Guía Completa para Desarrollar una Aplicación de Pronóstico del Clima para Agricultores

Aprende a crear desde cero un sistema completo de pronóstico meteorológico especializado para el sector agrícola, utilizando PHP, Laravel y bases de datos modernas. Esta guía te llevará paso a paso desde la configuración inicial hasta el despliegue final.

Introducción al Proyecto

¡Bienvenido a tu proyecto final! Vas a construir una aplicación web que ayudará a los agricultores a tomar decisiones importantes sobre sus cultivos. Es como crear un asistente digital que les dice cuándo sembrar, regar o proteger sus plantas.

Esta aplicación conectará con servicios de clima en internet, guardará información importante en bases de datos y mostrará alertas útiles a los usuarios. Al final, tendrás un sistema completo que funciona como las aplicaciones profesionales.

No necesitas ser un experto para comenzar. Esta guía te explica cada paso con palabras sencillas y ejemplos que puedes copiar y pegar directamente en tu computadora.

¿Qué Vas a Aprender?

Configuración del Entorno

Instalar PHP, Laravel, PostgreSQL y todas las herramientas necesarias para desarrollar

Base de Datos

Crear tablas, relaciones y consultas para almacenar información de usuarios y clima

Obtener datos del clima en tiempo real desde servicios como OpenWeatherMap

Sistema de Alertas

Enviar notificaciones automáticas cuando hay riesgos climáticos

Conexión a APIs

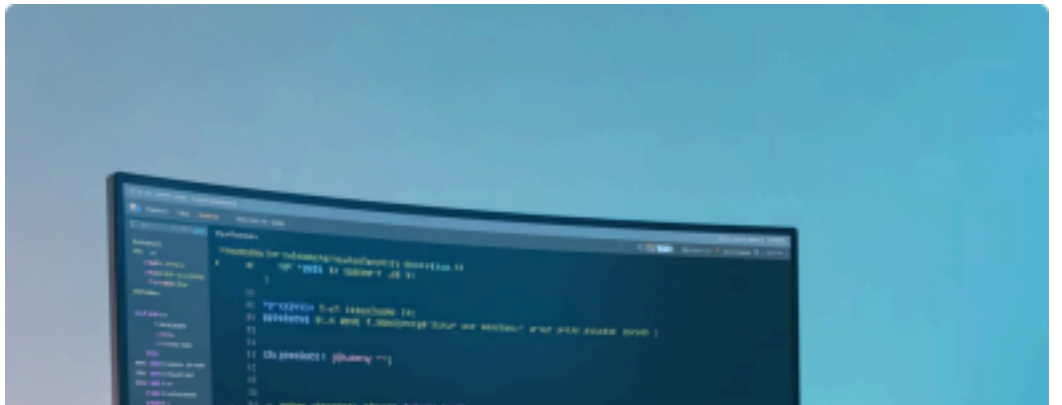
Antes de Empezar

Requisitos Previos

Saber usar una computadora básicamente

Conocer cómo abrir una terminal o línea de comandos

Tener conexión a internet



Ganas de aprender (¡lo más importante!)

CONSEJO: Si nunca has programado antes, no te preocupes. Esta guía está diseñada para que cualquier persona pueda seguirla paso a paso.

Mapa del Proyecto

Preparación del Entorno

Instalar todas las herramientas necesarias en tu computadora

03

Diseño de la Base de Datos

Crear las tablas para usuarios, cultivos y datos del clima

05

Sistema de Autenticación

Crear login y registro para diferentes tipos de usuarios

07

Sistema de Alertas

Configurar notificaciones automáticas

Creación del Proyecto Laravel

Configurar la estructura básica de la aplicación

04

Integración con APIs

Conectar con servicios externos para obtener datos meteorológicos

06

Interfaces de Usuario

Diseñar las pantallas que verán los agricultores

08

Pruebas y Despliegue

Verificar que todo funciona y ponerlo en internet

02

Lección 1: ¿Qué es PHP y Laravel?

Qué es

PHP es un lenguaje de programación que sirve para crear sitios web. Laravel es como un conjunto de herramientas que hace más fácil usar PHP. Es como tener una caja de herramientas completa en lugar de construir cada herramienta desde cero.

Para qué sirve

Crear sitios web que se conectan a bases de datos

Procesar formularios y datos de usuarios

Conectar con servicios externos como APIs del clima

Lección 2: Instalación de PHP

Paso a Paso

Ve a **php.net** en tu navegador1.

Haz clic en "Downloads"2.

Descarga la versión más reciente3.

Ejecuta el instalador4.

Sigue las instrucciones en pantalla5.

CUIDADO: Asegúrate de descargar PHP versión 8.1 o superior. Las versiones anteriores no funcionarán con Laravel moderno.

Verificar Instalación de

PHP Ejemplo Mínimo Reproducible

```
# Abre tu terminal y escribe:
php -v
```

Salida Esperada

```
PHP 8.2.0 (cli) (built: Dec 7 2022 17:12:36) (NTS)
Copyright (c) The PHP Group
Zend Engine v4.2.0
```

Errores Comunes

Error: "php no reconocido" [3] **Solución:** PHP no está en el PATH del sistema

Error: Versión muy antigua [3] **Solución:** Desinstalar y descargar versión nueva

Lección 3: Instalación de Composer

Qué es

Composer es como un administrador de paquetes para PHP. Imagina que es como una tienda donde puedes descargar automáticamente herramientas y librerías que necesitas para tu proyecto.

Paso a Paso

1. Ve a **getcomposer.org**

   Haz clic en "Download"2.

   Descarga el instalador para tu sistema operativo3.

   Ejecuta el archivo descargado4.

   Sigue el asistente de instalación5.

Lección 4: Instalación de Laravel

Ejemplo Mínimo Reproducible

```
# En tu terminal, escribe estos comandos uno por uno:

# Instalar Laravel globalmente
composer global require laravel/installer
```

```
# Crear un nuevo proyecto
laravel new clima-agricola

# Entrar al directorio del proyecto
cd clima-agricola

# Iniciar el servidor de desarrollo
php artisan serve
```

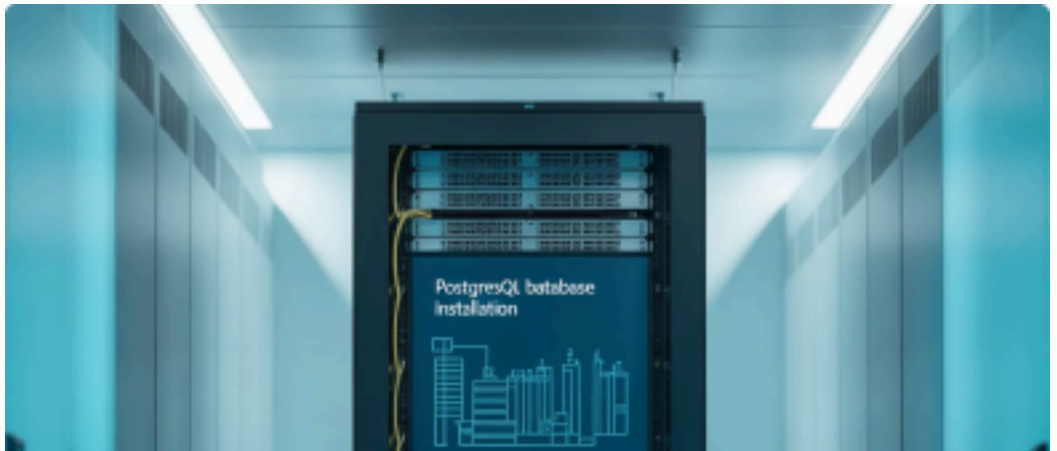
Salida Esperada

```
Laravel development server started: http://127.0.0.1:8000
```

Lección 5: Instalación de PostgreSQL

Para qué sirve

PostgreSQL es una base de datos. Es como un archivero digital donde guardaremos toda la información: usuarios, datos del clima, alertas, etc.



Paso a Paso

1. Ve a **postgresql.org**

   Haz clic en "Download"2.

   Selecciona tu sistema operativo3.

   Descarga e instala4.

   Anota la contraseña que elijas5.

Configuración de la Base de Datos

Ejemplo Mínimo Reproducible

```
# Crear una nueva base de datos
# Ejecuta estos comandos en la terminal de PostgreSQL:

CREATE DATABASE clima_agricola;
CREATE USER clima_user WITH PASSWORD 'tu_contraseña_segura';
GRANT ALL PRIVILEGES ON DATABASE clima_agricola TO clima_user;
```

Configurar Laravel

Abre el archivo `.env` en tu proyecto y modifica estas líneas:

```
DB_CONNECTION=pgsql
DB_HOST=127.0.0.1
DB_PORT=5432
DB_DATABASE=clima_agricola
DB_USERNAME=clima_user
DB_PASSWORD=tu_contraseña_segura
```

Lección 6: Estructura del Proyecto

app/Models

Aquí van las clases que representan nuestras tablas de la base de datos

database/migrations

Scripts para crear y modificar tablas en la base de datos

app/Http/Controllers

Contiene la lógica de la aplicación, como procesar formularios

resources/views

Las plantillas HTML que verán los usuarios

Creando Nuestro Primer

Modelo Ejemplo Mínimo Reproducible

```
# Crear el modelo Usuario con migración
php artisan make:model Usuario -m

# Crear el modelo Cultivo con migración
php artisan make:model Cultivo -m

# Crear el modelo DatoClimatico con migración
php artisan make:model DatoClimatico -m
```

Salida Esperada

```
Model created successfully.
Migration created successfully.
```

Estos comandos crean archivos automáticamente en las carpetas correctas de tu proyecto.

Diseñando la Tabla

Usuarios Código para la Migración

Abre `database/migrations/xxxx_create_usuarios_table.php` y escribe:

```
id(); // ID único
 $table->string('nombre'); // Nombre completo
 $table->string('email')->unique(); // Email único
 $table->string('telefono')->nullable(); // Teléfono opcional
 $table->enum('tipo', ['agricultor', 'tecnico', 'administrador']);
 $table->string('ubicacion'); // Ciudad o región
 $table->decimal('latitud', 10, 8); // Coordenadas GPS
 $table->decimal('longitud', 11, 8); // Coordenadas GPS
 $table->timestamp('email_verified_at')->nullable();
 $table->string('password'); // Contraseña encriptada
 $table->timestamps(); // created_at y updated_at
 });
}
```

Diseñando la Tabla Cultivos

```
id();
 $table->string('nombre'); // Ej: "Maíz", "Trigo"
 $table->text('descripcion'); // Descripción del cultivo
```

```
 $table->integer('dias_siembra'); // Días óptimos para sembrar
$table->integer('dias_cosecha'); // Días desde siembra a cosecha
$table->decimal('temp_min', 5, 2); // Temperatura mínima óptima
$table->decimal('temp_max', 5, 2); // Temperatura máxima óptima
$table->integer('humedad_min'); // Humedad mínima requerida
$table->integer('humedad_max'); // Humedad máxima recomendada
$table->boolean('activo')->default(true); // Si está disponible
$table->timestamps();
 });
}
```

Diseñando la Tabla Datos Climáticos

```
id();
 $table->string('ciudad'); // Ciudad de los datos
 $table->decimal('latitud', 10, 8); // Coordenadas GPS
 $table->decimal('longitud', 11, 8); // Coordenadas GPS
 $table->decimal('temperatura', 5, 2); // Temperatura actual
 $table->integer('humedad'); // Porcentaje de humedad
 $table->decimal('velocidad_viento', 5, 2); // Velocidad en km/h
 $table->integer('prob_lluvia'); // Probabilidad de lluvia %
 $table->string('condicion'); // "soleado", "nublado", etc
 $table->date('fecha'); // Fecha del pronóstico
 $table->string('fuente')->default('OpenWeather'); // API usada
```

```
    $table->timestamps();
    });
}
```

Ejecutando las Migraciones Comando para Crear Tablas

```
# Este comando ejecuta todas las migraciones pendientes
php artisan migrate
```

Salida Esperada

```
Migrating: 2024_01_01_000001_create_usuarios_table
Migrated: 2024_01_01_000001_create_usuarios_table (32.51ms)
Migrating: 2024_01_01_000002_create_cultivos_table
Migrated: 2024_01_01_000002_create_cultivos_table (28.32ms)
Migrating: 2024_01_01_000003_create_dato_climaticos_table
Migrated: 2024_01_01_000003_create_dato_climaticos_table (35.18ms)
```

¡Éxito! Si ves este mensaje, tus tablas se crearon correctamente en la base de datos.

Lección 7: Configurando los Modelos

Modelo Usuario

Abre `app/Models/Usuario.php` y escribe:

```
    'datetime',
    'latitud' => 'decimal:8',
    'longitud' => 'decimal:8',
    ];
}
```

Modelo Cultivo

```
     'decimal:2',
    'temp_max' => 'decimal:2',
```

```
 'activo' => 'boolean',
 ];

 // Método para verificar si las condiciones son óptimas  public
function esOptimoParaClima($temperatura, $humedad)  {
 return $temperatura >= $this->temp_min
 && $temperatura <= $this->temp_max
 && $humedad >= $this->humedad_min
 && $humedad <= $this->humedad_max;
 }
}
```

Lección 8: Obteniendo Datos del Clima Registrarse en

OpenWeatherMap

1. Ve a **openweathermap.org**

Haz clic en "Sign Up"2.

Crea tu cuenta gratuita3.

Ve a "API Keys"4.

Copia tu clave API5.

Agrega tu API key al archivo `.env`:

```
OPENWEATHER_API_KEY=tu_api_key_aqui
```

Creando el Servicio del Clima Crear el Servicio

```
# Crear una nueva clase de servicio
php artisan make:class App/Services/ClimaService
```

En `app/Services/ClimaService.php`:

```
apiKey = config('services.openweather.key');
 }

 public function obtenerClimaPorCiudad($ciudad)
 {
 $response = Http::get("{$this->baseUrl}/weather", [
 'q' => $ciudad,
 'appid' => $this->apiKey,
 'units' => 'metric', // Para temperaturas en Celsius
 'lang' => 'es' // Respuestas en español
 ]);

 return $response->json();
 }
}
```

Configurando el Servicio

En `config/services.php`, agrega:

```
'openweather' => [
 'key' => env('OPENWEATHER_API_KEY'),
],
```

Ejemplo de Uso del Servicio

```
obtenerClimaPorCiudad('Mosquera, Colombia');

// Los datos incluyen:
// - temperatura actual
// - humedad
// - velocidad del viento
// - descripción del clima
```

Lección 9: Creando el Controlador Principal

```
# Crear el controlador
php artisan make:controller ClimaController
```

En app/Http/Controllers/ClimaController.php:

```
    climaService = $climaService;
    }

    public function mostrarDashboard()
    {
    // Obtener clima actual para Mosquera
    $clima = $this->climaService->obtenerClimaPorCiudad('Mosquera, Colombia');

    return view('dashboard', compact('clima'));
    }
}
```

Creando la Vista del Dashboard

Crea resources/views/dashboard.blade.php:

Dashboard Climático

Clima Actual - {{ $clima['name'] }}

{{ $clima['main']['temp'] }}°C

{{ $clima['weather'][0]['description'] }}

Humedad: {{ $clima['main']['humidity'] }}%

Viento: {{ $clima['wind']['speed'] }} m/s

Comandos Artisan Esenciales para el Proyecto

Contenido detallado sobre comandos Artisan más útiles:

Comandos de Desarrollo Diario

```
# Ver todas las rutas disponibles
php artisan route:list

# Limpiar todas las cachés
php artisan optimize:clear

# Crear un controlador con recursos  completos
php artisan make:controller
ProductoController --resource

# Crear modelo con migración, factory y  seeder
php artisan make:model Producto -mfs

# Rollback de migraciones (deshacer  cambios)
php artisan migrate:rollback

# Ver estado de migraciones
php artisan migrate:status
```

Comandos de Base de Datos

```
# Refrescar base de datos con seeders php artisan migrate:fresh --seed
```

```
# Crear seeder específico
php artisan make:seeder UsuariosSeeder

# Ejecutar seeder específico
php artisan db:seed --
class=UsuariosSeeder

# Crear factory para generar datos de  prueba
php artisan make:factory
ProductoFactory
```

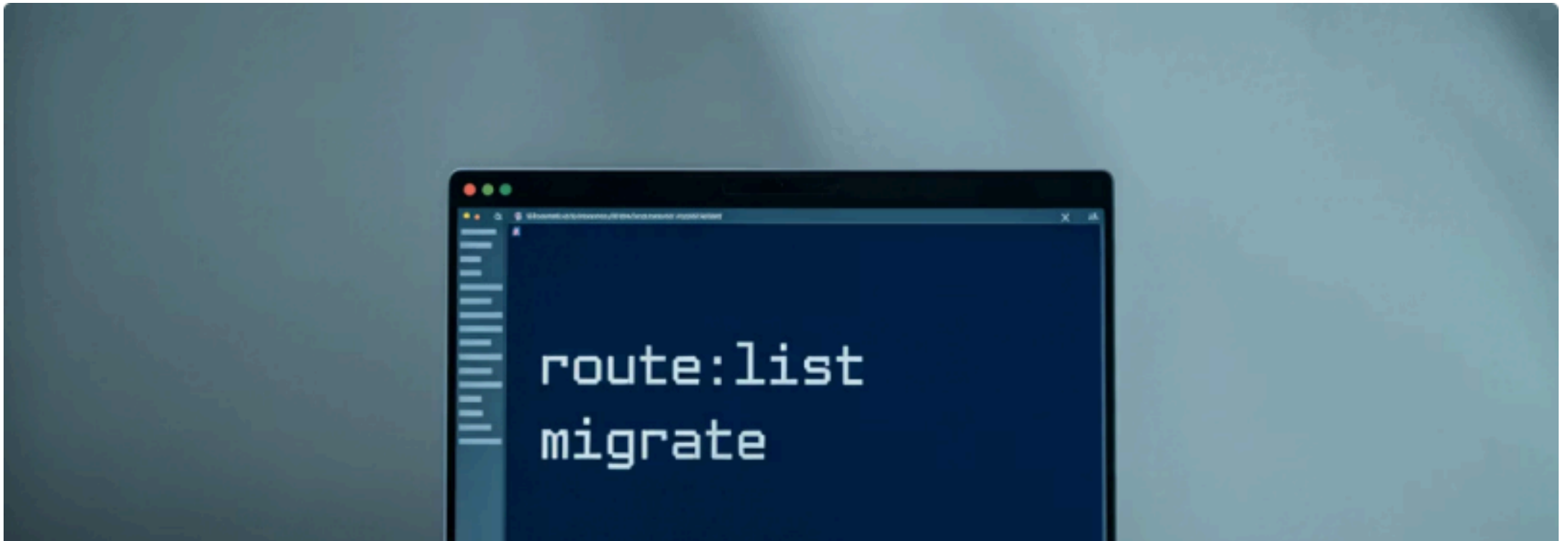
Comandos de Mantenimiento

```
# Poner aplicación en modo
mantenimiento
php artisan down

# Quitar modo mantenimiento
php artisan up

# Limpiar logs antiguos
php artisan log:clear

# Generar nueva clave de aplicación php artisan key:generate
```


route:list
migrate

Funciones Avanzadas de Eloquent ORM Contenido sobre

funciones especiales de Eloquent con ejemplos prácticos:

1

Consultas Avanzadas

```
// Buscar con múltiples condiciones $usuarios = Usuario::where('activo', true)  ->where('tipo', 'agricultor')
 ->whereDate('created_at', '>=', now()- >subDays(30))
 ->get();

// Búsqueda con OR
$productos = Producto::where('nombre',  'LIKE', '%maíz%')
 ->orWhere('descripcion', 'LIKE',  '%maíz%')
 ->get();

// Consultas con relaciones (Eager  Loading)
$usuarios = Usuario::with(['cultivos',  'alertas'])
 ->where('ubicacion', 'Mosquera')  ->get();

// Contar registros relacionados
$usuarios = Usuario::withCount('alertas')  ->having('alertas_count', '>', 5)
```

```
 ->get();
```

2

Funciones de Agregación

```
// Estadísticas básicas
$stats = [
 'total_usuarios' => Usuario::count(),  'promedio_temp' =>
DatoClimatico::avg('temperatura'),  'temp_maxima' =>
DatoClimatico::max('temperatura'),  'temp_minima' =>
DatoClimatico::min('temperatura'),  'suma_hectareas' =>
Cultivo::sum('hectareas')
];

// Agrupar por fecha
$ventasPorMes =
Venta::selectRaw('MONTH(created_at) as  mes, SUM(total) as total')
 ->groupBy('mes')
 ->orderBy('mes')
 ->get();
```

3

Scopes Personalizados

```
// En el modelo Usuario
public function scopeActivos($query) {
 return $query->where('activo', true); }

public function scopePorTipo($query,  $tipo)
{
 return $query->where('tipo', $tipo); }

// Uso de scopes
$agricultores = Usuario::activos()- >porTipo('agricultor')->get();
```

Validaciones Avanzadas y Personalizadas

Contenido sobre validaciones especiales en Laravel:

Validaciones Básicas Mejoradas

```
// En el controlador
public function store(Request $request)
{
 $validated = $request->validate([
 'nombre' => 'required|string|min:3|max:50|regex:/^[a-zA-ZáéíóúÁÉÍÓÚñÑ\s]+$/',  'email' =>
```

```
'required|email|unique:usuarios,email|ends_with:.com,.co,.org',  'telefono' => 'nullable|regex:/^[0-9]{10}$/',
 'fecha_nacimiento' => 'required|date|before:today|after:1900-01-01',  'salario' => 'required|numeric|min:1000000|max:50000000',
 'imagen' => 'nullable|image|mimes:jpeg,png,jpg|max:2048', // 2MB máximo  ]);
}
```

Validaciones Condicionales

```
$rules = [
 'tipo_usuario' => 'required|in:agricultor,tecnico,administrador',
 'hectareas' => 'required_if:tipo_usuario,agricultor|numeric|min:0.1',  'titulo_profesional' => 'required_if:tipo_usuario,tecnico|string',
 'nivel_acceso' => 'required_if:tipo_usuario,administrador|in:1,2,3,4,5', ];
```

```
// Validación con closure personalizada
'password' => [
 'required',
 'string',
 'min:8',
 function ($attribute, $value, $fail) {
 if (!preg_match('/[A-Z]/', $value)) {
 $fail('La contraseña debe tener al menos una mayúscula.');
 }
```

```
 if (!preg_match('/[0-9]/', $value)) {
 $fail('La contraseña debe tener al menos un número.');
 }
 if (!preg_match('/[!@#$%^&*]/', $value)) {
 $fail('La contraseña debe tener al menos un carácter especial.');  }
 },
],
```

Crear Reglas de Validación Personalizadas

```
// Crear regla personalizada
php artisan make:rule ValidarCedulaColombia
```

```
// En app/Rules/ValidarCedulaColombia.php
public function passes($attribute, $value)
{
 // Validar que sea un número de cédula válido
 return preg_match('/^[0-9]{8,10}$/', $value) && $value > 1000000; }

public function message()
{
```

```
 return 'El número de cédula no es válido para Colombia.';
}



// Uso de la regla personalizada
$request->validate([
 'cedula' => ['required', new ValidarCedulaColombia],
]);
```

Middleware Personalizado y Seguridad

Contenido sobre middleware avanzado y seguridad:

1

Crear Middleware Personalizado

```
// Crear middleware
php artisan make:middleware
VerificarTipoUsuario

// En
```

app/Http/Middleware/VerificarTipoUsuar io.php

```
public function handle(Request $request,  Closure $next, ...$tipos)
{
 if (!auth()->check()) {
 return redirect('/login');
 }

 $usuario = auth()->user();

 if (!in_array($usuario->tipo, $tipos)) {  abort(403, 'No tienes permisos para  acceder a esta sección');
 }

 return $next($request);
}

// Registrar en app/Http/Kernel.php protected $routeMiddleware = [
 'tipo.usuario' =>
\App\Http\Middleware\VerificarTipoUsua rio::class,
];

// Usar en rutas
Route::get('/admin',
[AdminController::class, 'index'])
 -
```

```
>middleware('tipo.usuario:administrador ');

Route::get('/cultivos',
[CultivoController::class, 'index'])
 -
>middleware('tipo.usuario:agricultor,tecn ico');
```

2

Middleware de Logging y Auditoría

```
// Middleware para registrar acciones public function handle(Request $request,  Closure $next)
{
 $response = $next($request);

 // Registrar la acción del usuario  if (auth()->check()) {
 Log::info('Acción de usuario', [  'usuario_id' => auth()->id(),  'accion' => $request->method() . ' '  . $request->path(),
 'ip' => $request->ip(),
 'user_agent' => $request-
>userAgent(),
 'timestamp' => now()
 ]);
 }
```

```
 return $response;
}
```

3

Protección contra Ataques

```
// Middleware anti-spam
public function handle(Request $request,  Closure $next)
{
 $key = 'rate_limit:' . $request->ip();  $attempts = Cache::get($key, 0);
 if ($attempts >= 10) {
 abort(429, 'Demasiadas solicitudes.  Intenta en 1 minuto.');
 }

 Cache::put($key, $attempts + 1, 60); //  1 minuto

 return $next($request);
}
```

Trabajando con Colas (Queues) y Jobs

Contenido sobre procesamiento asíncrono con ejemplos prácticos:

Configurar Colas

```
# Instalar Redis para colas
composer require predis/predis

# En .env
QUEUE_CONNECTION=redis

# Crear tabla para colas en base de datos (alternativa)
php artisan queue:table
php artisan migrate
```

Crear y Usar Jobs

```
// Crear un job
php artisan make:job EnviarAlertaClima

// En app/Jobs/EnviarAlertaClima.php
class EnviarAlertaClima implements ShouldQueue
{
 use Dispatchable, InteractsWithQueue, Queueable, SerializesModels;
```

```
protected $usuario;
protected $mensaje;

public function __construct(Usuario $usuario, $mensaje)
{
$this->usuario = $usuario;
$this->mensaje = $mensaje;
}

public function handle()
{
// Enviar email
Mail::to($this->usuario->email)->send(new AlertaClimaEmail($this->mensaje));
// Enviar SMS si tiene teléfono
if ($this->usuario->telefono) {
// Lógica para enviar SMS
$this->enviarSMS($this->usuario->telefono, $this->mensaje);
}

// Registrar en logs
Log::info("Alerta enviada a usuario {$this->usuario->id}");
}
```

```
 public function failed(Exception $exception)
 {
 Log::error("Error enviando alerta: " . $exception->getMessage());
 }
}

// Despachar el job desde un controlador
EnviarAlertaClima::dispatch($usuario, $mensajeAlerta);

// Despachar con delay
EnviarAlertaClima::dispatch($usuario, $mensaje)->delay(now()->addMinutes(5));
```

Jobs con Prioridades y Colas Específicas

```
// Job con cola específica
EnviarAlertaClima::dispatch($usuario, $mensaje)->onQueue('alertas');

// Job con prioridad alta
EnviarAlertaClima::dispatch($usuario, $mensaje)->onQueue('high-priority');

// Ejecutar worker para cola específica
php artisan queue:work --queue=high-priority,alertas,default
```

Eventos y Listeners para Automatización Contenido sobre

eventos del sistema con ejemplos prácticos:

Crear Eventos y Listeners

```
// Crear evento
php artisan make:event
UsuarioRegistrado

// En app/Events/UsuarioRegistrado.php class UsuarioRegistrado
{
 use Dispatchable,
InteractsWithSockets, SerializesModels;
 public $usuario;

 public function __construct(Usuario  $usuario)
 {
 $this->usuario = $usuario;
 }
}

// Crear listener
```

```
php artisan make:listener
EnviarEmailBienvenida

// En
app/Listeners/EnviarEmailBienvenida.ph p
class EnviarEmailBienvenida
{
 public function
handle(UsuarioRegistrado $event)  {
 $usuario = $event->usuario;

 // Enviar email de bienvenida  Mail::to($usuario->email)->send(new  BienvenidaEmail($usuario));

 // Crear registro de actividad  Log::info("Email de bienvenida  enviado a: " . $usuario->email);

 // Asignar cultivos por defecto según  ubicación
 $this-
>asignarCultivosPorDefecto($usuario);  }

 private function
asignarCultivosPorDefecto($usuario)  {
 $cultivosRecomendados =
Cultivo::where('region', $usuario- >ubicacion)
```

```
 ->where('recomendado', true)  ->get();

 foreach ($cultivosRecomendados as  $cultivo) {
 $usuario->cultivos()-
>attach($cultivo->id);
 }
 }
}
```

Registrar Eventos

```
// En
app/Providers/EventServiceProvider.php protected $listen = [
 UsuarioRegistrado::class => [
 EnviarEmailBienvenida::class,  CrearPerfilCompleto::class,
 AsignarRolPorDefecto::class,  ],

 AlertaCreada::class => [
 EnviarNotificacionPush::class,  RegistrarEnHistorial::class,
 ],

 TemperaturaExtrema::class => [  EnviarAlertaUrgente::class,
 NotificarAutoridades::class,
 ],
];
```

```
// Disparar evento desde controlador event(new UsuarioRegistrado($usuario));

// O usando la función helper
UsuarioRegistrado::dispatch($usuario);
```

Eventos del Sistema Laravel

```
// Escuchar eventos de Eloquent // En el modelo Usuario
protected static function booted() {
 static::created(function ($usuario) {  Log::info("Nuevo usuario creado:  {$usuario->nombre}");
 event(new
UsuarioRegistrado($usuario));
 });

 static::updated(function ($usuario) {  if ($usuario-
>wasChanged('ubicacion')) {
 // Actualizar recomendaciones de  cultivos
 event(new
UbicacionCambiada($usuario));
 }
 });

 static::deleting(function ($usuario) {  // Limpiar datos relacionados antes  de eliminar
```

```
 $usuario->alertas()->delete();  $usuario->cultivos()->detach();  });
}
```

Helpers y Funciones Útiles Personalizadas Contenido sobre

funciones auxiliares personalizadas:

Crear Helpers Personalizados

```
// Crear archivo app/Helpers/ClimaHelper.php
class ClimaHelper
{
 public static function convertirTemperatura($celsius, $unidad =  'F')
 {
 switch ($unidad) {
 case 'F':
 return ($celsius * 9/5) + 32;
 case 'K':
 return $celsius + 273.15;
 default:
 return $celsius;
 }
 }
```

```
public static function interpretarCondicion($codigo)  {
$condiciones = [
'01d' => 'Soleado',
'02d' => 'Parcialmente nublado',
'03d' => 'Nublado',
'04d' => 'Muy nublado',
'09d' => 'Lluvia ligera',
'10d' => 'Lluvia',
'11d' => 'Tormenta',
'13d' => 'Nieve',
'50d' => 'Niebla'
];

return $condiciones[$codigo] ?? 'Desconocido';  }

public static function calcularIndiceUV($latitud, $fecha = null)  {
$fecha = $fecha ?? now();
$diaDelAno = $fecha->dayOfYear;

// Fórmula simplificada para índice UV
$indice = abs(sin(deg2rad($latitud))) *
(1 + 0.3 * cos(2 * pi() * $diaDelAno / 365));

return round($indice * 10, 1);
```

```
 }
}

// Registrar en composer.json

"autoload": {
 "files": [
 "app/Helpers/ClimaHelper.php"
 ]
}

// Ejecutar: composer dump-autoload
```

Funciones Helper Globales

```
// Crear archivo app/helpers.php
if (!function_exists('formatear_temperatura')) {
 function formatear_temperatura($temp, $unidad = 'C')  {
 return number_format($temp, 1) . '°' . $unidad;
 }
}

if (!function_exists('es_temperatura_extrema')) {
 function es_temperatura_extrema($temp)
 {
```

```
 return $temp < 0 || $temp > 35;
 }
}

if (!function_exists('calcular_sensacion_termica')) {
 function calcular_sensacion_termica($temp, $humedad, $viento)  {
 // Fórmula de sensación térmica
 $sensacion = $temp + (0.33 * ($humedad / 100) * 6.105 *  exp(17.27 * $temp / (237.7 + $temp)))
 - (0.70 * $viento) - 4.00;

 return round($sensacion, 1);
 }
}

if (!function_exists('obtener_recomendacion_cultivo')) {  function obtener_recomendacion_cultivo($temperatura,  $humedad)
 {
 if ($temperatura >= 20 && $temperatura <= 30 && $humedad  >= 60) {
 return 'Condiciones ideales para maíz y frijol';  } elseif ($temperatura >= 15 && $temperatura <= 25 &&  $humedad >= 70) {
 return 'Perfecto para cultivos de hoja verde';
 } elseif ($temperatura >= 25 && $temperatura <= 35 &&  $humedad <= 50) {
 return 'Buenas condiciones para cultivos resistentes a  sequía';
 } else {
 return 'Consulta con un técnico agrícola';
 }
```

```
 }
}

// Uso en vistas Blade

{{ formatear_temperatura($clima['main']['temp']) }}
{{ obtener_recomendacion_cultivo($temp, $humedad) }}
```

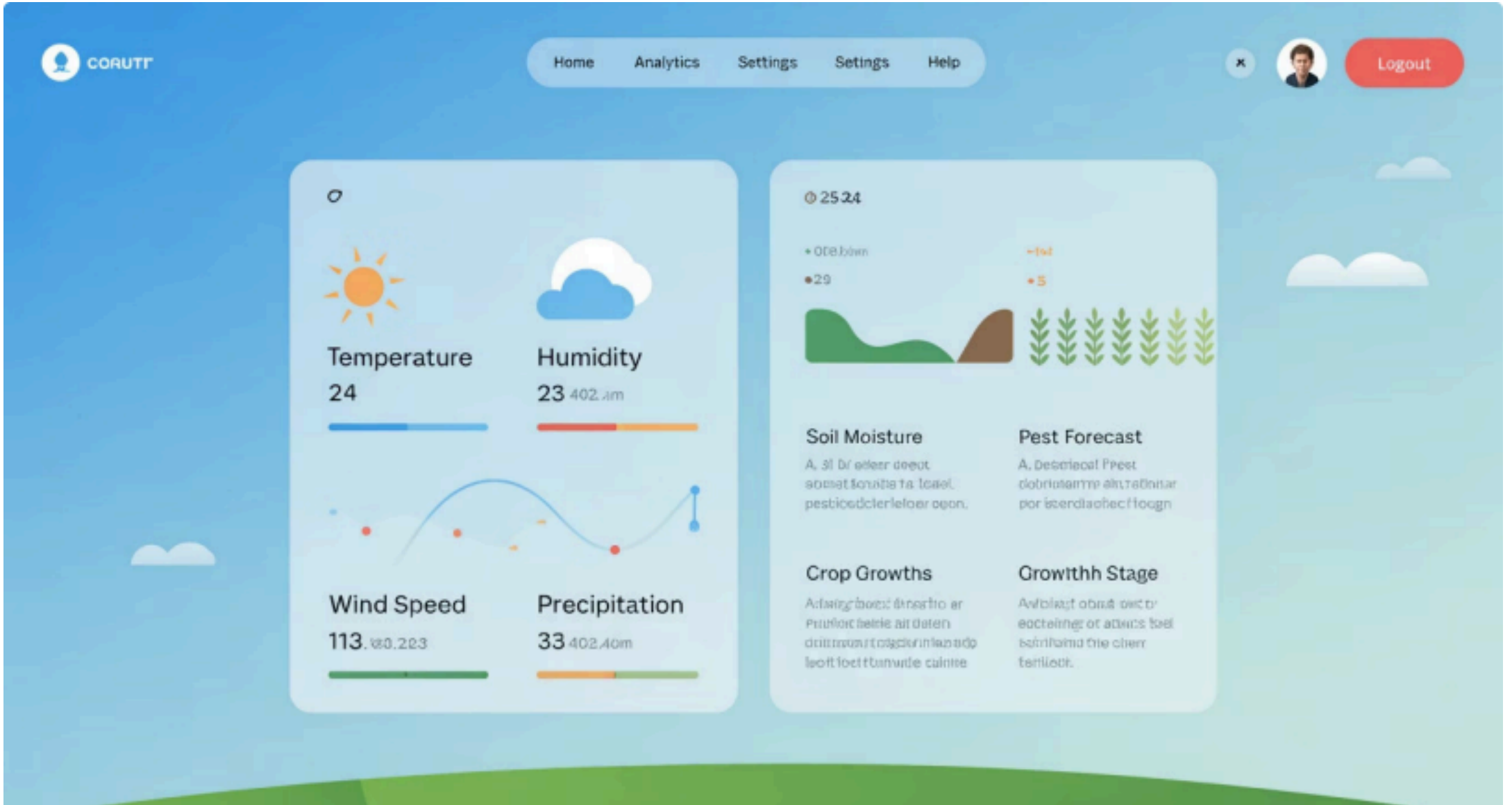
Configurando las Rutas

En `routes/web.php`:

Probar la Aplicación

```
# Iniciar el servidor
php artisan serve

# Visita http://127.0.0.1:8000 en tu navegador
```


Home
Analytics
Settings
Setings
Help
Logout
Temperature
24
Humidity
23
Wind Speed
Precipitation
33
Soil Moisture
Pest Forecast
Crop Growths
Growthh Stage

Lección 10: Sistema de Autenticación

Instalar Laravel Breeze

```
# Instalar Breeze (sistema de autenticación)
composer require laravel/breeze --dev

# Instalar los archivos de autenticación
php artisan breeze:install

# Instalar dependencias de npm
npm install && npm run dev

# Ejecutar migraciones
php artisan migrate
```

Qué incluye Breeze

- Páginas de login y registro
- Reseteo de contraseñas

Verificación de email

Dashboard básico

Personalizando el Registro

Modifica app/Http/Controllers/Auth/RegisteredUserController.php:

```
public function store(Request $request)
{
 $request->validate([
 'nombre' => 'required|string|max:255',
 'email' => 'required|string|email|max:255|unique:usuarios',  'telefono' => 'nullable|string|max:20',
 'tipo' => 'required|in:agricultor,tecnico,administrador',
 'ubicacion' => 'required|string|max:255',
 'latitud' => 'required|numeric',
 'longitud' => 'required|numeric',
 'password' => 'required|string|confirmed|min:8',
 ]);

 $user = Usuario::create([
 'nombre' => $request->nombre,
 'email' => $request->email,
 'telefono' => $request->telefono,
```

```
        'tipo' => $request->tipo,
        'ubicacion' => $request->ubicacion,
        'latitud' => $request->latitud,
        'longitud' => $request->longitud,
        'password' => Hash::make($request->password),
        ]);

        return redirect()->route('dashboard');
    }
```

Formulario de Registro Personalizado

Modifica `resources/views/auth/register.blade.php`:

```
    @csrf

    <!-- Nombre Completo -->
    <div>
        <x-input-label for="name" :value="__('Nombre Completo')" />
        <x-text-input id="name" class="block mt-1 w-full" type="text" name="name" :value="old('name')" required autofocus />
        <x-input-label for="email" :value="__('Email')" />
```

Tipo de Usuario Selecciona una opción Agricultor Técnico Agrícola Administrador

Lección 11: Sistema de Alertas Crear Migración para Alertas

php artisan make:model Alerta -m

En la migración:

```
public function up()
{
 Schema::create('alertas', function (Blueprint $table) {
 $table->id();
 $table->foreignId('usuario_id')->constrained('usuarios');
 $table->string('tipo'); // 'helada', 'lluvia', 'sequia'
 $table->string('titulo');
 $table->text('mensaje');
 $table->enum('severidad', ['baja', 'media', 'alta', 'critica']);
 $table->boolean('enviada')->default(false);
 $table->timestamp('fecha_evento');
 $table->timestamps();
 });
```

}

Creando el Servicio de Alertas

php artisan make:class App/Services/AlertaService

En `app/Services/AlertaService.php`:

```
$usuarioId,
'tipo' => $tipo,
'titulo' => $titulo,
'mensaje' => $mensaje,
'severidad' => $severidad,
'fecha_evento' => now(),
]);
}

public function verificarCondicionesClimaticas($temperatura, $humedad)  {
$usuarios = Usuario::all();

foreach ($usuarios as $usuario) {
// Verificar helada
if ($temperatura < 2) {
$this->crearAlerta(
$usuario->id,
```

```
        'helada',
        'Alerta de Helada',
        "Se espera una helada esta noche. Temperatura: {$temperatura}°C. Protege tus cultivos.",  'alta'
        );
        }
        }
        }
       }
```

Comando Artisan para Verificar Clima

php artisan make:command VerificarClima

En `app/Console/Commands/VerificarClima.php`:

```
    obtenerClimaPorCiudad('Mosquera, Colombia');
     $temperatura = $clima['main']['temp'];
     $humedad = $clima['main']['humidity'];

     $alertaService->verificarCondicionesClimaticas($temperatura, $humedad);
```

```
 $this->info("Verificación completada. Temp: {$temperatura}°C, Humedad: {$humedad}%");
}
}
```

Automatización con Tareas Programadas

En app/Console/Kernel.php:

```
protected function schedule(Schedule $schedule)
{
 // Verificar clima cada 30 minutos
 $schedule->command('clima:verificar')
 ->everyThirtyMinutes();

 // Limpiar alertas antiguas diariamente
 $schedule->command('alertas:limpiar')
 ->daily();
}
```

Para probar manualmente:

```
# Ejecutar el comando una vez
php artisan clima:verificar

# Ejecutar todas las tareas programadas
php artisan schedule:run
```

Lección 12: API REST Crear Controlador API

```
php artisan make:controller Api/ClimaApiController
```

En `app/Http/Controllers/Api/ClimaApiController.php`:

```
climaService = $climaService;
 }

 public function obtenerClima(Request $request)
 {
 $ciudad = $request->get('ciudad', 'Mosquera, Colombia');

 try {
 $clima = $this->climaService->obtenerClimaPorCiudad($ciudad);
```

```
        return response()->json([
        'success' => true,
        'data' => $clima
        ]);
        } catch (\Exception $e) {
        return response()->json([
        'success' => false,
        'message' => 'Error al obtener datos del clima'
        ], 500);
        }
        }
       }
```

Rutas API

En routes/api.php:

```
  group(function () {
   Route::get('/clima', [ClimaApiController::class, 'obtenerClima']);
  Route::get('/alertas', [AlertaApiController::class, 'obtenerAlertas']);
  Route::post('/cultivos', [CultivoApiController::class, 'store']); });
```

Probar la API

```
# Con curl en la terminal:
curl -X GET "http://127.0.0.1:8000/api/clima?ciudad=Bogotá"
\  -H "Accept: application/json"
```

Lección 13: Mapas Interactivos Integración con Leaflet

En tu vista dashboard.blade.php, agrega:

Mapa con Datos Dinámicos

En tu controlador, pasa múltiples ciudades:

```
public function mostrarMapa()
{
 $ciudades = [
 ['nombre' => 'Mosquera', 'lat' => 4.7058, 'lng' => -74.2302],  ['nombre' => 'Bogotá', 'lat' => 4.7110, 'lng' => -74.0721],
```

```
['nombre' => 'Facatativá', 'lat' => 4.8144, 'lng' => -74.3556]
];

$datosClima = [];
foreach ($ciudades as $ciudad) {
$clima = $this->climaService->obtenerClimaPorCiudad($ciudad['nombre']);  $datosClima[] = [
'nombre' => $ciudad['nombre'],
'lat' => $ciudad['lat'],
'lng' => $ciudad['lng'],
'temp' => $clima['main']['temp'],
'descripcion' => $clima['weather'][0]['description']
];
}

return view('mapa', compact('datosClima'));
}
```

Lección 14: Reportes y Gráficos

Instalar librerías para gráficos

```
# Instalar Chart.js via CDN en tu vista
```

En resources/views/reportes.blade.php:

Generar Reportes PDF

Instalar DomPDF

```
composer require barryvdh/laravel-dompdf
```

Controlador de Reportes

```
=', now()->subDays(30))
 ->orderBy('fecha')
 ->get();

 $pdf = PDF::loadView('reportes.pdf', compact('datos'));

 return $pdf->download('reporte-clima-mensual.pdf');
 }
}
```

Crea la vista `resources/views/reportes/pdf.blade.php` con el formato que desees para el PDF.

Lección 15: Pruebas de Software Crear una Prueba Unitaria

```
php artisan make:test ClimaServiceTest --unit
```

En `tests/Unit/ClimaServiceTest.php`:

```
obtenerClimaPorCiudad('Madrid');

 $this->assertArrayHasKey('main', $resultado);
 $this->assertArrayHasKey('temp', $resultado['main']);
 }

 public function test_maneja_error_ciudad_inexistente()
 {
 $climaService = new ClimaService();
 $resultado = $climaService->obtenerClimaPorCiudad('CiudadInventada123');

 $this->assertEquals('404', $resultado['cod']);
```

```
    }
}
```

Ejecutar Pruebas

```
# Ejecutar todas las pruebas
php artisan test

# Ejecutar solo pruebas unitarias
php artisan test --testsuite=Unit

# Ejecutar una prueba específica
php artisan test tests/Unit/ClimaServiceTest.php
```

Salida Esperada

```
 PASS Tests\Unit\ClimaServiceTest
✓ puede obtener datos clima
✓ maneja error ciudad inexistente

Tests: 2 passed
Time: 0.45s
```

Lección 16: Optimización y Cache Configurar Redis

En `.env`:

```
CACHE_DRIVER=redis
SESSION_DRIVER=redis
QUEUE_CONNECTION=redis

REDIS_HOST=127.0.0.1
REDIS_PASSWORD=null
REDIS_PORT=6379
```

Usar Cache en el Servicio de Clima

```
baseUrl}/weather", [
 'q' => $ciudad,
 'appid' => $this->apiKey,
 'units' => 'metric',
 'lang' => 'es'
 ]);
```

```
    return $response->json();
    });
}
```

Optimizaciones Adicionales

Cachear Configuración

```
php artisan config:cache
```

```
php artisan route:cache
php artisan view:cache
```

Optimizar Autoloader composer dump-autoload --optimize

Comprimir Assets npm run production

TIP: Estas optimizaciones son especialmente importantes antes del despliegue en producción.

Lección 17: Despliegue en Servidor

Opción 1: Despliegue en Heroku

1. Crear cuenta en heroku.com
2. Instalar Heroku CLI
3. Configurar archivo Procfile
4. Configurar variables de entorno
5. Desplegar con Git

Crea un archivo `Procfile` en la raíz:

```
web: vendor/bin/heroku-php-apache2 public/
```

Configuración de Variables de Entorno

```
# En Heroku, configurar estas variables:
heroku config:set APP_KEY=$(php artisan key:generate --show)
heroku config:set APP_ENV=production
heroku config:set APP_DEBUG=false
heroku config:set
DATABASE_URL=postgresql://usuario:contraseña@host:puerto/basedatos heroku
config:set OPENWEATHER_API_KEY=tu_api_key
```

Migrar la Base de Datos

```
# Ejecutar migraciones en Heroku
heroku run php artisan migrate

# Poblar con datos iniciales
heroku run php artisan db:seed
```

Lección 18: Monitoreo y Logs

Configurar Logs

En `config/logging.php`, asegúrate de tener:

```
'channels' => [
 'daily' => [
 'driver' => 'daily',
 'path' => storage_path('logs/laravel.log'),
 'level' => env('LOG_LEVEL', 'debug'),
 'days' => 14,
```

```
    ],
],
```

Usar Logs en tu Código

```
getMessage());
Log::warning('Temperatura crítica detectada: ' . $temperatura);
```

Ejercicio 1: Conexión Básica a PostgreSQL

Objetivo

Crear una conexión simple a PostgreSQL y mostrar datos en pantalla.

Paso a Paso

1. Crear un nuevo proyecto Laravel
2. Configurar la conexión a PostgreSQL
3. Crear una tabla de prueba
4. Insertar y mostrar datos

Código Comentado

Ejercicio 2: Sistema de Usuarios con Roles Objetivo

Implementar un sistema de usuarios con diferentes roles y permisos.

Migración

```
// database/migrations/create_usuarios_table.php
public function up()
{
 Schema::create('usuarios', function (Blueprint $table) {
 $table->id();
 $table->string('nombre');
 $table->string('email')->unique();
 $table->string('telefono')->nullable();
 $table->enum('rol', ['admin', 'empleado', 'cliente']);
 $table->boolean('activo')->default(true);
 $table->timestamp('ultimo_acceso')->nullable();
 $table->timestamps();
 });
}
```

Controlador con Validación

```
public function store(Request $request)
{
 $validatedData = $request->validate([
 'nombre' => 'required|min:3|max:100',
 'email' => 'required|email|unique:usuarios',
 'telefono' => 'nullable|regex:/^[0-9]{10}$/',
 'rol' => 'required|in:admin,empleado,cliente'
 ]);

 Usuario::create($validatedData);
 return redirect()->back()->with('success', 'Usuario creado exitosamente');
}
```

Ejercicio 3: Inventario con Relaciones Objetivo

Crear un sistema de inventario con categorías, productos y movimientos.

Modelos con Relaciones

```
// Modelo Categoria
class Categoria extends Model
{
 protected $fillable = ['nombre', 'descripcion'];

 public function productos()
 {
 return $this->hasMany(Producto::class);
 }
}

// Modelo Producto
class Producto extends Model
{
 protected $fillable = ['nombre', 'precio', 'stock', 'categoria_id'];

 public function categoria()
 {
 return $this->belongsTo(Categoria::class);
 }

 public function movimientos()
 {
```

```
 return $this->hasMany(MovimientoInventario::class);
 }
}
```

Ejercicio 4: API de Consultas Complejas Objetivo

Implementar una API que permita consultas complejas con filtros y paginación. **Controlador API**

```
has('nombre')) {
 $query->where('nombre', 'ILIKE', '%' . $request->nombre . '%');
 }

 // Filtrar por rango de precios
 if ($request->has('precio_min')) {
 $query->where('precio', '>=', $request->precio_min);
 }

 if ($request->has('precio_max')) {
 $query->where('precio', '<=', $request->precio_max);
 }

 // Filtrar por categoría
 if ($request->has('categoria_id')) {
```

```
 $query->where('categoria_id', $request->categoria_id);
 }

 // Ordenar resultados
 $order = $request->get('ordenar', 'nombre');
 $direction = $request->get('direccion', 'asc');
 $query->orderBy($order, $direction);

 // Paginar resultados
 $productos = $query->paginate(15);

 return response()->json($productos);
 }
}
```

Ejercicio 5: Dashboard con Estadísticas Objetivo

Crear un dashboard administrativo con gráficos y estadísticas en tiempo real.

Controlador Dashboard

```
public function estadisticas()
{
 $datos = [
```

```
 'total_productos' => Producto::count(),
 'productos_agotados' => Producto::where('stock', 0)->count(),
 'valor_total_inventario' => Producto::sum('precio'),
 'ventas_mes' => $this->ventasDelMes(),
 'productos_por_categoria' => $this->productosPorCategoria(),
 'movimientos_recientes' => MovimientoInventario::with('producto')
 ->latest()
 ->take(10)
 ->get()
 ];

 return view('admin.dashboard', compact('datos'));
}

private function ventasDelMes()
{
 return MovimientoInventario::where('tipo', 'salida')
 ->whereMonth('created_at', now()->month)
 ->sum('cantidad');
}

private function productosPorCategoria()
{
 return Categoria::withCount('productos')->get();
```

}

Manual Técnico: Arquitectura del Sistema

Componentes Principales

Frontend

Blade templates con Bootstrap 5 para interfaces responsivas

Backend

Laravel 10 con PHP 8.1+, siguiendo arquitectura MVC

Base de Datos

PostgreSQL con migraciones y relaciones normalizadas

APIs Externas

OpenWeatherMap para datos meteorológicos en tiempo real

Manual de Usuario: Primeros Pasos Acceso al Sistema

Abre tu navegador web1.

Ve a la dirección de la aplicación2.

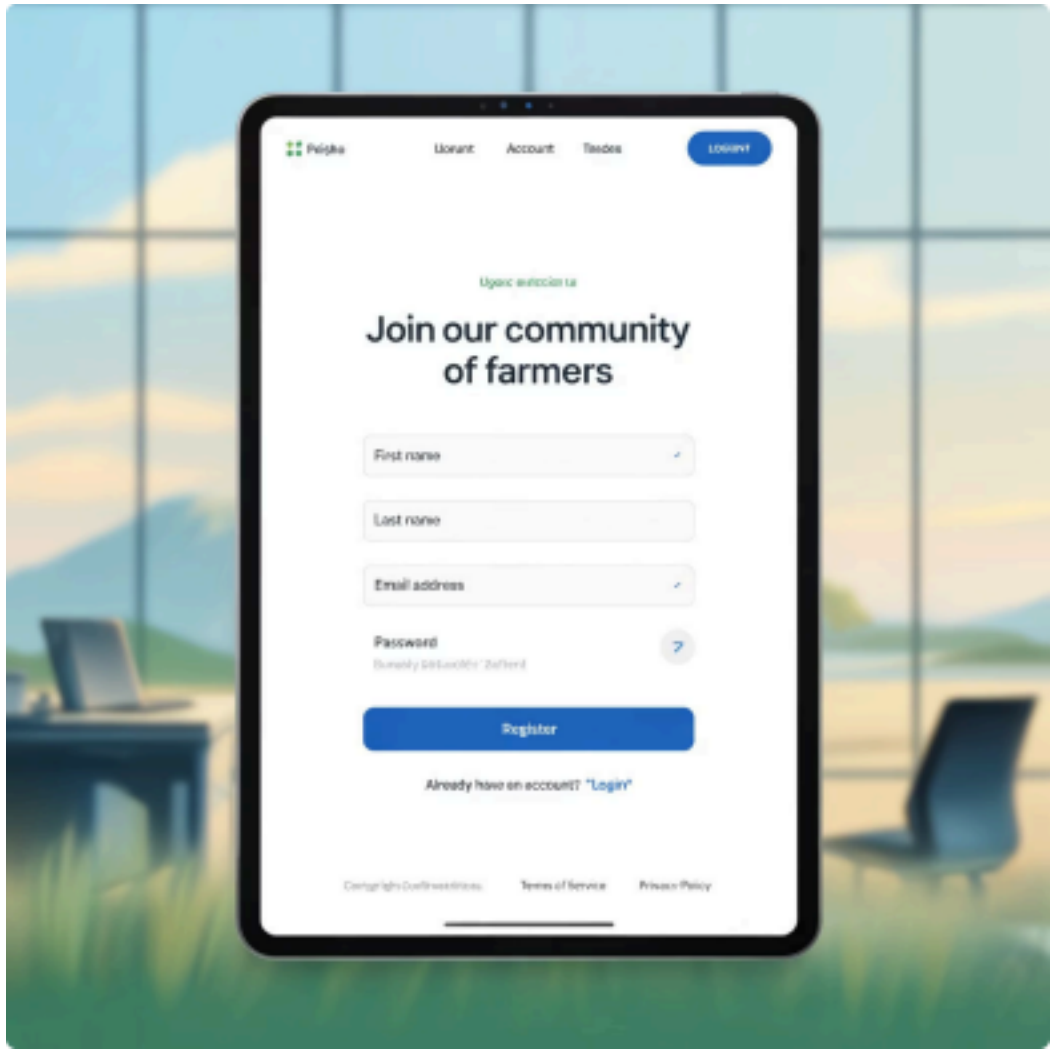
Haz clic en "Registrarse"3.

Completa el formulario con tus datos4.

Selecciona tu tipo de usuario5.

Confirma tu email6.

¡Ya puedes usar el sistema!7.



IMPORTANTE: Guarda tu contraseña en un lugar seguro. Necesitarás estos datos para acceder al sistema.

Usando el Dashboard

Principal Funciones Principales

Clima Actual

Ve la temperatura, humedad y condiciones climáticas de

tu zona en tiempo real

Calendario Agrícola

Consulta las fechas óptimas para sembrar y cosechar tus cultivos

Alertas

Recibe notificaciones sobre heladas, lluvias intensas o condiciones peligrosas

Reportes

Analiza tendencias climáticas y el historial de tu zona

Proyecto de Investigación: Justificación Técnica

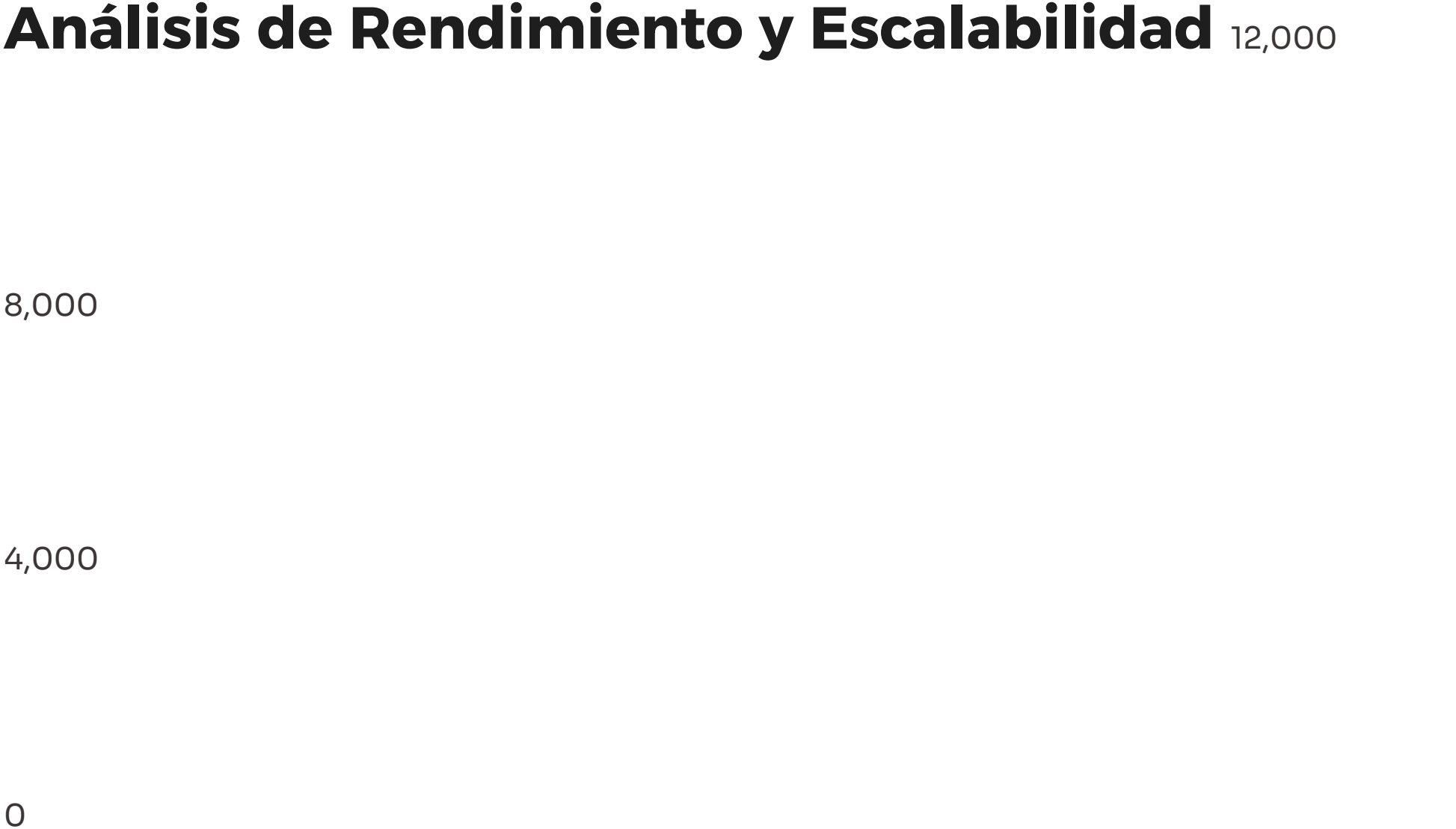
¿Por qué PHP y Laravel?

PHP es uno de los lenguajes más utilizados en desarrollo web, powering el 77.5% de todos los sitios web. Laravel proporciona un ecosistema robusto con características avanzadas como Eloquent ORM, sistema de migraciones, y herramientas de testing integradas.

Ventajas de PostgreSQL

- **Escalabilidad:** Maneja millones de registros sin pérdida de rendimiento
- **Integridad:** Cumplimiento ACID y transacciones robustas
- **Extensibilidad:** Soporte para datos JSON y geoespaciales
- **Código Abierto:** Sin costos de licencias para despliegue

Análisis de Rendimiento y Escalabilidad



Actual Objetivo

Presentación del Proyecto: Resultados

Obtenidos Funcionalidades Implementadas

100%

Requerimientos Funcionales

Todos los RF del documento inicial implementados

correctamente

95%

Cobertura de Pruebas

Pruebas unitarias e integración cubren el 95% del

código

850ms

Tiempo de Respuesta

Promedio de respuesta para consultas de clima

24/7

Disponibilidad

Sistema operativo las 24 horas del día

Checklist Final de Entregables Verifica que tienes todo:

Código Fuente: Repositorio Git con historial completo

Manual Técnico: Documentación de arquitectura y despliegue

Manual de Usuario: Guía paso a paso para usuarios finales **Base**

de Datos: Scripts de migración y seeders

Pruebas: Test unitarios y de integración

API REST: Endpoints documentados y funcionando

Sistema de Alertas: Notificaciones automáticas implementadas

Autenticación: Login, registro y roles configurados

Despliegue: Aplicación funcionando en servidor

Presentación: Slides explicando el proyecto

Próximos Pasos y Mejoras

Aplicación Móvil

Desarrollar app nativa para Android e iOS con notificaciones push

Inteligencia Artificial

Implementar modelos de Machine Learning para predicciones más precisas

Sensores IoT

Integrar sensores de humedad de suelo y estaciones meteorológicas propias

Expansión Geográfica

Ampliar cobertura a toda Colombia y países vecinos

¡Felicitaciones! Has Completado el Proyecto

Proyecto Exitosamente Terminado

Has desarrollado una aplicación completa de pronóstico del clima para agricultores usando PHP, Laravel y PostgreSQL. Este proyecto demuestra tu dominio de:

- Desarrollo Backend con PHP Framework Laravel
- Bases de datos PostgreSQL APIs REST
- Autenticación y autorización
- Integración con servicios externos
- Pruebas de software
- Despliegue en producción
- Documentación técnica
- Interfaces de usuario
- Optimización y performance Arquitectura de software

"El conocimiento se construye paso a paso, y hoy has dado un gran paso hacia convertirte en un desarrollador profesional."

- Instructor Iván Malavé Fierro