних для наповнення таблиці бази даних admins

2.2.4 Застосування міграцій, фабрик та посівів даних у реальних проектах

При написанні реальних проектів, міграції, фабрики та посіви даних можуть бути використані для різних цілей, таких як:

– Розробка нових функцій, що вимагають змін в структурі бази даних. Міграції дозволяють забезпечити послідовне та контрольоване внесення змін в базу даних.

– Тестування взаємодії додатка з базою даних. Фабрики та посіви даних допомагають швидко створювати реалістичні тестові дані для перевірки відповідності додатка очікуваним результатам.

– Навчання нових розробників. Міграції та посіви даних можуть бути використані для створення навчального середовища з прикладами бази даних, що допоможуть новим розробникам швидше розібратися в структурі та функціоналі проекту.

Використання міграцій, фабрик та посівів даних є ключовим аспектом при розробці сучасних веб-додатків на Laravel. Розуміння та ефективне застосування цих інструментів є важливим навиком для розробників, які працюють з базами даних. Завдяки їх гнучкості та масштабованості, міграції, фабрики та посіви даних можуть допомогти розробникам ефективно працювати над проектами будь-якого розміру та складності.

2.3 Обробка та валідація запитів

Запити (Requests) в Laravel представляють вхідні дані, що надходять від користувачів до веб-додатку. Запити можуть містити різні типи даних, такі як рядки, числа, файли, масиви тощо. Laravel надає клас Request, який допомагає обробляти інформацію, що надходить від користувачів. «To obtain an instance of the current HTTP request via dependency injection, you should type-hint the Illuminate\Http\Request class on your route closure or controller method. The incoming request instance will automatically be injected by the Laravel service container». (Для отримання доступу екземпляра поточного запиту HTTP за допомогою ін’єкції залежностей, вам слід ввести підказку класу Illuminate\Http\Request у методі закриття маршруту або контролера. Екземпляр вхідного запиту буде автоматично впроваджено контейнером служби Laravel) [5]. На рисунку 2.3 зображено приклад доступу до HTTP запиту у методі store контролера UserController.

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class UserController extends Controller
{
    /**
     * Store a new user.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return \Illuminate\Http\Response
     */
    public function store(Request $request)
    {
        $name = $request->input('name');

        //
    }
}
```

Рисунок 2.3 – Отримання доступу до екземпляра запиту HTTP у методі контролера

2.3.1 Отримання вхідних даних

В Laravel можна отримати вхідні дані з різних типів запитів, таких як GET, POST, PUT, DELETE тощо. Незалежно від типу запиту, ви можете використовувати такі методи класу Request для отримання даних:

– input() – дозволяє отримати значення елемента форми за ім'ям;

– only() – дозволяє отримати підмножину вхідних даних за масивом ключів;

– except() – дозволяє отримати всі вхідні дані, окрім зазначених ключів;

– has() – дозволяє перевірити, чи містить запит вказаний ключ. Це корисно, коли ви хочете виконати певні дії тільки тоді, коли користувач надіслав певні дані;

– all() – дозволяє отримати всі вхідні дані запиту у вигляді масиву. Це корисно, коли вам потрібно обробити всі дані разом.

Також доступ до вхідних даних можна отримати використовуючи динамічні властивості екземпляра класу Illuminate\Http\Request. Наприклад, щоб отримати доступ до значення поля name та записати його в змінну , потрібно буде звернутися до запиту таким чином:

$name = $request–>name;

При використанні динамічних властивостей Laravel спочатку шукатиме значення параметра в даних, які надіслав користувач. Якщо цього параметра там немає, тоді Laravel шукатиме поле в параметрах відповідного маршруту.

2.3.2 Робота з файлами

Для отримання завантажених користувачем файлів, Laravel пропонує використовувати метод file() або динамічні властивості. Нижче описано код для отримання екземпляру класу Illuminate\Http\UploadedFile, який надає різні методи для взаємодії з файлом:

$file = $request–>file(‘photo’);

$file = $request–>photo;

Щоб впевнитись, що користувач в запиті дійсно відправив файл, можна викликати метод hasFile(), який поверне значення типу boolean.

Щоб зберегти завантажений файл, ви зазвичай використовуєте одну з налаштованих файлових систем, які можна налаштувати у файлі config/filesystems.php. Клас UploadedFile має метод store(), який забезпечує збереження файлу на одну з файлових систем.

Метод store приймає шлях, де має зберігатися файл відносно налаштованого кореневого каталогу файлової системи. Цей шлях не повинен містити ім’я файлу, оскільки автоматично буде згенеровано унікальний ідентифікатор, який буде використовуватися як ім’я файлу.

Метод store також приймає додатковий другий аргумент для імені диска, який попередньо повинен бути налаштований у масиві disks за шляхом config/filesystems.php. Метод поверне шлях до файлу відносно кореня диска:

$path = $request–>photo–>store(‘images’);

$path = $request–>photo–>store(‘images’, ‘s3’);

Якщо розробник хоче задати власне ім’я для збереженого файлу, тоді слід використати метод storeAs(), який прийме першим параметром – шлях для збереження, другим – ім’я файлу, а третім – назва диску для збереження. Приклад використання методу storeAs:

$path = $request–>photo–>storeAs(‘images’, ‘filename.jpg’);

$path = $request–>photo–>storeAs(‘images’, ‘filename.jpg’, ‘s3’);

2.3.3 Валідація вхідних даних запиту

При написанні реальних проектів, розробник не може передбачити, які вхідні дані він отримає із запиту користувача. Наприклад, користувач у полі «Ім’я» вводить занадто довгий рядок, або замість числа запит відправляє рядок тексту. Для уникнення таких проблем та для покращення якості обробки запитів, Laravel забезпечує програмістів зручним способом перевірки даних – валідацією. Валідація вхідних даних є важливою частиною процесу розробки веб-додатків, оскільки вона дозволяє перевірити, чи відповідають введені дані вимогам, встановленим для їх формату, типу та правильності.

З коробки, всі контролери Laravel використовують ValidatesRequests trait, який забезпечує контроллер методом validate(). Першим аргументом він приймає екземпляр класу Illuminate\Http\Request, який містить у собі вхідні дані форми, а другий параметр передбачає масив правил, де ключ масиву – це назва поля форми, а значення це масив правил, на які це поле перевіряється. На рисунку 2.4 показано приклад валідації даних через метод validate(). Хоча коду багато не написано для цього функціоналу, однак ці чотири рядки виконують багато роботи. Спочатку ми явно визначаємо поля, які очікуємо, і застосовуємо правила (відокремлені символом |) до кожного з них окремо. Далі, метод validate() перевіряє вхідні дані з змінної екземпляра класу Illuminate\Http\Request $request (це означає, що він може використовувати методи для отримання даних $request–>all() або $request–>input()) та визначає, чи є вони коректними.

```php
// routes/web.php
Route::get('recipes/create', 'RecipesController@create');
Route::post('recipes', 'RecipesController@store');
// app/Http/Controllers/RecipesController.php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class RecipesController extends Controller
{
    public function create()
    {
        return view('recipes.create');
    }

    public function store(Request $request)
    {
        $this->validate($request, [
            'title' => 'required|unique:recipes|max:125',
            'body' => 'required'
        ]);

        // Recipe is valid; proceed to save it
    }
}
```

Рисунок 2.4 – Приклад використання методу validate() для перевірки даних [6]

Якщо дані є правильними, метод validate завершує свою роботу, і ми можемо перейти до методу контролера, щоб зберегти дані або провести будь-які інші маніпуляції із цими даними.

Однак завжди існує ймовірність, що дані які отримує контроллер є неправильними, тоді виникає помилка екземпляру ValidationException. Цей вийняток містить інструкції для маршрутизатора щодо того, як обробити таку помилку. Якщо запит є Ajax (або якщо він запитує JSON як відповідь), то виняток створить відповідь JSON, що містить помилки валідації. Якщо ні, то виняток поверне перенаправлення на попередню сторінку, разом з усім введеними користувачем даними та помилками валідації, що ідеально підходить для повторного заповнення невдалої форми та показу деяких помилок.

Окрім такої можливості валідації даних, фреймворк пропонує ще декілька варіантів для обробки вхідних даних. Наприклад, екземпляр класу Illuminate\Http\Request також має метод validate(). Цей метод практично не відрізняється від попереднього, так як логіка всередині відбувається така сама,

однак у цьому випадку першим аргументом функція приймає вже набір правил, а не екземпляр класу запиту.

Якщо програмісту не потрібно працювати в контролері або з якоїсь іншої причини попередні варіанти обробки даних запиту не підходять, Laravel дозволяє вручну створити екземпляр класу Validator та перевірити результат на правильність. Створюючи екземпляр валідатора, потрібно передати масив вхідних даних як перший параметр та правила валідації як другий. Клас Validator має метод fails(), викликавши який ми можемо перевірити чи дані пройшли процес валідації чи ні. Якщо ж метод fails() поверне булеве значення true, це означатиме, що вхідні дані є неправильними. Після цього ми зможемо передати клас валідатора у метод withErrors() для перенаправлення з помилками. Виглядатиме ця реалізація наступним чином:

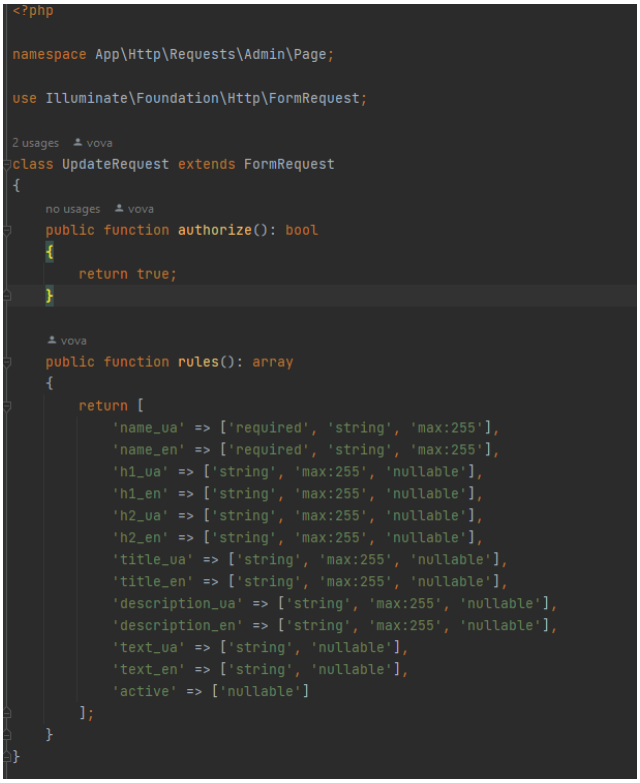
Проте існує також і третій варіант для валідації вхідних даних – це використання класів Form Request. Цей варіант реалізації функціоналу для валідації є кращим, так як логіка обробки даних запиту винесена в окремий файл, що зменшує кількість коду усередині контроллера. Звичайно, коли потрібно перевірити на правильність невелику к-сть даних, підійдуть і попередні варіанти, однак для перевірки великої кількості полів форми слід використовувати екземпляри класів Form Request.

Для створення нового класу Form Request необхідно виконати в консолі команду:

php artisan make:request StoreRequest

Запустивши цю команду в консолі, буде створено об'єкт запиту форми, який знаходитиметься за таким шляхом: app/Http/Requests/StoreRequest.php.

Кожен клас запиту форми надає один або два публічних методи. Перший – rules(), який повинен повернути масив правил валідації для цього запиту. Другий метод є необов’язковим – authorize(); якщо він повертає true, то користувач має дозвіл на виконання цього запиту, однак якщо false, запит користувача буде відхилено. На рисунку 2.5 зображено приклад реалізації класу UpdateRequest, який перевіряє велику кількість вхідних даних.



```php
<?php

namespace App\Http\Requests\Admin\Page;

use Illuminate\Foundation\Http\FormRequest;

2 usages  vova
class UpdateRequest extends FormRequest
{
    no usages  vova
    public function authorize(): bool
    {
        return true;
    }

    vova
    public function rules(): array
    {
        return [
            'name_ua' => ['required', 'string', 'max:255'],
            'name_en' => ['required', 'string', 'max:255'],
            'h1_ua' => ['string', 'max:255', 'nullable'],
            'h1_en' => ['string', 'max:255', 'nullable'],
            'h2_ua' => ['string', 'max:255', 'nullable'],
            'h2_en' => ['string', 'max:255', 'nullable'],
            'title_ua' => ['string', 'max:255', 'nullable'],
            'title_en' => ['string', 'max:255', 'nullable'],
            'description_ua' => ['string', 'max:255', 'nullable'],
            'description_en' => ['string', 'max:255', 'nullable'],
            'text_ua' => ['string', 'nullable'],
            'text_en' => ['string', 'nullable'],
            'active' => ['nullable']
        ];
    }
}
```

Рисунок 2.5 – Реалізація класу UpdateRequest для перевірки вхідних даних

Тепер, коли об'єкт запиту форми створено ми можемо його використати у контролері. Логіка його використання доволі проста – усе, що потрібно це у методі контролера замінити екземпляр класу Illuminate\Http\Request на створений об’єкт форми запиту. З коробки, цей клас перевіряє введення користувача та авторизує запит. Якщо вхідні дані неправильні, то він буде діяти так само, як і метод validate() в контролері, перенаправляючи користувача на попередню сторінку зі збереженим введенням та відповідними повідомленнями про помилки. І якщо користувач не авторизований, Laravel поверне 403 відповідь сервера (доступ заборонено) та не виконає код маршруту.

Окрім наведених правил валідації, які були використані раніше, можна ознайомитися з всім списком правил (Додаток А), які пропонує користувачеві фреймворк Laravel.

Загалом, валідація вхідних даних є важливою частиною розробки веб-додатків. Laravel надає зручні інструменти для валідації вхідних даних, які дозволяють перевірити їх правильність та попередити можливі помилки.

2.4 Архітектурний шаблон MVC

Архітектурний шаблон MVC (Model-View-Controller) є одним з найбільш популярних шаблонів проектування для розробки веб-додатків. Цей шаблон розділяє програму на три компоненти: Модель, Представлення та Контролер, що робить процес розробки більш структурованим і зрозумілим для розробників. Крім того, використання цього шаблону дозволяє підтримувати розділення між логікою додатку та його інтерфейсом користувача, що полегшує модифікацію коду та забезпечує більшу стійкість до змін.

На рисунку 2.6 показано простий приклад роботи цього шаблону:

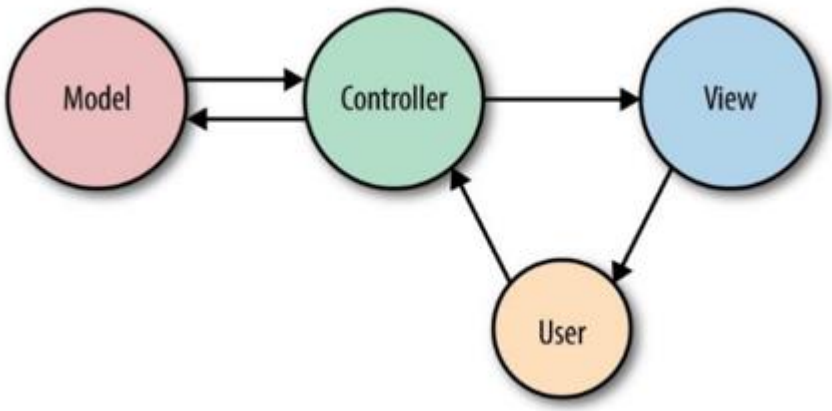


Рисунок 2.6 – Базова ілюстрація роботи архітектурного шаблону MVC [6]

MVC розділяє компоненти інтерфейсу користувача, такі як представлення та моделі. Представлення має забезпечити, що його відображення показує поточний стан моделі. Коли дані моделі змінюються, вона повідомляє всі види, які на неї підписані, що дозволяє кожному виду оновити своє вигляд. Таким чином, можна прикріплювати багато різних видів до моделі, щоб мати різні представлення, не змінюючи код моделі.

Перед тим, як описати реалізацію цього патерну у фреймворку Laravel, потрібно описати, за що відповідає кожен із його компонентів. Модель представляє логіку даних та ділових правил програми. Вона відповідає за

збереження, представлення та обробку даних, а також взаємодію з базою даних. Представлення є відповідальним компонентом за відображення даних користувачеві. Він перетворює дані моделі у відповідні візуальні форми, які можуть бути представлені на екрані. Контролер є зв’язним елементом між моделлю та видом. Він обробляє вхідні дані користувача, здійснює відповідні зміни у моделі та оновлює перегляд.

У фреймворку Laravel реалізація шаблону MVC дуже проста та інтуїтивно зрозуміла для розробників. Контролери в Laravel є класами, які обробляють запити користувачів та взаємодіють з моделями та представленнями. Моделі в Laravel реалізовані за допомогою Eloquent ORM, який забезпечує прості, але потужні способи взаємодії з базами даних, а представлення відображають дані користувачам у вигляді HTML-сторінок.

Архітектурний шаблон MVC дозволяє розробникам ефективно розробляти складні мультимедійні системи зберігання контенту, використовуючи фреймворк Laravel. Застосування MVC у розробці програмного забезпечення, має багато особливостей та переваг. Наприклад, цей шаблон є досить гнучким та масштабованим, що сприяє створенню модульних та масштабованих систем. Зміни в одному компоненті (модель, перегляд або контролер) зазвичай не вимагають суттєвих змін в інших компонентах, що дозволяє легко адаптувати систему до нових вимог або технологій. Проекти, які реалізовані за цим патерном легкі у тестуванні та налагодженні, так як усі компоненти є незалежними один від одного. Це забезпечує більш якісний код та меншу кількість помилок у програмі.

За замовчуванням, у фреймворку Laravel контролери знаходяться в папці app/Http/Controllers, моделі – в папці app/Models, а представлення – в папці resources/views. Однак, таку структуру можна змінити залежно від потреб проекту.

Для нашої підсистеми, моделі можуть слугувати для представлення мультимедійного контенту, такого як фотографії, відео та аудіофайли, а також для збереження та обробки метаданих, пов'язаних з ними. Контролери можуть

обробляти завантаження файлів від користувачів та забезпечувати збереження даних у моделях. Представлення можуть використовувати шаблони Blade для створення динамічних сторінок, на яких відображається мультимедійний контент. Це може включати відображення списків аудіо– та відеофайлів, а також детальні сторінки з додатковою інформацією про кожен елемент контенту.

«MVC also lets you change the way a view responds to user input without changing its visual presentation. You might want to change the way it responds to the keyboard, for example, or have it use a pop-up menu instead of command keys. MVC encapsulates the response mechanism in a Controller object. There is a class hierarchy of controllers, making it easy to create a new controller as a variation on an existing one.» (MVC також дозволяє змінювати спосіб реагування відображення на вхід користувача без зміни його візуальної презентації. Наприклад, ви можете змінити спосіб реагування на клавіатуру або використати спливаюче меню замість командних клавіш. MVC інкапсулює механізм відповіді в об'єкті контролера. Існує ієрархія класів контролерів, що дозволяє легко створювати новий контролер як варіацію на існуючому) [7].

У підсумку, використання шаблону MVC в Laravel дозволяє розробникам створювати структуровані та підтримувані веб-додатки. Використання об'єктно-орієнтованого програмування та вбудованих функцій Laravel, таких як система маршрутизації та бібліотеки, дозволяє зменшити час та зусилля, необхідні для розробки веб-додатків. Загалом, використання шаблону MVC в Laravel є важливим аспектом при створенні високоякісних веб-додатків.

2.5 Шаблон видів Blade

Blade – це потужна система шаблонів, використовувана в PHP фреймворку Laravel, яка дозволяє розробникам створювати гнучкі та зручні шаблони веб-додатків з чітким розділенням логіки та представлення. Усі шаблони Blade скомпільовані у звичайний PHP-код і зберігаються в кеш-пам’яті, доки їх не буде змінено, що сприяє швидкому відображенню представлення та оптимізації

роботи програми. Розташовані зазвичай шаблони у каталозі resources/views та мають розширення файлів .blade.php. Для того щоб відобразити вид на сторінці, у Laravel є допоміжна глобальна функція view(), яка першим параметром приймає шлях до файлу, а другим масив даних, який передає дані для відображення.

Blade використовує спеціальний синтаксис, який дозволяє вбудовувати PHP-код у HTML-шаблони. Наприклад, для виведення змінної name всередині HTML-тегу буде використано наступний синтаксис:

{{ $name }}

Якщо ж виникає ситуація, коли в шаблон передається змінна, яка вже включає у себе HTML-теги і потрібно, щоб змінна не була екранована, тоді потрібно використати наступну конструкцію:

{!! $variable !!}

Однак використання такого синтаксису без екранування може бути небезпечним, так як вид не є захищеним від XSS (Cross-Site Scripting) атак, тому розробник повинен обережно використовувати такого типу конструкцію. [8]

Одними з основних компонентів Blade шаблонів є його директиви. Директиви Blade – це вбудовані команди, які дозволяють виконувати певні операції всередині шаблоні. Вони спрощують роботу з видами, додають гнучкості та забезпечують чистоту коду. Загалом директиви можна поділити на декілька категорій: умовні, циклові, директиви для аутентифікації, стекові, директиви секції, середовища, CSS класів та стилів. Детальніше з кожною директивою цих груп можна ознайомитися у додатку Б.

Blade також дозволяє вам створювати власні директиви за допомогою методу directive у сервіс-провайдері. Приклад створення власної директиви зображено на рисунку 2.7.

```
public function boot()
{
    Blade::directive( name: 'datetime', function ($expression) {
        return "<?php echo ($expression)->format('m/d/Y H:i'); ?>";
    });
}
```

Рисунок 2.7 – Створення власної директиви datetime

Після створення, розробник може використовувати свою директиву в шаблонах Blade, використовуючи ключовий символ «@» перед її назвою. Проте, програміст повинен знати, що при створенні власних директив потрібно дотримуватися правил безпеки, таких як екранування виводу, щоб запобігти атаці XSS.

Ще одним із провідних переваг використання шаблонізатору Blade є його компоненти. Компоненти – це можливість розділити шаблон на незалежні частини, які можна використовувати в різних місцях програми. Компоненти дозволяють забезпечити модульність та перевикористання коду, що сприяє підтримці та розширенню додатку. Щоб створити компонент на основі класу, ви можете використати команду artisan make:component. Щоб проілюструвати, як використовувати компоненти, ми створимо простий компонент TableRow. Команда make:component розмістить компонент у каталозі app/View/Components:

php artisan make:component TableRow

Щоб відобразити компонент, ви можете використати тег компонента Blade в одному зі своїх шаблонів Blade. Теги компонентів Blade починаються з рядка «x-», за яким слідує назва класу компонента написана в стилі kebab case. Якщо клас компонентів вкладено глибше в каталозі app/View/Components, ви можете використовувати «.» символ для позначення вкладеності каталогу. Наприклад, якщо ми припустимо, що компонент розташований у app/View/Components/Inputs/Button.php, ми можемо відобразити його так [8]:

<x-inputs.button/>

Якщо розробнику потрібно передавати дані компонентам Blade, тоді це можливо здійснити за допомогою атрибутів HTML. Жорстко закодовані примітивні значення можуть бути передані компоненту за допомогою простих рядків атрибутів HTML. Вирази та змінні PHP слід передавати до компонента через атрибути, які використовують символ «:» як префікс:

<x-user-profile :name=«$name» :surname=«$surname»/>

Для того, щоб ці дані можна було відобразити в шаблоні, потрібно їх ініціалізувати в конструкторі класу, який відповідає за відображення компоненту. Детальний приклад показано на рисунку 2.8.

```php
<?php

namespace App\View\Components;

use Illuminate\View\Component;

no usages
class UserProfile extends Component
{

    private string $name;
    1 usage
    private string $surname;

    no usages
    public function __construct(string $name, string $surname)
    {
        //
        $this->name = $name;
        $this->surname = $surname;
    }
    public function render()
    {
        return view( view: 'components.user-profile');
    }
}
```

Рисунок 2.8 – Реалізація класу компоненту UserProfile

Загалом шаблони Blade є досить універсальним та потужним інструментом, який забезпечує додаток повною функціональністю та швидкодією.

2.6 Локалізація

Локалізація в Laravel – це механізм, що дозволяє використовувати різні мовні версії вашої програми для різних користувачів на різних мовах.

Laravel забезпечує потужний механізм локалізації, який дозволяє додавати переклади для різних мов, та легко використовувати їх у вашому коді. Для роботи з локалізацією Laravel використовує клас Illuminate\Translation\Translator, який дозволяє вам здійснювати переклади на різні мови.

Щоб розпочати роботу з локалізацією в Laravel, спочатку необхідно додати потрібні мовні файли. Файли з перекладами зберігаються в директорії resources/lang проекту, в папках з назвами мов. Для кожної мови створюється окрема папка, яка включає в себе файли, наприклад, для англійської мови папка буде мати назву en, а для української мови – ua.

Крім того, Laravel також підтримує відміну слів для різних мов. Наприклад, в англійській мові слово «apple» має три форми – «apple», «apples» та «apples» для одних, багатьох або нульової кількості предметів. Laravel автоматично вибирає відповідну форму слова, використовуючи правила відмінювання для поточної мови.

У файлах з перекладами зазвичай зберігаються ключові слова та їх переклади. На рисунку 2.9 зображено приклад файлу interface.php для української локалізації.

```php
<?php

return [
    'titles' => [
        'about-project' => 'Про проект',
        'virtual-studio' => 'Віртуальна студія',
        'virtual-viewers-zone' => 'Віртуальна глядацька зона',
        'games' => 'Ігри',
    ],

    'games' => [
        'gothic-treasures' => [
            'title' => '2D додаток-гра «Готські скарби» 2D додаток-гра
            'text' => 'Виконаний з урахуванням візуального відтворення
        ],
        'multimedia-quiz' => [
            'title' => 'Мультимедійна вікторина',
            'text' => 'Мультимедійна вікторина слугуватиме узагальненням
        ],
    ],
];
```

Рисунок 2.9 – Реалізація української локалізації у файлі interface.php

Після того, як файли з перекладами були додані, ви можете використовувати їх у своєму коді, використовуючи метод trans() або __() для перекладу ключових слів на потрібну мову.

В Laravel також є можливість використовувати різні мови в залежності від мови браузера користувача. Для цього використовується middleware SetLocale, який автоматично встановлює мову відповідно до мови браузера користувача.

Загалом, Laravel має потужну та зручну систему локалізації, що дозволяє створювати додатки, які можуть легко працювати з різними мовами.

РОЗДІЛ 3
РОЗРОБКА ТА РЕАЛІЗАЦІЯ ПІДСИСТЕМИ ЗБЕРЕЖЕННЯ МУЛЬТИМЕДІЙНОГО КОНТЕНТУ

3.1 Проектування архітектури бази даних

Одним із першочергових завдань при розробці реального проекту є проектування бази даних. На цьому етапі програміст передбачає, які таблиці буде містити база даних, якими даними вони будуть наповнені, як правильно налаштувати зв'язки між таблицями тощо. Також важливим аспектом при розробці БД є реалізований програмний інтерфейс користувача. Так як на етапі проектування архітектури бази даних, програмний інтерфейс історичного музею ще не є готовим, структура буде передбачена на основі потреб користувача.

Насамперед потрібно зрозуміти, які саме дані будуть зберігатися до БД. Так як підсистема повинна бути захищеною, для отримання доступу до неї в подальшому буде впроваджена авторизація, для реалізації якої потрібна таблиця *admins*. Ця таблиця бази даних слугуватиме для збереження облікових даних авторизованих користувачів (логін, ім'я, електронна пошта, пароль). Її структура показана на рисунку 3.1:

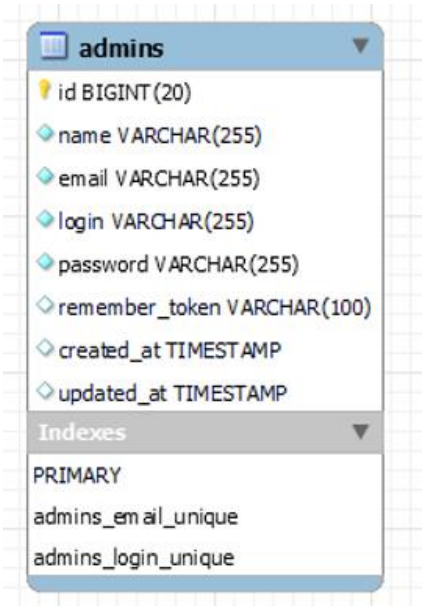


Рисунок 3.1 – Структура таблиці бази даних *admins*

Одним із наступних завдань при розробці підсистеми зберігання мультимедійного контенту є реалізація сервісу зберігання текстових даних для сторінок. Дана підсистема є мультимовною та включатиме 2 мови: українську та англійську. На основі цієї інформації можна зробити висновок, що для збереження даних для текстових сторінок потрібно буде створити дві таблиці – *pages* та *page_contents*. Перша таблиця є основною, вона зберігатиме загальні дані для конкретної сторінки: унікальний ідентифікатор сторінки (url), який використовуватиметься для формування посилання на цю сторінку, параметр активності (active), який відповідає за те, чи потрібно сторінку показувати на платформі чи ні, та параметр сортування (sort), завдяки якому можливо відсортувати сторінки за певним порядком для виводу в програмному інтерфейсі. Таблиця *page_contents* посилатиметься на таблицю *pages*, та зберігатиме усі текстові дані для доступних мов підсистеми (назва сторінки, заголовок, підзаголовок, мета-теги сторінки та текст). На рисунку 3.2 продемонстровано структуру обох цих таблиць.

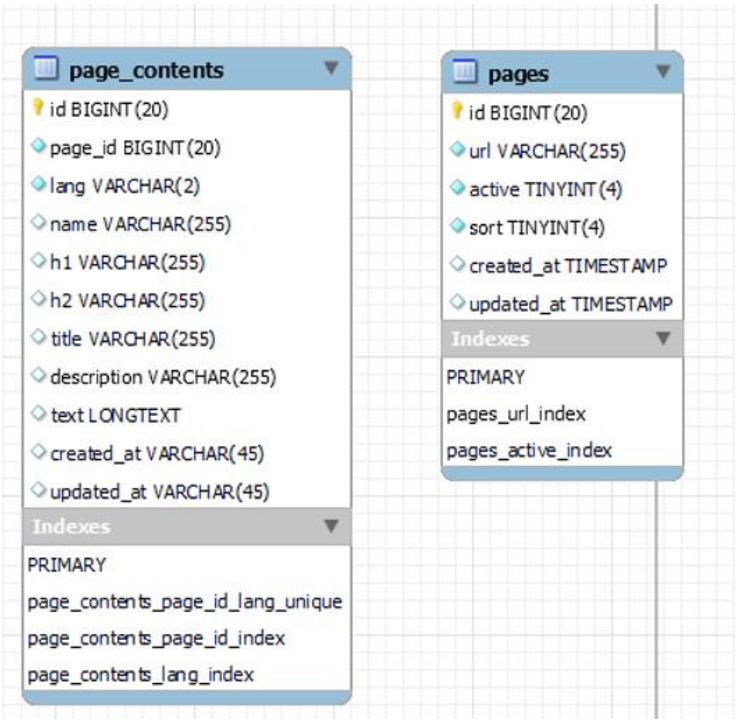


Рисунок 3.2 – Структура таблиць бази даних *pages* та *page_contents*

Для реалізації наступних завдань, а саме зберігання відео та зображень, найкращим варіантом буде спроектувати одну таблицю, яка включатиме у себе будь-які файли медіа та матиме зв’язок з конкретною сторінкою. Для реалізації такої задачі найкращим варіантом буде використати бібліотеку Laravel-medialibrary від команди розробників Spatie. Інструкції по встановленню та використанню цього пакету можна знайти на сайті самих розробників [9]. Після встановлення цієї бібліотеки та запуску міграцій в командному рядку, буде спроектовано нову таблицю яка матиме назву *media*. Структура цієї таблиці показано на рисунку 3.3.

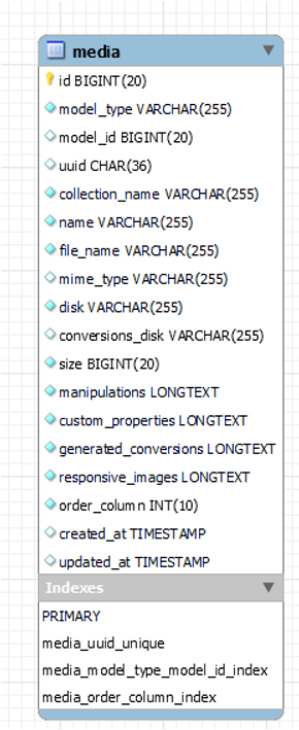


Рисунок 3.3 – Структура таблиці бази даних *media*

Загалом архітектура бази даних проекту включатиме у себе лише 4 таблиці: *admins*, *page*, *page_contents* та *media*. Проте функціональність підсистеми зберігання мультимедійного контенту не залежить від кількості створених таблиць, навпаки чим менше таблиць у базі даних, тим підсистема швидше працюватиме. Наступним кроком розробки є програмування можливості користувача авторизуватися у підсистему.

3.2 Авторизація користувачів

Першим кроком у виконанні цієї задачі є створення шляху маршрутизації для сторінки логіну. Хорошою практикою вважається розділення файлів маршрутизації для програмного інтерфейсу користувачів та для адміністративної підсистеми керування контентом. Тому для початку у папці проекту routes потрібно створити новий файл, який слід назвати admin.php. На рисунку 3.4 зображено структуру каталогу routes.

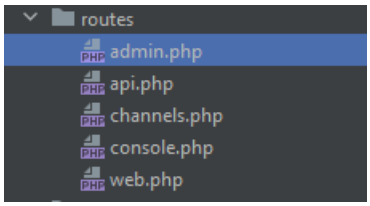


Рисунок 3.4 – Структура папки routes

Далі у методі boot() класу RouteServiceProvider потрібно зареєструвати цей файл, додавши якийсь спеціальний префікс, наприклад *admin*. На рисунку 3.5 зображено реєстрація файлу admin.php. Тепер у цьому файлі можна створювати різні шляхи маршрутизації для підсистеми.

```
Route::middleware( middleware: 'web')
    ->prefix( prefix: 'admin')
    ->name( value: 'admin.')
    ->namespace($this->namespace)
    ->group(base_path( path: 'routes/admin.php'));
```

Рисунок 3.5 – Реєстрація файлу маршрутизації admin.php

Для реалізації авторизації користувача потрібно написати 3 шляхи маршрутизації: перший для відображення сторінки логіну користувача, другий – для авторизації, та третій – для виходу з облікового запису. Також потрібно створити відразу й шлях до головної сторінки адміністративної панелі. Усі ці шляхи маршрутизації показано на рисунку 3.6.

```php
<?php

use App\Http\Controllers\Admin\Auth\LoginController;
use App\Http\Controllers\Admin\IndexController;
use Illuminate\Support\Facades\Route;

Route::middleware( middleware: 'guest:admin')->group(function () {
    /* Show login page */
    Route::get( uri: '/login/', [LoginController::class, 'index'])->name( name: 'login.page');

    /* Authenticate user */
    Route::post( uri: '/login/', [LoginController::class, 'store'])->name( name: 'login');
});

Route::group(['middleware' => ['auth:admin']], static function () {

    /* Logout */
    Route::get( uri: '/logout/', [LoginController::class, 'logout'])->name( name: 'logout');

    /* ADMIN HOME PAGE */
    Route::get( uri: '/', action: IndexController::class)->name( name: 'index');
});
```

Рисунок 3.6 – Шляхи маршрутизації для авторизації користувача

Так як фреймворк Laravel є побудованим на шаблоні проектування MVC, потрібно створити контролер, який оброблятиме запити по вказаних шляхах маршрутизації. Цей контролер відповідатиме за авторизацію, тому його слід назвати LoginController.php. Всередині контролера потрібно створити ті методи, які вказані для цього контролера у файлі маршрутизації admin.php, а саме – index(), store(), logout(). Метод index() створений для того, щоб відобразити сторінку для авторизації користувача, тому все що потрібно для цього зробити – це створити вигляд цієї сторінки. Після реалізації методу index(), сторінка авторизації матиме такий вигляд, як зображено на рисунку 3.7.

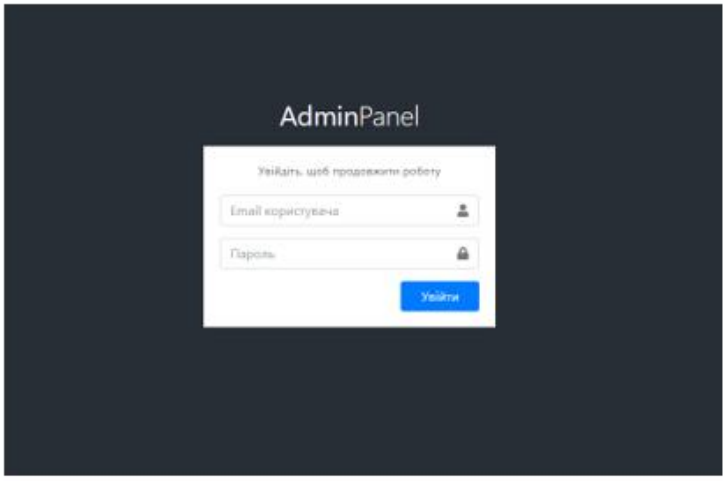


Рисунок 3.7 – Вигляд сторінки авторизації користувача

При нажатті на кнопку «Увійти», відправиться POST запит з даними форми, який у свою чергу попаде до методу store(Request $request), який параметром приймає тіло запиту. Провалідувавши дані, за допомою фасаду Auth та його методу attempt(), перегенерується сесія та користувача перенаправить на головну сторінку адміністративної панелі. Якщо ж користувача не вийде авторизувати, або виникне помилка валідації його перенаправить назад на сторінку авторизації.

Останнім етапом в реалізації авторизації, потрібно реалізувати функціонал виходу з облікового запису. Візуальна частина для можливості виходу з облікового запису показано на рисунку 3.8.

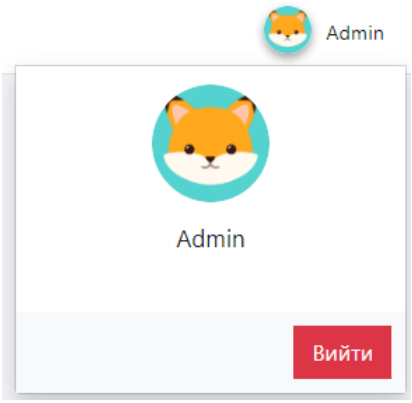


Рисунок 3.8 – Візуальний вигляд функціоналу для виходу з облікового запису

При натисненні на кнопку «Вийти», користувач відправить GET запит, який реалізує метод logout(Request $request) у контролері LoginController. Розробка функціоналу є досить простою: фасад Auth для цього має спеціальний метод logout(). Все, що залишається програмісту це викликати цей метод, знищити попередні дані сесії, перегенерувати токен сесії та перенаправити користувача на сторінку авторизації.

Код реалізації контролера та вигляду авторизації та виходу із облікового запису наведений у додатку Б.

3.3 Реалізація структури виглядів

Розробка структури виглядів досить важливою при проектуванні підсистеми. Кожна сторінка платформи матиме багато схожих компонентів, такі як: header, footer, sidebar, загальні скрипти написані на мові JavaScript та основні стилі (шрифти, задній фон). Для того, щоб реалізувати таку структуру, фреймворк Laravel має спеціальну директиву *@extends*, яку слід використовувати для вказання макету сторінки. Макет сторінки – це загальний blade файл, який матиме спільний набір компонентів для усіх сторінок, які наслідуватимуть даний макет. Хорошою практикою є створення макету як Laravel компоненту, а весь унікальний контент кожної сторінки буде визначений як $slot властивість цього макету. Загальний вигляд інтерфейсу адміністративної панелі зображено на рисунку 3.9.

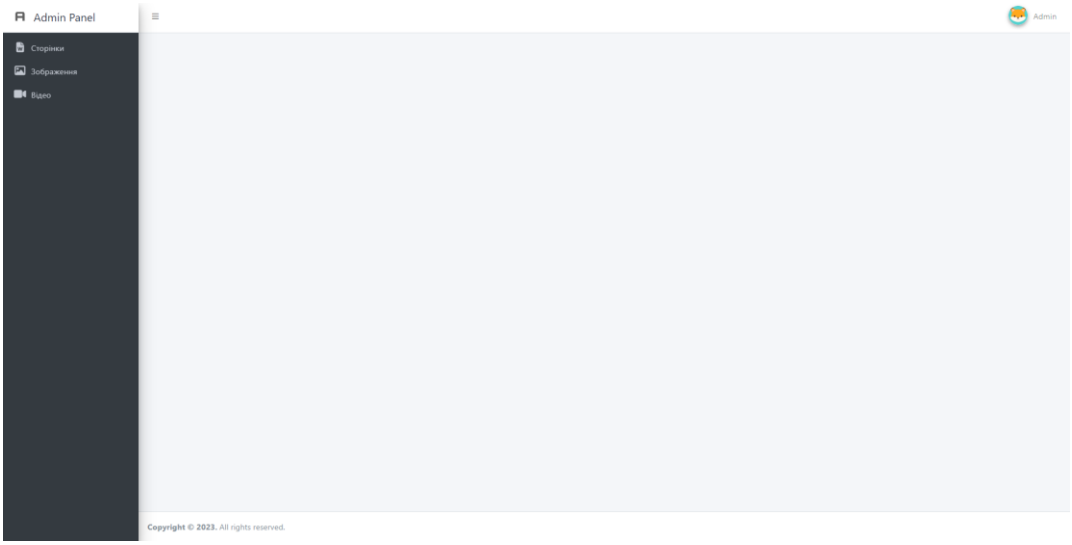


Рисунок 3.9 – Вигляд головної сторінки програмного інтерфейсу

За допомогою цієї візуальної частини, можна чітко прослідкувати використання на сторінці трьох основних компонентів: header, footer та sidebar. Після верстки цих компонентів за допомогою бібліотеки Bootstrap [10], потрібно створити загальний компонент макету – layout.blade.php. Далі в HTML тезі <body> слід підключити ці компоненти в правильному порядку, а основну

частину, яка буде у кожної сторінки унікальною, необхідно визначити через змінну $slot. Повний приклад із вихідним кодом компонентів header, sidebar, footer та макету layout можна переглянути у додатку В.

Щоб розробити загальний вигляд інтерфейсу головної сторінки, потрібно створити новий blade файл для цієї сторінки. На рисунку 3.10 зображено структуру вигляду головної сторінки підсистеми.

```
<x-layouts.admin.layout>
    <x-slot name="title">Admin Panel</x-slot>
    <x-slot name="description">Admin Panel</x-slot>

    <div class="content-wrapper">
        <x-admin.message></x-admin.message>

        <!-- Content Header (Page header) -->
        <section class="content-header">
        </section>

        <!-- Main content -->
        <section class="content container-fluid">
        </section>
        <!-- /.content -->
    </div>
</x-layouts.admin.layout>
```

Рисунок 3.10 – Структура файлу головної сторінки підсистеми

3.4 Розробка сервісу для зберігання текстових даних

Спочатку необхідно з’ясувати, які дії користувачу необхідні для роботи з функціоналом збереження текстових даних сторінок. Насамперед, користувач повинен мати змогу вибрати конкретну сторінку для редагування, можливість перегляду та редагування даних вибраної сторінки. Для цього створимо контролер PageController та шляхи маршрутизації типу *resource* та зазначимо методи, які нам потрібні для реалізації цих задач, а саме: index, edit, update. Зареєструєм дані маршрутизації таким чином:

Route::resource('pages', PageController::class)–>only('index', 'edit', 'update');

Реалізувати функціонал вибору необхідної сторінки можна у вигляді табличного представлення (виводу усіх сторінок у вигляді таблиці) або за допомогою випадаючого списку з пошуковим рядком так кнопкою підтвердження. Об'єктивно другий варіант є кращим та зручнішим для реалізації підсистеми, вигляд цього інтерфейсу зображений на рисунку 3.11.

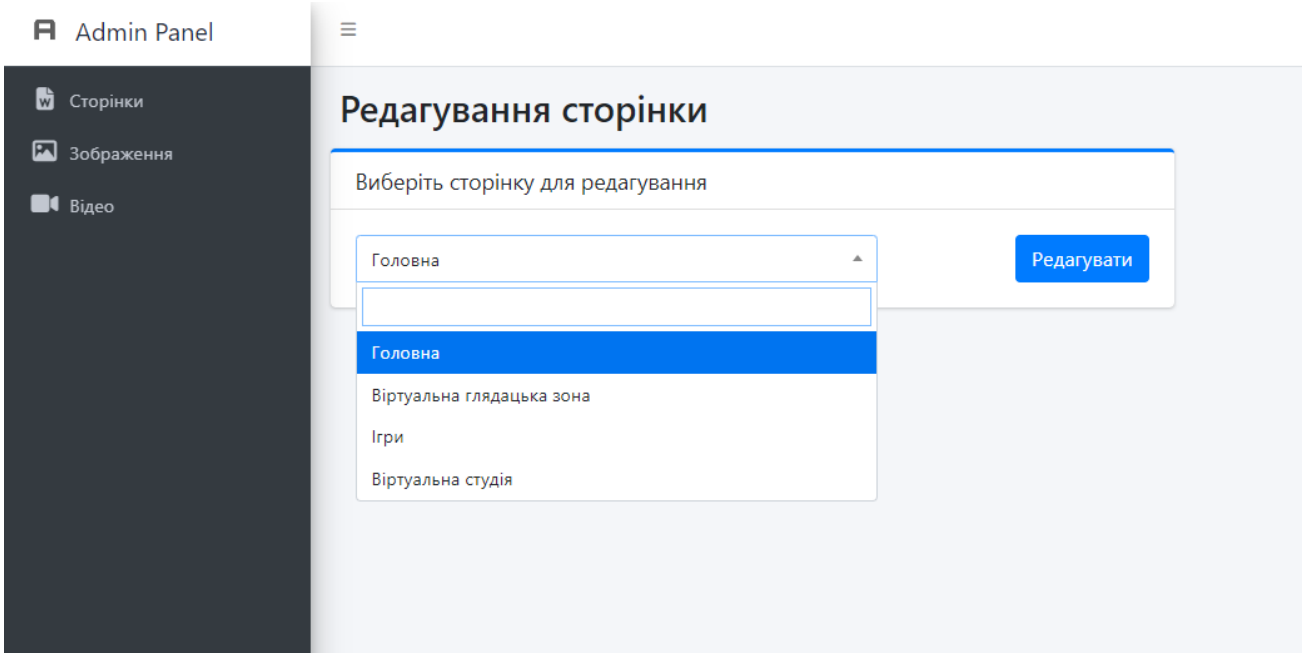


Рисунок 3.11 – Сторінка для вибору сторінки для редагування

Для розробки методу index() необхідно записати усі дані з таблиці *pages* та дані контенту з таблиці *page_contents* в змінну, яку передамо у вигляд. Нижче зображено реалізацію методу index():

```
public function index() {
    $items = Page::with('contents')–>get();
    return view('admin.pages.site–pages.index', compact('items'));
}
```

Після вибраної сторінки та натиснення на кнопку «Редагувати» адміністратора перенаправить на сторінку редагування цієї сторінки.

На рисунку 3.12 зображено дизайн сторінки редагування головної сторінки.

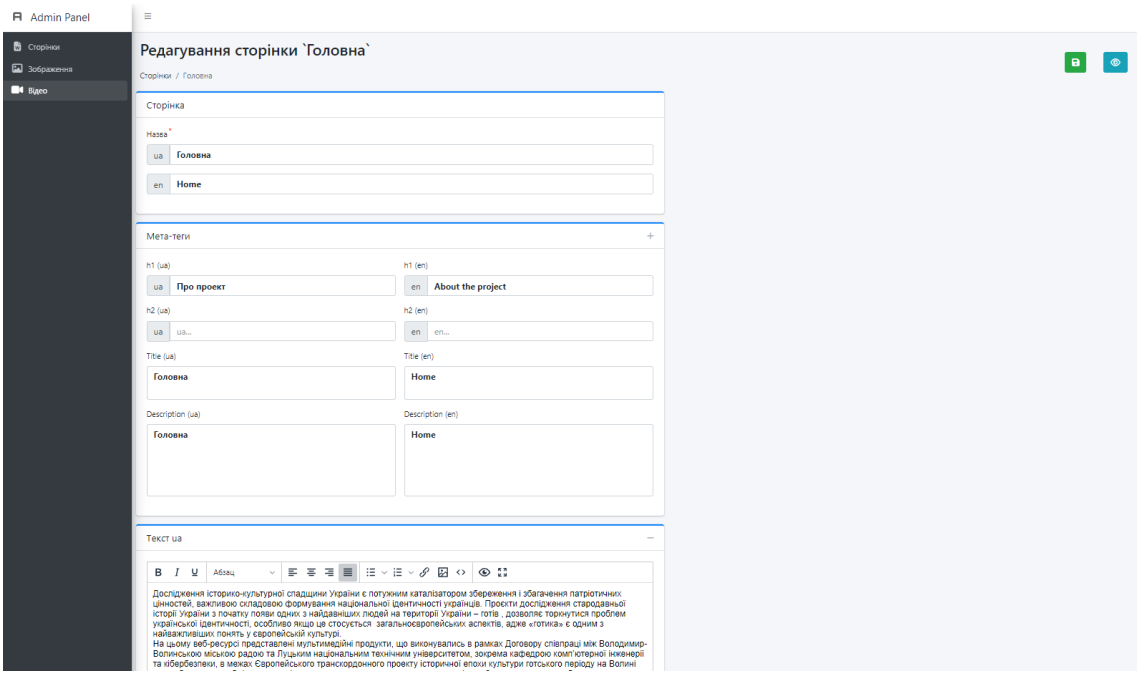


Рисунок 3.12 – Програмний інтерфейс сторінки редагування текстових даних

Вона поділена на декілька блоків: блок з редагуванням назви сторінки та параметру відображення на платформі, блок із мета-тегами та блоки з редагуванням тексту сторінки. Також для зручності користувача передбачено кілька кнопок: зберігання контенту та перегляд цієї сторінки на платформі. Метод edit() відповідає за віддачу конкретного вигляду вибраної сторінки, для цього він приймає параметр $id, який визначає ідентифікатор запису в базі даних. У нашому випадку цей ідентифікатор визначає конкретну вибрану сторінку, тому на основі нього можна надіслати запит до БД, щоб отримати дані для цієї сторінки. Метод edit() у контролері має наступний вигляд:

```
public function edit(int $id)
{
    $item = Page::find($id);
    if (! $item) {
    return redirect(route('admin.pages.index'))
        –>withErrors('Сторінку не знайдено');
    }
    $title = "Сторінка '{$item–>name}'";
```

```
        return view('admin.pages.site-pages.edit', compact('item', 'title'));
    }
```

Після заповнення усіх необхідних даних форми та підтвердження відправки форми, відбувається запит типу PATCH, який обробляється у методі update(). Перед обробкою даних у самому методі потрібно створити клас типу FormRequest для валідації вхідних даних. Після валідації, необхідно розподілити дані форми на дані, які потрібно оновити в таблиці *pages* та на дані для таблиці *page_contents*. За стандартами програмування SOLID [11], логіку збереження можна винести в окремий клас, наприклад StorePageData.php. Це робиться для того, щоб не робити функцію update() надто великою. Нижче приведено код методу update():

```
public function update(UpdateRequest $request, Page $page) {
    try {
        (new StorePageData())–>handle($page, $request–>validated());
        $request–>session()–>flash('message', 'Оновлено успішно!');
    } catch (\RuntimeException $exception) {
        return back()
            –>withInput()
            –>withErrors($exception–>getMessage());
    } catch (\Throwable $exception) {
        return back()
            –>withInput()
            –>withErrors('Серверна помилка!');
    }
    return redirect(route('admin.pages.edit', [$page–>id])); }
```

У додатку Г приведено весь інший код, який використовується для реалізації модуля збереження текстових даних.

3.5 Створення функціоналу збереження зображень

Розробка функціоналу збереження зображень є більш складною системою, ніж збереження текстових даних. Цей функціонал передбачає завантаження зображень з локального диску користувача, після чого він зможе відредагувати зображення: змінити його розмір, приблизити або вибрати конкретну частину зображення, яку виділить користувач. На рисунку 3.13 зображено вигляд програмного інтерфейсу галереї, з можливістю додавання нового фото, або видалення вже існуючого.

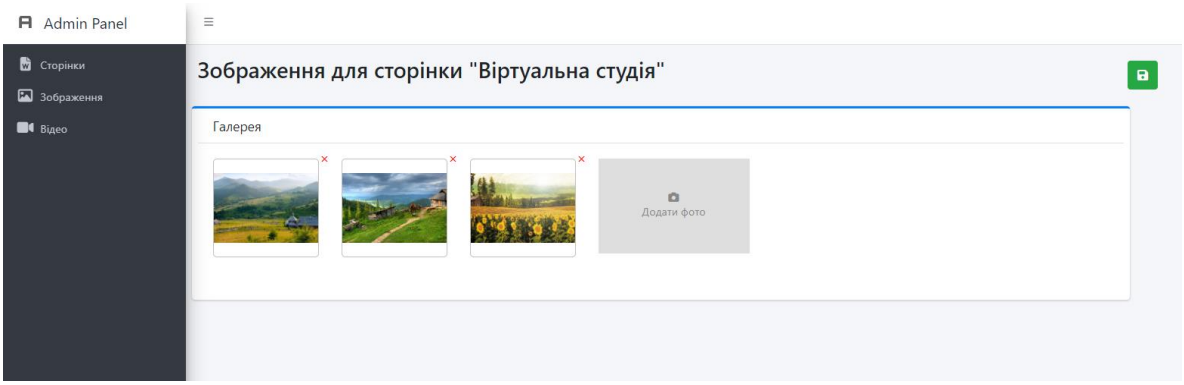


Рисунок 3.13 – Вигляд програмного інтерфейсу галереї

Для цього створимо новий шлях маршрутизації та контролер ImageController типу resource з двома методами: index та store.

Route::resource('images', ImageController::class)–>only('index', 'store');

Метод index() відповідає за зовнішній вигляд галереї. Зображення, які користувач зберігатиме, статично будуть прив’язані для сторінки віртуальної студії, так як лише ця сторінка відповідає за відображення фото на платформі. Тому спочатку потрібно зробити запит до бази даних, щоб отримати модель Page для сторінки віртуальної студії. Щоб система зберігала фото коректно, також потрібно налаштувати модель Page. Для цього бібліотека Laravel-medialibrary надає спеціальний файл (trait) InteractsWithMedia, який потрібно підключити всередині файлу моделі. Крім цього, також щоб забезпечити, що модель Page має

зв’язок з моделлю цієї бібліотеки Media, модель Page повинна реалізувати інтерфейс HasMedia. Після цих налаштувань, всередині моделі Page потрібно зареєструвати колекції медіа, для того, щоб при збереженні в базі даних можна було розрізнити чи цей файл є зображенням, чи іншим типом даних. Для цього потрібно переопреділити метод registerMediaCollections() всередині моделі, та всередині нього зареєструвати медіа колекцію з назвою images.

Наступним етапом буде підключення бібліотеки Croppie JS [12]. За допомогою неї користувач зможе змінити розмір зображення, виділити конкретну область фото, яку хоче зберегти, в окремому модальному вікні. Це модальне вікно з’явиться після вибору зображення з локального диску користувача. На рисунку 3.14 показано вигляд модального вікна для редагування вибраного фото.

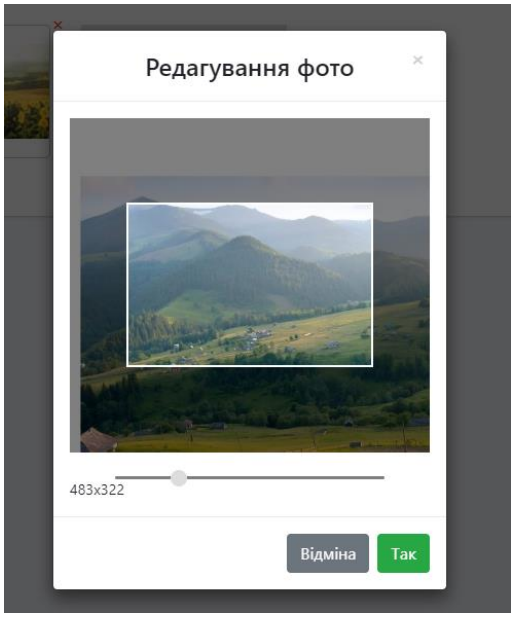


Рисунок 3.14 – Модальне вікно для редагування зображення

Після дії підтвердження в модальному вікні, нове зображення відобразиться в списку, однак воно ще не буде збережене. Для того щоб відправити запит на збереження фото, потрібно натиснути кнопку збереження, яка показана на рисунку 3.15.

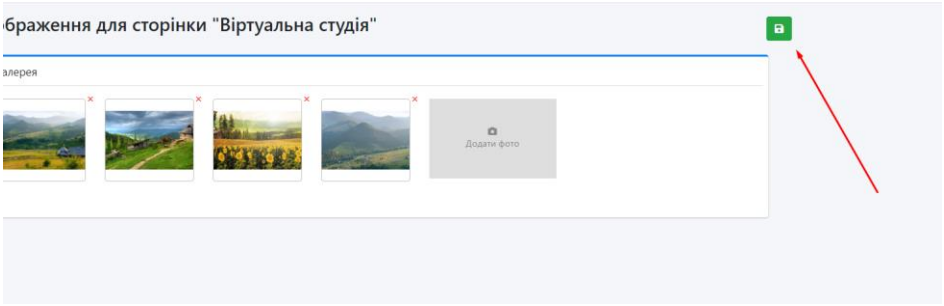
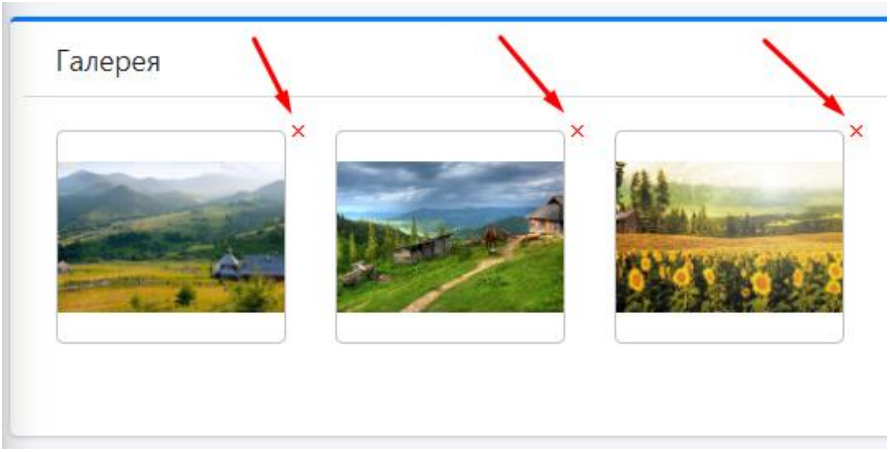


Рисунок 3.15 – Кнопка для збереження зображення

Після натиснення на цю кнопку відправиться запит, який оброблятиме метод store(). В цілях безпеки, спочатку потрібно перевірити чи існує на той момент ще сторінка, до якої повинне прив'язатися зображення, так як бувають випадки, коли якийсь інший адміністратор підсистеми видалив дану сторінку.

Далі, так як бібліотека Croppie JS перетворює завантажений файл у рядок типу base64, ми повинні отримати масив даних із такими рядками з вхідних даних форми. Модель Page має метод addMediaFromBase64(), який приймає в себе аргумент – рядок base64. Також після виклику цього методу потрібно викликати метод toMediaCollection() та передати в тіло функції назву медіа колекції, яку раніше зареєстрували в моделі Page, у нашому випадку – images. Тепер через цикл необхідно перебрати усі значення масиву даних та на кожен з рядків викликати метод addMediaFromBase64(). Зображення збережуться до папки /storage/app/public та до таблиці media бази даних додадуться записи про ці файли, які прив'язані до сторінки віртуальної студії. Якщо ніяких помилок не відбулося, користувача перенаправить на сторінку галереї.

Також для користувача передбачений функціонал видалення зображень. Біля кожного фото у правому верхньому куті є знак для видалення, як це показано на рисунку 3.16.



Розділ 3.16 – Знаки видалення для завантажених зображень

Натиснувши на цей знак, користувачу відобразиться модальне вікно для підтвердження видалення фото (рисунок 3.17).

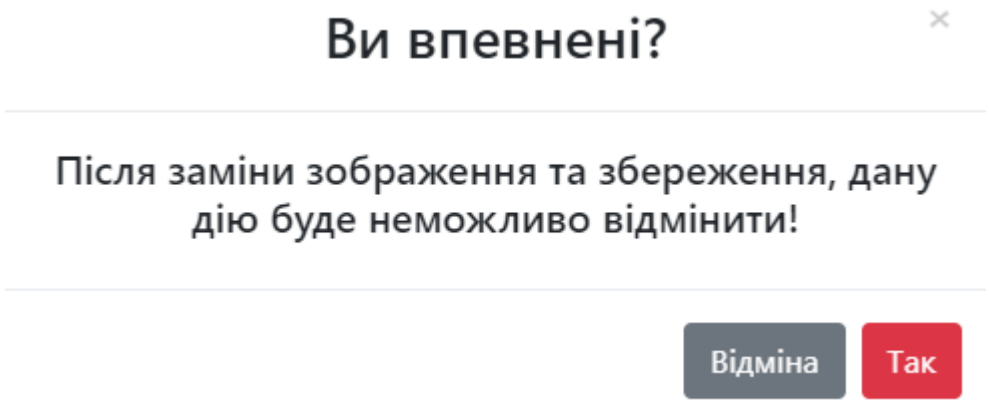


Рисунок 3.17 – Модальне вікно для підтвердження видалення або заміни зображення

Однак після підтвердження фото ще не буде видалене, поки користувач не натисне кнопку збереження. Це зроблено для зручності користувача, так як, можливо користувачу потрібно буде видалити не одне фото, або, наприклад, видалити фото та замінити його. Функціонал видалення також обробляється методом store(). Із запиту ми отримуємо масив ідентифікаторів для медіа файлів, які потрібно видалити. Тоді за допомогою реляційного зв’язку між таблицями

pages і *media*, можливо видалити записи з бази даних. Коли записи з бази даних буде видалено, бібліотека Laravel-medialibrary видалить файли з диску проекта.

У додатку Д приведено код, який демонструє функціонал збереження та видалення медіа зображень.

3.6 Розробка модуля зберігання відео-файлів

Фінальним етапом розробки підсистеми зберігання мультимедійного контенту є програмування модуля для керування відео. Цей компонент за візуальним виглядом буде дуже схожий на компонент галереї зображень, так як на ньому також потрібно вивести список мініатюр (перших кадрів відео) з можливістю їх видалення. Також буде присутній компонент для додавання відео.

Для створення цього компоненту використано бібліотеку FilePond [13], яка забезпечує динамічне завантаження зображень без перезавантаження сторінки, можливість перетягнути одне або декілька відео у завантажувач файлів, перевірку на тип файлів, ліміт по розміру та кількості файлів.

На рисунку 3.18 зображено дизайн сторінки для керування відео контентом.

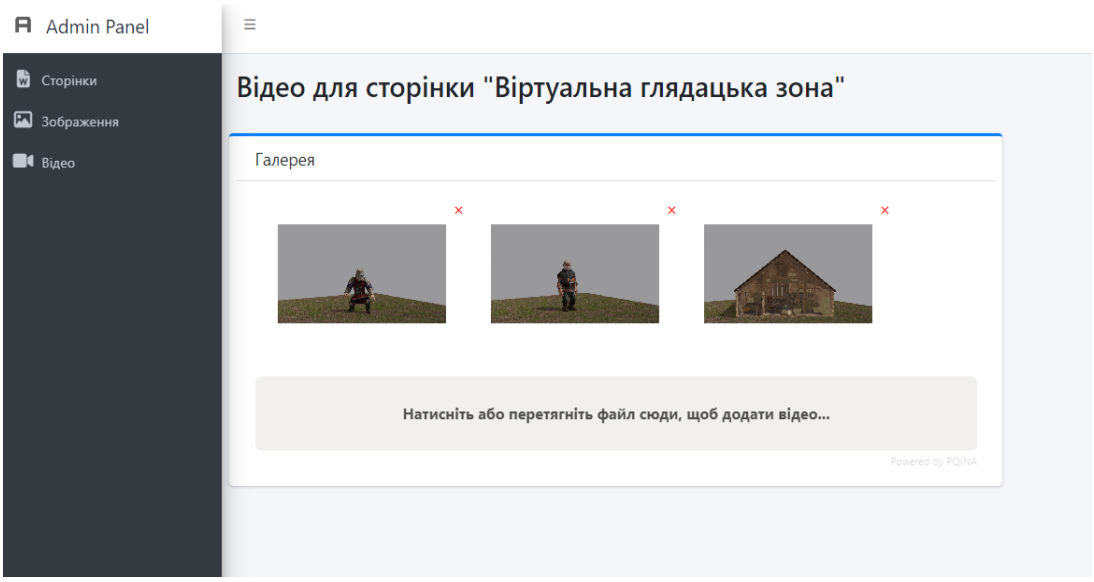


Рисунок 3.18 – Дизайн сторінки модуля керування відео контентом

Для того, щоб у базі даних розрізняти файли, прив’язані до сторінок, потрібно у методі registerMediaCollections() моделі Page зареєструвати ще одну медіа колекцію – videos. Реалізувати збереження відео буде складніше, ніж той самий модуль галереї зображень, так як завантаження та видалення відбуватиметься динамічно. Для цього створимо контролер VideoController та зареєструємо новий шлях маршрутизації:

Route::resource('videos', VideoController::class)–>only('index', 'show', 'store', 'destroy');

У контролері буде передбачено 4 методи:

– index: рендеринг вигляду сторінки для керування відео контентом;

– show: повертає JSON об’єкт із списком усіх записів відео сторінки;

– store: метод для валідації та динамічного збереження відео;

– destroy: метод для динамічного видалення, який повертає список об’єктів відео.

Збереження відео відбуватиметься схожим чином, як збереження фотографій, за винятком того, що за один запит зберігатиметься лише одне відео, яке буде додане з використання методу addMediaFromRequest(). Більше ніж одне відео додавати не варто, так як запит буде довше відпрацьовувати і дані будуть більше займати оперативної пам’яті, а якщо відео будуть великого розміру та в великій кількості, один запит може не завершитися до кінця, так як по часу виконання перевищить ліміт браузера.

Функціонал видалення відпрацьовує так само, як і для галереї зображень, але так як процес видалення відео є динамічним, на сторінці відсутня кнопка збереження. Після натиснення на кнопку підтвердження у модальному вікні відправляється запит на видалення.

У додатку Е показано реалізацію контролера для отримання, зберігання та видалення відео контенту.

ВИСНОВКИ

В кваліфікаційній роботі було розроблено підсистему зберігання мультимедійного контенту.

У ході написання кваліфікаційної роботи вирішено такі задачі:

Проведено аналіз найпопулярніших платформ у галузі мультимедії та з’ясовано головні аспекти, якими керується користувач при виборі веб-платформи.

Проаналізовано та досліджено увесь інструментарій платформи GitHub для використання цього сервісу в якості платформи для спільної розробки програмного забезпечення.

Досліджено такі інструменти розробки, як: мови програмування, середовище для керування контейнерами Docker, фреймворки мови PHP, текстові редактори (IDE). На основі проведених досліджень технологій було вибрано найкращий функціональний спектр інструментів для розробки функціональної підсистеми.

Здійснено загальний огляд роботи фреймворку Laravel.

Досліджено інструменти для роботи з базами даних, обробку та валідацію запитів, локалізацію, структуру виглядів шаблону Blade у фреймворці Laravel.

Розглянуто архітектурний шаблон програмування MVC (Model View Controller), на основі якого реалізовано фреймворк Laravel.

Спроектовано базу даних для підсистеми зберігання мультимедійного контенту.

Реалізовано функціонал авторизації та виходу з облікового запису користувача.

Розроблено модуль для керування сторінками платформи та їх текстовими даними.

Створено функціонал для роботи з графічними зображеннями та відео контентом.

Розроблену підсистему збереження мультимедійного контенту можна використовувати у галузі веб-розробки. Реалізована модульність цієї системи є простою для інтегрування цього сервісу у будь-який додаток.

Мета роботи, яка була поставлена на початку, була повністю досягнута.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Що таке Docker і навіщо він? URL: https://qagroup.com.ua/publications/shcho-take-docker-i-navishcho-vin/ (дата звернення: 01.05.2023).

2. Laradock: веб-сайт. URL: https://laradock.io/ (дата звернення: 01.05.2023).

3. GitHub: веб-сайт. URL: https://github.com/ (дата звернення: 14. 04.2023).

4. Request Lifecycle – Laravel – The PHP Framework For Web Artisans: веб-сайт. URL: https://laravel.com/docs/8.x/lifecycle (дата звернення: 16.04.2023)

5. HTTP Requests – Laravel – The PHP Framework For Web Artisans: веб-сайт. URL: https://laravel.com/docs/8.x/requests (дата звернення: 18.04.2023).

6. M. Stauffer. Laravel: Up & Running: A Framework for Building Modern PHP Apps. O'Reilly Media, 2017, с. 82-206.

7. E. Gamma, R. Helm, R. Johnson, J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software, 1994, с. 5.

8. Blade Templates – Laravel – The PHP Framework For Web Artisans: веб-сайт. URL: https://laravel.com/docs/8.x/blade (дата звернення: 20.04.2023).

9. Introduction | laravel-medialibrary | Spatie: веб-сайт. URL: https://spatie.be/docs/laravel-medialibrary/v10/introduction (дата звернення: 27.04.2023).

10. Bootstrap · The most popular HTML, CSS, and JS library in the world.: веб-сайт. URL: https://getbootstrap.com/ (дата звернення: 08.05.2023).

11. B. Joshi. Beginning SOLID Principles and Design Patterns for ASP.NET Developers, 2016, с. 45.

12. Croppie – a simple javascript image cropper – Foliotek: веб-сайт. URL: https://foliotek.github.io/Croppie/ (дата звернення: 08.05.2023).

13. Easy File Uploading With JavaScript | FilePond: веб-сайт. URL: https://pqina.nl/filepond/ (дата звернення: 08.05.2023).

ДОДАТКИ

Додаток А

Список правил валідації у фреймворку Laravel

required: вимагає, щоб поле було заповненим

email: перевіряє, що значення поля є дійсною електронною адресою

numeric: перевіряє, що значення поля є числовим

integer: перевіряє, що значення поля є цілим числом

digits: перевіряє, що значення поля складається з цифр та має певну кількість цифр

min: вимагає, щоб значення поля було більше або рівним певному значенню

max: вимагає, щоб значення поля було менше або рівним певному значенню

size: вимагає, щоб розмір файлу (для файлових полів) був меншим або рівним певному значенню

in: перевіряє, що значення поля знаходиться в масиві допустимих значень

not_in: перевіряє, що значення поля не знаходиться в масиві допустимих значень

regex: перевіряє, що значення поля відповідає певному регулярному виразу

date: перевіряє, що значення поля є коректною датою

date_format: перевіряє, що значення поля відповідає певному формату дати

after: перевіряє, що значення поля є датою, що починається після певної дати

before: перевіряє, що значення поля є датою, що закінчується перед певною датою

confirmed: перевіряє, що значення поля співпадає з підтвердженим значенням (наприклад, для підтвердження паролю)

accepted: вимагає, щоб значення поля було true, 1, «1», «yes», або «on» (для прапорців)

active_url: перевіряє, що значення поля є дійсним URL

alpha: перевіряє, що значення поля складається лише з букв (a-z)

alpha_dash: перевіряє, що значення поля складається лише з букв, цифр та знаків дефісу та підкреслення

alpha_num: перевіряє, що значення поля складається лише з букв та цифр

array: перевіряє, що значення поля є масивом

before_or_equal: перевіряє, що значення поля є датою, що починається до або дорівнює певній даті

boolean: перевіряє, що значення поля є булевим типом (true або false)

date_equals: перевіряє, що значення поля дорівнює певній даті

different: перевіряє, що значення поля відрізняється від значення іншого поля

dimensions: перевіряє, що розміри зображення (для файлових полів) відповідають певним вимогам

distinct: перевіряє, що значення поля унікальні в межах масиву або колекції

file: перевіряє, що значення поля є файлом

image: перевіряє, що значення поля є зображенням

in_array: перевіряє, що значення поля знаходиться у масиві іншого поля

json: перевіряє, що значення поля є коректним JSON рядком

mimes: перевіряє, що файл має певний тип MIME

nullable: дозволяє полю мати значення null

present: перевіряє, що значення поля присутнє в запиті

required_if: вимагає, щоб поле було заповненим, якщо значення іншого поля дорівнює певному значенню

required_unless: вимагає, щоб поле було заповненим, якщо значення іншого поля не дорівнює певному значенню

required_with: вимагає, щоб поле було заповненим, якщо певні інші поля присутні в запиті

required_with_all: вимагає, щоб поле було заповненим, якщо всі певні інші поля присутні в запиті

required_without: вимагає, щоб поле було заповненим, якщо певні інші поля відсутні в запиті

required_without_all: вимагає, щоб поле було заповненим, якщо всі певні інші поля відсутні в запиті

timezone: перевіряє, що значення поля є коректним часовим поясом

url: перевіряє, що значення поля є коректним URL.

Додаток Б

Логіка сторінки авторизації користувача

```
<?php
namespace App\Http\Controllers\Admin\Auth;

use App\Http\Controllers\Controller;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;

class LoginController extends Controller
{
    public function index()
    {
        return view('admin.pages.auth.login');
    }

    public function store(Request $request): \Illuminate\Http\RedirectResponse
    {
        $validated = $request->validate([
            'email' => 'required|email',
            'password' => 'required|string',
        ]);

        if (Auth::guard('admin')->attempt(['email' => $validated['email'], 'password' => $validated['password']], true)) {
            $request->session()->regenerate();
            return redirect()->intended(route('admin.index'));
        }

        return redirect()->back()->withInput($request->only('login'));
    }

    public function logout(Request $request)
    {
        Auth::guard('admin')->logout();
        $request->session()->invalidate();
        $request->session()->regenerateToken();
        return redirect(route('admin.login.page'));
    }
```

```
      }


<!DOCTYPE html>
<html lang="ru">
    <head>
        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <title>Admin Panel | Log in</title>
        <meta content="width=device-width, initial-scale=1, maximum-scale=1,
user-scalable=no" name="viewport">
        <link rel="stylesheet" href="{{ mix('/css/index.css', 'admin') }}">
        <script defer
src="https://use.fontawesome.com/releases/v5.15.4/js/all.js" integrity="sha384-
rOA1PnstxnOBLzCLMcre8ybwbTmemjzdNlILg8O7z1lUkLXozs4DHonlDtnE7fpc"
crossorigin="anonymous"></script>
        <style>@media print {#ghostery-purple-box {display:none
!important}}</style>
</head>
    <body class="hold-transition login-page justify-content-start"
style="position: relative; background: url(../../admin/img/sign.bg.jpg) no-
repeat center center, #272e35; background-size: cover; background-attachment:
fixed;">
        <x-admin.message></x-admin.message>
        <div class="login-box">
            <div class="login-logo"><span
style="color:white;"><b>Admin</b>Panel</span></div>
            <div class="card"><div class="card-body login-card-body">
                    <p class="login-box-msg">Увійдіть, щоб продовжити роботу</p>
                    <form action="{{ route('admin.login') }}" method="post"
autocomplete="off">{{ csrf_field() }}<div class="input-group mb-3">
                            <input type="text" name="email" tabindex="1"
autofocus
                                class="form-control" placeholder="Email
користувача"
                                required value="{{ old('email') }}">
                            <div class="input-group-append"><div class="input-
group-text">
                              <span class="fas fa-
user"></span></div></div></div>
                        <div class="input-group mb-3">
                            <input type="password" name="password" tabindex="2"
                                class="form-control" placeholder="Пароль"
required>
                            <div class="input-group-append"><div class="input-
group-text"><span class="fas fa-lock"></span></div></div></div>
                        <div class="row">
                            <div class="col-4 ml-auto">
                                <button
                                    type="submit"
                                    name="submit"
                                    tabindex="3"
                                    class="btn btn-primary btn-block">
                                    Увійти
                                </button>
                            </div>
                        </div>
                    </form>
                </div> </div></div></body></html>
```

Додаток В

Код загального макету адміністративної панелі та її компонентів

```
<nav class="main-header navbar navbar-expand navbar-white navbar-light">
    <ul class="navbar-nav"><li class="nav-item">
            <a class="nav-link js-push-menu-butt" data-widget="pushmenu"
href="javascript:void(0);" role="button"><i class="fas fa-bars"></i></a>
        </li>
    </ul>
    <ul class="navbar-nav ml-auto">
        <li class="nav-item dropdown user-menu">
            <a href="#" class="nav-link dropdown-toggle js--custom-
dropdown">
                <img src="{{ asset('/admin/img/avatar.png') }}"
class="user-image img-circle elevation-2" alt="User Image">
                <span class="d-none d-md-inline">{{ Auth::user()->name
}}</span>
            </a>
            <ul class="dropdown-menu dropdown-menu-lg ">
                <li class="user-header">
                    <img src="{{ asset('/admin/img/avatar.png') }}"
class="img-circle" alt="User Image">
                    <p>{{ Auth::user()->name }}</p>
                </li>
                <li class="user-footer">
                    <div class="float-right">
                        <a href="{{ route('admin.logout') }}" class="btn
btn-danger btn-flat">Вийти</a>
                    </div></li></ul>
        </li>
    </ul>
</nav>
<aside class="main-sidebar sidebar-dark-primary elevation-4">
    <a href="{{ config('app.url') }}" class="brand-link bg-white">
        <img src="/admin/img/logo.webp" alt="logo" class="brand-image"
style="opacity: .8">
        <span class="brand-text font-weight-light"><b>Admin
Panel</b></span>
    </a>
    <div class="sidebar">
        <nav class="mt-2">
```

```
                <ul class="nav nav-pills nav-sidebar nav-child-indent flex-column" data-widget="treeview">
                    <x-admin.sidebar-nav-link :href="route('admin.pages.index')" active="pages.*" icon-class="fas fa-file-word">
                        Сторінки
                    </x-admin.sidebar-nav-link>
                    <x-admin.sidebar-nav-link :href="route('admin.images.index')" active="images.*" icon-class="fa-regular fa-image">
                        Зображення
                    </x-admin.sidebar-nav-link>
                    <x-admin.sidebar-nav-link :href="route('admin.videos.index')" active="videos.*" icon-class="fa-solid fa-video">
                        Відео
                    </x-admin.sidebar-nav-link>
                </ul>
            </nav>
        </div>
    </aside>
<footer class="main-footer">
    <strong>
            Copyright © {{ date("Y") }}.
    </strong>
    All rights reserved.
</footer>
<!DOCTYPE html>
    <html lang="uk">
    <head>
        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta content="width=device-width, initial-scale=1, maximum-scale=1, user-scalable=no" name="viewport">
        <link rel="icon" type="image/svg+xml" href="/admin/img/logo.webp">
        <link rel="alternate icon" href="/admin/img/logo.webp">
        <title>{{ $title }}</title>
        <meta name="description" content="{{ $description }}">
        <meta name="csrf-token" content="{{ csrf_token() }}">
        <link rel="preload" as="style" href="{{ mix('/css/index.css', 'admin') }}" crossorigin>
```

```
        <link  rel="stylesheet"  href="{{  mix('/css/index.css',  'admin')  }}"
crossorigin>
        <link                                                    rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.3.0/css/all.min.css"
integrity="sha512-
SzlrxWUlpfuzQ+pcUCosxcglQRNAq/DZjVsC0lE40xsADsfeQoEypE+enwcOiGjk/bSuGGKHEyjSoQ1z
VisanQ==" crossorigin="anonymous" referrerpolicy="no-referrer" />
        @stack('css')
    </head>
    <body class="sidebar-mini layout-fixed">
        <div class="wrapper">
            @include('admin.main.header')
            @include('admin.main.sidebar')
            {{ $slot }}
            @include('admin.main.footer')
        </div>
        <script src="{{ mix('/js/vendor.js', 'admin') }}"></script>
        @if($showIndexScripts)
            <script src="{{ mix('/js/index.js', 'admin') }}"></script>
        @endif
        <script src="/admin/plugins/tinymce/tinymce.min.js"></script>
        <script type="text/javascript">
            $('ul.nav-sidebar a.nav-link.active')
                .closest('ul.nav-
treeview').css('display','block').closest('li.nav-item').addClass('menu-
open').find('a.nav-link:first').addClass('active');
        </script>
        @stack('scripts')
    </body>
    </html>
```

Додаток Г

Код для реалізації модуля редагування текстових даних

```
<x-layouts.admin.layout>
      <x-slot name="title">{{ $title ?? '' }}</x-slot>
      <x-slot name="description">{{ $title ?? '' }}</x-slot>
      <div class="content-wrapper"><x-admin.message></x-admin.message>
      <section class="content-header">
      <div class="container-fluid d-sm-flex justify-content-between align-items-
center block-title"><div class="d-inline">
      <h1 class="mb-3">Редагування сторінки `{{ $item->name }}`
      </h1><ol class="breadcrumb mb-2 mb-sm-0">
      <li class="breadcrumb-item">
      <a href="{{ route('admin.pages.index') }}">Сторінки</a></li>
      <li class="breadcrumb-item active">{{ $item->name }}</li></ol></div>
      <button type="submit" form="form" class="btn btn-success saved">
      <i class="fa fa-save"></i></button>
      <a  href="{{  $item->getUrl()  }}"  target="_blank"  class="btn  btn-info
position-fixed" style="right: 17%;"><i class="fa fa-eye"></i></a></div>
      </section><section class="content container-fluid">
      <form  id="form"  action="{{  route('admin.pages.update',  [$item->id])  }}"
method="POST" enctype="multipart/form-data" autocomplete="off">
      @csrf @method('PATCH')
      <div class="row max-width"> <div class="col-md-12 col-lg-10">
      <div class="row"> <div class=" col-sm-12 col-md-12 col-lg-12">
      <div class="card card-outline card-primary">
      <div class="card-header"><h3 class="card-title">Сторінка</h3></div>
      <div class="card-body row"><div class="form-group col-12 mb-0">
      <label>Назва<span class="required-star"></span></label>
      @foreach (config('app.locales') as $lang)
      @php set_locale($lang); @endphp
      <div class="input-group mb-3">
      <span class="input-group-prepend">
      <div class="input-group-text">{{ $lang }}</div></span>
      <input type="text" name="name_{{ $lang }}"
      value='{{ old("name_$lang", $item->name) }}'
      class="form-control" required></div>@endforeach
      @php set_locale(config('app.locale')); @endphp </div>
      @if($item->url !== 'index')
      <div class="form-group mb-0 col-12 text-right">
      <div class="custom-control custom-checkbox d-inline-block">
```

```
      <input type="hidden" name="active" value="0">
      <input  type="checkbox"  id="viewed"  name="active"  class="custom-control-
input" @if($item->active) checked @endif value="1">
      <label for="viewed" class="custom-control-label">Відображати</label>
      </div></div>
      @endif </div></div></div></div>
      <div class="row">
      <div class="col-sm-12 col-xl-12">
      <div class="card card-outline card-primary">
      <div class="card-header cursor-pointer" data-card-widget="collapse">
      <h3 class="card-title">Мета-теги</h3>
      <div class="card-tools">
      <button    type="button"    class="btn    btn-tool    btn-sm"    data-card-
widget="collapse" data-toggle="tooltip" title="Collapse">
      <i class="fas fa-plus"></i></button>
      </div></div>
      <div class="card-body">
      <div class="row">
      @foreach(config('app.locales') as $lang)
      @php set_locale($lang); @endphp
      <div class="col-12 col-md-6">
      <div class="form-group">
      <label class="font-weight-normal">h1 ({{ $lang }})</label>
      <div class="input-group">
      <div class="input-group-prepend">
          <div class="input-group-text">{{ $lang }}</div>
      </div>
      <input type="text" name="h1_{{ $lang }}" data-lang="{{ $lang }}"
          value='{{ old("h1_$lang", $item->h1 ?? '') }}'
          class="form-control font-weight-bold"
          placeholder="{{ $lang }}...">
      </div></div>
      <div class="form-group">
      <label class="font-weight-normal">h2 ({{ $lang }})</label>
      <div class="input-group">
      <div class="input-group-prepend">
          <div class="input-group-text">{{ $lang }}</div>
      </div>
      <input type="text" name="h2_{{ $lang }}"
          value='{{ old("h2_$lang", $item->h2 ?? '') }}'
          data-lang="{{ $lang }}"
          class="form-control font-weight-bold"
```

```
          placeholder="{{ $lang }}...">
      </div></div>
      <div class="form-group">
      <label class="font-weight-normal">Title ({{ $lang }})</label>
      <textarea   class="form-control   font-weight-bold"name="title_{{   $lang
}}"rows="2" placeholder="{{ $lang }}...">{{ old("title_$lang", $item->title ??
'') }}</textarea></div><div class="form-group">
      <label class="font-weight-normal">Description ({{ $lang }})</label>
      <textarea class="form-control font-weight-bold" name="description_{{ $lang
}}"
      rows="5"
      placeholder="{{ $lang }}...">{{ old("description_$lang", $item->description
?? '') }}</textarea></div></div>@endforeach
      @php set_locale(config('app.locale')); @endphp
</div></div></div></div></div>
      <div class="row">
      <div class="col-sm-12 col-xl-12">
      @foreach(config('app.locales') as $lang)
      @php set_locale($lang); @endphp
      <div class="card card-outline card-primary card-custom-tabs">
      <div class="card-header cursor-pointer" data-card-widget="collapse">
      <h3 class="card-title">Текст {{ $lang }}</h3>
      <div class="card-tools">
      <button    type="button"    class="btn    btn-tool    btn-sm"    data-card-
widget="collapse" data-toggle="tooltip" title="Collapse">
      <i class="fas fa-minus"></i></button>
      </div></div>
      <div class="card-body">
      <div class="overflow-auto col-12 p-0">
      <div class="fade active show"
      aria-labelledby="{{ $lang }}">
      <textarea class="tinymce" name="text_{{ $lang }}"
      placeholder="{{    $lang    }}...">{{    old("text_$lang",    $item->text)
}}</textarea>
      </div></div></div></div>@endforeach</div></div>
      <div class="row mt-4">
      <div class="col-sm-12 col-md-12 col-lg-12">
      <button type="submit" class="btn btn-success float-right btn-styles">
      <i class="fa fa-save"></i>Зберегти</button>
      </div></div></div></div></form></section></div>
      </x-layouts.admin.layout>
```

```
<?php

namespace App\Actions\Admin\Page;

use App\Models\Page;
use App\Models\PageContent;

class StorePageData
{
    public function handle(Page $page, array $data) {
        if (empty($data))
           throw new \RuntimeException('Дані про сторінку відсутні!');
        if  (isset($data['active']))  $page->update(['active'  =>  (int)
$data['active']]);
        foreach (config('app.locales') as $locale) {
            $content = PageContent::firstOrNew( ['page_id' => $page->id,
'lang' => $locale]); $contentData = [];
            foreach ($page->getLocaleFields() as $field) {
                $contentData[$field]    =    $data["{$field}_{$locale}"];
}
            $content->fill($contentData);
            if (! $content->save()) {
                throw    new    \RuntimeException('Помилка    збереження
контенту!');
            }
}
return true;
  }
}
```

Додаток Д

Функціонал збереження та видалення зображень

```
class DeleteImages
{
    public function handle(Page $page, array $removeImageIds)
    {
        if (! empty($removeImageIds)) {
            foreach ($removeImageIds as $id => $deleted) {
                if ($deleted) {
                    $page->media->where('id', $id)->first()->delete();
                }
            }
        }
    }
}

class StoreImages
{
    public function handle(Page $page, array $images)
    {
        if (! empty($images)) {
            foreach ($images as $image) {
                $page->addMediaFromBase64($image)
                    ->toMediaCollection('images');
            }
        }
    }
}


class ImageController extends Controller
{
    protected const URL = 'virtual-studio';

    public function index()
    {
        $page = Page::query()->where('url', self::URL)->firstOrFail();

        return view('admin.pages.images.index', compact('page'));
    }
```

```
        public function store(Request $request)
        {
            $page = Page::where('url', self::URL)->first();
            if (! $page) {
                return redirect(route('admin.index'))
                   ->withErrors('Сторінку не знайдено!');
            }
            try {
                (new  StoreImages)->handle($page,  $request->get('text-images',
[]));
                (new           DeleteImages)->handle($page,           $request-
>get('delete_gallery', []));
                $request->session()->flash('message', 'Оновлено успішно!');
            } catch (\Throwable $exception) {
                return back()
                    ->withInput()
                    ->withErrors($exception->getMessage());
            }
      return redirect(route('admin.images.index'));
      }
      }
```

Додаток Е

Реалізація модуля керування відео контентом

```
class StoreVideo
{
    public function handle($file, Page $page): Media
    {
        if (strpos($file->getClientMimeType(), 'video') === false) {
            throw new \Exception('Доданий файл не є формату відео', 422);
        }
        $media=$page->addMediaFromRequest('video')
            ->toMediaCollection('videos');
        if (! $media->id) {
            throw new \Exception('Помилка збереження відео до бд', 500);
        }
        return $media;
    }
}
```

```
<?php

namespace App\Http\Controllers\Admin;

use App\Actions\Admin\Video\StoreVideo;
use App\Http\Controllers\Controller;
use App\Models\Page;
use Illuminate\Http\Request;
use Spatie\MediaLibrary\MediaCollections\Models\Media;

class VideoController extends Controller
{
    protected const URL = 'virtual-viewers-zone';

    public function index()
    {
        $page = Page::query()->where('url', self::URL)->first();
        if (! $page) {
            return redirect(route('admin.index'))->withErrors('Сторінки не
існує!');
        }
        return view('admin.pages.videos.index', compact('page'));
```

```
        }

        public function store(Request $request)
        {
            if (! $request->hasFile('video')) {
                return response()->json(['error' => 'Файл відсутній.'], 400);
            }
            $request->validate(['video' => 'required|file']);
            $page = $this->getPage();
            try {
                $media=app(StoreVideo::class)->handle($request->file('video'),
$page);
            } catch (\Throwable $exception) {
                return response()->json(['error' => $exception->getMessage()],
$exception->getCode());
            }
            return $media->file_name;
        }

    public function show(): \Illuminate\Http\JsonResponse
        {
            $videos = $this->getPage()->getMedia('videos')->sortBy('id');

          return response()->json(['videos' => $videos->toArray()]);
        }

        public function destroy(Media $video)
        {
            $video->delete();

            $videos = $this->getPage()->getMedia('videos')->sortBy('id');
            return response()->json(['videos' => $videos->toArray()]);
        }

        protected function getPage()
        {
            $page = Page::query()->where('url', self::URL)->first();
            if (! $page) {
                return response()->json(['error' => 'Сторінки не існує, будь
ласка перезавантажте сторінку'], 404);
            }
            return $page;}}
```