Laravel 치트시트

설치 및 프로젝트 설정

Laravel은 PHP를 위한 표현력 있고 우아한 문법을 가진 웹 애플리케이션 프레임워크입니다.

- **프로젝트 생성**: Composer를 사용하여 새로운 Laravel 프로젝트를 생성합니다. `composer create-project laravel/laravel <project_name>`
- **개발 서버 실행**: 내장된 개발용 서버를 구동합니다. `php artisan serve`
- **환경 설정**: `.env` 파일에서 데이터베이스, 메일 서비스 등 애플리케이션의 주요 환경 설정을 관리합니다.
- **애플리케이션 키 생성**: 보안을 위한 암호화 키를 생성하여 `.env` 파일에 자동으로 등록합니다. `php artisan key:generate`

Artisan CLI

Artisan은 Laravel에서 제공하는 강력한 커맨드 라인 인터페이스(CLI)로, 개발 중 반복되는 작업을 자동화해 줍니다.

- **명령어 목록 확인**: `php artisan list`
- **컨트롤러 생성**: 비즈니스 로직을 처리할 컨트롤러 파일을 생성합니다. `php artisan make:controller MyController`
- **모델 및 관련 파일 생성**: 데이터베이스와 상호작용할 모델과 함께 마이그레이션(-m), 팩토리(-f), 시더(-s) 파일을 한 번에 생성합니다. `php artisan make:model Post -mfs`
- **마이그레이션 실행**: 정의된 스키마를 바탕으로 데이터베이스 테이블을 생성하거나 수정합니다. `php artisan migrate`
- **라우트 목록 확인**: 정의된 모든 URL 경로 목록을 출력합니다. `php artisan route:list`
- **Tinker (REPL)**: 애플리케이션의 환경에서 직접 PHP 코드를 실행하고 데이터를 조작할 수 있는 대화형 쉘입니다.

라우팅 (Routing)

사용자의 요청 URL을 특정 컨트롤러나 로직에 연결합니다. 웹 경로는 `routes/web.php`, API 경로는 `routes/api.php`에 정의합니다.

- **기본 라우트 정의**:

```
use App\Http\Controllers\PostController;

// GET 요청을 처리하여 인덱스 페이지 표시
Route::get('/posts', [PostController::class, 'index']);
// POST 요청을 처리하여 데이터 저장
Route::post('/posts', [PostController::class, 'store']);
```

- **리소스 라우트**: CRUD에 필요한 표준 라우트들을 한 줄로 생성합니다. `Route::resource('photos', PhotoController::class);`
- **라우트 파라미터**: URL의 특정 부분을 변수로 전달받습니다. `Route::get('/posts/{id}', function ($id) { ... });`
- **이름 있는 라우트**: 코드 내에서 URL 대신 참조할 수 있는 고유 이름을 부여합니다.

컨트롤러 (Controllers)

HTTP 요청에 대한 응답 로직을 처리하며, 주로 `app/Http/Controllers` 디렉토리에 위치합니다.

```
namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\Post;

class PostController extends Controller
{
    public function index()
    {
        // 모든 포스트 데이터를 조회하여 뷰(View)로 전달합니다.
        $posts = Post::all();
        return view('posts.index', ['posts' => $posts]);
    }

    public function show(Post $post) // 라우트 모델 바인딩 활용
    {
        // ID 대신 모델 인스턴스를 직접 주입받아 처리합니다.
        return view('posts.show', ['post' => $post]);
    }
}
```

블레이드 템플릿 (Blade)

Laravel의 강력하고 직관적인 템플릿 엔진으로, HTML 내에서 PHP 코드를 깔끔하게 작성할 수 있게 돕습니다.

- **변수 출력**: `{{ $variable }}` 형식을 사용하여 데이터를 안전하게 출력합니다.
- **주석 작성**: `{{-- 블레이드 전용 주석입니다. --}}`
- **제어 구조**:
  - 조건문: `@if`, `@elseif`, `@else`, `@endif`
  - 반복문: `@foreach`, `@forelse` (배열이 비었을 때 `@empty` 처리 가능)
  - 인증 확인: `@auth` (로그인 시), `@guest` (비로그인 시)
- **레이아웃 상속**:
  - 부모 레이아웃 확장: `@extends('layouts.app')`
  - 섹션 정의: `@section('content') ... @endsection`
  - 섹션 출력: `@yield('title')`
- **컴포넌트 및 부분 뷰**: `@include('partials.header')`를 통해 뷰를 재사용합니다.

엘로퀀트 ORM (Eloquent)

데이터베이스 작업을 객체 지향적으로 처리할 수 있게 해주는 Laravel의 강력한 Active Record 패턴 구현체입니다.

- **모델 정의**: `app/Models/Post.php`와 같이 클래스를 정의하여 테이블과 매핑합니다.
- **데이터 조회**:
  - 전체 조회: `Post::all()`
  - ID로 조회: `Post::find(1)`
  - 조건부 조회: `Post::where('status', 'active')->orderBy('id', 'desc')->get();`
- **데이터 저장 및 수정**:
  - 생성: `Post::create(['title' => '제목', 'content' => '내용']);`
  - 수정: 기존 인스턴스를 찾아 값을 변경한 후 `save()`를 호출합니다.
- **데이터 삭제**: `$post->delete();`

관계 정의 (Relationships)

테이블 간의 연관 관계를 메서드 형태로 정의하여 쉽게 데이터를 가져올 수 있습니다.

- **주요 관계 유형**:
  - 일대다(1:N): `hasMany()`, `belongsTo()`
  - 다대다(N:N): `belongsToMany()`
  - 일대일(1:1): `hasOne()`, `belongsTo()`
- **데이터 로딩 전략**:
  - **Eager Loading**: `User::with('posts')->get();` (N+1 문제 방지를 위해 미리 로드)
  - **Lazy Loading**: `$user->posts` (필요한 시점에 쿼리 실행)

미들웨어 (Middleware)

HTTP 요청이 애플리케이션으로 들어오기 전이나 나갈 때 가로채어 특정 작업을 수행(인증, 로깅, 필터링 등)하는 메커니즘입니다.

- **미들웨어 생성**: `php artisan make:middleware CheckAge`
- **등록 및 적용**: 생성한 미들웨어는 `app/Http/Kernel.php`에 등록하여 특정 라우트나 그룹에 적용합니다.

유효성 검사 (Validation)

사용자가 입력한 데이터의 정당성을 검사하여 데이터 무결성을 보장합니다.

- **폼 요청 클래스**: 별도의 요청 클래스를 생성하여 컨트롤러 로직을 깔끔하게 유지합니다. `php artisan make:request StorePostRequest`
- **검사 규칙 정의**: `rules()` 메서드에 각 필드별 제약 사항을 작성합니다.

```
return [
    'title' => 'required|max:255',
    'content' => 'required',
];
```

- **에러 피드백**: 유효성 검사 실패 시 자동으로 이전 페이지로 리다이렉트되며, 뷰에서 `@error` 지시어를 통해 에러 메시지를 표시할 수 있습니다.