Лабораторна робота №6

Знайомство з Laravel: встановлення, конфігурація,
робота з БД та HTTP. Eloquent ORM, Auth, CRUD.

Мета роботи

Навчитися встановлювати та налаштовувати фреймворк Laravel, розуміти його архітектуру MVC, працювати з міграціями та Eloquent ORM, реалізовувати автентифікацію, повний CRUD для моделей та завантаження файлів.

Необхідне програмне забезпечення

- PHP версії 8.1 або вище
- Composer (менеджер залежностей для PHP)
- MySQL або MariaDB (СУБД)
- Node.js та npm (для збірки фронтенду)
- phpMyAdmin або DBeaver (для перегляду БД)
- VS Code або PhpStorm (редактор коду)

Частина 1: Базовий додаток

1. Встановлення Laravel

Laravel — це PHP-фреймворк, що реалізує архітектурний шаблон MVC (Model–View–Controller). Він автоматизує рутинні задачі: маршрутизацію, підключення до бази даних, автентифікацію тощо. Замість того, щоб писати весь цей код вручну, розробник зосереджується на бізнес-логіці додатку.

Що таке MVC?

Компонент	Відповідає за	Приклад у Laravel
Model	Дані та логіка роботи з БД	app/Models/Product.php
View	Відображення (HTML-шаблони)	resources/views/products/index.blade.php
Controller	Обробка запитів, зв'язок M та V	app/Http/Controllers/ProductController.php

Крок 1 — Перевірка середовища

Перш ніж встановлювати Laravel, переконайтесь, що у вас встановлені необхідні інструменти. Відкрийте термінал і виконайте:

Команди для перевірки (термінал):

```
php -v

composer -V
```

 Примітка: Якщо Composer не встановлений — завантажте його з https://getcomposer.org. Для Windows є окремий інсталятор (.exe).

Крок 2 — Створення проєкту

Команда нижче завантажить Laravel та створить структуру проєкту у папці laravel-app. Це займе 1–3 хвилини залежно від швидкості інтернету.

Встановлення та запуск:

```
composer create-project laravel/laravel laravel-app

cd laravel-app

php artisan serve
```

Команда	Що робить

composer create-project laravel/laravel laravel-app	Завантажує Laravel і всі його залежності
cd laravel-app	Переходить у папку щойно створеного проєкту
php artisan serve	Запускає локальний веб-сервер на порті 8000

 Порада: Після команди php artisan serve відкрийте браузер і перейдіть за адресою http://127.0.0.1:8000. Якщо ви бачите стартову сторінку Laravel — встановлення пройшло успішно.

Завдання 1: Встановлення Laravel

1. Перевірте версію PHP та Composer командами з інструкції.
2. Створіть новий проєкт командою composer create-project.
3. Запустіть сервер за допомогою php artisan serve.
4. Переконайтесь, що стартова сторінка Laravel відкривається в браузері.
5. Зробіть скриншот стартової сторінки для звіту.

2. Конфігурація додатку (.env)

Laravel зберігає всі налаштування у спеціальному файлі .env (environment — оточення). Цей підхід є стандартною практикою в індустрії: паролі та секретні ключі не потрапляють у код, а налаштування можна змінювати без правки PHP-файлів.

⚠ Увага: Файл .env не можна публікувати у відкритий репозиторій (GitHub). Він вже додається у .gitignore автоматично. Ніколи не зберігайте реальні паролі у відкритому доступі.

Основні параметри .env

Файл .env (приклад):

```
APP_NAME=Laravel # Назва додатку

APP_ENV=local # Середовище: local / production

APP_DEBUG=true # Показувати помилки (тільки для розробки!)

APP_URL=http://127.0.0.1:8000


DB_CONNECTION=mysql # Тип СУБД

DB_HOST=127.0.0.1 # Адреса сервера БД

DB_PORT=3306 # Порт MySQL

DB_DATABASE=laravel_db # Назва вашої БД
```

```
DB_USERNAME=root # Ім'я користувача

DB_PASSWORD=1234 # Пароль (замініть на свій)
```

Підключення бази даних — покроково

- Відкрийте phpMyAdmin (зазвичай http://localhost/phpmyadmin).
- Натисніть «Нова БД» та введіть назву: laravel_db.
- Відкрийте файл .env у редакторі та заповніть поля DB_DATABASE, DB_USERNAME, DB_PASSWORD.
- Виконайте у терміналі: php artisan migrate
- Якщо у phpMyAdmin з'явились таблиці — підключення працює.

Завдання 2: Налаштування .env і підключення БД

1. Відкрийте файл .env у корені проєкту.
2. Створіть базу даних laravel_db у phpMyAdmin.
3. Заповніть параметри DB_* у файлі .env.
4. Виконайте php artisan migrate та переконайтесь у відсутності помилок.
5. Перевірте у phpMyAdmin, що таблиці (users, migrations тощо) створились.

3. Робота з БД — Міграції

Міграції — це PHP-файли, які описують структуру таблиць бази даних. Замість того щоб вручну створювати таблиці в phpMyAdmin, ви описуєте їх у коді. Це дозволяє всій команді мати однакову структуру БД і відстежувати зміни через Git.

 Примітка: Думайте про міграції як про «версійний контроль» для бази даних. Так само як Git зберігає історію змін коду — міграції зберігають історію змін структури БД.

Створення міграції для таблиці Users

Команда нижче генерує файл міграції у папці database/migrations/. Laravel автоматично додає дату у назву файлу, щоб визначити порядок виконання.

Генерація файлу міграції:

```
php artisan make:migration create_users_table
```

Структура таблиці Users

database/migrations/..._create_users_table.php:

```
Schema::create('users', function (Blueprint $table) {

$table->id(); // Автоінкрементний первинний ключ

$table->string('name'); // ПІБ, обов'язкове поле
```

```
$table->string('email')->unique(); // Email, унікальний у таблиці

$table->string('password'); // Хеш пароля (не сам пароль!)

$table->integer('age')->nullable(); // Вік, не обов'язкове

$table->string('role')->default('user'); // Роль: 'user' або 'admin'

$table->string('phone')->nullable(); // Телефон, не обов'язкове

$table->timestamps(); // created_at та updated_at

});
```

Метод Blueprint	Тип у MySQL	Пояснення
$table->id()	BIGINT UNSIGNED AUTO_INCREMENT	Унікальний ідентифікатор запису
$table->string('name')	VARCHAR(255)	Текстове поле до 255 символів
->unique()	(індекс)	Гарантує відсутність дублікатів
->nullable()	NULL allowed	Поле може бути порожнім
->default('user')	DEFAULT 'user'	Значення за замовчуванням
$table->timestamps()	created_at, updated_at	Laravel оновлює автоматично

Виконання міграцій:

```
php artisan migrate
```

Завдання 3: Міграції та схема БД

1. Вивчіть файл міграції для таблиці users за схемою вище.
2. Переконайтесь, що всі поля (name, email, password, age, role, phone) присутні.
3. Виконайте php artisan migrate.
4. Відкрийте phpMyAdmin та перевірте структуру таблиці users.
5. Поясніть у звіті: чому пароль зберігається як хеш, а не відкритим текстом?

4. Eloquent ORM — робота з БД як з об'єктами

Eloquent — це вбудований у Laravel інструмент для роботи з базою даних. Замість написання SQL-запитів вручну, ви працюєте з PHP-об'єктами. Кожна таблиця у БД відповідає одному класу-моделі.

Без Eloquent (SQL)	З Eloquent (PHP-об'єкти)
SELECT * FROM products	Product::all()

SELECT * FROM products WHERE id = 1	Product::find(1)
INSERT INTO products (...) VALUES (...)	Product::create([...])
DELETE FROM products WHERE id = 1	$product->delete()

Створення моделі Product

Прапор -m означає, що разом з моделлю одразу створюється файл міграції. Це зручно — не потрібно виконувати дві окремі команди.

Генерація моделі та міграції:

```
php artisan make:model Product -m
```

Клас моделі

app/Models/Product.php:

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Product extends Model

{

// $fillable – список полів, дозволених для масового заповнення

// Це захист від Mass Assignment вразливості

protected $fillable = [

'name',

'category',

'price',

'description',

'image',

];

}
```

⚠ Увага: Поле у $fillable означає, що його можна заповнити через Product::create([...]). Поля, яких немає у $fillable, ігноруватимуться. Це важливий захист від атак.

Основні операції Eloquent

Приклади Eloquent-запитів:

```
// Отримати всі товари
$products = Product::all();

// Знайти товар за ID
$product = Product::find(1);

// Знайти або кинути помилку 404
$product = Product::findOrFail(1);

// Створити новий товар
Product::create([
'name' => 'Ноутбук',
'category' => 'Електроніка',
'price' => 25000,
]);

// Оновити
$product->update(['price' => 23000]);

// Видалити
$product->delete();

// Фільтрація
$cheap = Product::where('price', '<', 5000)->get();
```

Завдання 4: Знайомство з Eloquent ORM

1. Створіть модель Product командою php artisan make:model Product -m.
2. Відкрийте файл міграції та додайте поля: name, category, price, description, image.
3. Виконайте php artisan migrate.
4. У файлі моделі заповніть масив $fillable потрібними полями.
5. Протестуйте Eloquent через php artisan tinker: виконайте Product::all() і Product::create([...]).

5. Створення повноцінного додатку

У цьому розділі ви зберете всі вивчені компоненти в єдиний додаток з автентифікацією, CRUD-операціями та завантаженням зображень.

Вимоги до додатку — Частина 1

Функція	Опис та вимоги
Авторизація	Вхід як адміністратор (без реєстрації для звичайних користувачів)
Seeder	Початкові дані: admin@admin.com / password
CRUD Users	Список, створення, редагування, видалення користувачів
CRUD Products	Список, створення, редагування, видалення товарів
Поля Users	ПІБ (обов'язк.), email, password, вік, роль, телефон
Поля Products	Назва (обов'язк.), категорія (список), ціна, опис, зображення (мін. 100x100)
Міграції	Усі таблиці створюються через міграції
Зображення	Зберігати у storage/app/public через symlink
Контролери	Resource controllers з методами index, create, store, edit, update, destroy

5.1 Автентифікація (Laravel Breeze)

Laravel Breeze — офіційний стартовий кіт для автентифікації. Він генерує готові сторінки логіну, реєстрації, форми скидання пароля та middleware для захисту маршрутів.

Встановлення Breeze:

```
composer require laravel/breeze --dev

php artisan breeze:install

npm install

npm run dev

php artisan migrate
```

Після виконання цих команд Laravel створить:

- Сторінку входу: /login
- Middleware auth для захисту приватних маршрутів
- Систему сесій для збереження стану користувача
- Blade-шаблони у resources/views/auth/

Видалення реєстрації

Оскільки додаток призначений лише для адміністратора, потрібно прибрати маршрути реєстрації. Відкрийте файл routes/auth.php та закоментуйте або видаліть рядки, пов'язані з register:

routes/auth.php — видалення реєстрації:

```
// Знайдіть і закоментуйте (або видаліть) ці рядки:

// Route::get('register', [RegisteredUserController::class, 'create'])

// ->name('register');

// Route::post('register', [RegisteredUserController::class, 'store']);
```

5.2 Seeder — початковий адміністратор

Seeder — це PHP-клас, що заповнює базу даних початковими (тестовими) даними. Ми створимо адміністратора, щоб мати можливість увійти в систему одразу після встановлення.

Генерація seeder:

```
php artisan make:seeder AdminSeeder
```

database/seeders/AdminSeeder.php:

```
<?php


namespace Database\Seeders;


use App\Models\User;

use Illuminate\Support\Facades\Hash;

use Illuminate\Database\Seeder;


class AdminSeeder extends Seeder

{

public function run(): void

{

User::create([

'name' => 'Administrator',

'email' => 'admin@admin.com',

'password' => Hash::make('password'), // хешуємо пароль!

'role' => 'admin',

]);

}
```

```
}
```

Запуск seeder:

```
php artisan db:seed --class=AdminSeeder
```

 Порада: Hash::make('password') перетворює рядок 'password' на безпечний хеш. Laravel автоматично перевіряє хеш при вході. Ніколи не зберігайте паролі відкритим текстом!

5.3 CRUD для Products — Resource Controller

Генерація контролера:

```
php artisan make:controller ProductController --resource
```

Прапор --resource генерує контролер з усіма необхідними методами одразу. Вам залишається лише заповнити їх логікою.

Метод контролера	HTTP-метод	URL	Призначення
index()	GET	/products	Список усіх товарів
create()	GET	/products/create	Форма створення
store()	POST	/products	Збереження нового товару
show()	GET	/products/{id}	Деталі одного товару
edit()	GET	/products/{id}/edit	Форма редагування
update()	PUT/PATCH	/products/{id}	Збереження змін
destroy()	DELETE	/products/{id}	Видалення товару

Приклад методу store()

app/Http/Controllers/ProductController.php — метод store():

```
public function store(Request $request)

{

// 1. Валідація вхідних даних

$data = $request->validate([

'name' => 'required|min:2|max:100',

'category' => 'required',

'price' => 'required|numeric|min:0',

'description' => 'nullable|max:1000',

]);
```

```
// 2. Збереження у БД

Product::create($data);


// 3. Перенаправлення зі спалах-повідомленням

return redirect()->route('products.index')

->with('success', 'Товар успішно додано!');

}
```

Реєстрація маршрутів

routes/web.php:

```
// routes/web.php


use App\Http\Controllers\ProductController;


// Один рядок замінює 7 маршрутів!

Route::resource('products', ProductController::class)->middleware('auth');
```

 Примітка: Route::resource автоматично реєструє всі 7 маршрутів (index, create, store, show, edit, update, destroy). Middleware 'auth' гарантує, що лише авторизовані користувачі мають доступ.

5.4 Завантаження зображень

Laravel зберігає завантажені файли у папці storage/app/public/. Щоб зробити їх доступними через браузер, потрібно створити символічне посилання (symlink) з public/storage на storage/app/public.

Одноразова команда для налаштування:

```
php artisan storage:link
```

Збереження зображення у контролері:

```
// У методі store() або update() контролера:


if ($request->hasFile('image')) {

// Зберегти файл і отримати шлях

$path = $request->file('image')->store('products', 'public');

$data['image'] = $path; // зберігаємо шлях у БД

}


// У Blade-шаблоні для відображення:
```

```
// <img src="{{ asset('storage/' . $product->image) }}">
```

Завдання 5: Базовий додаток з CRUD

1. Встановіть Laravel Breeze та налаштуйте автентифікацію.
2. Видаліть маршрути реєстрації, залишивши лише вхід.
3. Створіть AdminSeeder та запустіть його. Перевірте вхід: admin@admin.com / password.
4. Створіть ProductController --resource та заповніть методи index, create, store, edit, update, destroy.
5. Реалізуйте завантаження зображень через storage з валідацією розміру (мін. 100x100).
6. Зареєструйте маршрути через Route::resource з middleware auth.
7. Протестуйте весь CRUD: додавання, перегляд, редагування та видалення товару.

Частина 2: Розширений функціонал

У другій частині ви розширите базовий додаток: додасте таблицю рахунків (Invoices) зі зв'язками між моделями, валідацію через окремі класи, пагінацію, інтерактивні таблиці, надсилання Email та підтримку кількох мов.

Вимоги до розширеного функціоналу

Функція	Технологія/Деталі
Invoices	ID, UserID (FK), ProductID (FK), дата, кількість товару, загальна сума
Валідація	Окремі Request-класи (php artisan make:request)
Пагінація	10 записів на сторінку для Users, Products, Invoices
DataTables	Datatables.net — пошук, сортування, фільтрація
Email	SMTP (Mailtrap для тестування) — лист при створенні користувача
i18n	resources/lang — підтримка двох мов (наприклад, UK та EN)
Тема	AdminLTE або аналог Bootstrap-адмін-теми

6. Таблиця Invoices та зв'язки між моделями

Invoice (рахунок/накладна) — це запис про покупку: який користувач придбав який товар, у якій кількості та на яку суму. Таблиця invoices пов'язана з users і products через зовнішні ключі (Foreign Keys).

Генерація міграції:

```
php artisan make:migration create_invoices_table
```

database/migrations/..._create_invoices_table.php:

```
Schema::create('invoices', function (Blueprint $table) {

$table->id();

// Зовнішній ключ на таблицю users

$table->foreignId('user_id')->constrained()->onDelete('cascade');

// Зовнішній ключ на таблицю products

$table->foreignId('product_id')->constrained()->onDelete('cascade');

$table->date('invoice_date'); // Дата рахунку

$table->integer('quantity'); // Кількість товару

$table->decimal('total_amount', 10, 2); // Загальна сума
```

```
$table->timestamps();

});
```

 Примітка: Метод constrained() автоматично визначає назву таблиці з назви поля (user_id → таблиця users). onDelete('cascade') означає: якщо користувача видалено — всі його рахунки теж видаляться.

Зв'язки між моделями (Relationships)

Eloquent дозволяє отримувати пов'язані дані через методи-зв'язки. Це елегантніше і безпечніше, ніж писати JOIN-запити вручну.

Визначення зв'язків у моделях:

```
// app/Models/Invoice.php

class Invoice extends Model

{

protected $fillable = ['user_id', 'product_id', 'invoice_date', 'quantity',
'total_amount'];


// Invoice belongs to User

public function user()

{

return $this->belongsTo(User::class);

}


// Invoice belongs to Product

public function product()

{

return $this->belongsTo(Product::class);

}

}


// app/Models/User.php — додати метод:

public function invoices()

{

return $this->hasMany(Invoice::class);

}
```

Використання зв'язків:

```
// Отримати ім'я покупця рахунку:

$invoice->user->name;


// Отримати назву товару:

$invoice->product->name;


// Всі рахунки конкретного користувача:

$user->invoices;


// 3 eager loading (без N+1 проблеми):

$invoices = Invoice::with(['user', 'product'])->paginate(10);
```

 Порада: Завжди використовуйте with(['user', 'product']) (eager loading), якщо відображаєте список рахунків. Без цього Laravel виконає окремий SQL-запит для кожного рядка (проблема N+1).

7. Валідація через Request-класи

У базовій частині валідація знаходиться прямо в контролері. Для більших проєктів прийнято виносити її в окремі класи — це робить код чистішим і повторно використовуваним.

Генерація Request-класу:

```
php artisan make:request StoreProductRequest
```

app/Http/Requests/StoreProductRequest.php:

```
<?php


namespace App\Http\Requests;


use Illuminate\Foundation\Http\FormRequest;


class StoreProductRequest extends FormRequest

{

// Хто має право виконати цей запит?

public function authorize(): bool

{

return true; // або перевірка ролі

}
```

```
// Правила валідації
public function rules(): array
{
return [
'name' => 'required|min:3|max:100',
'category' => 'required|in:Electronics,Clothing,Food',
'price' => 'required|numeric|min:0',
'description' => 'nullable|max:1000',
'image' => 'nullable|image|min_dimensions:100,100',
];
}

// Повідомлення про помилки (необов'язково)
public function messages(): array
{
return [
'name.required' => 'Назва товару є обов'язковою.',
'price.numeric' => 'Ціна повинна бути числом.',
];
}
}
```

Використання у контролері:

```
// У контролері — замінити Request на StoreProductRequest:
public function store(StoreProductRequest $request)
{
// Валідація вже виконана автоматично!
$data = $request->validated();
Product::create($data);
return redirect()->route('products.index');
}
```

8. Пагінація

Пагінація дозволяє відображати дані частинами (сторінками), а не всі одразу. Це зменшує навантаження на сервер і покращує UX.

Контролер та шаблон:

```
// Замість Product::all() — використовувати paginate():

$products = Product::paginate(10);


// У Blade-шаблоні додати кнопки навігації:

{{ $products->links() }}
```

 Примітка: Laravel автоматично генерує HTML-кнопки пагінації, сумісні з Bootstrap та Tailwind CSS. Жодного додаткового коду не потрібно!

9. Надсилання Email

Laravel має вбудовану систему надсилання пошти. Для тестування використовуйте Mailtrap.io — він перехоплює листи і показує їх у веб-інтерфейсі, не надсилаючи реальним отримувачам.

Налаштування Mailtrap у .env

.env — налаштування пошти:

```
MAIL_MAILER=smtp

MAIL_HOST=sandbox.smtp.mailtrap.io

MAIL_PORT=2525

MAIL_USERNAME=ваш_username_з_mailtrap

MAIL_PASSWORD=ваш_password_з_mailtrap

MAIL_FROM_ADDRESS=noreply@example.com

MAIL_FROM_NAME="Laravel App"
```

Клас листа та надсилання:

```
// Генерація класу листа:

php artisan make:mail UserCreatedMail


// app/Mail/UserCreatedMail.php

class UserCreatedMail extends Mailable

{

public function __construct(public User $user) {}
```

```
public function envelope(): Envelope

{

return new Envelope(subject: 'Вітаємо в системі!');

}


public function content(): Content

{

return new Content(view: 'emails.user-created');

}

}


// Надсилання у контролері (після створення User):

Mail::to($user->email)->send(new UserCreatedMail($user));
```

10. DataTables — інтерактивні таблиці

DataTables.net — JavaScript-бібліотека, що перетворює звичайну HTML-таблицю на інтерактивну: з пошуком, сортуванням за колонками та пагінацією на клієнті.

Підключення DataTables (CDN):

```
<!-- Підключення у Blade-шаблоні: -->

<link rel="stylesheet"
href="https://cdn.datatables.net/1.13.6/css/dataTables.bootstrap5.min.css">

<script
src="https://cdn.datatables.net/1.13.6/js/jquery.dataTables.min.js"></script>


<!-- Ініціалізація: -->

<script>

$(document).ready(function () {

$('#products-table').DataTable({

language: { url: '//cdn.datatables.net/plug-ins/1.13.6/i18n/uk.json' }

});

});

</script>
```

11. Багатомовність (i18n)

Laravel підтримує локалізацію через файли у папці resources/lang/. Можна легко перемикати мову додатку.

Файли перекладу та їх використання:

```
// resources/lang/uk/messages.php

return [

'welcome' => 'Ласкаво просимо',

'products' => 'Товари',

'create_button' => 'Додати товар',

];


// resources/lang/en/messages.php

return [

'welcome' => 'Welcome',

'products' => 'Products',

'create_button' => 'Add Product',

];


// Використання у Blade:

{{ __('messages.welcome') }}

{{ __('messages.products') }}
```

12. Адмін-тема (AdminLTE)

AdminLTE — популярна готова Bootstrap-адмін-тема для Laravel. Вона надає готовий sidebar, header, картки, таблиці — без необхідності верстати їх з нуля.

Встановлення:

```
composer require jeroennoten/laravel-adminlte

php artisan adminlte:install
```

Завдання 6: Розширений функціонал (Частина 2)

1. Створіть міграцію та модель Invoice з полями та зовнішніми ключами.
2. Додайте зв'язки belongsTo/hasMany у моделях Invoice, User, Product.
3. Реалізуйте CRUD для Invoices з автоматичним розрахунком total_amount = quantity × price.
4. Винесіть валідацію Products та Users у окремі Request-класи.
5. Замініть Product::all() на paginate(10) у всіх контролерах. Додайте {{ $items->links() }} у шаблони.
6. Підключіть DataTables до таблиць Users, Products, Invoices.
7. Зареєструйтесь на Mailtrap.io, налаштуйте .env та надсилайте лист при створенні User.
8. Додайте мінімум 2 мови (uk/en). Реалізуйте перемикач мови у навігації.
9. Встановіть AdminLTE та адаптуйте шаблони під цю тему.

Критерії оцінювання та чеклист

Частина 1 — Базовий додаток (50 балів)

#	Критерій	Бали	Перевірка
1	Laravel встановлено, стартова сторінка відкривається	5	Скриншот браузера
2	.env налаштований, php artisan migrate без помилок	5	Скриншот phpMyAdmin
3	Міграції для users і products створені правильно	5	Структура таблиць
4	Моделі з $fillable та Eloquent-операції працюють	5	php artisan tinker
5	Авторизація через Breeze, вхід admin@admin.com/password	10	Демо входу
6	CRUD Products повністю реалізований	10	Тестування форм
7	Завантаження зображень (storage + symlink)	5	Зображення відображається
8	Маршрути через Route::resource з middleware auth	5	php artisan route:list

Частина 2 — Розширений функціонал (50 балів)

#	Критерій	Бали	Перевірка
1	Таблиця invoices з FK, зв'язки між моделями	8	Схема БД + код
2	Request-класи для валідації Products/Users	7	Код Request-класу
3	Пагінація (10 записів) у всіх списках	5	Навігація по сторінках
4	DataTables: пошук та сортування	8	Демо пошуку
5	Email при створенні User (Mailtrap)	8	Скриншот листа у Mailtrap
6	Багатомовність (мін. 2 мови, перемикач)	7	Демо перемикання
7	Адмін-тема (AdminLTE або аналог)	7	Зовнішній вигляд

Корисні ресурси

Ресурс	URL	Що знайти
Laravel Docs	https://laravel.com/docs	Офіційна документація
Mailtrap	https://mailtrap.io	Тестування Email
DataTables	https://datatables.net	Документація та приклади
AdminLTE	https://adminlte.io	Документація теми
Laravel Breeze	https://laravel.com/docs/starter-kits	Стартовий кіт авторизації
Laracasts	https://laracasts.com	Відеоуроки з Laravel

Бажаємо успіху у виконанні лабораторної роботи!