Hábitos y su impacto en la calidad del sueño

Documentación Técnica del Sistema

Ángel Ortiz - 67001080

Desarrollo de Software

Contenido

1. Resumen ejecutivo

1.1 Descripción general del sistema

SleepHabits es una aplicación web de registro y análisis de hábitos de sueño, diseñada para capturar información básica de personas (sujetos), sus sesiones de sueño y los factores de estilo de vida asociados. El sistema permite:

- Registrar y gestionar sujetos con datos básicos (nombre, edad, género y etiquetas/hábitos).
- Registrar sesiones de sueño diarias con horario de acostarse, despertar, número de despertares y una eficiencia de sueño calculada.
- Asociar cada sujeto con tags o hábitos (por ejemplo, *café tarde*, *pantallas noche*, *ejercicio*, *alcohol*), que posteriormente se utilizan para generar indicadores agregados.
- Visualizar en un dashboard sencillo:
    - La duración promedio del sueño (horas) a lo largo de los días.
    - La relación entre hábitos y calidad del sueño, medida a través del promedio de eficiencia por tag.

La API se apoya en una base de datos en Supabase (PostgreSQL) y en Supabase Storage para almacenar archivos adjuntos opcionales asociados a los registros de sueño (por ejemplo, reportes de dispositivos de monitoreo).

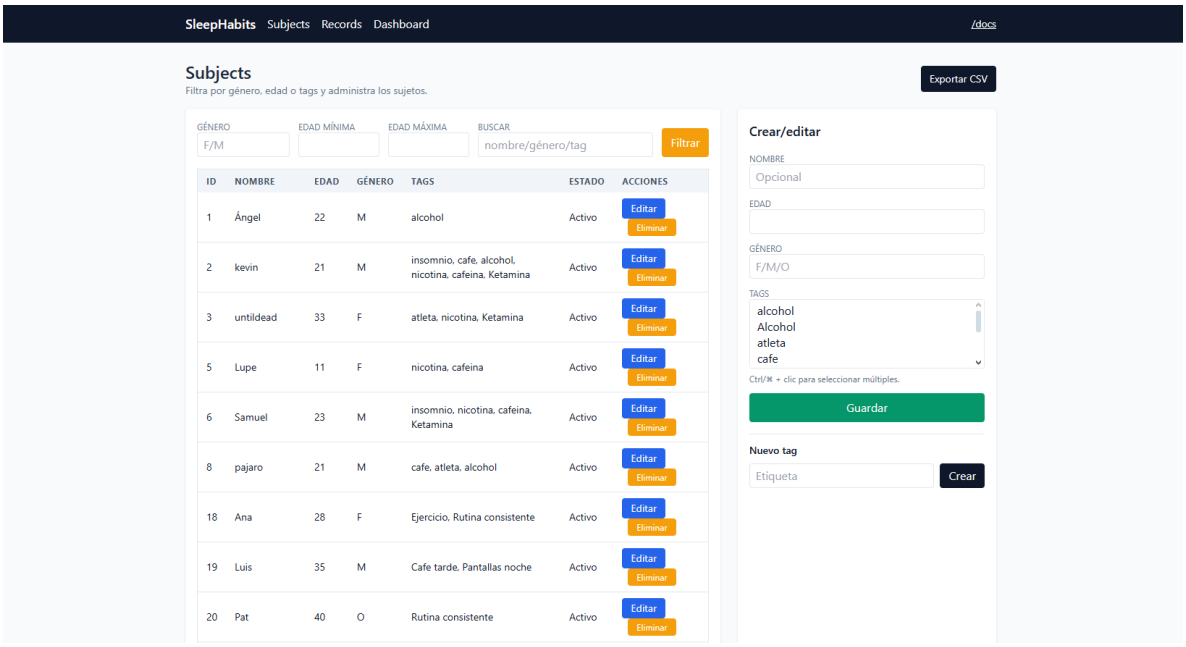

Figura 1. Vista general de la aplicación con las secciones *Subjects*, *Records* y *Dashboard*.

1.2 Principales funcionalidades

Las principales funcionalidades del sistema se pueden agrupar en tres módulos:

1. Gestión de sujetos (Subjects)
    - Crear, editar y eliminar lógicamente sujetos.
    - Registrar nombre, edad y género (restringido a F, M, O).
    - Asignar y gestionar tags/hábitos (por ejemplo, *alcohol*, *cafeína*, *ejercicio*, *rutina consistente*).
    - Filtros por género, rango de edad y búsqueda por nombre/género/tag.
    - Exportar la lista de sujetos en formato CSV.
2. Gestión de registros de sueño (Sleep Records)
    - Crear nuevos registros de sueño para un sujeto seleccionado.
    - Validar coherencia entre:
        - Fecha del registro.
        - Hora de acostarse (bedtime) y hora de despertar (wakeup).

- Cálculo automático de la duración del sueño (horas) y de la eficiencia de sueño (%) a partir de los horarios.
- Registro de número de despertares y notas opcionales.
- Carga opcional de archivos a Supabase Storage (por ejemplo, reportes en PDF) y almacenamiento de la URL asociada al registro.
- Filtros por rango de fechas, género y sujeto.
- Exportación de los registros en formato CSV.

3. Dashboard y reportes
   - Gráfico de duración promedio del sueño (horas) por día a partir de los registros almacenados.
   - Gráfico de hábitos y calidad del sueño, que muestra la eficiencia promedio (%), la duración promedio (horas) y el número de registros asociados a cada tag.

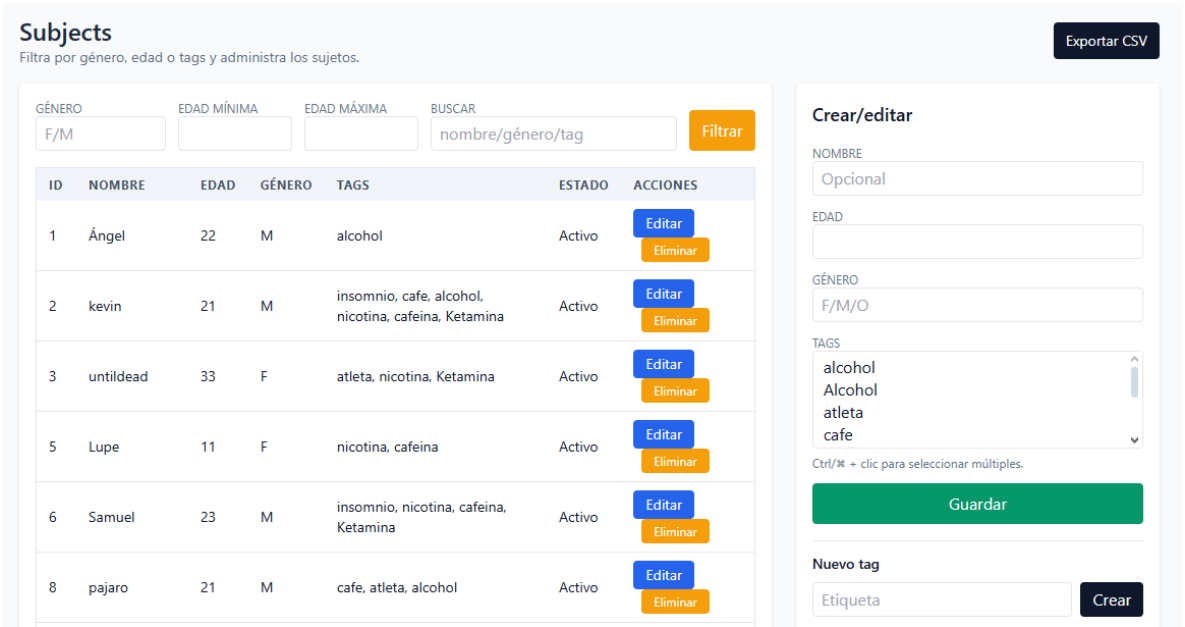

Figura 2. Vista de Subjects con filtros, listado, formulario de creación/edición y manejo de tags.

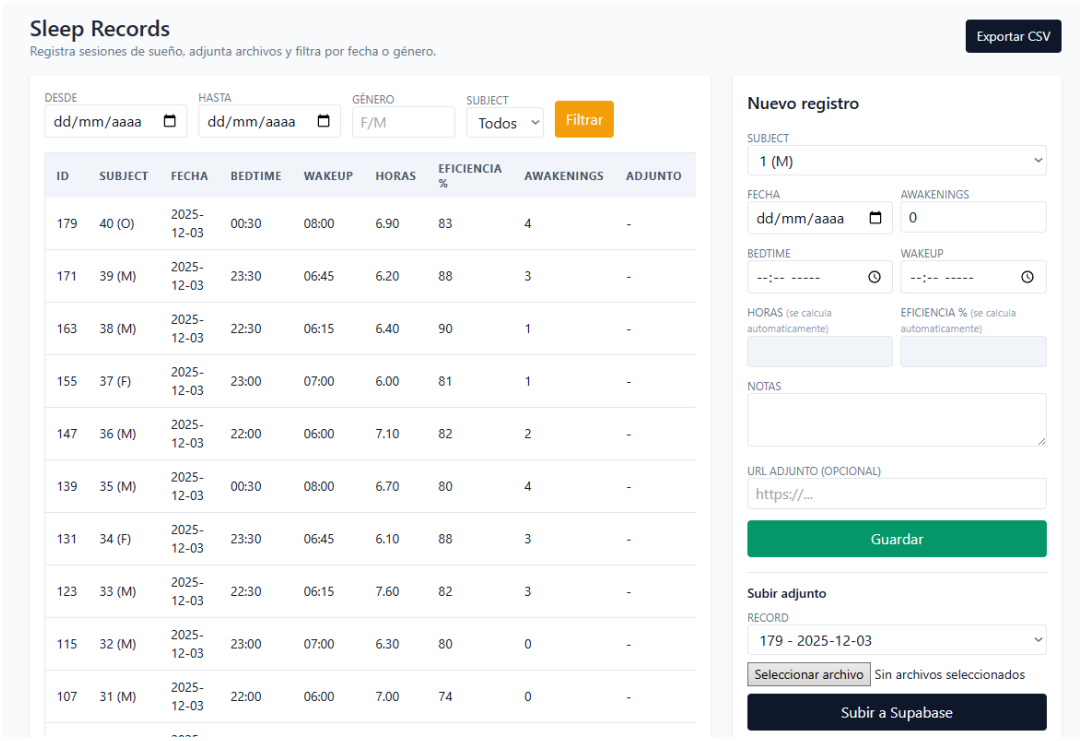

Figura 3. Vista de Sleep Records con registros, filtros, formulario y módulo de subida a Supabase.

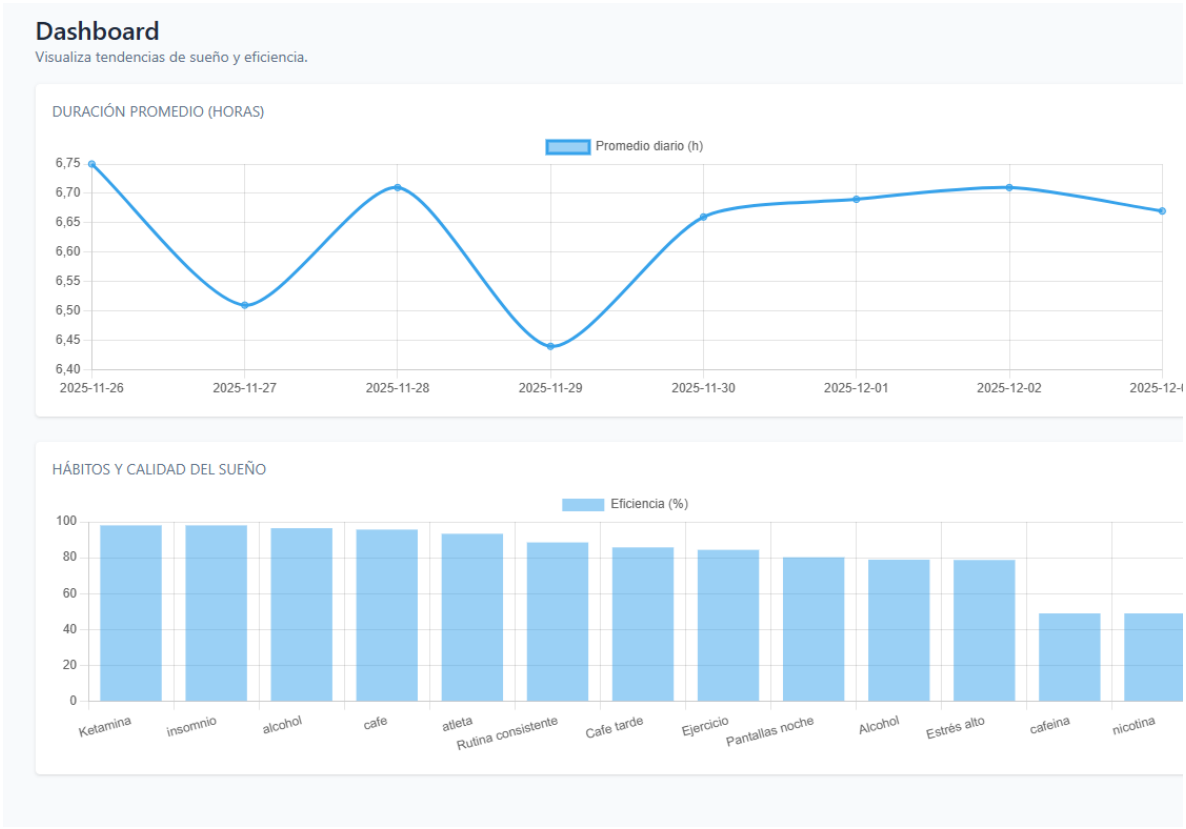

Figura 4. Dashboard con gráficos de duración promedio y hábitos/calidad del sueño.

1.3 Tecnologías utilizadas

El proyecto está construido con un stack sencillo y coherente con un entorno académico y de prototipo:

- Backend / API
    - Python 3.11
    - FastAPI para la definición de rutas, validación de datos y generación automática de documentación (/docs).
    - SQLAlchemy (ORM) para el mapeo objeto–relacional entre las entidades Python y las tablas de la base de datos.
    - asyncpg como driver asíncrono para PostgreSQL.
    - Manejo de configuración mediante dotenv y variables de entorno.
- Base de datos y almacenamiento
    - Supabase (PostgreSQL) como base de datos principal.
    - Supabase Storage para almacenamiento de archivos adjuntos.
    - Conexión segura mediante SSL, configurada en db/db.py.
- Frontend
    - Plantillas Jinja2 integradas en FastAPI (templates/base.html, subjects.html, records.html, dashboard.html).
    - HTML + CSS ligero con clases utilitarias (Tailwind-like / estilos simples).
    - JavaScript clásico para manejar formularios, validaciones en cliente y llamadas fetch() a la API.
    - Chart.js para la generación de gráficos del dashboard.
- Otros
    - Gestión de dependencias mediante requirements.txt.
    - Servidor de desarrollo con Uvicorn y recarga automática (--reload).

2. Introducción

2.1 Contexto y motivación

La calidad del sueño es un factor determinante en la salud física, mental y el desempeño diario. Sin embargo, en la práctica muchas personas no registran de forma sistemática sus hábitos de sueño, ni cuentan con herramientas sencillas para relacionar esos hábitos con la calidad percibida o con datos objetivos como la duración y la eficiencia del descanso.

En contextos académicos y de investigación aplicada, suele necesitarse un sistema ligero que permita:

- Registrar datos de sueño de manera estructurada.
- Asociar estos datos con hábitos o factores de estilo de vida.
- Visualizar tendencias y relaciones básicas sin requerir una plataforma compleja o comercial.

SleepHabits nace como un proyecto didáctico/técnico que cumple esa función: demostrar cómo construir una API y una interfaz web sencilla que registren, persistan y analicen información sobre hábitos de sueño, usando tecnologías modernas pero accesibles (FastAPI, Supabase, Chart.js).

2.2 Problema que aborda el sistema

El sistema aborda principalmente los siguientes problemas:

1. Falta de registro estructurado de datos de sueño
   Muchos registros personales se hacen de manera informal (notas sueltas, aplicaciones no estructuradas), lo que dificulta luego el análisis.
2. Dificultad para relacionar hábitos con indicadores de calidad del sueño
   Las personas pueden intuir que el café tarde, las pantallas de noche o el alcohol afectan su descanso, pero sin datos es difícil visualizar patrones.
3. Ausencia de una herramienta sencilla para pruebas o pilotos
   En un entorno de laboratorio, curso o proyecto piloto se requiere una herramienta que:
   - Sea fácil de desplegar en local.
   - Tenga un modelo de datos claro.

- Permita exportar la información para análisis posteriores en otras herramientas (por ejemplo, Python, R, Excel).

SleepHabits responde a estos problemas ofreciendo una API REST y una interfaz web ligera, con validaciones básicas y capacidad de generar reportes, que puede utilizarse como prototipo, herramienta docente o punto de partida para sistemas más complejos.

2.3 Objetivos del proyecto

Objetivo general

Desarrollar una aplicación web sencilla para el registro, gestión y visualización de hábitos y registros de sueño, que permita vincular factores de estilo de vida con indicadores de calidad de sueño (duración y eficiencia), utilizando una arquitectura moderna basada en FastAPI y Supabase.

Objetivos específicos

1. Implementar un modelo de datos que represente:
    - Sujetos con características demográficas básicas (nombre, edad, género).
    - Registros de sueño diarios con horarios, duración, eficiencia y despertares.
    - Tags o hábitos asociados a cada sujeto.
2. Desarrollar una API REST que permita:
    - Crear, consultar, actualizar y eliminar lógicamente sujetos, registros y tags.
    - Generar reportes en CSV y JSON para análisis posteriores.
3. Construir una interfaz web que:
    - Facilite el registro de datos mediante formularios con validaciones de entrada.
    - Permita filtrar y explorar la información.
    - Muestre gráficas de tendencias y agregados de sueño.
4. Integrar Supabase Storage para el manejo de archivos adjuntos asociados a los registros de sueño.

5. Proveer scripts de carga de datos de ejemplo (seed_data.py) para alimentar el sistema y demostrar el funcionamiento del dashboard con datos realistas.

2.4 Alcance y limitaciones

Alcance

- El sistema cubre la gestión básica de sujetos y registros de sueño, con validaciones lógicas sencillas (por ejemplo, género limitado a F/M/O, coherencia de fechas y horas).
- Permite asignar múltiples tags/hábitos a cada sujeto y utilizar esos tags como base para reportes agregados.
- Incluye un dashboard con:
    - Duración promedio del sueño por día.
    - Eficiencia promedio por hábito, con información adicional de horas promedio y número de registros.
- Soporta la carga y almacenamiento de adjuntos en Supabase Storage, referenciados desde cada registro de sueño.
- Ofrece endpoints de exportación en formato CSV para:
    - Sujetos (/api/reports/subjects.csv)
    - Registros de sueño (/api/reports/records.csv)

Limitaciones

- No es un producto listo para uso clínico ni una herramienta médica certificada; se trata de un prototipo académico/técnico.
- Las métricas de eficiencia se calculan a partir de reglas simples definidas en el backend, sin integración con dispositivos médicos ni algoritmos avanzados.
- No hay autenticación de usuarios ni gestión de roles; se asume un entorno controlado (uso local o en un servidor de pruebas).
- La interfaz es deliberadamente sencilla y no incluye optimizaciones avanzadas de experiencia de usuario (por ejemplo, no hay paginación en el dashboard, ni gráficos muy complejos).

- La política de seguridad de Supabase (RLS) puede requerir ajustes adicionales para un entorno productivo; el proyecto está pensado para un entorno de demostración.

3. Arquitectura general del sistema

3.1 Visión global

SleepHabits está construido como una aplicación web clásica de tres capas: interfaz, lógica de negocio y datos. El usuario solo ve un sitio sencillo con tres secciones —Subjects, Sleep Records y Dashboard—, pero detrás de eso hay una API en FastAPI que habla con una base de datos PostgreSQL alojada en Supabase y, cuando hace falta, con un bucket de almacenamiento para los archivos adjuntos.

Cuando el navegador pide una página, FastAPI renderiza una plantilla HTML con Jinja2, le inyecta los datos necesarios (por ejemplo, la lista de sujetos o de registros) y devuelve el resultado. Para las partes más dinámicas, como el dashboard, el frontend no vuelve a cargar la página completa, sino que hace peticiones fetch() a endpoints JSON (los /api/reports/...) y allí recibe los datos que luego se dibujan con Chart.js.

En la capa de datos todo se organiza alrededor de un modelo relacional sencillo: sujetos, registros de sueño, etiquetas/hábitos y las relaciones entre ellos. SQLAlchemy sirve como puente: desde el código Python se trabaja con objetos (Subject, SleepRecord, Tag, etc.) y la librería se encarga de traducir esas operaciones a SQL contra Supabase.

3.2 Componentes principales

Aunque el proyecto se ve grande en el explorador de VS Code, en realidad se puede entender como un conjunto de piezas muy claras:

- main.py es el punto de entrada. Allí se crea la aplicación FastAPI, se registran los routers, se configura el directorio de plantillas, los archivos estáticos y se llama a init_db() al arrancar para asegurarse de que la base de datos está lista.
- En la carpeta db/ vive toda la lógica de conexión a Supabase:
    - db.py se encarga de leer la DATABASE_URL del .env, construir el engine asíncrono de SQLAlchemy con SSL y exponer funciones como get_engine() y get_session_maker(). También tiene init_db(), que crea las tablas si aún no existen.
    - session.py define la dependencia get_db, que los routers usan para trabajar con una sesión de base de datos por petición.

- La carpeta models/ contiene la representación del modelo de datos:
    - En entities.py se definen las clases SQLAlchemy que corresponden a las tablas reales: Subject, Tag, SubjectTag, SleepRecord y, si se usan, SleepStage y LifestyleFactors.
      Subject representa a la persona; *SleepRecord* representa una noche concreta; *Tag* y *SubjectTag* permiten asociar hábitos al sujeto; y *SleepRecord* se enlaza con posibles detalles como etapas de sueño o factores de estilo de vida.
- La lógica de la aplicación está organizada en módulos dentro de routers/:
    - subjects.py maneja todo lo relacionado con los sujetos: listado, creación, edición, marcado como eliminado y asignación de tags. Aquí se valida, por ejemplo, que el género solo pueda ser F, M u O.
    - tags.py gestiona la creación y consulta de etiquetas/hábitos.
    - sleep_records.py es el corazón del registro de datos: recibe las fechas y horas, calcula la duración y la eficiencia del sueño, valida que las combinaciones de fecha y horas tengan sentido y permite filtrar por rangos y por sujeto.
    - uploads.py se ocupa de subir archivos a Supabase Storage y guardar la URL en el registro correspondiente.
    - reports.py concentra los reportes agregados: el promedio diario de horas (/timeseries), la relación entre hábitos (tags) y calidad de sueño (/habits_quality) y la exportación de CSV de sujetos y registros.
- Para completar el cuadro, templates/ contiene las vistas HTML:
    - base.html define la estructura general, la barra de navegación y carga los scripts (incluido Chart.js).
    - subjects.html, records.html y dashboard.html son las pantallas principales que ya se ven en las capturas: gestión de sujetos, gestión de registros y dashboard.
- Finalmente, el script seed_data.py sirve para poblar la base de datos con datos de ejemplo de forma controlada. Se apoya en el mismo sistema de sesiones que usa la aplicación y crea sujetos, hábitos y registros para varios días, justo lo necesario para que el dashboard cobre vida.

3.3 Flujo de información en una operación típica

Una forma fácil de entender la arquitectura es seguir un caso de uso de principio a fin. Por ejemplo: registrar una nueva noche de sueño y verla reflejada en el dashboard.

1. El usuario abre la página de Sleep Records, llena el formulario (sujeto, fecha, hora de ir a la cama, hora de despertar, número de despertares) y pulsa Guardar.
2. El navegador envía una petición HTTP al endpoint correspondiente de sleep_records.py. FastAPI recibe los datos, construye el modelo SleepRecord, calcula la duración (en horas) y la eficiencia, comprueba que la fecha no esté vacía y que las horas tengan sentido, y finalmente inserta el registro a través de SQLAlchemy en la tabla sleep_records.
3. Cuando el usuario va al Dashboard, la vista HTML se carga y, en cuanto el JavaScript se ejecuta, se hacen dos peticiones fetch():
    - /api/reports/timeseries devuelve la serie temporal diaria con el promedio de horas dormidas.
    - /api/reports/habits_quality devuelve, por cada tag/hábito, la eficiencia media, duración media y número de registros.
4. Con esas dos respuestas, dashboard.html usa Chart.js para dibujar:
    - Una gráfica de líneas con la duración promedio diaria.
    - Una gráfica de barras donde cada barra representa un hábito y su eficiencia media, con un tooltip que muestra también horas e n.

Todo esto ocurre sin que el usuario tenga que saber nada de la base de datos ni de Supabase: solo ve que, al registrar más noches, las curvas del dashboard cambian.

3.4 Integración con Supabase

La integración con Supabase aparece en dos frentes: la base de datos PostgreSQL y el almacenamiento de archivos.

En el lado de la base de datos, la URL de conexión se define en el archivo .env como DATABASE_URL. El módulo db/db.py se encarga de leerla, limpiar parámetros redundantes y construir un engine asíncrono con soporte SSL. Para evitar problemas de certificados, se utiliza el bundle de certifi, de manera que la conexión cifrada con Supabase pase la verificación. Al iniciar la aplicación (ya sea con uvicorn o al ejecutar

seed_data.py), init_db() asegura que las tablas definidas en models.entities estén creadas en la instancia de Supabase.

Para los adjuntos, Supabase Storage aporta un bucket —por ejemplo, sleep-uploads— donde se guardan los archivos relacionados con los registros de sueño (informes, capturas, etc.). El módulo supabase_client.py crea un cliente a partir de SUPABASE_URL y SUPABASE_ANON_KEY. El router uploads.py recibe el archivo desde el formulario de la UI, lo sube al bucket y guarda en la base de datos únicamente la URL de acceso. Desde la vista de registros, el usuario puede ver o recuperar ese enlace.

En este proyecto el enfoque es claramente didáctico: las claves están pensadas para uso en entorno controlado y la prioridad es mostrar el flujo completo de una aplicación real, más que endurecer al máximo la seguridad. Aun así, la infraestructura es perfectamente escalable: activando RLS en las tablas y moviendo las claves sensibles a un entorno seguro, el mismo diseño podría evolucionar hacia un servicio multiusuario más robusto.

4. Modelo de datos y diseño de la base de datos

4.1 Visión general del modelo

El modelo de datos de SleepHabits está pensado para ser sencillo, pero lo bastante expresivo como para soportar análisis básicos de sueño. Todo el sistema gira en torno a tres conceptos principales:

- El sujeto (la persona que duerme).
- El registro de sueño (una noche concreta).
- Los hábitos o factores asociados (tags que describen conductas o condiciones).

A partir de ahí se añaden tablas auxiliares para detalles más finos, como etapas de sueño o factores de estilo de vida. La base de datos se implementa en PostgreSQL (vía Supabase) y se accede a ella desde SQLAlchemy, de modo que la definición de las tablas está sincronizada con las clases del módulo models.entities.

En el *Schema Visualizer* de Supabase el modelo se ve como un conjunto compacto de seis tablas: subjects, sleep_records, sleep_stages, lifestyle_factors, tags y subject_tags.

4.2 Tablas principales

4.2.1 subjects

La tabla subjects representa a las personas cuyos hábitos de sueño se están registrando. Es el punto de partida de casi todas las relaciones.

Campos más relevantes:

- id (PK entero, autoincremental): identificador interno.
- name (varchar): nombre opcional del sujeto. Se permite dejarlo vacío para escenarios de anonimato.
- age (entero): edad en años.
- gender (varchar): género codificado como F, M u O (otro). En la lógica de la aplicación se valida que solo se usen esos tres valores.
- is_deleted (booleano, no nulo): implementa el borrado lógico. Cuando es true, el sujeto deja de aparecer en la interfaz, pero su historial queda preservado.
- created_at (timestamp): marca la fecha de creación del registro. Se usa para ordenamientos y, en el futuro, para auditoría básica.

La tabla subjects no elimina registros físicamente; siempre se marca el borrado con is_deleted. Esto permite mantener la integridad referencial con los registros de sueño existentes.

4.2.2 sleep_records

sleep_records es la tabla que almacena las noches de sueño. Cada fila representa una sesión de sueño asociada a un sujeto y a una fecha concretas.

Campos más relevantes:

- id (PK entero).
- subject_id (FK a subjects.id): indica a quién pertenece ese registro.
- record_date (date): fecha de referencia de la noche. Se asume que es la fecha de inicio del descanso.
- bedtime y wakeup_time (timestamp): marcas de tiempo completas de inicio y fin del sueño. Se almacenan como timestamp (sin zona horaria) para simplificar los cálculos.
- sleep_duration (float): cantidad de horas dormidas, calculada por la API a partir de las horas de inicio y fin.

- sleep_efficiency (float): indicador porcentual (0–100) calculado en el backend. Representa qué tan “aprovechado” fue el tiempo en cama.
- awakenings (int): número de despertares durante la noche.
- attachment_url (varchar, opcional): URL del archivo almacenado en Supabase Storage (por ejemplo, un informe o una captura).
- notes (varchar): campo libre para observaciones.
- is_deleted (booleano) y created_at (timestamp): mismos criterios que en subjects.

En la lógica de negocio las duraciones y eficiencias **no se piden al usuario**; se derivan de los datos de entrada. De este modo se evita que haya inconsistencias entre la hora de ir a la cama, la hora de despertarse y las horas declaradas.

4.2.3 tags y subject_tags

Para representar hábitos o condiciones (por ejemplo, “Cafe tarde”, “Ejercicio”, “Pantallas noche”) se utiliza la tabla tags. Cada fila es una etiqueta textual:

- id (PK entero).
- name (varchar, único): nombre del hábito/condición.

La asociación entre sujetos y tags se modela con una tabla intermedia subject_tags, que implementa una relación muchos-a-muchos:

- subject_id (FK a subjects.id).
- tag_id (FK a tags.id).

En la práctica, subject_tags significa: “el sujeto X tiene el hábito Y como característica general”. A diferencia de otros modelos donde la etiqueta se asocia a un registro de sueño concreto, aquí el hábito se liga a la persona, y luego los reportes (/habits_quality) calculan las métricas usando todos los registros de ese sujeto.

4.3 Tablas de detalle: etapas de sueño y factores de estilo de vida

El modelo incluye dos tablas que enriquecen el registro de sueño con información adicional, pensadas para escenarios más avanzados:

4.3.1 sleep_stages

sleep_stages almacena un resumen de cómo se distribuyó el tiempo de sueño entre las distintas fases:

- id (PK).
- sleep_record_id (FK a sleep_records.id).
- rem_percentage, deep_percentage, light_percentage (floats): porcentajes de tiempo en sueño REM, profundo y ligero.

Aunque en la versión actual del dashboard se terminó usando el gráfico de hábitos en lugar de las etapas, la tabla está diseñada para que, si más adelante se dispone de datos de algún dispositivo o aplicación, se pueda representar la distribución de etapas sin modificar el resto del modelo.

4.3.2 lifestyle_factors

lifestyle_factors se centra en información de estilo de vida vinculada a una noche concreta:

- id (PK).
- sleep_record_id (FK a sleep_records.id).
- caffeine_consumption, alcohol_consumption, smoking_status, exercise_frequency (varchar): campos cualitativos que podrían codificarse (por ejemplo, “ninguno / moderado / alto”).

En este proyecto los factores de estilo de vida funcionan como un complemento a los tags: permiten representar aspectos más específicos por noche, mientras que los tags son rasgos generales del sujeto. De nuevo, esto se deja preparado para futuras evoluciones sin cargar de complejidad el MVP que se entrega.

4.4 Relaciones y cardinalidades

El diagrama de la base de datos se resume en las siguientes relaciones:

- **Subjects ↔ SleepRecords**:
  Un sujeto puede tener muchos registros de sueño (1:N).
  Un registro de sueño pertenece exactamente a un sujeto.
- **Subjects ↔ Tags (vía SubjectTags)**:
  Un sujeto puede tener muchos tags, y un tag puede aplicarse a muchos sujetos (N:M).

Esta relación se materializa con la tabla subject_tags, donde cada fila conecta un subject_id con un tag_id.

- **SleepRecords ↔ SleepStages**:
  Relación 1:1 o 1:0..1: un registro de sueño puede tener, como mucho, un resumen de etapas. En la práctica, es 0..1 porque no todos los registros necesitarán ese nivel de detalle.
- **SleepRecords ↔ LifestyleFactors**:
  Mismo patrón que con sleep_stages: una fila opcional que describe factores de estilo de vida específicos de esa noche.

Estas relaciones permiten, por ejemplo, responder a preguntas como:

- "¿Cómo evoluciona el promedio de horas de sueño por fecha?" (reporte timeseries, basado en sleep_records).
- "¿Qué hábitos se asocian con mayor eficiencia de sueño?" (reporte habits_quality, que combina tags, subject_tags, subjects y sleep_records).

4.5 Reglas de integridad y validación de negocio

Aunque muchas reglas se implementan directamente en la lógica Python, el diseño del modelo las facilita:

- Los campos is_deleted evitan problemas de claves foráneas al "eliminar" datos. Los registros de sueño de un sujeto no se borran, pero dejan de mostrarse cuando el sujeto queda marcado como inactivo.
- gender se trata como un conjunto cerrado de valores. La API valida que solo se admitan F, M u O, de modo que los reportes por género sean consistentes.
- En sleep_records, la combinación (subject_id, record_date) se trata como única en la lógica de negocio: el código no crea dos registros para la misma persona y la misma fecha, sino que evita duplicados (o, si se quisiera, podría actualizar el existente).
- Tiempos y fechas se validan antes de guardar: los formularios exigen que la fecha esté presente, que las horas estén bien formadas y que el cálculo de duración no genere valores negativos.

Además, los reportes (/reports/timeseries y /reports/habits_quality) siempre filtran por is_deleted = false tanto en sujetos como en registros, de forma que las estadísticas nunca se "contaminen" con datos ya descartados en la interfaz.

4.6 Extensibilidad del modelo

Un objetivo importante del diseño fue que la base de datos pudiera crecer sin romper lo existente. Algunas decisiones van en esa línea:

- Dejar SleepStages y LifestyleFactors como tablas opcionales pero ya enlazadas a sleep_records abre la puerta a integrar datos de dispositivos, cuestionarios u otros instrumentos.
- Usar tags como catálogo abierto de hábitos permite añadir nuevos comportamientos (p. ej. “Siesta larga”, “Trabajo nocturno”, “Viaje”) sin cambios de esquema: solo hay que crear el tag y asignarlo.
- El uso de created_at en subjects y sleep_records hace viable, en el futuro, implementar filtros temporales más complejos o auditorías de cambios.

En definitiva, el modelo actual es pequeño, pero no es un callejón sin salida: está preparado para crecer hacia un sistema de seguimiento de sueño mucho más completo sin perder la simplicidad que el proyecto necesita para su sustentación.

5. Lógica de negocio y validaciones

La aplicación no se limita a guardar lo que el usuario escribe: aplica una capa de reglas de negocio para que los datos tengan sentido clínico y estadístico antes de llegar a la base de datos.

5.1 Creación y edición de sujetos

Cuando se crean o editan sujetos:

- El género se restringe a los valores F, M u O. Cualquier otro valor se rechaza con un mensaje de error claro.
- La edad debe ser numérica y mayor a cero; edades negativas o vacías no se aceptan.
- El campo nombre se valida para evitar cadenas vacías o puramente espacios, aunque sigue siendo un identificador “humano”, no una clave técnica.
- El borrado es siempre lógico: al eliminar un sujeto desde la interfaz se marca is_deleted = true, pero se conservan sus registros de sueño para no romper reportes ni relaciones.

Estas validaciones se hacen primero en el formulario (front) y se refuerzan en el backend mediante los esquemas de entrada (Pydantic) y las restricciones del modelo.

5.2 Registro de noches de sueño

Para cada SleepRecord la lógica de negocio hace varias comprobaciones:

- La fecha del registro es obligatoria; sin fecha no se permite guardar.
- Las horas de inicio (bedtime) y fin (wakeup_time) se verifican como horas válidas y coherentes.
  Si el fin es anterior al inicio, se asume el cruce de medianoche (por ejemplo, dormir a las 23:00 y despertar a las 06:30 del día siguiente).
- La duración del sueño (sleep_duration) no la ingresa el usuario: se calcula automáticamente en horas a partir de las marcas de tiempo.
- La eficiencia del sueño (sleep_efficiency) se deriva según una fórmula simple (p. ej. porcentaje del tiempo en cama realmente dormido, acotado entre 0 y 100). Se valida que el resultado final no salga de ese rango.
- El número de despertares debe ser un entero mayor o igual a cero.

Además, la combinación (subject_id, record_date) se trata como única a nivel de lógica: el código evita crear registros duplicados para el mismo día y sujeto.

5.3 Gestión de hábitos (tags) y calidad de sueño

Los tags describen hábitos o condiciones generales del sujeto (ejercicio, café tarde, pantallas, rutina consistente). La lógica asociada es:

- Solo se crean tags nuevos cuando no existe ya uno con el mismo nombre, evitando duplicados en tags.
- La tabla de unión subject_tags se llena cuidando de no repetir la misma pareja (subject_id, tag_id).
- El reporte /api/reports/habits_quality calcula, por cada tag:
  - Promedio de eficiencia de sueño.
  - Promedio de horas dormidas.
  - Número de noches consideradas (n).

Ese cálculo se hace siempre filtrando sujetos y registros no eliminados (is_deleted = false), de modo que estadísticas y dashboard están alineados con lo que se ve en la interfaz.

5.4 Validaciones en la interfaz

Antes de enviar cualquier dato al backend, el formulario aplica validaciones mínimas:

- Revisa que campos obligatorios (fecha, horas, edad, género) estén diligenciados.
- Limita la selección de género a tres opciones.
- Muestra mensajes de error sencillos cuando algo está mal (por ejemplo: fechas vacías, formato de hora incorrecto o valores fuera de rango).

De esta forma, el backend valida “en serio” y la interfaz filtra errores frecuentes, haciendo que el flujo de registro sea más robusto y agradable.

6. Endpoints de la API

Aunque el usuario interactúa principalmente mediante formularios y vistas HTML, debajo hay un conjunto de endpoints estructurados que exponen la funcionalidad principal.

6.1 Endpoints de vistas (HTML)

Estos endpoints devuelven plantillas Jinja:

- GET /
  Redirige al dashboard o a la vista principal de registros.
- GET /records
  Lista de registros de sueño, con filtros básicos y acceso a edición/eliminación.
- GET /subjects
  Vista de sujetos donde se crean, editan y marcan como eliminados.
- GET /dashboard
  Renderiza el dashboard con el gráfico de duración promedio por día y el gráfico de hábitos vs. calidad de sueño.

Detrás de estas vistas se usan servicios de la capa de datos para obtener los registros válidos (no eliminados) y construir el contexto que se pasa a las plantillas.

6.3 Endpoints utilitarios (exportación)

Para facilitar análisis externos:

- GET /api/reports/records.csv
  Exporta todos los SleepRecord (no eliminados) en formato CSV, incluyendo género del sujeto y URL del adjunto, si existe.
- GET /api/reports/subjects.csv
  Exporta sujetos con sus tags asociados en una sola fila, tags separados por comas.

Estos endpoints permiten que un docente o analista descargue los datos y los lleve a Excel, R, Python, etc., sin tener que entrar a la base de datos.

7. Seguridad y manejo de secretos

Aunque es un proyecto académico, se aplican prácticas básicas de seguridad para que la app sea desplegable en entornos reales.

7.1 Manejo de credenciales y variables de entorno

- La cadena de conexión a la base de datos (DATABASE_URL) no está hardcodeada en el código.
- Se usa python-dotenv para leer variables desde .env en desarrollo y variables de entorno reales en Render.
- Claves sensibles de Supabase (SUPABASE_URL, SUPABASE_ANON_KEY, SUPABASE_SERVICE_ROLE_KEY) también se cargan desde entorno y no se exponen en el repositorio público.

Esto permite cambiar de base de datos o de proveedor sin tocar el código fuente, solo actualizando el entorno.

7.2 Conexión segura a la base de datos (SSL)

En db/db.py se construye un contexto SSL:

- Por defecto, se usa el bundle de certificados de certifi, con verificación estricta del certificado del servidor.
- Para entornos restringidos (como algunos equipos locales con antivirus que interceptan TLS), se habilita una opción de desarrollo DB_SSL_INSECURE=1 que desactiva la verificación.

Este modo está pensado solo para pruebas; en producción se mantiene la verificación activa.

De esta forma, la app puede conectarse a Supabase usando TLS sin recurrir a configuraciones inseguras permanentes.

7.3 Otros aspectos de seguridad

- No se exponen endpoints de administración sin plantillas; las operaciones se hacen desde la UI de la app.
- No se implementa autenticación de usuarios finales (no era requisito del proyecto), pero la estructura de rutas y el patrón FastAPI permitirían añadir JWT o sesiones más adelante.
- La lógica de negocio valida y normaliza datos de entrada, reduciendo el riesgo de inconsistencias o valores malformados.

8. Despliegue y operación del sistema

La aplicación está pensada para ejecutarse tanto en local como en la nube.

En entorno local, se utiliza:

- Python con entorno virtual (.venv).
- Servidor de desarrollo con uvicorn main:app --reload.
- Variables de entorno cargadas desde .env (cadena de conexión a Supabase, claves de Storage, flags de SSL).

En producción, el despliegue se realiza en Render como un servicio web:

- El código se toma directamente del repositorio público de GitHub.
- El comando de inicio es:
- gunicorn -k uvicorn.workers.UvicornWorker -w 2 -b 0.0.0.0:$PORT main:app
- Render inyecta las variables de entorno necesarias (incluida DATABASE_URL) y asigna un puerto dinámico ($PORT) al que se liga Gunicorn.

La base de datos sigue residiendo en Supabase (PostgreSQL gestionado), lo que permite que el mismo backend funcione con el mismo esquema tanto en desarrollo como en producción. El servicio se expone a través de una URL pública del tipo:

https://sleephabits.onrender.com

desde donde se accede a las vistas /records, /subjects y /dashboard.

9. Pruebas y validación

Dado el carácter académico del proyecto, las pruebas se centraron en la funcionalidad crítica de captura y visualización:

1. **Pruebas de validación de formulario**
    - Se verificó que el backend rechaza edades negativas, géneros no válidos y fechas incoherentes (por ejemplo, hora de despertar anterior a la de dormir).
    - Se comprobó que, ante errores, la interfaz muestra mensajes claros y mantiene los campos ingresados cuando es posible.
2. **Pruebas de consistencia de datos**
    - Se revisó que cada SleepRecord tenga siempre un sujeto asociado y que las banderas is_deleted funcionen como “soft delete” (los datos no desaparecen de la base, pero dejan de aparecer en la UI y en los reportes).
3. **Pruebas de reportes**
    - Se alimentó la base con datos sintéticos (por ejemplo, 3 sujetos con varios días consecutivos) para verificar:
        - Cálculos de promedio diario de horas de sueño.
        - Agregación por hábitos en el endpoint /api/reports/habits_quality.
    - Se compararon resultados del frontend con consultas directas a la base (SQL) para confirmar que las cifras coinciden.
4. **Pruebas de despliegue**
    - Se realizaron pruebas de arranque en Render para asegurar:
        - Conectividad SSL correcta con Supabase.
        - Lectura de variables de entorno en producción.
        - Respuesta adecuada del dashboard y de las rutas principales.

No se implementó un conjunto formal de tests automatizados (por ejemplo, con pytest), pero la estructura actual del código permite añadirlos progresivamente.

10. Limitaciones y trabajo futuro

Aunque la API cumple los objetivos del proyecto, se identifican varias limitaciones y oportunidades de mejora:

- **Autenticación y multiusuario**
  Actualmente no hay manejo de usuarios finales. Todas las operaciones se asumen bajo un único contexto de uso. En un escenario real, sería necesario:
  - Añadir autenticación (por ejemplo, JWT o sesiones).
  - Aislar los datos de cada usuario o grupo.
- **Modelo simplificado de sueño**
  El modelo se basa en:
  - Duración total.
  - Eficiencia (porcentaje).
  - Número de despertares.
  - Hábitos asociados a tags.
    No se consideran todavía otros factores relevantes como cronotipo, calidad subjetiva, consumo de medicamentos, o datos de dispositivos wearables.
- **Escalabilidad**
  El sistema está dimensionado para pocos usuarios y un volumen moderado de registros. Para escalar:
  - Podrían ajustarse los parámetros de pool de conexiones.
  - Separar capa de lectura (reportes) de la capa transaccional.
  - Introducir caché para endpoints muy consultados (por ejemplo, reportes agregados).
- **Visualizaciones**
  Los gráficos actuales resuelven dos preguntas básicas:
  - ¿Cuántas horas se duerme, en promedio, cada día?

    - ¿Cómo se relacionan ciertos hábitos con la eficiencia y duración del sueño?
      A futuro se podrían incluir:
    - Comparaciones entre sujetos.
    - Indicadores de regularidad (desviación estándar en horarios).
    - Alertas visuales de “malas rachas” de sueño.

- **Pruebas automatizadas y documentación técnica detallada**
  El proyecto cuenta con descripción técnica (este documento) y validación manual, pero aún faltan:
    - Tests unitarios y de integración.
    - Documentación de la API en formato OpenAPI más comentada y guías de uso para terceros.

11. Conclusiones

El proyecto SleepHabits API demuestra que es posible construir, con tecnologías modernas pero accesibles, una pequeña plataforma de registro y análisis de hábitos de sueño:

- Combina FastAPI y SQLAlchemy asíncrono para ofrecer una API limpia, con separación clara entre modelos, rutas y capa de acceso a datos.
- Se apoya en Supabase PostgreSQL como base de datos gestionada, aprovechando tablas relacionales para sujetos, hábitos y registros de sueño.
- Implementa un dashboard web ligero, usando plantillas Jinja y Chart.js, que permite:
    - Visualizar la evolución de la duración del sueño en el tiempo.
    - Relacionar hábitos declarados (ejercicio, consumo de café, pantallas nocturnas, rutinas consistentes, etc.) con la calidad del sueño (eficiencia) y la duración promedio.
- Introduce validaciones de negocio razonables (género restringido, fechas coherentes, eliminación lógica), que acercan la aplicación a un escenario real, más allá de un simple CRUD sin restricciones.

- Está preparado para ser desplegado en la nube (Render), con configuración externa mediante variables de entorno y conexión segura a la base de datos mediante TLS.

En términos académicos, esta API sirve como:

- Un ejemplo completo de aplicación full stack ligera (backend + dashboard) en el mundo Python.
- Un laboratorio donde se pueden extender modelos, reglas y visualizaciones para explorar relaciones entre hábitos y salud del sueño.
- Una base sobre la cual se podría construir, en el futuro, un sistema más avanzado con autenticación, analítica más sofisticada y soporte para datos provenientes de dispositivos o diarios de sueño más detallados.