MoodTracker — Cambios implementados y explicación para no programadores

1) Objetivo general

Hemos mejorado el corazón del sistema para que MoodTracker pueda:

- Guardar configuraciones del sistema de forma centralizada (settings) y leerlas fácilmente.
- Registrar alertas cuando ciertos indicadores bajan de un umbral.
- Llevar un registro (auditoría) de "quién hizo qué y cuándo" dentro de la aplicación.
- Generar datos de ejemplo de manera realista para demos y pruebas.

La idea es que el sistema sea más confiable, configurable y fácil de mantener.

---

2) Explicación en lenguaje sencillo

- «Settings» (configuraciones): Son parámetros que controlan el comportamiento del sistema sin tener que tocar el código. Ejemplos: cuántas respuestas mínimas hacen falta para mostrar datos anónimos, cuál es el umbral que dispara una alerta, o a qué hora enviar recordatorios.
- «Alerts» (alertas): Avisos automáticos para señalar situaciones que requieren atención (por ejemplo, un indicador de bienestar bajo en un departamento). Guardan a qué departamento afectan, quién la creó y en qué semana ocurrió.
- «Audit logs» (bitácora/auditoría): El historial de acciones importantes. Sirve para responder preguntas como: "¿quién cerró esta alerta y cuándo?", o "¿quién cambió una configuración?".
- «Factories y seeders»: Herramientas para generar datos de ejemplo de forma rápida y realista. Así podemos probar el sistema y mostrar demos sin depender de datos reales.

---

3) Qué hemos cambiado exactamente (alto nivel)

1. Usuarios (users)
    - Añadimos la columna `consent_at` (marca cuándo la persona aceptó el consentimiento) y soportamos `department_id` opcional si no existía.
    - El modelo `User` ahora entiende `consent_at` como fecha.
2. Configuraciones (settings)
    - Creamos la tabla `settings` con campos `key` (clave) y `value` (valor en formato JSON).
    - Creamos el modelo `Setting` y un seeder `SettingsSeeder` que inserta valores iniciales clave:
        - `anon_threshold = 5` (respuestas mínimas para proteger el anonimato)
        - `iec_alert_threshold = 60` (umbral de alerta para indicadores)
        - `send_window = "Mon 17:00-18:00"` (ventana horaria para recordatorios)
    - Añadimos un helper global `setting($key, $default)` para leer configuraciones en 1 línea.
3. Alertas (alerts)

- Creamos la tabla `alerts` con relaciones a `departments` y a `users` (quien la creó).
- Definimos estados simples: `open` o `closed` y la semana (`period_week`) a la que aplica.
- Añadimos el modelo `Alert` con relaciones: `department()` y `createdBy()`.

4. Auditoría (audit_logs)

- Creamos la tabla `audit_logs` con: `actor_id` (quién), `action` (qué), `entity_type` y `entity_id` (sobre qué) y `meta` (detalles extra).
- Solo guarda `created_at` (no se edita; un log no se modifica, se agrega).
- Creamos el modelo `AuditLog` y un método estático para registrar acciones fácilmente.

5. Datos de ejemplo (factories)

- Creamos factories realistas para `Company`, `Department`, `User`, `Alert`, `AuditLog` y `Setting`.
- Esto permite generar datos demo con una sola línea en la consola de Laravel (Tinker).

---

4) ¿Por qué era necesario?

- Configuraciones centralizadas: Evita "cablear" números en el código. Si mañana el umbral cambia de 60 a 65, se actualiza en una tabla, no en múltiples archivos de código.
- Alertas claras: Permiten detectar y priorizar situaciones que requieren intervención (por ejemplo, equipos con indicadores bajos).
- Trazabilidad (audit logs): Aporta transparencia y facilita el diagnóstico de incidencias y el cumplimiento normativo: siempre se sabe quién hizo cada cambio y cuándo.
- Datos demo realistas: Facilita pruebas y demostraciones sin depender de información sensible.

---

5) Cómo se usa en el día a día (ejemplos prácticos)

1. Leer una configuración en el código

```
$minRespuestas = setting('anon_threshold', 5);
```

Se usa en cálculos para decidir si mostrar datos desglosados o no (protección del anonimato).

2. Disparar una alerta

```
$umbral = setting('iec_alert_threshold', 60);
if ($igbDepartamento < $umbral) {
    // Crear alerta para ese departamento y semana
}
```

3. Registrar una acción en la auditoría

```
AuditLog::log('alert.closed', auth()->id(), 'Alert', $alertId, ['reason' =>
'manual_close']);
```

4. Ver quién creó una alerta

```
$alert = Alert::with('createdBy', 'department')->find($id);
$alert->createdBy->name; // nombre de la persona que la creó
```

5. Generar datos demo en consola (Tinker)

```
\App\Models\Alert::factory()->count(5)->create();
\App\Models\AuditLog::factory()->count(10)->create();
```

---

6) Qué archivos agregamos o tocamos

- Migraciones:
  - `add_consent_at_and_optional_department_to_users.php`
  - `create_settings_table.php`
  - `create_alerts_table.php`
  - `create_audit_logs_table.php`
- Modelos:
  - `App\Models\Setting` (cast de `value` como JSON)
  - `App\Models\Alert` (relaciones con `Department` y `User`)
  - `App\Models\AuditLog` (método estático `log(...)`)
  - `App\Models\Company`, `App\Models\Department` (soporte para factories)
  - `App\Models\User` (soporte para `consent_at` y factories)
- Seeders:
  - `SettingsSeeder` (valores iniciales clave)
  - `SetUsersConsentNullSeeder` (opcional para dejar `consent_at` en null)
  - Registro de `SettingsSeeder` en `DatabaseSeeder` (solo `local`/`staging`).
- Helper global:
  - `app/helpers.php` con la función `setting($key, $default = null)`.
- Factories (datos demo):
  - `CompanyFactory`, `DepartmentFactory`, `UserFactory`,
  - `AlertFactory`, `AuditLogFactory`, `SettingFactory`.

---

7) Cómo probar que todo funciona

1. Ver que existen las tablas:

```
Schema::hasTable('settings');
Schema::hasTable('alerts');
Schema::hasTable('audit_logs');
```

2. Probar lecturas de settings:

```
setting('anon_threshold');
setting('iec_alert_threshold');
setting('send_window');
```

3. Crear datos demo:

```
\App\Models\User::factory()->create();
\App\Models\Alert::factory()->create();
\App\Models\AuditLog::factory()->create();
\App\Models\Setting::factory()->create();
```

4. Ver relaciones:

```
\App\Models\Alert::with(['department','createdBy'])->latest()->first();
```

---

8) Beneficios para el negocio

- Decisiones basadas en datos: Alertas y auditoría aportan contexto y trazabilidad.
- Menos dependencia del equipo técnico: Cambiar umbrales y horarios desde “settings”.
- Cumplimiento y transparencia: Auditoría clara de acciones clave.
- Aceleración de demos y pruebas: Factories generan datos creíbles en segundos.

---

9) Próximos pasos sugeridos (opcionales)

- Mostrar en el dashboard los valores activos de configuración (umbral, ventana horaria, etc.).
- Crear un job/cron que use `send_window` para enviar recordatorios en la franja adecuada.
- Añadir panel simple para editar “settings” desde la interfaz (con permiso de admin).

---

10) Glosario rápido

- Seeder: Script que rellena la base de datos con datos iniciales.
- Factory: Plantilla para crear datos de prueba realistas.
- Helper: Función utilitaria disponible en todo el proyecto.
- Umbral (threshold): Valor a partir del cual se dispara una condición (ej.: alerta).

- Auditoría (audit log): Registro histórico de acciones importantes.

Si necesitas una versión abreviada para ejecutivos o para el README principal, podemos generar un resumen de 8–10 líneas.