МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Київський національний університет будівництва і архітектури

РОЗРОБКА ВЕБДОДАТКІВ НА ОСНОВІ ФРЕЙМВОРКУ LARAVEL

Методичні вказівки
до виконання лабораторних робіт
для здобувачів першого (бакалаврського) рівня
вищої освіти спеціальності F3 «Комп'ютерні науки»

Київ 2026

УДК: 004.42:004.738.5
Р64

Укладачі: Ю. О. Науменко, канд. техн. наук
А. Г. Хаддад, керівник ФОП

Рецензент І. А. Ачкасов, д-р техн. наук, професор

Відповідальна за випуск Т. А. Гончаренко, д-р техн. наук, професор

Затверджено на засіданні кафедри інформаційних технологій, протокол № 9 від 6 травня 2026 року.

В авторській редакції.

Розробка вебдодатків на основі фреймворку Laravel [електронний
Р64 ресурс] : методичні вказівки до виконання лабораторних робіт/ уклад.: Ю.О. Науменко, А. Г. Хаддад. – Київ: КНУБА, 2026. – 25 с.

Містять рекомендації й типові завдання, що відображають рівень засвоєння курсу, зміст, порядок оформлення і вказівки до виконання окремих лабораторних робіт.

Призначено для здобувачів першого (бакалаврського) рівня вищої освіти спеціальності F3 «Комп'ютерні науки» .

ЗМІСТ

Загальні положення

Дисципліна «Розробка вебдодатків на основі фреймворку Laravel» є важливою складовою підготовки фахівців у галузі комп'ютерних технологій, оскільки вона спрямована на формування практичних навичок створення сучасних, масштабованих та безпечних вебзастосунків. Актуальність курсу зумовлена домінуючою роллю екосистеми PHP та фреймворку Laravel у професійній веброзробці, що дає змогу автоматизувати бізнес-процеси та впроваджувати складні архітектурні рішення.

Методичні вказівки охоплюють повний цикл розробки: від налаштування професійного середовища (Laravel Herd, Composer, Node.js) та вивчення структури проєкту до розгортання готового продукту на сервері (деплой) з використанням принципів CI/CD. Мета вказівок – закріпити теоретичні знання про патерн MVC (Model-View-Controller) та сформувати стійкі навички роботи з маршрутизацією, шаблонізацією Blade, обробкою форм та взаємодією з базами даних через ORM Eloquent.

Окремий акцент у навчальному процесі зроблено на забезпеченні безпеки та розмежуванні доступу. Здобувачі вивчають механізми автентифікації та авторизації, використання Middleware, а також захист від типових вебзагроз (наприклад, через впровадження CSRF-захисту та валідації даних). Це закладає фундамент для побудови інформаційних систем, орієнтованих на реальні потреби користувачів та організацій.

Після виконання лабораторних робіт здобувач повинен уміти:

✓ налаштовувати робоче середовище та використовувати інструментарій Artisan для автоматизації рутинних завдань;

✓ проєктувати гнучку систему маршрутизації та створювати динамічні інтерфейси за допомогою компонентів Blade;

✓ реалізовувати надійну логіку валідації запитів та обробки користувацького вводу;

✓ моделювати структури баз даних, використовуючи міграції, сидери та складні зв'язки в Eloquent (hasMany, belongsTo);

✓ інтегрувати сучасні фронтенд-інструменти (наприклад, Vite) у процес розробки;

✓ виконувати деплой застосунку та налаштовувати процеси підтримки проєкту.

✓ впроваджувати механізми автентифікації, забезпечуючи реєстрацію, безпечний вхід та вихід користувачів із системи з використанням хешування паролів;

✓ застосовувати Middleware для гнучкого розмежування прав доступу та фільтрації HTTP-запитів на рівні окремих маршрутів застосунку;

✓ реалізовувати розширений функціонал, зокрема завантаження медіафайлів на сервер, створення профілів користувачів та інтеграцію соціальної автентифікації;

✓ оптимізувати продуктивність вебзастосунку через впровадження механізмів кешування та створення індексів у базах даних для підвищення швидкодії системи.

Для полегшення засвоєння матеріалу наводяться інструкції з конфігурації інструментів та практичні вказівки до реалізації проєктів.

Завершивши вивчення дисципліни, здобувач повинен уміти самостійно: проєктувати архітектуру сучасних вебзастосунків на основі патерну MVC; впроваджувати складну бізнес-логіку через контролери та Eloquent ORM; професійно моделювати структури баз даних за допомогою міграцій та сидерів. Окрім цього, здобувач навчиться розробляти динамічні та інтерактивні інтерфейси на базі Blade-шаблонів, реалізовувати надійні системи автентифікації та авторизації (Gates, Policies), а також забезпечувати безпечну валідацію користувацького вводу та захист даних. Отримані компетенції дозволяють створювати масштабовані програмні рішення, орієнтовані на реальні потреби користувачів, та здійснювати їх розгортання на сервері з використанням сучасних принципів CI/CD.

Лабораторна робота №1
Тема: Налаштування середовища розробки та створення Laravel-проєкту

Мета: Ознайомитися з екосистемою фреймворку Laravel, навчитися налаштовувати професійне середовище розробки (Laravel Herd, PHP, Composer, Node.js), вивчити внутрішню структуру проєкту та опанувати базові команди інструментарію Artisan.

Короткі теоретичні відомості

Laravel: сучасний PHP-фреймворк, побудований на патерні MVC (Model-View-Controller), що призначений для створення масштабованих вебзастосунків із чистою архітектурою.

Composer та залежності: Основним інструментом управління екосистемою PHP є менеджер залежностей Composer. Він дає змогу автоматизувати встановлення фреймворку, бібліотек та сторонніх пакетів, забезпечуючи дотримання стандартів автозавантаження класів.

Artisan CLI: Laravel містить вбудований інтерфейс командного рядка Artisan, який дає змогу розробнику уникати дублювання коду та автоматизувати рутинні завдання, такі як створення компонентів, міграцій баз даних та керування конфігурацією застосунку.

Архітектура Front Controller: Застосунки на Laravel використовують патерн Front Controller, де єдиний вхідний скрипт (зазвичай index.php) обробляє всі вхідні HTTP-запити в циклі «запит-відповідь» та делегує їх відповідним частинам системи.

Конфігурація та безпека: Важливою частиною проєкту є файл .env, який містить налаштування локального середовища, включаючи ключі безпеки та параметри підключення до бази даних. Використання окремого конфігураційного файлу дає змогу гнучко адаптувати застосунок під різні сервери без зміни основного коду.

Стандарти та середовище: Професійна розробка передбачає дотримання стандартів PHP-FIG, зокрема PSR-1, який рекомендує відокремлювати декларації класів і функцій від логіки, що викликає побічні ефекти. Для локальної розробки використовуються спеціалізовані середовища, такі як Laravel Herd або PHP Web Server, що дають змогу імітувати роботу реального сервера.

Структура проєкту:

- /app/ : містить основний код застосунку (моделі, контролери, Middleware).
- /config/ : файли конфігурації всього застосунку.
- /routes/ : визначення маршрутів (наприклад, web.php для вебсторінок).
- /resources/ : Blade-шаблони, стилі та JS-файли.
- /database/ : міграції, фабрики та сидери для роботи з базами даних.
- /.env: файл налаштувань локального середовища (підключення до БД, ключі безпеки).

Порядок виконання:

Крок 1. Підготовка робочого середовища. Встановіть необхідне програмне забезпечення, яке включає Laravel Herd для керування версіями PHP та локальним сервером, менеджер пакетів Composer та платформу Node.js для роботи з фронтенд-інструментами.

Крок 2. Створення та ініціалізація проєкту. Відкрийте термінал та виконайте команду створення нового проєкту через Composer, після чого перейдіть у робочу директорію застосунку за допомогою команди cd.

Крок 3. Запуск та перевірка працездатності. Використовуйте інструментарій Artisan для запуску вбудованого сервера розробки та переконайтеся, що вітальна сторінка Laravel доступна у браузері за локальною адресою.

Крок 4. Дослідження архітектури застосунку. Ознайомтеся з призначенням ключових директорій, зокрема папки app для бізнес-логіки, resources для шаблонів та routes для визначення маршрутів застосунку.

Контрольні запитання (самоконтроль)

1. Яке головне призначення фреймворку Laravel у сучасній веброзробці та які переваги він надає порівняно з чистим PHP?
2. Опишіть роль менеджера залежностей Composer та поясніть, чому він є критично важливим для екосистеми Laravel?
3. Які функції виконує файл .env у структурі проєкту та чому параметри безпеки слід тримати саме в цьому файлі?
4. Назвіть основні команди Artisan, які використовуються для запуску сервера та перегляду списку доступних інструментів автоматизації?

Лабораторна робота №2
Тема: Маршрутизація та робота з Blade-шаблонами

Мета: Опанувати механізми побудови системи маршрутизації у Laravel, навчитися створювати гнучкі інтерфейси за допомогою двигуна шаблонізації Blade та реалізувати успадкування макетів для сторінок застосунку.

Короткі теоретичні відомості

Маршрутизація: Це механізм, який даєзмогу зіставляти URL-адреси вхідних HTTP-запитів із конкретною логікою застосунку або контролерами. Основні веб-маршрути визначаються у файлі routes/web.php, де можна використовувати різні методи, такі як GET для отримання сторінок та POST для обробки даних.

Blade: Це потужний двигун шаблонізації Laravel, який дає змогу змішувати HTML із короткими директивами. На відміну від звичайного PHP, Blade-файли мають розширення .blade.php і підтримують чистий синтаксис для виводу даних та керування логікою.

Успадкування шаблонів: Процес створення базового макета (layout), який містить спільні елементи, такі як шапка та підвал сайту. Дочірні сторінки розширюють цей макет, вставляючи унікальний контент у спеціально визначені зони.

Директиви Blade: Спеціальні команди, що починаються із символу @. Наприклад, @extends вказує на батьківський макет, @section визначає блок контенту, а @yield позначає місце у макеті, де цей контент буде відображено.

Передача даних: Механізм відправки змінних із маршруту до представлення (view). Це дає змогу робити сторінки динамічними, відображаючи інформацію, отриману з логіки застосунку.

Порядок виконання:

Крок 1. Створення базового макета застосунку. У директорії resources/views створіть файл layout.blade.php. Підготуйте стандартну HTML-структуру та додайте директиву @yield('content') у тіло сторінки, щоб визначити місце для динамічного вмісту.

Крок 2. Розробка дочірніх сторінок. Створіть файли welcome.blade.php та about.blade.php. У кожному файлі використайте директиву @extends для

підключення макета та @section('content') для написання унікального тексту кожної сторінки.

Крок 3. Оголошення маршрутів у системі. Відкрийте файл routes/web.php та створіть два маршрути для головної сторінки та сторінки "Про нас". Використовуйте функцію view() для повернення відповідних шаблонів.

Крок 4. Реалізація динамічного виводу даних. Модифікуйте один із маршрутів так, щоб він передавав масив із заголовком сторінки. У Blade-шаблоні виведіть це значення, використовуючи подвійні фігурні дужки.

Крок 5. Налаштування навігації. Додайте до базового макета навігаційне меню. Використовуйте іменовані маршрути та хелпер route() для створення коректних посилань між сторінками застосунку.

Контрольні запитання (самоконтроль)

1. Що таке маршрутизація у Laravel та в якому файлі зазвичай визначаються маршрути для вебінтерфейсу?
2. Які переваги надає використання Blade-шаблонів порівняно зі стандартним PHP-кодом у HTML-файлах?
3. Поясніть різницю між директивами @yield та @section у контексті створення макетів?
4. Яким чином можна передати змінну з файлу маршрутів до представлення (view)?
5. Для чого використовуються іменовані маршрути та як згенерувати посилання на них у шаблоні?

Лабораторна робота №3

Тема: Реєстрація та автентифікація користувачів. Валідація даних

Мета: Опанувати механізми створення систем захищеного входу до застосунку, навчитися реалізовувати реєстрацію та авторизацію користувачів, вивчити методи хешування паролів та принципи валідації вхідних даних для забезпечення безпеки системи.

Короткі теоретичні відомості

Автентифікація: Це процес перевірки особи користувача, який зазвичай полягає у порівнянні введених облікових даних із записами у базі даних. У Laravel цей процес тісно пов'язаний із сесіями, що дає змогу застосунку «пам'ятати» користувача після успішного входу.

Валідація: Процес перевірки вхідних даних на відповідність встановленим правилам, таким як обов'язковість заповнення, довжина рядка або унікальність електронної пошти. Це запобігає збереженню некоректної інформації та захищає від типових вебзагроз.

Хешування: Метод безпечного зберігання паролів, при якому пароль перетворюється на незворотний рядок символів. Laravel використовує надійні алгоритми (наприклад, Bcrypt), що робить неможливим відновлення реального пароля навіть у разі витоку даних із бази.

Blade-директиви доступу: Спеціальні команди @auth та @guest, які дозволяють динамічно змінювати інтерфейс залежно від статусу користувача. Директива @auth відображає контент лише авторизованим особам, тоді як @guest: лише неавторизованим відвідувачам.

Middleware: Проміжне програмне забезпечення, яке фільтрує HTTP-запити до застосунку. Наприклад, Middleware 'auth' перевіряє, чи увійшов користувач у систему, і якщо ні: перенаправляє його на сторінку логіну.

Порядок виконання:

Крок 1. Налаштування бази даних. Відкрийте файл .env та вкажіть параметри підключення до вашої бази даних. Після цього виконайте команду php artisan migrate, щоб створити стандартні таблиці для користувачів, які постачаються разом із Laravel.

Крок 2. Створення маршрутів для автентифікації. У файлі routes/web.php оголосіть маршрути для відображення форм реєстрації та

входу (GET-запити), а також маршрути для обробки надісланих даних і виходу із системи (POST-запити).

Крок 3. Розробка форм у Blade-шаблонах. Створіть представлення для реєстрації та логіну. Використовуйте директиву @csrf у кожній формі для захисту від міжсайтової підробки запитів та додайте вивід повідомлень про помилки валідації для кожного поля.

Крок 4. Реалізація логіки контролера. Створіть AuthController за допомогою Artisan. Напишіть методи для реєстрації, де здійснюється хешування пароля функцією Hash::make(), та для входу, де використовується метод Auth::attempt() для перевірки даних.

Крок 5. Захист інтерфейсу та маршрутів. Модифікуйте головний макет застосунку, використовуючи директиви @auth та @guest для відображення кнопки виходу або посилань на реєстрацію. Застосуйте Middleware до маршрутів, які потребують обов'язкової авторизації.

Контрольні запитання (самоконтроль)

1. У чому полягає принципова різниця між процесами автентифікації та авторизації у вебзастосунках?
2. Чому зберігання паролів у відкритому вигляді є критичною помилкою безпеки та як хешування вирішує цю проблему?
3. Яке призначення директиви @csrf у формах Laravel та від якого типу атак вона захищає?
4. Як працюють директиви @auth та @guest всередині Blade-шаблонів?
5. Поясніть роль Middleware у процесі обмеження доступу до певних сторінок сайту.

Лабораторна робота №4

Тема: Авторизація через Gates і Policies.
Використання @can та $this->authorize() для захисту CRUD-операцій

Мета: Опанувати механізми розмежування прав доступу в Laravel, навчитися створювати та застосовувати Gates для простих перевірок і Policies для захисту дій над моделями, а також реалізувати контроль відображення елементів інтерфейсу залежно від прав користувача.

Короткі теоретичні відомості

Авторизація: Це процес перевірки прав автентифікованого користувача на виконання певних дій із ресурсами застосунку. На відміну від автентифікації, яка встановлює особу, авторизація визначає, що саме цій особі дозволено робити.

Gates: Це замикання (closures), які зазвичай використовуються для простої логіки авторизації, не прив'язаної до конкретної моделі. Вони ідеально підходять для перевірки глобальних прав, наприклад, чи є користувач адміністратором системи.

Policies: Це класи, які організовують логіку авторизації навколо конкретної моделі або ресурсу. Кожен метод класу політики відповідає певній дії, такій як перегляд, створення, оновлення або видалення запису (CRUD-операції).

Метод authorize(): Спеціальний метод контролера, який дає змогу швидко перевірити права доступу. Якщо користувач не має відповідних прав, Laravel автоматично перерве виконання запиту та поверне HTTP-відповідь із кодом 403 Forbidden.

Директива @can: Потужний інструмент шаблонізатора Blade, який дає змогу приховати або відобразити фрагменти HTML-коду залежно від дозволів користувача. Це забезпечує відповідність інтерфейсу реальним можливостям доступу.

Порядок виконання:

Крок 1. Генерація класу політики. Використовуйте вбудований інструментарій Artisan для створення нової політики, яка буде асоційована з вашою моделлю даних для подальшого захисту ресурсів.

Крок 2. Опис логіки доступу. Відкрийте створений файл політики та визначте умови для методів оновлення та видалення записів, наприклад: дозвольте ці дії лише автору контенту.

Крок 3. Впровадження захисту в контролері. У методах контролера, які відповідають за зміну даних, додайте виклик методу authorize(), щоб гарантувати, що запит виконується уповноваженою особою.

Крок 4. Створення глобального Gate. У сервіс-провайдері застосунку визначте правило для перевірки ролі адміністратора, яке можна буде використовувати для обмеження доступу до панелі керування.

Крок 5. Налаштування динамічного інтерфейсу. Модифікуйте ваші Blade-шаблони, обгорнувши кнопки редагування та видалення в директиви @can, щоб забезпечити коректне відображення елементів.

Крок 6. Тестування обмежень. Спробуйте виконати дії під різними обліковими записами та переконайтеся, що несанкціоновані спроби доступу завершуються помилкою з кодом 403

Контрольні запитання (самоконтроль)

1. У чому полягає принципова відмінність між процесами автентифікації та авторизації в Laravel?
2. У яких конкретних випадках доцільніше використовувати Gates замість розробки повноцінних класів Policies?
3. Як працює механізм автоматичного переривання запиту під час використання методу authorize() у контролері?
4. Яке призначення Blade-директиви @can та як вона допомагає приховати логіку доступу від кінцевого користувача?

Лабораторна робота №5

Тема: Проєктування структури застосунку. Моделювання бази даних. Створення сидерів та основного функціоналу

Мета: Опанувати принципи проєктування архітектури вебзастосунків у середовищі Laravel, навчитися створювати міграції для опису схем таблиць, використовувати сидери для автоматичного заповнення бази даних та реалізувати базову CRUD-логіку для взаємодії з моделями.

Короткі теоретичні відомості

Eloquent ORM: Це вбудована система об’єктно-реляційного відображення Laravel, яка дає змогу розробнику взаємодіяти з базою даних за допомогою об’єктно-орієнтованого синтаксису PHP замість написання прямих SQL-запитів. Кожна таблиця в базі даних має відповідну модель, через яку здійснюється маніпуляція даними, що забезпечує прозору архітектуру коду.

Міграції: Це інструмент, який виконує роль системи контролю версій для бази даних. Вони дають змогу описувати структуру таблиць, типи полів та індекси безпосередньо в коді, що гарантує ідентичність бази даних на різних етапах розробки та деплою.

Сидери та наповнення даних: Це спеціальні класи, призначені для автоматизації процесу наповнення бази даних початковими або тестовими даними. Використання сидерів дає змогу швидко створювати демонстраційне середовище та перевіряти роботу складних функцій застосунку без ручного введення інформації.

CRUD-логіка: Абревіатура, що позначає чотири основні функції управління даними: створення (Create), читання (Read), оновлення (Update) та видалення (Delete). У Laravel реалізація цієї логіки зазвичай зосереджена в контролерах, які взаємодіють з моделями Eloquent для обробки запитів користувача.

Порядок виконання

Крок 1. Проєктування схеми даних. Визначте основні сутності вашого проєкту та створіть відповідні моделі разом із міграціями за допомогою команди Artisan, що дають змогу одночасно підготувати об'єктну структуру та опис таблиць у базі даних.

Крок 2. Опис структури таблиць у міграціях. Відкрийте створені файли міграцій та додайте опис полів, використовуючи методи об’єкта Blueprint. Налаштуйте типи даних, такі як рядки для заголовків, довгі тексти для контенту та зовнішні ключі для забезпечення зв’язків між таблицями.

Крок 3. Виконання міграцій та налаштування зв’язків. Запустіть процес міграції для створення фізичних таблиць у базі даних та пропишіть методи Eloquent relationships у класах моделей для реалізації зв'язків типу "один до багатьох" або "належить одному".

Крок 4. Створення та запуск сидерів. Згенеруйте класи сидерів для автоматичного заповнення бази даних початковою інформацією. Напишіть логіку створення декількох записів через модель та виконайте команду наповнення бази.

Крок 5. Реалізація базової CRUD-функціональності. Створіть контролер для управління основною сутністю проєкту та реалізуйте методи для відображення списку записів і створення нових елементів через вебформу.

Контрольні запитання (самоконтроль)

1. Яку роль відіграє патерн MVC та система Eloquent ORM у побудові архітектури Laravel-проєктів?
2. Яку проблему вирішує використання міграцій при спільній роботі над проєктом у команді?
3. Чим відрізняється призначення сидерів від призначення міграцій у життєвому циклі застосунку?
4. Як реалізувати зв’язок між таблицями на рівні моделей Eloquent (наприклад, hasMany або belongsTo)?
5. Що таке CRUD-функціональність та як вона пов'язана з HTTP-методами запитів?
6. Яка Artisan-команда дає змогу видалити всі існуючі таблиці та заново виконати міграції разом із сидерами?

Лабораторна робота №6

Тема: Реалізація розширених можливостей: коментарі, вкладені зв'язки, завантаження файлів. Розширена валідація

Мета: Навчитися реалізовувати складні архітектурні рішення у Laravel, опанувати роботу з вкладеними зв'язками Eloquent для створення системи коментарів, вивчити методи безпечного завантаження та зберігання медіафайлів, а також впровадити розширену валідацію даних через класи FormRequest.

Короткі теоретичні відомості

Вкладені зв'язки Eloquent: Для реалізації функціоналу коментарів або профілів користувачів використовуються складні відносини між моделями, такі як «один до багатьох» (hasMany) та «належить до» (belongsTo). Це дає змогу автоматично отримувати всі пов'язані дані: наприклад, коментарі до конкретного поста разом із даними про їхніх авторів через ланцюжки методів.

Система зберігання файлів: Laravel надає абстракцію над файловою системою за допомогою фасаду Storage, що дає змогу легко працювати з локальними дисками або хмарними сховищами. Важливим етапом є створення символічного посилання між папкою сховища та публічною директорією застосунку через команду storage:link для доступу до завантажених зображень через веббраузер.

Розширена валідація через FormRequest: У разі ускладнення логіки перевірки даних правила валідації доцільно виносити в окремі класи запитів. Це дає змогу відокремити бізнес-логіку від контролера, спростити тестування та забезпечити повторне використання правил для різних методів: наприклад, під час створення та редагуванні профілю користувача.

Оптимізація продуктивності: Під час роботи з великою кількістю вкладених даних важливо уникати проблеми запитів N+1, використовуючи метод eager loading (with). Також на цьому етапі розробки розглядаються питання індексації полів бази даних та кешування результатів складних вибірок для підвищення швидкодії системи.

Порядок виконання:

Крок 1. Проєктування бази даних для коментарів. Створіть модель та міграцію для коментарів, додавши необхідні поля для тексту та зовнішні ключі (user_id, post_id) для зв’язку з користувачами та публікаціями.

Крок 2. Налаштування відносин у моделях. У класах моделей пропишіть методи для зв'язків Eloquent, щоб забезпечити можливість отримання коментарів для кожного поста та автора для кожного коментаря.

Крок 3. Підготовка середовища до завантаження файлів. Використайте команду Artisan для створення символічного посилання між сховищем та публічною папкою, а також перевірте налаштування дисків у файлі конфігурації filesystem.php.

Крок 4. Створення та налаштування FormRequest. Згенеруйте новий клас запиту для обробки форм із файлами та опишіть розширені правила валідації, включаючи перевірку типів MIME та максимального розміру зображень.

Крок 5. Реалізація логіки завантаження у контролері. Напишіть код для збереження файлів на диску за допомогою методу store() та запису шляху до файлу в базу даних для подальшого виводу в інтерфейсі.

Крок 6. Відображення вкладених даних та медіафайлів. Модифікуйте Blade-шаблони для виводу списку коментарів під кожним постом та додайте теги для коректного відображення завантажених зображень за допомогою хелпера asset().

Крок 7. Впровадження захисту та безпеки. Додайте перевірки авторизації для видалення коментарів або зміни файлів, щоб гарантувати: лише власники контенту мають доступ до цих дій.

Крок 8. Оптимізація вибірки даних. Використайте метод with() у контролері для завантаження коментарів разом із постами, щоб значно зменшити кількість запитів до бази даних та пришвидшити роботу сторінки.

Контрольні запитання (самоконтроль)

1. У чому Яким чином реалізуються вкладені зв’язки між моделями для побудови системи коментарів?
2. Яка роль команди storage:link у забезпеченні доступу до завантажених файлів через браузер?
3. Які переваги надає використання класів FormRequest порівняно з валідацією безпосередньо в контролері?

4. Як правильно валідувати завантажені зображення (тип, розмір, розширення) у середовищі Laravel?

5. Що таке проблема запитів N+1 і як її ефективно вирішити при відображенні постів із коментарями?

6. Для чого використовується метод store() під час роботи з об’єктом завантаженого файлу?

7. Як реалізувати видалення фізичного файлу з диска під час видалення відповідного запису з бази даних?

8. Які інструменти Laravel дають змогу оптимізувати роботу з великою кількістю медіаданих?

9. Чим відрізняється локальний диск зберігання від публічного у конфігурації файлової системи застосунку?

10. Як забезпечити цілісність даних у разі видалення поста, що має вкладені коментарі?

Лабораторна робота №7

Тема: Розгортання застосунку. Налаштування середовища. Деплой на сервер та базові принципи CI/CD

Мета: Навчитися готувати Laravel-застосунок до роботи у виробничому середовищі, опанувати процес деплою на сервер, налаштувати параметри безпеки та ознайомитися з основами автоматизації через CI/CD.

Короткі теоретичні відомості

Виробниче середовище: На відміну від етапу розробки, середовище продакшену вимагає суворих налаштувань безпеки та продуктивності. Ключовим аспектом є переведення параметра APP_DEBUG у значення false у файлі .env, що запобігає витоку конфіденційної інформації про структуру коду та бази даних у разі виникнення помилок.

Оптимізація застосунку: Перед розгортанням необхідно виконати кешування конфігурації, маршрутів та Blade-шаблонів за допомогою спеціальних команд Artisan. Це дає змогу серверу значно швидше обробляти запити, оскільки фреймворку не потрібно кожного разу аналізувати файли налаштувань та маршрутизації під час виконання скриптів.

Безпека та конфіденційність: Кожен Laravel-застосунок має унікальний ключ APP_KEY, який використовується для шифрування сесій та інших конфіденційних даних. При розгортанні на новому сервері важливо правильно згенерувати цей ключ та забезпечити коректні права доступу до папок storage та bootstrap/cache, щоб сервер мав можливість записувати логи та кеш-файли.

CI/CD (Безперервна інтеграція та доставка): Сучасний підхід до розробки передбачає автоматизацію процесу тестування та деплою. За допомогою таких інструментів, як GitHub Actions, розробники можуть налаштувати автоматичне розгортання коду на сервер після кожного успішного оновлення репозиторію, що мінімізує людський фактор та пришвидшує реліз нових функцій.

Логування та моніторинг: Після запуску проєкту важливо налаштувати систему збору логів для відстеження помилок у реальному часі. Laravel надає гнучкі інструменти для запису подій у файли або зовнішні сервіси, що дає змогу адміністраторам швидко реагувати на критичні збої у роботі системи та забезпечувати стабільну підтримку проєкту.

Порядок виконання:

Крок 1. Підготовка конфігурації до продакшену. Оновіть файл .env на сервері, встановивши параметри APP_ENV у значення production та вимкнувши режим відлагодження. Переконайтеся, що всі ключі безпеки та паролі до бази даних вказані правильно.

Крок 2. Оптимізація фронтенд-ресурсів. Використайте інструментарій Vite для збирання активів застосунку в режимі продакшену. Це створить мінімізовані версії стилів та скриптів для максимально швидкого завантаження сторінок користувачами.

Крок 3. Налаштування серверного середовища. Підготуйте сервер до роботи (наприклад, на базі Linux із встановленим Nginx та PHP-FPM). Створіть виробничу базу даних та налаштуйте користувача з обмеженими правами доступу.

Крок 4. Встановлення залежностей. Завантажте код проєкту на сервер та виконайте інсталяцію PHP-пакетів через Composer з прапорцем --no-dev. Це дозволить уникнути встановлення інструментів, призначених лише для розробки та тестування.

Крок 5. Робота з базою даних на сервері. Запустіть міграції для створення структури таблиць у виробничій базі даних. Використовуйте спеціальні прапорці Artisan, щоб підтвердити виконання операції у виробничому середовищі без запиту з боку системи.

Крок 6. Кешування системних файлів. Виконайте серію команд для кешування конфігурації, маршрутів та представлень. Це є обов'язковим кроком для забезпечення високої швидкодії Laravel-застосунку під великим навантаженням.

Крок 7. Налаштування автоматичного деплою. Ознайомтеся із принципами роботи CI/CD та налаштуйте автоматичне виконання кроків розгортання при пуші в основну гілку репозиторію за допомогою доступних інструментів автоматизації.

Контрольні запитання (самоконтроль)

1. Чому параметр APP_DEBUG обов'язково має бути вимкнений на робочому сервері?
2. Які команди Artisan використовуються для кешування конфігурації та маршрутів перед деплоєм?
3. Що таке CI/CD та які переваги цей підхід надає у процесі підтримки вебпроєкту?
4. Для чого призначена папка storage/logs та як вона допомагає у моніторингу роботи застосунку?
5. Яким чином забезпечується безпека конфіденційних даних у файлі .env на сервері?
6. Які права доступу на файловій системі необхідно встановити для коректної роботи папок storage та cache?
7. Чим відрізняється процес встановлення залежностей через Composer на етапі розробки та на етапі деплою?

Довідник ключових команд інструментарію Artisan та Composer

composer create-project laravel/laravel [name]: основна команда для ініціалізації нового проєкту та автоматичного завантаження всіх необхідних залежностей фреймворку.

php artisan serve: команда для запуску вбудованого сервера розробки, який дозволяє тестувати застосунок у локальному браузері за адресою http://127.0.0.1:8000.

php artisan route:list: команда для виводу повного списку всіх зареєстрованих маршрутів застосунку, їхніх імен та методів HTTP, що є критично важливим для налагодження маршрутизації.

php artisan make:controller [Name]: генерація нового класу контролера для обробки бізнес-логіки та взаємодії з представленнями.

php artisan make:controller [Name] --resource: створення ресурсного контролера, який одразу містить набір методів для реалізації повного циклу операцій створення, читання, оновлення та видалення.

php artisan make:model [Name] -m: одночасне створення моделі Eloquent та відповідного файлу міграції для опису структури таблиці в базі даних.

php artisan make:migration [name]: створення нового файлу міграції для внесення змін до існуючої схеми бази даних, наприклад: додавання нових стовпців або індексів.

php artisan migrate: запуск процесу виконання всіх нових міграцій для оновлення структури бази даних на сервері розробки або продакшену.

php artisan migrate:fresh --seed: радикальна операція, яка видаляє всі існуючі таблиці, повторно запускає всі міграції та наповнює базу новими тестовими даними через сидери.

php artisan make:seeder [Name]: створення спеціального класу для автоматизації наповнення таблиць бази даних початковими або демонстраційними даними.

php artisan db:seed: запуск процесу наповнення бази даних, який виконує всі зареєстровані класи-сидери.

php artisan make:request [Name]: генерація нового класу FormRequest для реалізації розширеної логіки валідації вхідних даних від користувача.

php artisan make:policy [Name] --model=[Model]: створення класу політики, автоматично прив’язаного до конкретної моделі для розмежування прав доступу.

php artisan make:middleware [Name]: генерація нового проміжного програмного забезпечення для фільтрації та обробки HTTP-запитів.

php artisan storage:link: критично важлива команда для створення зв’язку між внутрішнім сховищем файлів та публічною папкою застосунку.

php artisan key:generate: створення унікального ключа шифрування застосунку APP_KEY, який забезпечує безпеку сесій та зашифрованих даних користувачів.

php artisan optimize:clear: команда для повного очищення всіх видів кешу застосунку, включаючи кеш конфігурації, маршрутів та відрендерених Blade-шаблонів.

php artisan config:cache: створення об’єднаного та прискореного файлу конфігурації для підвищення швидкодії системи у виробничому середовищі.

Список літератури

1. Stauffer, Matt. Laravel: Up & Running: A Framework for Efficient PHP Development. Sebastopol: O'Reilly Media, 2019. 544 p.. (Примітка: дані про видавництво та кількість сторінок не містяться в джерелах і взяті з відкритих даних).
2. Laracasts. Освітня платформа для Laravel-розробників [Електронний ресурс]. – Режим доступу: https://laracasts.com.
3. Laravel Documentation. Офіційна документація фреймворку [Електронний ресурс]. – Режим доступу: https://laravel.com/docs.
4. Smith, Matt. PHP Crash Course: The Complete, Modern, Hands-on Guide. San Francisco: No Starch Press, 2025. 679 p..
5. PHP Manual. Офіційна документація мови PHP [Електронний ресурс]. – Режим доступу: https://www.php.net/manual/.
6. PHP the Right Way. Збірник найкращих практик сучасного PHP-програмування [Електронний ресурс]. – Режим доступу: https://phptherightway.com/.
7. PHP-FIG (PHP Framework Interop Group). Стандарти кодування PSR (PSR-1, PSR-3, PSR-4) [Електронний ресурс]. – Режим доступу: https://www.php-fig.org/psr/.
8. OWASP (Open Web Application Security Project). Рекомендації з безпеки вебзастосунків [Електронний ресурс]. – Режим доступу: https://owasp.org.

Навчально-методичне видання

РОЗРОБКА ВЕБДОДАТКІВ НА ОСНОВІ ФРЕЙМВОРКУ LARAVEL

Методичні вказівки
до виконання лабораторних робіт
для здобувачів першого (бакалаврського) рівня
вищої освіти спеціальності F3 «Комп'ютерні науки»

Укладачі: **Науменко** Юрій Олександрович,
Хаддад Антон Георгович

Випусковий редактор *Ю.М. Долгополова*
Комп’ютерне верстання *Ю.М. Долгополової*

Ум. друк. арк. 1.39. Обл.-вид. арк. 1,5
Електронний документ. Вид № 93/V-26.

Виконавець і виготовлювач

Київський національний університет будівництва і архітектури
Проспект Повітряних Сил, 31, Київ, Україна, 03037

Свідоцтво про внесення до Державного реєстру суб’єктів
видавничої справи ДК № 808 від 13.02.2002