📋 DOCUMENTACIÓN COMPLETA DEL PROYECTO gbnEmotions

Versión: 1.0
Fecha: Diciembre 2024
Autor: Equipo de Desarrollo
Tecnologías: Laravel 12, PHP 8.2+, Tailwind CSS, Alpine.js, SQLite

---

📖 ÍNDICE

🎯 RESUMEN EJECUTIVO

Descripción del Proyecto

gbnEmotions es una aplicación web desarrollada en Laravel 12 diseñada para la gestión y registro del estado emocional de empleados en tiempo real. La aplicación permite a los empleados registrar sus emociones diarias a través de un formulario interactivo, proporcionando insights valiosos para el bienestar organizacional.

Objetivos del Sistema

- ☑ Registro de emociones por empleado
- ☑ Interfaz multilingüe (Francés, Inglés, Español)
- ☑ Sistema de autenticación robusto
- ☑ Integración con APIs externas
- ☑ Interfaz responsiva y moderna
- ☑ Almacenamiento seguro de datos

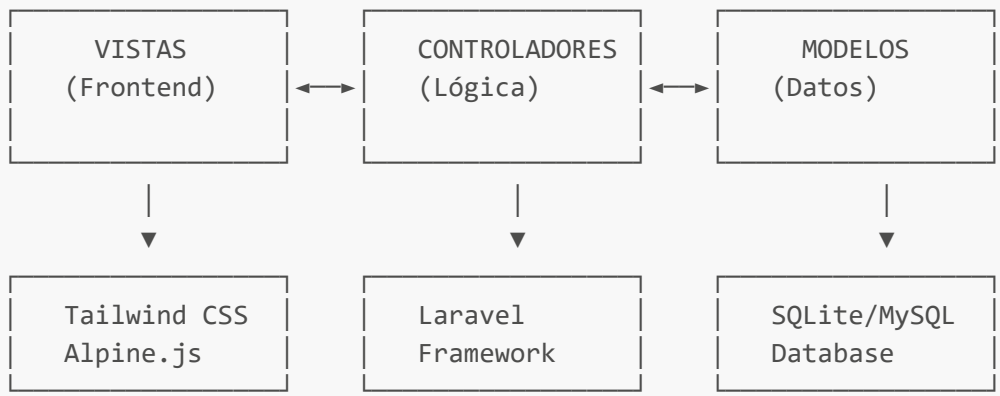
Tecnologías Principales

- **Backend:** Laravel 12 (PHP 8.2+)
- **Frontend:** Tailwind CSS, Alpine.js
- **Base de Datos:** SQLite (desarrollo) / MySQL (producción)
- **Build Tool:** Vite
- **Autenticación:** Laravel Breeze

🏗️ ARQUITECTURA DEL SISTEMA

Patrón de Arquitectura

El proyecto sigue el patrón **MVC (Modelo-Vista-Controlador)** de Laravel, proporcionando una separación clara de responsabilidades:



Flujo de Datos

1. **Usuario** accede a la aplicación
2. **Router** direcciona la petición al controlador correcto
3. **Controlador** procesa la lógica de negocio
4. **Modelo** interactúa con la base de datos
5. **Vista** renderiza la respuesta al usuario

📁 ESTRUCTURA DEL PROYECTO

Organización de Directorios

```
gbnEmotions/
├── app/                          # Lógica principal de la aplicación
│   ├── Http/                     # Controladores y middleware
│   │   ├── Controllers/          # Controladores de la aplicación
│   │   │   ├── Auth/            # Controladores de autenticación
│   │   │   ├── MoodEmotionController.php
│   │   │   ├── ApiConsumerController.php
```

```
│   │   │   └── ProfileController.php
│   │   └── Requests/            # Validación de formularios
│   ├── Models/                   # Modelos Eloquent
│   │   ├── User.php
│   │   └── MoodEmotion.php
│   └── Providers/               # Proveedores de servicios
├── bootstrap/                    # Archivos de inicialización
├── config/                       # Archivos de configuración
│   ├── app.php                  # Configuración general
│   ├── database.php             # Configuración de BD
│   └── auth.php                 # Configuración de autenticación
├── database/                     # Base de datos
│   ├── migrations/              # Migraciones de BD
│   ├── seeders/                 # Datos de prueba
│   └── factories/               # Generadores de datos
├── lang/                         # Archivos de idiomas
│   ├── en/                      # Inglés
│   ├── es/                      # Español
│   └── fr/                      # Francés
├── public/                       # Archivos públicos
│   ├── index.php                # Punto de entrada
│   └── js/                      # JavaScript compilado
├── resources/                    # Recursos sin compilar
│   ├── css/                     # Estilos CSS
│   ├── js/                      # JavaScript
│   └── views/                   # Vistas Blade
│       ├── auth/                # Vistas de autenticación
│       ├── components/          # Componentes reutilizables
│       ├── layouts/             # Layouts principales
│       └── moods/               # Vistas de emociones
├── routes/                       # Definición de rutas
│   ├── web.php                  # Rutas web
│   └── auth.php                 # Rutas de autenticación
├── storage/                      # Archivos generados
├── tests/                        # Tests automatizados
├── vendor/                       # Dependencias de Composer
├── .env                         # Variables de entorno
├── artisan                      # CLI de Laravel
├── composer.json                # Dependencias PHP
├── package.json                 # Dependencias JavaScript
├── tailwind.config.js           # Configuración de Tailwind
└── vite.config.js               # Configuración de Vite
```

Archivos Clave y su Función

Archivo	Función	Importancia
`artisan`	CLI de Laravel	Comandos de desarrollo
`composer.json`	Dependencias PHP	Gestión de paquetes
`package.json`	Dependencias JS	Gestión de frontend

Archivo	Función	Importancia
`.env`	Variables de entorno	Configuración
`tailwind.config.js`	Configuración CSS	Estilos de la app
`vite.config.js`	Build tool	Compilación de assets

⚙️ CONFIGURACIÓN DEL ENTORNO

Requisitos del Sistema

- **PHP:** 8.2 o superior
- **Composer:** Última versión estable
- **Node.js:** 18 o superior
- **NPM:** Última versión estable
- **Base de Datos:** SQLite (desarrollo) / MySQL (producción)

Instalación Inicial

1. Clonar el Proyecto

```
git clone [URL_DEL_REPOSITORIO]
cd gbnEmotions
```

2. Instalar Dependencias PHP

```
composer install
```

3. Instalar Dependencias JavaScript

```
npm install
```

4. Configurar Variables de Entorno

```
cp .env.example .env
php artisan key:generate
```

5. Configurar Base de Datos

```
# Para SQLite (desarrollo)
touch database/database.sqlite
php artisan migrate

# Para MySQL (producción)
# Configurar .env con credenciales MySQL
php artisan migrate
```

6. Compilar Assets

```
npm run dev    # Desarrollo
npm run build  # Producción
```

Configuración de Entorno de Desarrollo

Variables de Entorno Importantes (.env)

```
APP_NAME=gbnEmotions
APP_ENV=local
APP_DEBUG=true
APP_URL=http://127.0.0.1:8000

DB_CONNECTION=sqlite
DB_DATABASE=/path/to/database.sqlite

CACHE_DRIVER=file
SESSION_DRIVER=file
QUEUE_CONNECTION=sync
```

BASE DE DATOS

Estructura de la Base de Datos

Tabla: `users`

```
CREATE TABLE users (
    id BIGINT PRIMARY KEY AUTOINCREMENT,
    name VARCHAR(255) NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    email_verified_at TIMESTAMP NULL,
    password VARCHAR(255) NOT NULL,
    remember_token VARCHAR(100) NULL,
    created_at TIMESTAMP NULL,
```

```
    updated_at TIMESTAMP NULL
);
```

Tabla: `mood_emotions`

```
CREATE TABLE mood_emotions (
    id BIGINT PRIMARY KEY AUTOINCREMENT,
    employee_id VARCHAR(255) NOT NULL,
    emotion VARCHAR(255) NOT NULL,
    answer_1 BOOLEAN NULL,
    answer_2 BOOLEAN NULL,
    answer_3 BOOLEAN NULL,
    diary_text TEXT NULL,
    created_at TIMESTAMP NULL,
    updated_at TIMESTAMP NULL
);
```

Migraciones Implementadas

1. Creación de Tabla de Emociones

- **Archivo:** `2025_07_22_083155_create_mood_emotions_table.php`
- **Función:** Crea la tabla inicial con campos básicos
- **Comando:** `php artisan make:migration create_mood_emotions_table`

2. Renombrado de Campo user_id

- **Archivo:** `2025_07_24_092334_rename_user_id_to_employee_id_in_mood_emotions.php`
- **Función:** Cambia `user_id` por `employee_id` y modifica el tipo a string
- **Comando:** `php artisan make:migration rename_user_id_to_employee_id_in_mood_emotions`

3. Agregado de Campos de Respuestas

- **Archivo:** `2025_07_28_135047_update_mood_emotions_table.php`
- **Función:** Agrega campos `answer_1`, `answer_2`, `answer_3` como booleanos
- **Comando:** `php artisan make:migration update_mood_emotions_table`

Modelo Eloquent: MoodEmotion

Características del Modelo

- **Tabla:** `mood_emotions`
- **Campos Fillable:** employee_id, emotion, answer_1, answer_2, answer_3, diary_text
- **Timestamps:** Habilitados (created_at, updated_at)

Relaciones

- **Usuario:** Relación con el modelo User (futuro desarrollo)

🔄 FLUJO DE LA APLICACIÓN

Flujo Principal: Registro de Emociones

```
graph TD
    A[Usuario accede a /moods/{employee_id}/create] --> B[Router direcciona a
MoodEmotionController@create]
    B --> C[Controlador renderiza vista create.blade.php]
    C --> D[Usuario ve formulario de emociones]
    D --> E[Usuario selecciona emoción]
    E --> F[JavaScript muestra preguntas adicionales]
    F --> G[Usuario responde preguntas]
    G --> H[Usuario envía formulario]
    H --> I[Router direcciona a MoodEmotionController@store]
    I --> J[Controlador valida datos]
    J --> K[Modelo guarda en base de datos]
    K --> L[Redirección con mensaje de éxito]
    L --> M[Usuario ve confirmación]
```

Flujo de Autenticación

```
graph TD
    A[Usuario accede a ruta protegida] --> B{¿Está autenticado?}
    B -->|No| C[Redirección a login]
    B -->|Sí| D[Acceso permitido]
    C --> E[Usuario ingresa credenciales]
    E --> F[Laravel Breeze valida]
    F --> G{¿Credenciales válidas?}
    G -->|No| H[Mensaje de error]
    G -->|Sí| I[Crear sesión]
    I --> J[Redirección a dashboard]
```

Flujo de Integración con API Externa

```
graph TD
    A[Usuario accede a /call-api] --> B[Router direcciona a
ApiConsumerController@callExternalApi]
    B --> C[Controlador hace petición HTTP a API externa]
    C --> D{¿API responde exitosamente?}
    D -->|No| E[Error 500 - Falló la llamada]
    D -->|Sí| F[Convertir respuesta JSON a array PHP]
    F --> G[Renderizar vista moods/index.blade.php]
    G --> H[Mostrar lista de employee IDs]
```

```
    H --> I[Usuario selecciona employee ID]
    I --> J[Redirección a formulario de emociones]
```

🧩 COMPONENTES DEL SISTEMA

Controladores Principales

1. MoodEmotionController

Ubicación: `app/Http/Controllers/MoodEmotionController.php`

Métodos:

- `create($employee_id)`: Muestra formulario de registro
- `store(Request $request, $employee_id)`: Guarda emoción en BD
- `index()`: Muestra historial de emociones

Responsabilidades:

- Validación de datos de entrada
- Interacción con modelo MoodEmotion
- Renderizado de vistas
- Manejo de mensajes de sesión

2. ApiConsumerController

Ubicación: `app/Http/Controllers/ApiConsumerController.php`

Métodos:

- `callExternalApi()`: Consume API externa de employee UIDs

Responsabilidades:

- Peticiones HTTP a APIs externas
- Manejo de respuestas JSON
- Gestión de errores de API

3. ProfileController

Ubicación: `app/Http/Controllers/ProfileController.php`

Métodos:

- `edit()`: Muestra formulario de perfil
- `update()`: Actualiza datos de perfil
- `destroy()`: Elimina cuenta de usuario

Modelos

1. User

Ubicación: `app/Models/User.php`

Características:

- Extiende de Authenticatable
- Usa Laravel Breeze para autenticación
- Campos: name, email, password, remember_token

2. MoodEmotion

Ubicación: `app/Models/MoodEmotion.php`

Características:

- Extiende de Model
- Tabla: mood_emotions
- Campos fillable definidos
- Sin relaciones actuales (preparado para futuro)

Componentes Blade

1. EmotionButton

Ubicación: `resources/views/components/emotion-button.blade.php`

Propósito: Botón reutilizable para selección de emociones

Props:

- `label`: Texto descriptivo del botón
- `emoji`: Emoji a mostrar

Estilos:

- Colores personalizados (#30CFD0, #FFB366)
- Efectos hover y focus
- Transiciones suaves

2. QuestionBlock

Ubicación: `resources/views/components/question-block.blade.php`

Propósito: Bloque contenedor para preguntas

Props:

- `emoji`: Emoji representativo
- `text`: Texto de la pregunta

Características:

- Layout responsivo
- Grid de 5 columnas para respuestas
- Estilos consistentes

3. LanguageSelector

Ubicación: `resources/views/components/language-selector.blade.php`

Propósito: Selector de idioma para la aplicación

Funcionalidad:

- Soporte para FR, EN, ES
- Indicador visual del idioma activo
- Cambio dinámico de idioma

RUTAS Y CONTROLADORES

Estructura de Rutas

Rutas Web Principales (`routes/web.php`)

```
// Página de bienvenida
Route::get('/', function () {
    return view('welcome');
});

// Dashboard (protegido)
Route::get('/dashboard', function () {
    return view('dashboard');
})->middleware(['auth', 'verified'])->name('dashboard');

// Rutas de emociones (públicas)
Route::get('/moods/{employee_id}/create', [MoodEmotionController::class,
'create'])
    ->name('moods.create');
Route::post('/moods/{employee_id}', [MoodEmotionController::class, 'store'])
    ->name('moods.store');

// Historial de emociones (autenticado)
Route::get('/emotion/history', [MoodEmotionController::class, 'index'])
    ->name('emotion.index');

// API externa
Route::get('/api/employee-uuids', function () {
    return response()->json(['E001', 'E002', 'E003']);
});
Route::get('/call-api', [ApiConsumerController::class, 'callExternalApi']);

// Rutas de perfil (autenticadas)
```

```
Route::get('/profile', [ProfileController::class, 'edit'])->name('profile.edit');
Route::patch('/profile', [ProfileController::class, 'update'])-
>name('profile.update');
Route::delete('/profile', [ProfileController::class, 'destroy'])-
>name('profile.destroy');
```

Rutas de Autenticación (`routes/auth.php`)

Rutas para Invitados:

- `register` - Registro de usuarios
- `login` - Inicio de sesión
- `forgot-password` - Recuperación de contraseña
- `reset-password` - Restablecimiento de contraseña

Rutas para Usuarios Autenticados:

- `verify-email` - Verificación de email
- `confirm-password` - Confirmación de contraseña
- `password.update` - Actualización de contraseña
- `logout` - Cerrar sesión

Middleware Utilizado

1. Auth Middleware

- **Propósito:** Proteger rutas que requieren autenticación
- **Aplicación:** Dashboard, perfil, historial de emociones

2. Verified Middleware

- **Propósito:** Verificar que el email del usuario esté confirmado
- **Aplicación:** Dashboard

3. Guest Middleware

- **Propósito:** Restringir acceso a usuarios ya autenticados
- **Aplicación:** Login, registro

🎨 VISTAS Y FRONTEND

Sistema de Vistas

Layout Principal

Archivo: `resources/views/layouts/app.blade.php`

Características:

- Estructura HTML base
- Integración con Vite para assets
- Secciones yield para contenido dinámico
- Soporte para header y main content

Vistas de Emociones

1. create.blade.php

Ubicación: `resources/views/moods/create.blade.php`

Funcionalidad:

- Formulario de registro de emociones
- Selección de emoción con emojis
- Preguntas dinámicas según emoción seleccionada
- Validación del lado cliente y servidor
- Mensajes de éxito y error

Características Técnicas:

- Uso de Tailwind CSS para estilos
- JavaScript para interactividad
- Integración con componentes Blade
- Formulario con CSRF protection

2. index.blade.php

Ubicación: `resources/views/moods/index.blade.php`

Funcionalidad:

- Lista de employee IDs obtenidos de API externa
- Enlaces a formularios de registro
- Layout responsivo

Frontend Technologies

1. Tailwind CSS

Configuración: `tailwind.config.js`

Características:

- Framework CSS utilitario
- Colores personalizados definidos
- Sistema de espaciado consistente
- Componentes reutilizables

Colores del Proyecto:

- Primary: #30CFD0 (turquesa)
- Secondary: #A18CD1 (morado)
- Accent: #FFB366 (naranja)

2. Alpine.js

Configuración: `resources/js/app.js`

Funcionalidad:

- Interactividad en el frontend
- Manejo de estados de componentes
- Transiciones suaves
- Validación del lado cliente

3. Vite

Configuración: `vite.config.js`

Propósito:

- Build tool para assets
- Hot module replacement
- Optimización para producción
- Integración con Laravel

JavaScript Personalizado

mood-form.js

Ubicación: `public/js/mood-form.js`

Funcionalidad:

- Manejo de selección de emociones
- Mostrar/ocultar preguntas dinámicamente
- Scroll suave a secciones
- Integración con traducciones del servidor

SISTEMA DE AUTENTICACIÓN

Laravel Breeze

Características Implementadas

- **Registro de usuarios** con validación
- **Inicio de sesión** con remember me
- **Recuperación de contraseña** por email
- **Verificación de email** opcional
- **Gestión de perfil** de usuario

- **Cerrar sesión** seguro

Controladores de Autenticación

Ubicación: `app/Http/Controllers/Auth/`

Controladores:

- `AuthenticatedSessionController` - Login/logout
- `RegisteredUserController` - Registro
- `PasswordResetLinkController` - Recuperación de contraseña
- `NewPasswordController` - Restablecimiento de contraseña
- `EmailVerificationPromptController` - Verificación de email
- `VerifyEmailController` - Proceso de verificación
- `ConfirmablePasswordController` - Confirmación de contraseña
- `PasswordController` - Actualización de contraseña

Vistas de Autenticación

Ubicación: `resources/views/auth/`

Vistas:

- `login.blade.php` - Formulario de inicio de sesión
- `register.blade.php` - Formulario de registro
- `forgot-password.blade.php` - Recuperación de contraseña
- `reset-password.blade.php` - Restablecimiento de contraseña
- `verify-email.blade.php` - Verificación de email
- `confirm-password.blade.php` - Confirmación de contraseña

Seguridad Implementada

1. CSRF Protection

- Tokens CSRF en todos los formularios
- Protección contra ataques Cross-Site Request Forgery

2. Validación de Datos

- Validación del lado servidor
- Reglas de validación específicas por formulario
- Mensajes de error personalizados

3. Hashing de Contraseñas

- Contraseñas hasheadas con bcrypt
- Salt automático para mayor seguridad

4. Rate Limiting

- Límites de intentos de login
- Protección contra ataques de fuerza bruta

INTEGRACIÓN CON APIs EXTERNAS

ApiConsumerController

Funcionalidad Principal

Método: `callExternalApi()`

Proceso:

1. Realiza petición HTTP GET a API externa
2. Verifica respuesta exitosa
3. Convierte JSON a array PHP
4. Renderiza vista con datos obtenidos

Configuración de API

URL de API: `http://127.0.0.1:8001/api/employee-uuids`

Formato de Respuesta Esperado:

```
["E001", "E002", "E003"]
```

Manejo de Errores

- **Respuesta exitosa:** Renderiza vista con datos
- **Error de API:** Aborta con error 500
- **Timeout:** Manejo de timeouts de red

Flujo de Integración

1. Petición a API Externa

```
$response = Http::acceptJson()
    ->get('http://127.0.0.1:8001/api/employee-uuids');
```

2. Validación de Respuesta

```
if ($response->successful()) {
    $data = $response->json();
    return view('moods.index', compact('data'));
}
```

```
```

3. Manejo de Errores

```
abort(500, 'Falló la llamada al Proveedor');
```

Consideraciones de Seguridad

1. Validación de URLs

- Verificar URLs de API antes de hacer peticiones
- Usar HTTPS en producción

2. Timeouts

- Configurar timeouts apropiados
- Manejar errores de timeout

3. Rate Limiting

- Implementar límites de peticiones a APIs externas
- Cachear respuestas cuando sea apropiado

🛠️ COMANDOS Y HERRAMIENTAS

Comandos Artisan Principales

Desarrollo

```
# Iniciar servidor de desarrollo
php artisan serve

# Ver todas las rutas
php artisan route:list

# Limpiar cache
php artisan cache:clear
php artisan config:clear
php artisan view:clear

# Consola interactiva
php artisan tinker
```

Base de Datos

```
```

```
# Crear migración
php artisan make:migration nombre_migracion

# Ejecutar migraciones
php artisan migrate

# Revertir última migración
php artisan migrate:rollback

# Revertir y volver a ejecutar
php artisan migrate:refresh

# Ver estado de migraciones
php artisan migrate:status

# Crear seeder
php artisan make:seeder NombreSeeder

# Ejecutar seeders
php artisan db:seed
```

Generación de Código

```
# Crear controlador
php artisan make:controller NombreController

# Crear modelo
php artisan make:model NombreModel

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

# Crear controlador con métodos CRUD
php artisan make:controller NombreController --resource

# Crear middleware
php artisan make:middleware NombreMiddleware
```

Autenticación

```
# Instalar Laravel Breeze
composer require laravel/breeze --dev
php artisan breeze:install

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

Comandos NPM

Desarrollo Frontend

```
# Instalar dependencias
npm install

# Compilar en desarrollo
npm run dev

# Compilar para producción
npm run build

# Ver dependencias
npm list
```

Scripts Disponibles

```
{
    "dev": "vite",
    "build": "vite build"
}
```

Comandos Composer

Gestión de Dependencias

```
# Instalar dependencias
composer install

# Actualizar dependencias
composer update

# Instalar paquete específico
composer require nombre-paquete

# Ver dependencias
composer show

# Autoload
composer dump-autoload
```

🚀 DESPLIEGUE Y MANTENIMIENTO

Preparación para Producción

1. Configuración de Entorno

```
APP_ENV=production
APP_DEBUG=false
APP_URL=https://tudominio.com

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=gbnemotions
DB_USERNAME=usuario
DB_PASSWORD=contraseña
```

2. Optimización de Laravel

```
# Cachear configuración
php artisan config:cache

# Cachear rutas
php artisan route:cache

# Cachear vistas
php artisan view:cache

# Optimizar autoloader
composer install --optimize-autoloader --no-dev
```

3. Compilación de Assets

```
npm run build
```

Mantenimiento

Tareas Programadas

```
# Limpiar cache periódicamente
php artisan cache:clear

# Verificar logs
tail -f storage/logs/laravel.log
```

```
# Backup de base de datos
php artisan backup:run
```

Monitoreo

- **Logs:** Revisar `storage/logs/laravel.log`
- **Errores:** Configurar notificaciones de errores
- **Rendimiento:** Monitorear tiempos de respuesta

Backup y Recuperación

Backup de Base de Datos

```
# SQLite
cp database/database.sqlite backup/database_backup.sqlite

# MySQL
mysqldump -u usuario -p gbnemotions > backup/database_backup.sql
```

Backup de Archivos

```
# Archivos de configuración
cp .env backup/.env_backup

# Archivos subidos
cp -r storage/app/public backup/uploads
```

🔧 TROUBLESHOOTING

Problemas Comunes y Soluciones

1. Error de Permisos

Síntoma: Error al escribir en storage o cache **Solución:**

```
chmod -R 775 storage bootstrap/cache
chown -R www-data:www-data storage bootstrap/cache
```

2. Error de Base de Datos

Síntoma: Error de conexión o tabla no encontrada **Solución:**

```
php artisan migrate:status
php artisan migrate
php artisan db:seed
```

3. Assets No Cargados

Síntoma: CSS/JS no se cargan correctamente **Solución:**

```
npm install
npm run dev
# o para producción
npm run build
```

4. Error de Autenticación

Síntoma: Problemas con login o registro **Solución:**

```
php artisan config:clear
php artisan cache:clear
composer dump-autoload
```

5. Error de API Externa

Síntoma: Error 500 al llamar API externa **Solución:**

- Verificar que la API externa esté funcionando
- Verificar URL y credenciales
- Revisar logs de Laravel

Logs y Debugging

Ubicación de Logs

- **Laravel:** `storage/logs/laravel.log`
- **PHP:** `/var/log/php/error.log`
- **Apache/Nginx:** `/var/log/apache2/` o `/var/log/nginx/`

Comandos de Debugging

```
# Ver logs en tiempo real
tail -f storage/logs/laravel.log

# Ver configuración actual
php artisan config:show
```

```
# Ver rutas registradas
php artisan route:list

# Ver variables de entorno
php artisan env
```

GLOSARIO TÉCNICO

Términos de Laravel

Término	Definición
Artisan	CLI de Laravel para comandos
Blade	Motor de plantillas de Laravel
Eloquent	ORM de Laravel para base de datos
Middleware	Filtros que procesan peticiones HTTP
Migration	Sistema de versionado de base de datos
Route	Definición de URL y su controlador
Controller	Clase que maneja lógica de negocio
Model	Clase que representa una tabla de BD
View	Archivo de presentación (HTML)
Seeder	Datos de prueba para la base de datos

Términos de Frontend

Término	Definición
Tailwind CSS	Framework CSS utilitario
Alpine.js	Framework JavaScript ligero
Vite	Build tool para assets
Component	Elemento reutilizable de interfaz
Responsive	Diseño que se adapta a diferentes pantallas
CSS Grid	Sistema de layout CSS
Flexbox	Sistema de layout CSS flexible

Términos de Base de Datos

Término	Definición

Término	Definición
SQLite	Base de datos ligera en archivo
MySQL	Sistema de gestión de base de datos
Migration	Script para modificar estructura de BD
Seeder	Datos iniciales para la base de datos
Eloquent	ORM de Laravel
Model	Representación de tabla en código

Términos de Desarrollo

Término	Definición
MVC	Patrón Modelo-Vista-Controlador
CRUD	Create, Read, Update, Delete
API	Interfaz de programación de aplicaciones
REST	Estilo de arquitectura para APIs
JSON	Formato de intercambio de datos
CSRF	Cross-Site Request Forgery protection

📞 CONTACTO Y SOPORTE

Información del Proyecto

- **Nombre:** gbnEmotions
- **Versión:** 1.0
- **Tecnología:** Laravel 12
- **Fecha de Documentación:** Diciembre 2024

Recursos Adicionales

- **Documentación Laravel:** https://laravel.com/docs
- **Documentación Tailwind:** https://tailwindcss.com/docs
- **Documentación Alpine.js:** https://alpinejs.dev/docs

Mantenimiento

Para actualizaciones, mejoras o soporte técnico, contactar al equipo de desarrollo.

Documento generado automáticamente - Última actualización: Diciembre 2024