ChatGPT

Modelo central de eventos y sistema de recompensas XP en LEFI

Eventos clave a registrar por módulo

Cada módulo principal de LEFI debe emitir eventos cuando ocurren acciones importantes del usuario. A continuación se listan los eventos clave por módulo (con sus códigos de módulo entre paréntesis):

- **Planificador Inteligente (M1)** – Eventos de productividad en la timeline:
- *Creación de bloque:* cuando el usuario crea una tarea o bloque de tiempo en su agenda. (Sirve para registrar uso activo de la planificación diaria).
- *Bloque marcado como completado:* cuando finaliza una tarea/bloque. Esto genera XP por tarea completada [1]. Si el bloque fue creado con asistencia de IA (integración con M4), se marca en el evento (p.ej. campo `payload.ai = true`) para recompensas adicionales [2].
- *Planificación diaria completada:* al concluir la planificación de un día completo. Puede ser un evento compuesto (ej. si el usuario planifica al menos N bloques en un día).

- *Racha de planificación:* evento especial cuando se acumulan X días seguidos planificando (ej. 14 días seguidos usando el planificador [3]). Este hito desbloquea una insignia de constancia ("Planificador Persistente") con bonificación XP [3].

- **Captura Rápida (M2)** – Eventos de captura de ideas y tareas al vuelo:

- *Nota/tarea capturada:* cada vez que el usuario captura una nota rápida (texto, voz o clip web) en su inbox. Registra la entrada en M2, usualmente ligada al Planificador (M1) para procesarla luego [4]. Se puede otorgar XP mínimo (p.ej. 1 XP por captura) para incentivar recoger ideas constantemente.

- **Segundo Cerebro - Notas PARA (M3)** – Eventos de gestión del conocimiento:

- *Creación de nota/proyecto:* cuando una idea del inbox se convierte en nota estructurada o proyecto en el sistema PARA [5].

- *Organización de conocimiento:* por ejemplo al mover notas entre Proyectos/Areas/Recursos/Archivo. Estos eventos indican aporte de datos de calidad (organización) pero podrían no dar XP directo, sino alimentar analíticas de uso.

- **Plantillas inteligentes / IA (M4)** – Eventos de uso de asistencia inteligente:

- *Aplicación de plantilla inteligente:* cuando el usuario aplica una "semana tipo" u otra plantilla sugerida por IA en su calendario [4].

- *Sugerencia de IA aceptada:* cuando la IA reorganiza la agenda o sugiere un bloque y el usuario la acepta. Esto puede registrarse como parte del evento de creación de bloque con IA (M1) o por separado. Otorga XP extra por uso efectivo de IA (p.ej. +25 XP por completar un bloque sugerido por IA [6]). Tras cierta cantidad, podría disparar un logro "Productividad Asistida" [7].

• **Analítica & Revisión (M5)** – Eventos de reflexión semanal y feedback:

• *Revisión semanal completada:* cuando el usuario completa su revisión productiva semanal en M5 (ej. los domingos por la tarde). Es una actividad de alto valor analítico que otorga una gran recompensa (p.ej. +40 XP) [8] .
• *Feedback al asistente de IA:* si durante la revisión el usuario brinda sugerencias o corrige recomendaciones de la IA, se registra este evento. Tiene recompensa adicional (por ejemplo +10 XP extra) por mejorar el sistema [9] .

• *Vista de analíticas consultada:* evento cuando el usuario revisa sus estadísticas personales. No genera XP, pero queda registrado para comprender interés en analítica.

• **Rutinas & Sueño (M6)** – Eventos de bienestar y hábito de sueño:

• *Registro de sueño:* cada mañana el usuario registra sus horas de sueño/calidad (manualmente o vía dispositivo). Este evento se almacena con datos de descanso (en `payload`, p.ej. horas dormidas, calidad). Otorga una pequeña recompensa XP acumulable por aporte de datos fisiológicos útiles [10] .
• *Completar rutina diaria:* si el módulo incluye rutinas (ej. rutina matinal/nocturna), marcar una rutina como completada genera un evento con pequeño XP (refuerzo positivo diario).

• *Streak de sueño:* evento logro cuando el usuario registra su sueño X días seguidos (p.ej. 7 días consecutivos [11]). Desbloquea la medalla "Descanso Constante" y recompensa bonus (+10 XP) por hábito saludable consistente [11] .

• **Automatizaciones (M7)** – Eventos de automatización de flujo:

• *Regla activada:* cuando se ejecuta automáticamente una regla definida por el usuario (ej. mover un bloque según hora). Inicialmente puede no asignar XP, pero queda registrado. Futuras extensiones podrían otorgar XP por crear/utilizar automatizaciones, incentivando eficiencia.

• **Conexiones & Grupos (M8)** – Eventos sociales y colaborativos:

• *Bloque compartido en grupo:* cuando un usuario comparte una tarea o bloque con su grupo/círculo (indicando uso colaborativo de la productividad).
• *Compromiso grupal cumplido:* cuando un usuario completa una tarea comprometida dentro de un grupo (por ej., completar un hábito grupal o un desafío compartido). Otorga XP social y refuerza la colaboración.

• *Interacción en grupo:* participación activa, como publicar un mensaje o dar feedback en un grupo. Puede registrar actividad pero con XP menor. Si el usuario participa activamente durante una racha (ej. 14 días participando [12]), se desbloquea la medalla "Colaborador Constante" con bonus (+15 XP) y quizá contenido colaborativo extra [12] .

• **Salud Femenina (M9)** – Eventos de seguimiento del ciclo y salud femenina:

• *Registro diario de ciclo:* ingreso de datos diarios del ciclo menstrual o síntomas. Similar a M6, aporta datos sensibles. Podría dar XP modesto por cada registro diario de salud [10] , enfatizando consistencia.
• *Ciclo completo registrado:* cuando el usuario completa un ciclo entero (ej. ~28 días de datos sin faltar). Desencadena un logro **"Ciclo Completado"** – insignia en el área de salud femenina con

recompensa (+15 XP) [13] . Además, puede desbloquear un **pack de contenido** especializado para ese módulo (p.ej. recomendaciones expertas adaptadas a su ciclo) [14] .
- *Módulo activado (opt-in):* evento especial cuando el usuario habilita voluntariamente un módulo sensible como M9 (salud femenina) o decide compartir datos anónimos para fines científicos. Se registra con tipo de módulo y consentimiento. Otorga una **recompensa ética simbólica**: insignia honorífica ("Colaborador/a por la Ciencia") en su perfil y quizá un pequeño XP de cortesía (+5) [15] . Más importante, habilita beneficios como acceso anticipado a funciones premium en agradecimiento a su aporte altruista [15] .

Cada uno de estos eventos captura **acciones valiosas** del usuario en su contexto, alineadas con los objetivos de cada módulo [16] [17] . De este modo, los datos generados en todos los módulos quedan registrados de forma consistente en un **sistema central de eventos**, proporcionando la base para calcular recompensas y analizar el uso en cada área de la app.

Transformación de eventos en XP, medallas y logros

Los eventos registrados se traducen en **puntos de experiencia (XP)** y otros reconocimientos de manera transparente. La estrategia es asignar **pequeñas recompensas frecuentes** por acciones cotidianas y **grandes recompensas** por hitos o acciones de alto valor [10] :

- **Puntos XP por acciones diarias:** Muchos eventos generan XP inmediato al ocurrir. Se definen valores base según la importancia:
- Acciones simples y frecuentes (e.g. crear o completar un bloque, registrar sueño un día) dan una cantidad baja de XP (p. ej. 1–5 XP). Son "micro-recompensas" acumulativas para reforzar hábitos diarios [10] .
- Acciones de mayor esfuerzo o valor (p. ej. completar la revisión semanal, cumplir un objetivo semanal) otorgan XP significativamente mayor (p. ej. 20–50 XP) [8] . Esto premia las contribuciones de calidad y animan a completar tareas más sustanciales.

- Ejemplo: marcar un bloque como **completado** podría dar 2 XP base, pero si el bloque se planificó con ayuda de IA (evento combinado M1+M4), se suman +25 XP adicionales por aprovechar la IA [7] . Completar la **revisión semanal** da ~40 XP dado su alto valor analítico [9] , y si además dejó feedback a la IA, +10 XP extra de bono [9] .

- **Insignias y logros por hitos:** Cuando ciertos eventos alcanzan una meta acumulativa o streak, el sistema otorga medallas/logros:

- **Rachas (streaks) positivas:** Al sostener un hábito X días seguidos (p.ej. 7 días de sueño, 14 de planificación, 14 de participación grupal), se desbloquea una medalla temática reconociendo la constancia (como *"Descanso Constante"*, *"Planificador Persistente"*, *"Colaborador Constante"*) [11] [12] . Cada logro viene con XP extra de bonificación (por ejemplo +10, +15 XP) como premio adicional [18] [19] .
- **Metas completas en módulos:** Lograr un ciclo completo en M9, completar cierto número de tareas con IA, etc., generan logros especiales (ej. *"Ciclo Completado"* en salud femenina con +15 XP [13]). Estos logros aparecen en la sección de perfil del usuario, mostrando un trofeo o badge.

- **Recompensas funcionales asociadas:** Muchas insignias no solo son simbólicas, sino que **desbloquean contenido o ventajas**. Por ejemplo, *Planificador Persistente* habilita packs/ plantillas avanzadas de productividad [20] , *Ciclo Completado* da acceso a contenido premium de salud femenina [14] , *Colaborador Constante* podría desbloquear desafíos o insignias de equipo adicionales [21] . Esto asegura que cada logro conlleva un beneficio real y no solo "puntos

vacíos” [22] . El usuario ve que su buen uso de la app le gana **nuevas funciones, conocimiento o reconocimiento tangible** en el ecosistema.

- **Niveles de usuario:** Los XP acumulados se asocian a un **nivel** global del usuario. A medida que suma XP de cualquier módulo, progresa en nivel (ej.: Nivel 1 a 2 a 3, etc.). Cada nivel requiere cierta cantidad de XP (creciente). Una **barra de progreso** muestra cuánto falta para el siguiente nivel. Los niveles sirven como indicador de experiencia general en la aplicación, motivando al usuario a seguir acumulando puntos para “subir de nivel” [23] . Se puede incorporar recompensas menores por subir nivel (p.ej. reconocimiento visual, confeti, quizás algún beneficio menor o simplemente satisfacción).

- **Mecánicas inspiradas en Duolingo/Me+:** Al igual que Duolingo, se emplean puntos de experiencia, niveles, insignias y rachas diarias para fomentar hábito [1] . La diferencia es un enfoque **ético**: no se castiga abandonar la racha, solo se premia retomarla [24] [25] . También se pueden incluir misiones diarias o semanales (quests) con pequeñas metas – estrategia que en Duolingo aumentó DAUs en 25% [26] – pero siempre alineadas con comportamientos beneficiosos en LEFI (ej. “planifica 3 días esta semana” en lugar de metas arbitrarias). Cada recompensa está atada a acciones valiosas del usuario, reforzando una relación ganar-ganar: el usuario mejora su vida/productividad y la plataforma reconoce ese esfuerzo [27] [28] .

En resumen, **cada evento significativo produce feedback positivo inmediato o a corto plazo**: ya sea en forma de XP incremental o un logro visible. Las recompensas en LEFI **siempre funcionan como refuerzo positivo** (encouragement) y nunca castigo [25] [29] . Si el usuario un día falla en su rutina, no pierde puntos; simplemente no gana esos XP, y la app lo anima a continuar sin hacerle sentir culpa [24] [30] . Este sistema de puntos y logros está diseñado para mantener motivado al usuario de manera saludable y sostenible en el tiempo [31] [32] .

Visualización del progreso (barra de XP y resumen semanal)

Para que el usuario perciba su avance y se mantenga motivado, LEFI incluirá **indicadores visuales claros del progreso**:

- **Barra de XP / Nivel:** En la pantalla principal (dashboard) o perfil se mostrará una barra de progreso con la XP acumulada hacia el siguiente nivel [33] . Por ejemplo, un widget diario podría indicar: *“Hoy: 15 XP ganados. Esta semana: 80 XP (de 100 XP necesarios) para subir a Nivel 3”* [33] . De este modo, el usuario tiene feedback inmediato de lo logrado en el día y sabe cuánto le falta para su próxima meta de nivel. La barra refuerza visualmente la continuidad: incluso si un día ganó poco, ve el progreso semanal avanzar [34] .

- **Resumen semanal de actividad:** Al cierre de cada semana, la app puede presentar un resumen tipo *“report”* con gráficos simples: XP total ganado cada día (ej. un gráfico de barras de lunes a domingo), logros desbloqueados en la semana, y mensajes de felicitación. Por ejemplo: “ *¡Genial! Completaste tu rutina 5 de 7 días y ganaste 80 XP esta semana. Sigue así para alcanzar Nivel 3 pronto.”* Estos resúmenes consolidan la información de la vista `vw_daily_xp` (ver sección siguiente) para que el usuario la entienda de un vistazo. Es similar a cómo apps de hábitos muestran semanas cumplidas o cómo Duolingo envía un email resumen semanal de XP.

- **Indicadores de racha y hábitos:** Junto a la barra de XP podría mostrarse un contador de racha (días consecutivos usando cierto módulo) u otros progresos específicos. Por ejemplo, un icono de fueguito con el número de días de racha de planificación, o un gráfico de “días con sueño

registrado" en la semana. Siempre se mostrará de forma **amigable y opcional**, sin generar ansiedad por perder la racha [32] [24]. Si la racha se rompe, el contador se reinicia silenciosamente pero quizá con un mensaje motivador de retomarla.

- **Sección de logros y medallas:** En el perfil del usuario habrá una "vitrina" de insignias logradas. Los logros pueden organizarse por categoría/módulo (Productividad, Salud, Colaboración, etc.) [35]. Cada categoría muestra los badges obtenidos y también los *pendientes* (atenuados o grises), insinuando próximos objetivos alcanzables [36]. Por ejemplo, bajo *Salud* aparecerá la medalla *"Descanso Constante"* si ya la obtuvo, y quizá una medalla *"Madrugador/a"* grisada si existe tal logro por despertar temprano X días. Esto invita a explorar funcionalidades: el usuario puede tocar un logro atenuado para ver qué requisito tiene, motivándolo de forma autónoma ("¿qué necesito para esta medalla?") [37]. La presentación será visualmente atractiva, estilo vitrina de trofeos, celebrando los éxitos.

- **Notificaciones y feedback inmediato:** Cada vez que el usuario gana XP o un logro, la app debe mostrar un pequeño refuerzo en tiempo real: por ejemplo, una animación o notificación in-app "+5 XP" con un mensaje como "¡Bien hecho!" [32]. Al desbloquear una insignia, un modal o pantalla de felicitación presentará el badge con su nombre y beneficio (similar a cómo Duolingo muestra un sello al completar una unidad). Estos momentos de celebración fortalecen el vínculo positivo con la app.

En conjunto, esta visualización constante del progreso convierte datos en motivación tangible. El usuario **siempre puede ver sus beneficios acumulados** – ya sea la barra de nivel subiendo, el conteo de días seguidos o su colección de medallas creciendo – lo que inspira a seguir participando regularmente en LEFI [1] [31].

Datos para analítica personal y monetización ética

El registro centralizado de eventos no solo impulsa la gamificación, sino que alimenta **analíticas personales** y la **monetización ética de datos** propuesta por LEFI:

- **Analítica personal por módulo:** Dado que cada evento incluye qué módulo y acción se realizó, es posible generar estadísticas detalladas para el usuario. Por ejemplo, usando los eventos podemos calcular:
- *Tiempo/productividad:* Cuántos bloques completa en promedio por día (M1), su consistencia semanal en planificación, etc.
- *Salud/bienestar:* Frecuencia de registro de sueño (M6) y horas promedio dormidas, consistencia en rutinas diarias, variabilidad del ciclo menstrual (M9), etc.
- *Colaboración:* Nivel de actividad en grupos (M8), días con interacciones sociales, etc.
- *Uso general:* Módulos más utilizados, rachas de días activos en la app, etc.

Estos datos alimentan vistas de **analítica personal** dentro de la app (M5). Por ejemplo, una sección de estadísticas podría mostrar "Horas de sueño vs. productividad" u otros insights combinados, gracias a que todos los eventos están en una tabla común fácil de consultar. También se pueden dar *insights* personalizados tipo *"Eres más productivo los días que duermes* $\geq 8h$*"* si la correlación se extrae de los eventos de sueño + tareas completadas (ejemplo de beneficio directo al usuario).

- **Economía de datos y monetización ética por módulo:** LEFI planea monetizar datos **agregados y anónimos** provenientes de estos eventos, sin comprometer la privacidad [38]. Organizar los eventos con un campo de tipo/módulo permite filtrar por área para crear *"paquetes de datos"* temáticos:

- Por ejemplo, los eventos de M6 (sueño) de todos los usuarios, agregados, pueden generar un informe sobre hábitos de descanso poblacionales. Similarmente, los eventos de M1/M2 (productividad) agregados muestran patrones de planificación efectivos. Estos insights por módulo pueden ofrecerse a investigaciones académicas, instituciones de salud o productividad, etc., **siempre de forma anónima y ética** [38] [39].
- Cada módulo activado por el usuario representa una categoría de datos valiosos. Si un usuario opta por compartir sus datos de cierto módulo (opt-in voluntario), esos eventos se etiquetan como *compartibles*. Luego, LEFI los incluye en el dataset agregado de ese módulo. **Nunca se vende un dato individual** ni se identifica a usuarios [40], solo tendencias colectivas (e.g. "X% de usuarios con rutina matinal reportan mejor rendimiento laboral").

- En recompensa, el usuario recibe **beneficios**: la plataforma deja claro que *"su esfuerzo y datos se traducen en mejoras tangibles para él"* [41]. Por ejemplo, al mantener completos sus registros de salud, desbloquea análisis personalizados más profundos; al cumplir retos de productividad, obtiene acceso anticipado a nuevas funciones de IA [41]. Incluso se pueden ofrecer recompensas fuera de la app, como descuentos en suscripción Premium o contenido exclusivo (webinars, ebooks) para usuarios de alto nivel, premiando su lealtad y aporte [42].

- **Integración con vistas de monetización:** Internamente, la tabla de eventos central puede usarse para construir vistas o informes que consoliden valor por módulo. Por ejemplo, una vista `vw_module_data_quality` que calcule métricas como "días activos por módulo por usuario" o "porcentaje de datos completos en módulo X". Estas métricas ayudan a identificar usuarios altamente consistentes en ciertos módulos, que podrían calificar para *packs* avanzados o ser casos de estudio (con permiso). También servirán para proyecciones: p.ej., si X% de usuarios usan M6 diariamente durante 6 meses, el dataset de sueño tiene cierto tamaño y valor para un estudio (útil en estimaciones financieras). Todo **respetando privacidad**: los eventos se agregan y anonimizan antes de cualquier uso externo [38].

- **Transparencia al usuario:** La plataforma puede incluso reflejar de vuelta al usuario cómo sus datos contribuyen. Por ejemplo, tras completar una semana de registros, mostrar mensajes del tipo: *"📊 Gracias a tus 7 registros de sueño esta semana, contribuyes anónimamente a un estudio sobre hábitos de descanso saludable"* [43]. Esto refuerza la motivación altruista: sus logros no solo le dieron XP, sino que ayudan a la comunidad científica, alineado con la misión ética de LEFI.

En síntesis, el **modelo central de eventos** se diseña no solo para el sistema de juego interno, sino también para habilitar un flujo de datos estructurado y modular. Esto permite **analítica personal enriquecida** (para el usuario) y **monetización ética** (para la empresa) sin sistemas separados: la misma tabla de eventos alimenta ambos, con las consideraciones de privacidad y consentimiento incorporadas [39] [44].

Diseño de la tabla central de eventos

El núcleo del modelo es una tabla de base de datos (en Supabase/Postgres) que consolida todos los eventos de todos los módulos. Esta **tabla** `events` tendrá un esquema flexible y extensible, por ejemplo:

```
CREATE TABLE events (
    id            BIGSERIAL PRIMARY KEY,
    user_id       UUID NOT NULL REFERENCES users(id),
    event_type    TEXT NOT NULL,       -- código del evento (acción o logro)
```

```
    event_time    TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    module        TEXT NOT NULL,       -- código de módulo, e.g. 'M1', 'M6'
    payload       JSONB,               -- datos adicionales del evento
    xp_points     INT NOT NULL DEFAULT 0  -- puntos XP otorgados por este
evento
);
```

Explicación de campos:

- `user_id`: Identifica al usuario que realizó la acción (en Supabase podría referenciar a la tabla de autenticación de usuarios). Esto permite agrupar eventos por usuario fácilmente.
- `event_type`: Nombre o código del evento ocurrido. Se utilizará una **nomenclatura consistente**, por ejemplo:
- Acciones directas: `"M1_BLOCK_COMPLETED"`, `"M6_SLEEP_LOG"`, `"M8_GROUP_POST"`.
- Logros/insignias: `"ACHIEVEMENT_SLEEP_STREAK_7"`, `"ACHIEVEMENT_CYCLE_COMPLETE"`, etc.
- Usar prefijos (`M#_` o `ACHIEVEMENT_`) facilita filtrar tipos similares. Alternativamente, se podría dividir en dos campos (p.ej. `event_category` y `event_action`), pero un código textual único es sencillo y extensible.
- `event_time`: Timestamp de cuándo ocurrió el evento (usando zona horaria consistente, e.g. UTC). Registra la secuencia temporal para análisis de series de tiempo (rachas, frecuencia diaria, etc.).
- `module`: Código del módulo origen (M1..M9, etc.). Redundante con el prefijo de `event_type`, pero útil para indexar y hacer agregaciones por módulo rápidamente. Se puede incluso hacer que `module` sea una *FK* a una tabla de módulos (tabla maestra que liste M1..M9) para mantener consistencia.
- `payload`: Datos en formato JSONB que describen detalles específicos del evento. Esto le da flexibilidad al esquema para nuevos módulos:
- Por ejemplo, en un evento `"M6_SLEEP_LOG"`, el JSON podría ser `{"hours":7.5,"quality": "Alta"}`.
- En un `"M1_BLOCK_COMPLETED"`, podría incluir `{"block_id": "...", "planned_duration": 30, "ai_assisted": true}`.
- En un `"ACHIEVEMENT_CYCLE_COMPLETE"`, incluir `{"cycle_length": 28,"start_date":"2025-07-01"}`.
- **Nota:** JSONB permite añadir campos sin modificar el esquema, ideal para que nuevos módulos agreguen info propia. También se pueden indexar campos JSON si luego se quieren consultar (ej. buscar eventos con `payload.ai_assisted = true`).
- `xp_points`: la cantidad de experiencia que otorgó este evento. Inicialmente puede ser 0 y luego actualizarse mediante lógica (trigger/función) según el tipo de evento. Alternativamente, podríamos no almacenarlo y calcularlo sobre la marcha uniéndolo con una tabla de valores XP; sin embargo, guardarlo simplifica las consultas de suma de XP y permite ajustes manuales si fuese necesario.

Ejemplo: cuando el usuario completa un bloque con ayuda de IA, la app inserta un evento:

```
INSERT INTO events(user_id, event_type, module, payload)
VALUES
  ('<uid>', 'M1_BLOCK_COMPLETED', 'M1', '{"block_id":"abc123","ai_assisted":
true}');
```

Luego, mediante un *trigger* o función, se le asignará `xp_points = 25` a ese evento por haber `ai_assisted=true` (ver sección de cálculo automático). Si el mismo bloque completado sin IA quizá obtendría `xp_points = 5` base.

Diseñando la tabla así, logramos un **log unificado** de todo lo que sucede relevante a recompensas. Es eficiente: con un solo INSERT por evento desde cualquier módulo, los datos aterrizan aquí sin necesidad de múltiples tablas separadas. Esto **facilita consultas globales** (p.ej., todos los eventos de un usuario, o filtrar eventos de tipo "ACHIEVEMENT_") y asegura consistencia en cómo guardamos los eventos de gamificación en toda la app.

Además, mantener un esquema simple (campos genéricos + JSON) asegura que agregar un nuevo tipo de evento no requiere migraciones de base de datos – solo definir el nuevo código de `event_type` y comenzar a insertarlo. La modularidad de LEFI se refleja en este diseño: los módulos están desacoplados pero comparten este **bus de eventos** común para reportar sus acciones.

Vista `vw_daily_xp` – XP diario por usuario

Para visualizar la experiencia acumulada por día (por ejemplo, en la barra de progreso semanal mencionada) crearemos una **vista SQL materializada** llamada `vw_daily_xp`. Esta vista resumirá la tabla de eventos, agrupando por usuario y fecha. Su estructura lógica sería:

```
CREATE OR REPLACE VIEW vw_daily_xp AS
SELECT
    user_id,
    date_trunc('day', event_time) AS fecha,
    SUM(xp_points) AS xp_dia
FROM events
GROUP BY user_id, date_trunc('day', event_time);
```

Funcionamiento: la vista redondea cada timestamp al día (ignorando la hora) y suma los puntos XP de todos los eventos de cada usuario en ese día. El resultado es una tabla virtual con columnas: `user_id`, `fecha` (ej. 2025-07-15) y `xp_dia` (total de XP obtenido por ese usuario en esa fecha).

Ejemplo de filas que `vw_daily_xp` podría producir:

user_id	fecha	xp_dia
user_123	2025-07-14	35
user_123	2025-07-15	10
user_456	2025-07-15	50

Esto indicaría que el usuario 123 ganó 35 XP el 14 de julio y 10 XP el 15 de julio, mientras que el usuario 456 ganó 50 XP el 15 de julio, etc. Con esta vista, es trivial obtener: - **XP de hoy por usuario:** `SELECT xp_dia FROM vw_daily_xp WHERE user_id = X AND fecha = CURRENT_DATE;` - **XP de la última semana:** sumando los últimos 7 días por usuario, o directamente expandiendo la vista:

```sql
SELECT user_id, date_trunc('week', fecha) AS semana, SUM(xp_dia)
FROM vw_daily_xp
GROUP BY user_id, date_trunc('week', fecha);
```

(O usar directamente la vista para cada día en el cliente).

En la implementación real, podríamos hacer `vw_daily_xp` una **vista materializada** que se refresque cada cierto intervalo o tras ciertos inserts, si el volumen de eventos es muy alto, para acelerar las lecturas en la app. Sin embargo, inicialmente una vista normal es suficiente dado que una consulta de agrupación por índice de usuario y fecha es eficiente.

Esta vista se relaciona directamente con la tabla de eventos: cada vez que se inserta un nuevo evento con XP, automáticamente afectará a la suma de ese día. Por ende, la barra de progreso diaria/semanal del usuario puede consultar esta vista para mostrar "XP ganado hoy" y "esta semana" sin cálculos complejos en el frontend.

Además, `vw_daily_xp` sirve como base para analítica personal: por ejemplo, la app podría graficar los valores de `xp_dia` de las últimas 4 semanas para mostrar la tendencia de actividad del usuario. También podemos combinar con la información de módulos si queremos un desglose (e.g., XP por módulo por día, con otro GROUP BY que incluya `module`).

En resumen, `vw_daily_xp` proporciona una **resumen sencillo pero poderoso** del compromiso diario de cada usuario con la app, alimentando la UI de gamificación (progreso semanal) y cualquier análisis temporal que necesitemos.

Cálculo automático de XP (funciones y triggers en el backend)

Para que el sistema sea consistente y fácil de mantener, la asignación de XP y el desbloqueo de logros deben ocurrir automáticamente en el servidor cada vez que se registra un evento. En Supabase/Postgres esto se puede lograr con:

- **Triggers de Postgres:** Una *trigger function* en la tabla `events` que, **después de insertar** un evento nuevo, calcule la recompensa correspondiente y actualice el registro (o inserte eventos adicionales si es un logro). El trigger garantiza que, sin importar desde qué módulo o parte del app se insertó el evento, las reglas de XP se apliquen uniformemente.

- **Funciones definidas por el usuario (RPC):** Alternativamente, usar un procedimiento almacenado que los clientes (módulos) llamen para registrar eventos. Esa función (`log_event()` por ejemplo) podría encapsular la lógica de asignar XP y crear logros. Esto es útil si queremos mantener la lógica en SQL en un solo lugar, a costa de requerir que los módulos llamen al RPC en lugar de insertar directamente en la tabla.

Un enfoque híbrido óptimo es definir un **trigger AFTER INSERT** que llame a una función `compute_xp_and_achievements()`. Así los módulos simplemente insertan eventos y la base de datos se encarga del resto. Es más difícil cometer errores desde el lado del cliente.

1. Asignación de XP mediante trigger:

La trigger function revisará el `event_type` recién insertado y asignará XP según reglas predefinidas. Podría implementarse con un CASE o una consulta a una tabla de configuración de puntos. Ejemplo simplificado de lógica en PL/pgSQL:

```
CREATE OR REPLACE FUNCTION compute_xp_and_achievements()
RETURNS TRIGGER AS $$
DECLARE
    base_xp INT;
    days_count INT;
BEGIN
    -- Asignar XP base según tipo de evento
    CASE NEW.event_type
      WHEN 'M1_BLOCK_COMPLETED' THEN
         base_xp := CASE
                      WHEN (NEW.payload->>'ai_assisted')::BOOLEAN IS TRUE
THEN 5 + 20
                      ELSE 5
                    END;
      WHEN 'M5_WEEKLY_REVIEW' THEN base_xp := 40;
      WHEN 'M5_AI_FEEDBACK'  THEN base_xp := 10;
      WHEN 'M6_SLEEP_LOG'    THEN base_xp := 2;
      WHEN 'M8_GROUP_TASK_DONE' THEN base_xp := 5;
      -- ... demás casos
      ELSE
         base_xp := 0;
    END CASE;
    -- Actualizar XP base del evento
    NEW.xp_points := base_xp;

    -- Regla de logro: 7 días seguidos registrando sueño (M6)
    IF NEW.event_type = 'M6_SLEEP_LOG' THEN
       SELECT COUNT(DISTINCT date_trunc('day', event_time))
       INTO days_count
       FROM events
       WHERE user_id = NEW.user_id
         AND event_type = 'M6_SLEEP_LOG'
         AND event_time >= (CURRENT_DATE - interval '6 days')
         AND event_time < CURRENT_DATE;
       IF days_count = 6 THEN
          -- Insertar evento de logro (7mo día consecutivo)
          INSERT INTO events(user_id, event_type, module, payload, xp_points)
          VALUES (NEW.user_id, 'ACHIEVEMENT_SLEEP_STREAK_7', 'M6',
                  jsonb_build_object('streak_days', 7), 10);
       END IF;
    END IF;
    -- (Reglas similares para otras rachas o logros: 14 días M1, 14 días M8,
etc.)

    RETURN NEW;
END;
```

```
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_events_after_insert
AFTER INSERT ON events
FOR EACH ROW
EXECUTE PROCEDURE compute_xp_and_achievements();
```

Explicación: La función arriba primero determina un `base_xp` según el tipo de evento (usando un CASE, donde por ejemplo completar un bloque da 5 XP y se suma +20 si `ai_assisted=true`, total 25 XP [7]). Luego asigna ese valor al campo `xp_points` del evento insertado.

Después, comprueba si el nuevo evento debería detonar un logro: - En el ejemplo, si se insertó un evento de registro de sueño (`M6_SLEEP_LOG`), la función cuenta cuántos días distintos en los últimos 6 días previos ya tenían registros. Si encuentra 6 (es decir, el usuario registró sueño cada uno de los 6 días anteriores consecutivamente), entonces el evento actual es el día 7 consecutivo. En tal caso, inserta automáticamente un nuevo evento extra de tipo `ACHIEVEMENT_SLEEP_STREAK_7` asociado al usuario, con un payload indicando 7 días y otorgando 10 XP [11]. Este evento de logro aparecerá en la tabla `events` igual que los demás, quedando registrado que el usuario obtuvo la medalla *"Descanso Constante"*. - Se pueden anidar más reglas similares: para M1 (14 días planificación) verificar 13 días previos, para M8 (14 días grupos) igual [12], etc. También para cosas como "ciclo completo": al insertar un evento que marque el final de ciclo (p.ej. un `M9_CYCLE_END`), la función podría verificar si hubo al menos X días de registros en el ciclo y entonces insertar el logro `ACHIEVEMENT_CYCLE_COMPLETE` con +15 XP [13].

Usar triggers asegura que **el cálculo ocurre en tiempo real** y de forma centralizada. El usuario crea una entrada de sueño el séptimo día y en ese mismo instante el servidor calcula que completó la racha y genera la recompensa. No depende de lógica en el cliente (que podría ser manipulable). Además, tener las reglas en SQL facilita ajustarlas en un solo lugar y potencialmente aprovechar la potencia de Postgres (consultas de fechas, etc.).

2. Uso de RPC/funciones: Alternativamente, podríamos implementar la función anterior como un procedimiento que se llame manualmente. Ejemplo conceptual: `SELECT log_event('M6_SLEEP_LOG', '{"hours":7}')`. La función SQL haría internamente lo mismo (insertar evento, asignar XP, crear logro si aplica) y retornaría quizás la fila insertada. Esto puede ser útil para depurar o si se quiere mayor control transaccional. No obstante, en Supabase los triggers funcionan bien y mantienen la capa de datos independiente de la aplicación.

3. Tabla de configuración para XP (opcional): En lugar de codificar los valores XP en la función, podríamos tener una tabla `event_types` con columnas `event_type` y `base_xp`, e incluso umbrales para logros (ej. un registro para `'M1_BLOCK_COMPLETED'` con base_xp 5, otro para `'ACHIEVEMENT_SLEEP_STREAK_7'` con base_xp 10, etc). La función entonces haría algo como:

```
SELECT base_xp INTO base_xp FROM event_types WHERE event_type =
NEW.event_type;
```

Esto permite ajustar puntos sin desplegar código. Y al añadir nuevos eventos, solo insertamos su config en `event_types`. Si un evento no tiene entrada, se asume XP 0. Este enfoque **data-driven** mejora la extensibilidad (un gerente de producto podría actualizar puntos vía DB en vez de pedir cambio de código).

4. Seguridad y consistencia: Se puede envolver la lógica de triggers en una transacción para que la inserción de un evento y cualquier evento adicional de logro ocurran juntos (en nuestro PL/pgSQL ya es atomic por función). Supabase también soporta **Row Level Security**, pero en este caso, dado que solo nuestro backend escribe en `events`, podemos controlar acceso para que los usuarios finales no manipulen sus puntos directamente.

5. Testing: Es recomendable probar la función de cálculo con distintos escenarios (ej. simular 7 días seguidos de sueño) para asegurar que los logros se otorgan correctamente y solo una vez. También manejar casos borde: si el usuario registra dos veces el mismo día, quizás contamos 1 día; si falta un día intermedio, la racha se rompe, etc. Estas reglas deben definirse claramente (p.ej., ¿qué constituye "14 días seguidos"? – se asume días naturales consecutivos con al menos un evento diario).

En conclusión, la **automatización de XP mediante lógica en el backend** garantiza un sistema de recompensas robusto. Los desarrolladores de cada módulo no necesitan implementar cálculo de puntos; solo disparan eventos. El sistema central decide los XP, crea las medallas pertinentes y actualiza las vistas de progreso. Esto reduce errores y asegura que la gamificación se comporta de forma consistente con las reglas de negocio definidas.

Modularidad y extensibilidad para futuros módulos

El diseño propuesto soporta la naturaleza modular de LEFI: nuevos módulos o funcionalidades pueden integrarse al sistema de eventos y XP **sin "romper" nada existente**. Algunas consideraciones para lograr esta extensibilidad:

- **Esquema flexible por diseño:** La tabla `events` usa campos genéricos (`event_type`, `payload`) en vez de columnas rígidas específicas. Esto significa que si mañana se añade el módulo **M10: Nutrición**, por ejemplo, podremos empezar a insertar eventos como `"M10_MEAL_LOG"` o `"M10_CALORIE_GOAL_MET"` con sus datos en JSON sin necesidad de alterar la estructura de la tabla. Los módulos actuales ignorarán esos nuevos tipos si no los conocen, y el sistema central los tratará simplemente como filas adicionales.

- **Convención de tipos y modularidad:** Al prefijar cada `event_type` con el código del módulo (M#), mantenemos separación lógica. Un módulo nuevo tendrá sus propios prefijos (M10, M11, etc.) evitando colisiones de nombres. También podemos filtrar fácilmente por `module` para aislar los eventos de ese componente. Esto encaja con la arquitectura modular de LEFI, donde cada componente es independiente pero **se comunica a través de interfaces comunes** [45]. Aquí, la interfaz común es nuestro registro de eventos.

- **Actualización de lógica de XP mínima:** Si añadimos un módulo nuevo con eventos que dan XP, solo hace falta:

  - Definir en la tabla de configuración de XP los nuevos tipos con sus valores (si usamos esa tabla) o actualizar el CASE en la función trigger para incluirlos.
  - (Opcional) Si el nuevo módulo tiene logros compuestos (p.ej. rachas), agregar las reglas correspondientes en la función trigger. Sin embargo, muchas mecánicas pueden reutilizar plantillas existentes: por ejemplo, cualquier módulo que requiera racha de X días puede usar una función genérica que reciba el `event_type` y X como parámetro para verificar rachas. Esto evita duplicar mucho código. Podríamos generalizar la comprobación de rachas en la función `compute_xp_and_achievements` para que sea data-driven (ej., una tabla de logros con `event_type` e `days_required`).

- Crear las entradas de descripción de logro para la interfaz (mostrar nombre e icono de la medalla, etc.), lo cual es independiente del backend.

- **Aislamiento de fallos:** Gracias a que cada módulo opera insertando en `events`, un módulo nuevo mal implementado no debería afectar el sistema global más allá de sus propios eventos. Por ejemplo, si M10 tuviera un bug y enviara eventos incorrectos, esos podrían filtrarse o ignorarse en análisis sin perjudicar la XP de otros módulos. La arquitectura modular asegura que incluso si un módulo se deshabilita o falla, los demás y el sistema de gamificación global siguen funcionando [45]. Los eventos de un módulo apagado simplemente dejarán de llegar, pero la tabla `events` conservará el historial y las métricas hasta ese punto.

- **Backward compatibility:** Si en un futuro se decide cambiar la definición de un evento o eliminarlo, es mejor marcarlo obsoleto que reutilizar su código para otra cosa. Mantener un catálogo de `event_type` válido ayuda. Por ejemplo, si `M4` cambia de propósito en el futuro, podríamos deprecar los viejos `event_type` y introducir nuevos, manteniendo los antiguos para no romper consultas históricas. Las vistas agregadas (como `vw_daily_xp`) seguirán funcionando ya que simplemente suman XP; no dependen de conocer el significado de cada evento.

- **Escalabilidad:** A medida que los módulos crecen, la tabla de eventos sí crecerá en filas. Es importante indexar por `user_id` y/o `event_time` para consultas rápidas. Podemos también partir por rangos de fechas si llega a ser muy grande (p.ej. particionado por año). Supabase/Postgres escala bien con millones de filas si está bien indexado. Además, podríamos implementar **barridos periódicos** para materializar algunas métricas históricas (ej. XP mensual) y así archivar eventos muy antiguos si fuera necesario, pero manteniendo la precisión de niveles/insignias conseguidos.

- **Incorporación de nuevos tipos de recompensas:** La modularidad no solo aplica a módulos funcionales, sino también a mecánicas de gamificación. Si en el futuro quisiéramos agregar, digamos, un *"Leaderboard"* global o entre amigos, podríamos usar los mismos eventos XP sumados por usuario para construirlo. Si quisiéramos implementar *"quests semanales"* (misiones transversales a varios módulos), podríamos definir event_types específicos (ej. `QUEST_2025W30_COMPLETED` otorgando XP) que se activan al cumplir condiciones combinadas de varios eventos. Todo esto sin cambiar la estructura básica; solo añadiendo más lógica de alto nivel que crea nuevos eventos cuando se cumplen criterios.

En conclusión, el modelo central de eventos está pensado para ser **futuro-proof**: nuevos módulos como *Salud Mental, Finanzas, Nutrición, etc.* podrán "enchufarse" al sistema de recompensas simplemente emitiendo eventos a la tabla central. Las mejores prácticas de **arquitectura modular** de LEFI [45] se cumplen, ya que el core de gamificación actúa como una pieza separada a la que todos los demás se conectan. Esto dará a LEFI la habilidad de escalar su experiencia de usuario (más áreas de la vida cubiertas) sin tener que rediseñar desde cero la motivación y engagement – el sistema de XP y logros crecerá orgánicamente con la plataforma.

Referencias Utilizadas: Duolingo, Me+ y Habitify (mecánicas de gamificación, rachas, XP); Documentos internos de LEFI (*Propuesta de Gamificación Ética*, *Diseño de Monetización Ética*, *LEFI – Tu Asistente Integral*) para alineamiento con la visión ética y estructura modular [46] [47]. Cada detalle del diseño busca equilibrar la **diversión y motivación** del usuario con **beneficios reales** y respeto por su bienestar, convirtiendo la gamificación en una ventaja competitiva ética para LEFI [48] [49].

---

1 4 5 31 32 45 48 49 LEFI – Tu asistente integral de productividad y bienestar.pdf
file://file-9GK4Y9Rabxan5dUDS3Zhbh

2 3 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 33 34 35 36 37 38 39 40 41 42 43 44 46 47 Propuesta de Gamificación Ética y Sistema de Recompensas para LEFI.pdf
file://file-5VP1ToHyfpNXdP51EBkV6g