PHP Laravel - Comprehensive Study Guide

Laravel Fundamentals (Core Concepts)

What is Laravel?

- PHP web application framework with expressive, elegant syntax
- Follows MVC (Model-View-Controller) architecture pattern
- Created by Taylor Otwell, first released in June 2011
- Current stable version: Laravel 10.x

Key Features

- Elegant syntax designed for developer happiness and productivity
- Built-in tools for routing, authentication, caching, and more
- Eloquent ORM for intuitive database interaction
- Blade templating engine for clean, powerful templates
- Artisan CLI for common tasks and code generation
- Laravel Mix for asset compilation
- Robust ecosystem with packages like Sanctum, Jetstream, Livewire

Laravel Architecture

Directory Structure

```
/app                # Core application code
  /Http             # Controllers, Middleware, Requests
  /Models           # Eloquent models
  /Providers        # Service providers
/bootstrap          # Framework bootstrap files
/config             # Configuration files
/database           # Migrations, seeds, factories
/public             # Entry point, assets
/resources          # Views, language files, uncompiled assets
/routes             # Route definitions
/storage            # Logs, compiled views, file uploads
/tests              # Test files
/vendor             # Composer dependencies
```

Request Lifecycle

1. Entry point through public/index.php
2. Request handled by HTTP kernel
3. Service providers are loaded and registered
4. Router matches request to a route
5. Route executes middleware
6. Controller or closure handles request
7. Response returned to client

Core Components

Routing

Define routes in routes/web.php or routes/api.php

```
// Basic route
Route::get('/greeting', function () {
    return 'Hello World';
});

// Route with parameters
Route::get('/user/{id}', [UserController::class, 'show']);

// Named routes
Route::get('/profile', [ProfileController::class, 'show'])->name('profile');

// Route groups
Route::middleware(['auth'])->group(function () {
    Route::get('/dashboard', [DashboardController::class, 'index']);
});
```

Controllers

Classes that organize request handling logic. Located in app/Http/Controllers

```
class UserController extends Controller
{
    public function index()
    {
        $users = User::all();
        return view('users.index', ['users' => $users]);
    }

    public function show($id)
    {
        $user = User::findOrFail($id);
        return view('users.show', ['user' => $user]);
    }
}
```

Models & Eloquent ORM

Represent database tables as elegant PHP classes. Located in app/Models

```
class User extends Model
{
    // Mass-assignable attributes
    protected $fillable = ['name', 'email', 'password'];

    // Hide attributes from serialization
    protected $hidden = ['password', 'remember_token'];

    // Define relationships
```

```
    public function posts()
    {
        return $this->hasMany(Post::class);
    }
}
```

Query Examples

```
// Basic queries
$users = User::all();
$user = User::find(1);
$activeUsers = User::where('status', 'active')->get();

// Relationships
$user = User::find(1);
$posts = $user->posts;

// Create
$user = User::create([
    'name' => 'John Doe',
    'email' => 'john@example.com',
    'password' => bcrypt('password123')
]);

// Update
$user = User::find(1);
$user->name = 'Jane Doe';
$user->save();

// Delete
$user = User::find(1);
$user->delete();
```

Views & Blade

Template engine that compiles to plain PHP. Located in resources/views

```
// resources/views/greeting.blade.php
<html>
    <body>
        <h1>Hello, {{ $name }}</h1>
    </body>
</html>

// In controller
return view('greeting', ['name' => 'James']);
```

Blade Directives

```
// Conditionals
@if($user->isAdmin)
    Admin content
@elseif($user->isManager)
    Manager content
@else
    User content
```

```
@endif

// Loops
@foreach($users as $user)
    {{ $user->name }}
@endforeach

// Layouts
// layouts/app.blade.php
<html>
    <head>
        <title>@yield('title')</title>
    </head>
    <body>
        @yield('content')
    </body>
</html>

// Using layouts
@extends('layouts.app')

@section('title', 'Page Title')

@section('content')
    <p>Content goes here</p>
@endsection
```

Middleware

Filter HTTP requests entering application. Located in app/Http/Middleware

```
class CheckAge
{
    public function handle($request, Closure $next)
    {
        if ($request->age < 18) {
            return redirect('home');
        }

        return $next($request);
    }
}

// Register in Kernel.php
protected $routeMiddleware = [
    'age' => \App\Http\Middleware\CheckAge::class,
];

// Use in routes
Route::get('adults-only', function () {
    // ...
})->middleware('age');
```

Database & Migrations

Configuration

- Set up in .env file and config/database.php
- Supports MySQL, PostgreSQL, SQLite, SQL Server

Migrations

Version control for database. Located in database/migrations

```
// Create migration
php artisan make:migration create_users_table

// Migration file
public function up()
{
    Schema::create('users', function (Blueprint $table) {
        $table->id();
        $table->string('name');
        $table->string('email')->unique();
        $table->timestamp('email_verified_at')->nullable();
        $table->string('password');
        $table->rememberToken();
        $table->timestamps();
    });
}

// Run migrations
php artisan migrate

// Rollback
php artisan migrate:rollback
```

Seeders

Populate database with test data. Located in database/seeders

```
class UserSeeder extends Seeder
{
    public function run()
    {
        User::factory(10)->create();

        User::create([
            'name' => 'Admin User',
            'email' => 'admin@example.com',
            'password' => bcrypt('password')
        ]);
    }
}

// Run seeders
php artisan db:seed
```

Factories

Generate fake model data for testing. Located in database/factories

```
class UserFactory extends Factory
{
    public function definition()
    {
        return [
            'name' => $this->faker->name(),
            'email' => $this->faker->unique()->safeEmail(),
            'password' => bcrypt('password'),
        ];
    }
}

// Use factory
$user = User::factory()->create();
$users = User::factory()->count(5)->create();
```

Authentication & Authorization

Authentication

Built-in authentication scaffolding. Options include Laravel Breeze, Jetstream, Fortify

```
// Basic auth setup
php artisan make:auth  // For older versions
// Or
composer require laravel/breeze --dev
php artisan breeze:install

// Auth helpers
Auth::user()  // Current user
Auth::id()    // Current user ID
Auth::check()  // Check if authenticated
```

Authorization (Policies & Gates)

```
// Define gate
Gate::define('update-post', function ($user, $post) {
    return $user->id === $post->user_id;
});

// Use gate
if (Gate::allows('update-post', $post)) {
    // User can update post
}

// Policy class
class PostPolicy
{
    public function update(User $user, Post $post)
    {
        return $user->id === $post->user_id;
```

```
    }
}

// Use policy in controller
$this->authorize('update', $post);
```

Forms & Validation

CSRF Protection

Automatic CSRF protection for forms

```
<form method="POST" action="/profile">
    @csrf
    <!-- Form fields -->
</form>
```

Validation

```
// In controller
public function store(Request $request)
{
    $validated = $request->validate([
        'title' => 'required|unique:posts|max:255',
        'body' => 'required',
        'published_at' => 'nullable|date',
    ]);

    // Validation passed, create post
    $post = Post::create($validated);
}

// Display errors in blade
@if ($errors->any())
    <div class="alert alert-danger">
        <ul>
            @foreach ($errors->all() as $error)
                <li>{{ $error }}</li>
            @endforeach
        </ul>
    </div>
@endif

// Display specific field error
@error('title')
    <div class="alert alert-danger">{{ $message }}</div>
@enderror
```

Form Requests

Custom request classes for complex validation

```
// Generate request class
php artisan make:request StorePostRequest
```

```
// In request class
public function rules()
{
    return [
        'title' => 'required|unique:posts|max:255',
        'body' => 'required',
    ];
}

// In controller
public function store(StorePostRequest $request)
{
    // Validation already happened
    $post = Post::create($request->validated());
}
```

API Development

API Routes

Define in routes/api.php. Automatically prefixed with /api

```
Route::get('/users', [UserController::class, 'index']);
```

API Resources

Transform models to JSON

```
// Generate resource
php artisan make:resource UserResource

// Resource class
class UserResource extends JsonResource
{
    public function toArray($request)
    {
        return [
            'id' => $this->id,
            'name' => $this->name,
            'email' => $this->email,
            'posts' => PostResource::collection($this->posts),
            'created_at' => $this->created_at,
        ];
    }
}

// In controller
return UserResource::collection(User::all());
// Or for single item
return new UserResource($user);
```

API Authentication

Laravel Sanctum for API tokens and SPA authentication

```
// Set up Sanctum
composer require laravel/sanctum
php artisan vendor:publish --provider="Laravel\Sanctum\SanctumServiceProvider"
php artisan migrate

// Create token
$token = $user->createToken('token-name')->plainTextToken;

// Protect routes
Route::middleware('auth:sanctum')->get('/user', function (Request $request) {
    return $request->user();
});
```

Testing

Types of Tests

- **Feature tests**: Test full HTTP requests
- **Unit tests**: Test individual classes
- Located in /tests directory

Example Tests

```
// Feature test
class UserTest extends TestCase
{
    public function test_user_can_see_profile()
    {
        $user = User::factory()->create();

        $response = $this->actingAs($user)
                         ->get('/profile');

        $response->assertStatus(200)
                 ->assertSee($user->name);
    }
}

// Unit test
public function test_can_get_formatted_name()
{
    $user = User::factory()->make([
        'first_name' => 'John',
        'last_name' => 'Doe',
    ]);

    $this->assertEquals('John Doe', $user->getFullName());
}

// Run tests
php artisan test
```

Artisan CLI

Key Commands

```
php artisan list               # List all commands
php artisan make:controller    # Create controller
php artisan make:model         # Create model
php artisan make:migration     # Create migration
php artisan serve              # Start development server
php artisan tinker             # Interactive shell
php artisan route:list         # List all routes
php artisan config:clear       # Clear config cache
php artisan queue:work         # Start queue worker
php artisan storage:link       # Create symlink
```

Custom Commands

```
// Generate command
php artisan make:command SendEmails

// Command class
class SendEmails extends Command
{
    protected $signature = 'mail:send {user} {--queue}';
    protected $description = 'Send emails to user';

    public function handle()
    {
        $userId = $this->argument('user');
        $queue = $this->option('queue');

        // Command logic
    }
}
```

Advanced Topics

Service Providers

Bootstrap components and register services. Located in app/Providers

```
class AppServiceProvider extends ServiceProvider
{
    public function register()
    {
        $this->app->singleton('example', function () {
            return new ExampleService();
        });
    }

    public function boot()
    {
        // Bootstrap logic
    }
}
```

Service Container

Dependency injection container

```
// Binding
$this->app->bind('HelpSpot\API', function () {
    return new HelpSpot\API('api-key');
});

// Resolving
$api = app('HelpSpot\API');
// Or
$api = App::make('HelpSpot\API');
```

Events & Listeners

```
// Generate event and listener
php artisan make:event UserRegistered
php artisan make:listener SendWelcomeEmail --event=UserRegistered

// Dispatch event
event(new UserRegistered($user));
```

Queues

Process background jobs

```
// Create job
php artisan make:job ProcessPodcast

// Job class
class ProcessPodcast implements ShouldQueue
{
    use Dispatchable, InteractsWithQueue, Queueable, SerializesModels;

    protected $podcast;

    public function __construct(Podcast $podcast)
    {
        $this->podcast = $podcast;
    }

    public function handle()
    {
        // Process the podcast
    }
}

// Dispatch job
ProcessPodcast::dispatch($podcast);
// Or with delay
ProcessPodcast::dispatch($podcast)->delay(now()->addMinutes(10));
```

Caching

```
// Store in cache
Cache::put('key', 'value', $seconds);

// Retrieve from cache
$value = Cache::get('key', 'default');

// Remove from cache
Cache::forget('key');

// Cache helper for expensive operations
$value = Cache::remember('users', $seconds, function () {
    return User::all();
});
```

Error Handling

- Configure in app/Exceptions/Handler.php
- Custom error pages in resources/views/errors

Recommended Best Practices

1. **Follow Laravel conventions** - Work with the framework, not against it
2. **Keep controllers thin** - Move business logic to services/models
3. **Use form requests** for complex validation logic
4. **Repository pattern** for database abstraction when needed
5. **Write tests** for critical functionality
6. **Use Laravel Mix** for asset compilation
7. **Use configuration files** instead of hardcoding values
8. **Use migrations** for all database changes
9. **Create reusable components** with Blade components
10. **Follow PSR standards** for code style