ChatGPT

Documentación Técnica del Sistema de Webinars

Resumen del Propósito del Sistema

El sistema es una plataforma backend para gestión de **webinars** educativos con contenido en video. Permite administrar usuarios (registrarse, autenticarse), gestionar instructores o **tutores**, crear y publicar **webinars** (que agrupan varios videos educativos), adjuntar **videos** con sus **materiales de apoyo** (archivos descargables complementarios como PDFs, enlaces, etc.), y hacer seguimiento a la **visualización** de los videos por parte de los usuarios (registro de progreso y finalización). El propósito principal es brindar una infraestructura para cursos en línea tipo webinar, donde un tutor imparte un webinar compuesto de videos y materiales, y los usuarios pueden consumir ese contenido de forma segura, llevando un registro de su avance. Adicionalmente, el sistema contempla métricas básicas de uso (analytics) para conocer qué webinars/videos son más vistos y el tiempo de visualización por usuario. En resumen, se ofrece un conjunto de APIs RESTful que cubren el ciclo completo: desde la autenticación de usuarios hasta la entrega de medios y recopilación de estadísticas, facilitando así la construcción de una plataforma de e-learning basada en webinars.

Arquitectura General del Backend

El backend está construido con **FastAPI** (framework web asíncrono en Python) junto con **SQLModel** (un ORM basado en Pydantic y SQLAlchemy) sobre una base de datos **MySQL**. La arquitectura sigue un estilo modular (incluso **microservicios** lógicos) separando las responsabilidades por contexto de negocio. Cada módulo o servicio maneja un agregado de datos específico (usuarios, tutores, webinars, etc.) y expone un conjunto de endpoints bajo un prefijo de ruta único (p. ej. `/auth/v1` para autenticación y usuarios) [1] [2]. Esto permite una escalabilidad y mantenibilidad superiores, ya que los módulos pueden desarrollarse, desplegarse y escalarse de forma independiente si es necesario.

A grandes rasgos, la arquitectura comprende:

- **API REST**: Implementada con FastAPI, define endpoints HTTP organizados por módulos. Las rutas siguen buenas prácticas REST (verbos HTTP para operaciones CRUD) y convenciones de versionado (`/v1`) y nombres en plural en español para los recursos (p. ej. `/usuarios`, `/tutores`).
- **Capa de Datos**: Los modelos de datos se definen con SQLModel, lo que combina la declaración de esquemas (como con Pydantic) con la capacidad de mapear a tablas SQL. Cada módulo tiene sus modelos de base de datos correspondientes. La comunicación con MySQL se realiza mediante sesiones de SQLModel/SQLAlchemy, aplicando ORM para consultas comunes y consultas SQL nativas cuando es necesario optimizar.
- **Base de Datos MySQL**: Se utiliza MySQL como sistema de gestión de base de datos relacional. La estructura de la base de datos incluye tablas para cada entidad principal (usuarios, tutores, webinars, videos, materiales, vistas de reproducción, etc.), con sus relaciones (por ejemplo, webinars relacionados a tutores, videos relacionados a webinars, etc.). Para algunas operaciones de lectura complejas (como métricas agregadas), se consideró el uso de **vistas SQL** o **consultas SQL crudas** dentro de la aplicación para simplificar la obtención de datos agregados.
- **Servicios/Módulos independientes**: El backend está dividido lógicamente en servicios como **ms-auth**, **ms-tutores**, **ms-webinars**, **ms-videos**, **ms-materiales**, **ms-playback**, etc., cada uno

correspondiente a un dominio. Internamente, en la implementación se organizan mediante *routers* de FastAPI registrados en la aplicación principal, cada uno con su prefix (`/auth/v1`, `/tutores/v1`, etc.), pero podrían desplegarse como microservicios separados en caso de requerirse. Esta separación por contexto garantiza un bajo acoplamiento.

- **Integración y Optionales**: Además de los módulos principales, el diseño prevé servicios opcionales como un módulo de **Analytics** (`/analytics/v1`) para cálculos de KPIs transversales (ej. top de videos, tiempos de reproducción) y un módulo **Media Gateway** (`/media/v1`) para la entrega segura de archivos multimedia (generación de URLs firmadas para videos y descargas de materiales) [3] [4]. Estos módulos complementarios están pensados para escalar funcionalidad especializada sin recargar los servicios principales.

En cuanto a despliegue, todos los servicios comparten la misma base de datos MySQL (en diferentes tablas) para simplificar la implementación académica, aunque en un entorno productivo podrían separarse las bases de datos por microservicio. La aplicación utiliza *dependency injection* de FastAPI para gestionar la sesión de base de datos en cada request y la verificación de seguridad (tokens JWT), asegurando transacciones atómicas y middleware de autenticación consistente. La arquitectura es altamente mantenible: cada módulo contiene su propia lógica de negocio, modelos y rutas, y se interactúa entre módulos vía referencias de IDs (evitando dependencias circulares). En caso de necesitar datos combinados de varios módulos (por ejemplo, para analytics), el enfoque es realizar consultas SQL con JOINs o vistas SQL definidas, más que romper la encapsulación de cada servicio.

Módulos del Sistema

A continuación, se detalla la organización modular del sistema, describiendo cada módulo con sus modelos de datos principales, procedimientos/vistas SQL relevantes y los endpoints expuestos por su API. Los módulos principales son: **Autenticación/Usuarios**, **Tutores**, **Webinars**, **Videos**, **Material de Apoyo**, y **Visualización (Vistas de Reproducción)**. Cada módulo corresponde a un microservicio lógico (prefijo `ms-` en el diseño) con su propio segmento de URL y lógica encapsulada.

Módulo de Autenticación y Usuarios (ms-auth)

Propósito: Gestiona la identidad de los usuarios y la autenticación. Incluye registro de nuevos usuarios, inicio/cierre de sesión (mediante JWT), y operaciones sobre el perfil del usuario. Este módulo mantiene el ciclo de vida del usuario (creación, activación/desactivación) y las reglas de seguridad básicas, como cambio de contraseña y roles de acceso [5] [6].

Modelo de Datos: El modelo principal es **Usuario**, definido como una tabla SQLModel. Sus campos incluyen: - `id` (UUID, clave primaria) - `username` (string único para login) - `email` (string único) - `hashed_password` (string con la contraseña hasheada) - `rol` (string o enum para el rol de usuario, e.g. "user", "admin", etc.) - `activo` (booleano, indica si el usuario está activo o ha sido desactivado)

Adicionalmente, para el proceso de autenticación, se manejan campos como `password` (solo para entrada del usuario, nunca almacenado en texto plano) y posiblemente un token de refresco asociado. Los datos sensibles (contraseña) se almacenan usando hashing seguro (p. ej. Bcrypt), y nunca se devuelven en respuestas. Se utilizan modelos Pydantic de entrada/salida para controlar qué campos se exponen: por ejemplo, un **UsuarioEntrada** (para registrar o actualizar, que incluye la contraseña en texto plano) y un **UsuarioSalida** (para respuestas, que omite la contraseña y otros campos sensibles).

Procedimientos Almacenados y Vistas SQL: No se implementó un procedimiento almacenado específico para usuarios, ya que las operaciones (registro, login, actualización) se manejan a nivel de

aplicación con la lógica de negocio apropiada. Tampoco se definen vistas SQL particulares para este módulo; las consultas (por ejemplo, listado de usuarios con filtros) se realizan mediante el ORM de SQLModel, aprovechando índices en las columnas buscables (username, email) para eficiencia.

Rutas de la API (Usuarios/Autenticación): Las rutas expuestas por este módulo llevan el prefijo `/auth/v1`. A continuación se presenta la lista de endpoints y sus descripciones:

Método	Endpoint	Descripción
POST	`/auth/v1/login`	Autentica a un usuario con `username`/`password` y devuelve tokens JWT (de acceso y refresco).
POST	`/auth/v1/register`	Registra un nuevo usuario con datos como nombre de usuario, email y contraseña (aplicando validaciones y hash de contraseña).
POST	`/auth/v1/refresh`	Renueva el token JWT de acceso usando un **refresh token** válido (permite extender la sesión del usuario).
GET	`/auth/v1/usuarios`	Lista los usuarios registrados (con paginación y filtros); **requiere rol admin** para su uso. [7]
GET	`/auth/v1/usuarios/{id}`	Obtiene el detalle de un usuario por su ID; normalmente restringido al propio usuario o admin.
PATCH	`/auth/v1/usuarios/{id}`	Actualiza datos de perfil de un usuario (ej. email, estado activo); requiere autenticación (el propio usuario o admin).
PATCH	`/auth/v1/usuarios/{id}/password`	Cambia la contraseña del usuario (verificando la contraseña previa); el propio usuario autenticado puede hacerlo. [8]
PATCH	`/auth/v1/usuarios/{id}/deactivate`	Desactiva la cuenta de un usuario (cambia su estado `activo` a false); solo ejecutable por un **admin**. [9]

Notas: Los endpoints de login, register y refresh no requieren un token JWT previo (son públicos). Los demás endpoints de usuario sí requieren autenticación por JWT; además, ciertas rutas están limitadas por rol (por ejemplo, solo un administrador puede listar usuarios o desactivar a alguien que no sea él mismo). Al autenticarse correctamente, el sistema emite un JWT que incluye la identificación del usuario y su rol para autorizar subsiguientes peticiones.

Módulo de Tutores (ms-tutores)

Propósito: Administra la información de los **tutores** o instructores que imparten webinars. Este módulo funciona como un catálogo de tutores, permitiendo crear nuevos tutores, actualizarlos, consultarlos, y obtener la lista de webinars asociados a cada tutor [10] [11]. Un tutor tiene relación con el módulo de webinars dado que un webinar es impartido por un tutor.

Modelo de Datos: El modelo **Tutor** incluye campos como: - `id` (UUID, clave primaria) - `nombre` (string, nombre completo del tutor) - `puesto` (string, por ejemplo el título o cargo del tutor, útil para mostrar su posición/profesión) - `bio` (texto opcional con biografía o descripción del tutor) - Posible `activo` (booleano, para marcar si sigue activo en la plataforma)

La relación principal es que un Tutor puede estar asociado a muchos webinars (relación 1 a N desde tutor hacia webinars). En la base de datos, típicamente el modelo Webinar incluye un campo `tutorId` como llave foránea a Tutor.

Procedimientos Almacenados y Vistas SQL: No se utilizaron procedimientos almacenados específicos para tutores. Las operaciones CRUD son sencillas y fueron implementadas con el ORM. Tampoco se definieron vistas SQL materializadas; para obtener, por ejemplo, los webinars de un tutor, se realiza una consulta filtrada (o un JOIN entre tutores y webinars según el caso) usando SQLModel. En un entorno modular, la ruta correspondiente hace una consulta al servicio de webinars filtrando por tutorId.

Rutas de la API (Tutores): Prefijo de rutas: `/tutores/v1`. Endpoints disponibles:

Método	Endpoint	Descripción
POST	`/tutores/v1/tutores`	Crea un nuevo tutor en el sistema (registro de un instructor con nombre, puesto, etc.).
GET	`/tutores/v1/tutores`	Lista todos los tutores registrados, con posibilidad de filtrar por nombre o puesto (por ejemplo, buscar tutores por nombre que contenga X o por puesto). [12]
GET	`/tutores/v1/tutores/{id}`	Obtiene el detalle de un tutor específico por su ID (incluye todos sus datos: nombre, bio, etc.).
PATCH	`/tutores/v1/tutores/{id}`	Actualiza los campos de un tutor existente (p. ej. actualizar su nombre, puesto o biografía).
GET	`/tutores/v1/tutores/{id}/webinars`	Lista todos los webinars que son impartidos por el tutor especificado (identificado por `{id}`). Esta ruta facilita obtener el catálogo de webinars de un tutor dado. [13] [14]

Notas: Todas las rutas de tutores requieren autenticación JWT (se asume que solo usuarios autenticados —por ejemplo administradores o quizás usuarios con ciertos permisos— pueden gestionar o ver tutores). La ruta de listar webinars de un tutor potencialmente podría ser pública si se quiere mostrar el catálogo a usuarios no autenticados, pero en esta implementación se maneja bajo autenticación estándar. Internamente, la implementación de `GET /tutores/{id}/webinars` puede realizar una consulta directa a la tabla de webinars filtrando por tutorId o delegar la solicitud al módulo de webinars para mayor aislamiento.

Módulo de Webinars (ms-webinars)

Propósito: Gestiona los **webinars** en sí mismos, que son los cursos o eventos compuestos por videos. Incluye la creación de nuevos webinars, su actualización, asignación de tutor, publicación o despublicación, y consulta de webinars existentes. Un webinar es la unidad central de contenido, tiene su propio ciclo de vida (borrador → publicado) y reglas de negocio, por ejemplo no se debería publicar un webinar si no tiene contenido, o permitir cambiar su tutor asignado [15] [16].

Modelo de Datos: El modelo **Webinar** contiene campos clave como: - `id` (UUID, clave primaria) - `titulo` (string, nombre del webinar o curso) - `descripcion` (texto, detalles del webinar) - `categoria` (string, categoría temática del webinar, ej. "Programación", "Marketing", etc.) - `dificultad` (string o enum, nivel de dificultad: básico, intermedio, avanzado) - `libre` (booleano, indica si el webinar es de acceso libre/gratuito para todos los usuarios autenticados o si es restringido) -

`publicado` (booleano, indica si el webinar está publicado y visible para usuarios; podría manejarse también con una fecha de publicación) - `tutorId` (UUID, referencia al Tutor que imparte el webinar) - Otros posibles: `fecha_creacion`, `fecha_publicacion` (timestamps para control interno)

El Webinar tiene relación con **Videos** (uno a muchos, un webinar contiene varios videos) y con **Tutores** (muchos a uno, cada webinar tiene un tutor).

Procedimientos Almacenados y Vistas SQL: No se definieron procedimientos almacenados específicos para la lógica de webinars. La publicación/despublicación de webinars se realiza mediante endpoints dedicados que actualizan el campo `publicado`. Sin embargo, para obtener métricas de webinars (como la tasa de finalización), se apoya en consultas sobre las **vistas de reproducción** relacionadas: por ejemplo, el cálculo de la *tasa de finalización* de un webinar puede lograrse mediante una consulta agregada que cuente cuántas sesiones de visualización (vistas) de ese webinar fueron marcadas como completadas frente al total de sesiones iniciadas. Esto se puede implementar dinámicamente con SQL (incluso podría crearse una **vista SQL** materializada o un procedimiento almacenado para dicho cálculo), pero en la implementación actual se realiza con una consulta ORM/SQL ad-hoc en el endpoint correspondiente [17]. No hay vistas SQL permanentes en la base de datos para webinars, más allá de posibles *views* temporales usadas para analytics.

Rutas de la API (Webinars): Prefijo de rutas: `/webinars/v1`. Endpoints expuestos:

Método	Endpoint	Descripción
POST	`/webinars/v1/webinars`	Crea un nuevo webinar (estado inicial típicamente no publicado, asociado a un tutor).
GET	`/webinars/v1/webinars`	Lista todos los webinars, con soporte de **filtros** por categoría, dificultad, flag libre, e identificador de tutor (`tutorId`). Esto permite paginar y buscar webinars según criterios. [18]
GET	`/webinars/v1/webinars/{id}`	Obtiene la información detallada de un webinar por ID (incluye título, descripción, tutor asignado, etc., y posiblemente la lista de videos si se decide incluir).
PATCH	`/webinars/v1/webinars/{id}`	Actualiza los datos de un webinar existente (por ejemplo, cambiar título, descripción, categoría, etc., si aún es un borrador o para corregir información).
POST	`/webinars/v1/webinars/{id}/publicacion`	Publica un webinar, cambiando su estado a `publicado=true`. Una vez publicado, se supone que está disponible para los usuarios en la plataforma. (Puede incluir validaciones, p. ej. verificar que tenga al menos un video antes de publicar).
DELETE	`/webinars/v1/webinars/{id}/publicacion`	Despublica un webinar, revirtiendo su estado a no publicado (borrador). Esto podría utilizarse para ocultar un webinar que ya no se ofrece o se está editando nuevamente.
PUT	`/webinars/v1/webinars/{id}/tutor`	Asigna o cambia el tutor de un webinar determinado. Se proporciona en la ruta el ID del webinar y en el cuerpo (o ruta) el nuevo `tutorId` para re-asociar. Esto actualiza la relación muchos-a-uno.

Método	Endpoint	Descripción
GET	`/webinars/v1/webinars/{id}/videos`	Lista todos los videos pertenecientes al webinar con ID dado. Es la forma de obtener el contenido (lista de lecciones) de un webinar. [19]
GET	`/webinars/v1/webinars/{id}/metricas/completion`	(Requiere módulo Analytics habilitado) Calcula la **tasa de finalización** del webinar especificado, es decir, qué porcentaje de sus sesiones de visualización fueron completadas exitosamente por los usuarios. Este endpoint cruza datos de webinars y vistas de reproducción.

Notas: Todas las rutas de webinars requieren autenticación; operaciones sensibles como crear/actualizar/publicar webinars probablemente requieren rol de **admin** o quizás un rol especial de **operador** de contenido. La asignación de tutor también sería restringida a admins. La tasa de finalización es parte de **Analytics**; si el módulo de analytics no está activo, esa ruta podría no estar disponible. En caso de estar disponible, internamente consulta la tabla de *vistas de reproducción* filtrando por webinarId y calculando la métrica (p. ej. mediante un `COUNT` condicional). Esto ejemplifica el uso de *consultas SQL agregadas* a través del ORM o potencialmente usando una vista SQL definida para resumir estos datos.

Módulo de Videos (ms-videos)

Propósito: Administra los **videos** individuales que conforman el contenido de los webinars. Permite crear videos, actualizarlos, asociarlos a webinars, y consultarlos. Cada video representa una lección o unidad dentro de un webinar y tiene metadatos importantes (ej. duración, URL del archivo, miniatura, si es gratuito o de acceso restringido) [20] [21] .

Modelo de Datos: El modelo **Video** incluye campos como: - `id` (UUID, clave primaria) - `titulo` (string, título o nombre del video) - `descripcion` (string, breve descripción del contenido del video) - `url` (string, URL o path de almacenamiento del archivo de video - puede ser un enlace a S3 u otro servidor de videos) - `miniatura` (string, URL de la imagen miniatura del video para previsualización) - `duracionSeg` (integer, duración del video en segundos) - `libre` (booleano, indica si este video en particular es gratuito/abierto o está restringido; complementa el flag del webinar, útil por si hay videos de muestra gratuitos en un webinar de pago) - `webinarId` (UUID, referencia al Webinar al que pertenece este video; puede ser nulo si el video se crea primero y luego se asigna)

Relaciones: cada Video pertenece a un Webinar (salvo que se permita crear videos sin asignar y luego ligar, pero en general es Webinar 1 - N Videos). También un Video puede tener múltiples Materiales de apoyo asociados (relación 1 a N hacia el módulo de materiales).

Procedimientos Almacenados y Vistas SQL: No se usaron procedimientos almacenados específicos para videos. Las consultas sobre videos (listar por webinar, etc.) son directas. Para listar los materiales de un video, existe un endpoint dedicado que realiza una consulta a la tabla de materiales filtrando por `videoId`. Esto podría haberse implementado con una **vista SQL** que una las tablas de videos y materiales para facilitar la obtención conjunta, pero en la solución actual simplemente se realiza una consulta con JOIN o una subconsulta desde la capa de aplicación cuando es requerido. En otras palabras, no hay vistas SQL permanentes para este módulo; el acceso a materiales relacionados se resuelve en la propia API mediante la ruta correspondiente o llamadas al módulo de materiales.

Rutas de la API (Videos): Prefijo de rutas: `/videos/v1`. Endpoints del módulo:

Método	Endpoint	Descripción
POST	`/videos/v1/videos`	Crea un nuevo video. Puede recibirse un `webinarId` opcional para asociarlo directamente a un webinar al crearlo, o el video puede crearse independientemente.
GET	`/videos/v1/videos/{id}`	Obtiene los detalles de un video por su ID (título, descripción, URL, duración, estado, etc.).
PATCH	`/videos/v1/videos/{id}`	Actualiza los metadatos de un video existente (por ejemplo, marcar si es libre o privado, actualizar la URL o miniatura, o la duración si cambió).
PUT	`/videos/v1/videos/{id}/webinar/{webinarId}`	Asocia un video existente a un webinar (o cambia su asociación a otro webinar). Este endpoint liga el video identificado con `{id}` al webinar identificado con `{webinarId}`.
GET	`/videos/v1/webinars/{webinarId}/videos`	Lista todos los videos que pertenecen al webinar dado. Permite a los clientes obtener la currícula (lista de lecciones en formato video) de un webinar específico. [19] [22]
GET	`/videos/v1/videos/{id}/materiales`	Lista todos los materiales de apoyo asociados al video con ID dado. Este endpoint facilita obtener los recursos adicionales de una lección sin tener que consultar manualmente el módulo de materiales. [23]

Notas: Las rutas de videos requieren autenticación. Crear, actualizar o asociar videos normalmente será tarea de un tutor o admin (según políticas, podría restringirse a admins o a tutores propietarios del webinar). La lista de videos de un webinar puede estar accesible a usuarios autenticados que tienen acceso al webinar (por ejemplo, si es un webinar libre o si el usuario tiene permiso). La ruta de listar materiales de un video es esencialmente una conveniencia; podría equivaler a llamar a `/materiales/v1/videos/{videoId}/materiales` del módulo de materiales. En implementación, se pudo manejar de las dos formas: o reenrutando la solicitud internamente, o directamente consultando la base de datos desde el servicio de videos. En cualquier caso, asegura que un usuario solo obtenga materiales de videos a los que tiene acceso (p. ej., si el video es libre o el usuario tiene acceso al webinar).

Módulo de Material de Apoyo (ms-materiales)

Propósito: Gestiona los **materiales de apoyo** asociados a videos. Estos materiales pueden ser archivos descargables (presentaciones, apuntes, PDF) o enlaces, que complementan el contenido de un video. El módulo permite crear material de apoyo para un video, actualizar sus datos, listar materiales de un video y contabilizar descargas. También aplica reglas de control de acceso, pues puede haber materiales marcados como gratuitos o restringidos según el video/webinar al que pertenezcan [24] [25].

Modelo de Datos: El modelo **MaterialApoyo** (Material de Apoyo) contiene: - `id` (UUID, clave primaria) - `nombre` (string, nombre descriptivo del material, p. ej. "Diapositivas de la Clase 1") - `url` (string, enlace de descarga o visualización del archivo) - `tipo` (opcional, tipo de archivo o recurso, p. ej. "PDF", "enlace externo", etc., para información) - `free` (booleano, indicador de si el material es libre o requiere tener acceso al video correspondiente; normalmente heredado del video: si el video es libre, el material podría ser libre, pero si el video es restringido, el material también) - `descargas` (integer, contador de cuántas veces se ha descargado o accedido el material) - `videoId` (UUID, referencia al Video al que pertenece este material)

Relaciones: cada Material está ligado a exactamente un Video (relación N a 1 hacia Video), y por ende indirectamente a un Webinar a través de ese Video.

Procedimientos Almacenados y Vistas SQL: No se implementaron procedimientos almacenados complejos en este módulo. La única operación no trivial es incrementar el contador de descargas; esto se resuelve con una simple actualización SQL (por ejemplo, `UPDATE materiales SET descargas = descargas + 1 WHERE id = ...`). Dado que es una operación atómica sencilla, no se creó un procedimiento almacenado, sino que se ejecuta directamente a través del ORM o con SQL nativo dentro de la ruta correspondiente. Tampoco existen vistas SQL para combinar datos de materiales con otros dominios; las consultas necesarias (como listar materiales de cierto video) se realizan filtrando por `videoId` en la tabla de materiales. En caso de integrarse un servicio de media separado, la entrega de materiales podría involucrar una URL firmada o un streaming, pero eso recae en el módulo Media opcional.

Rutas de la API (Materiales de Apoyo): Prefijo: `/materiales/v1`. Endpoints disponibles:

Método	Endpoint	Descripción
POST	`/materiales/v1/materiales`	Crea un nuevo material de apoyo asociado a un video (el `videoId` se provee normalmente en el cuerpo JSON).
GET	`/materiales/v1/materiales/{id}`	Obtiene el detalle de un material específico por su ID (incluyendo nombre, URL, etc., aunque la URL real de descarga podría requerir firma).
PATCH	`/materiales/v1/materiales/{id}`	Actualiza los datos de un material (nombre, URL, flag free). No suele cambiarse su asociación al video una vez creado, solo metadatos.
POST	`/materiales/v1/materiales/{id}/descargas`	Incrementa el contador de descargas del material indicado. Se invoca cuando un usuario descarga o accede al material, para llevar la métrica. [26]
GET	`/materiales/v1/videos/{videoId}/materiales`	Lista todos los materiales de apoyo pertenecientes al video con ID dado. Esto permite obtener en un solo request todos los recursos adicionales de una lección. [27]

Notas: Todas estas rutas requieren autenticación, ya que los materiales están ligados a contenido posiblemente restringido. Además, al acceder o descargar un material, típicamente se debería verificar que el usuario tenga permiso (p. ej., que el video no sea de un webinar de pago al que no está inscrito el usuario). Esa lógica de autorización se implementa comprobando el flag `free` y/o validando que el usuario haya iniciado una **vista** sobre el webinar correspondiente. En un escenario con el módulo **Media Gateway**, la ruta `GET /materiales/{id}/download` delegaría la verificación de JWT y entrega segura del archivo (por ejemplo, redirigiendo a un URL temporal) [28] .

Módulo de Visualización / Vistas de Reproducción (ms-playback)

Propósito: Gestiona las sesiones de **visualización** de los usuarios, es decir, el seguimiento del progreso de cada usuario en cada webinar. Cada sesión de visualización (denominada *Vista* o *VistasWebinar*) registra cuándo un usuario inicia ver un webinar, qué video está viendo, por dónde va (segundos vistos), si la sesión está activa, y si completó el contenido. Este módulo permite iniciar una sesión cuando el usuario comienza un webinar, actualizar su progreso, cambiar de video dentro de la misma sesión,

marcar como completado un webinar y cerrar la sesión de visualización [29] [30]. Es crucial para poder reanudar contenidos donde se dejaron y para obtener estadísticas de completitud.

Modelo de Datos: El modelo **Vista** (o **VistasWebinar**) tiene los campos siguientes: - `id` (UUID, clave primaria de la sesión de vista) - `userId` (UUID, referencia al Usuario que está viendo) - `webinarId` (UUID, referencia al Webinar que está siendo visualizado) - `videoId` (UUID, referencia al Video que el usuario está viendo actualmente dentro del webinar) - `posSeg` (integer, posición actual en segundos hasta donde el usuario ha visto del video actual) - `startedAt` (datetime, marca de tiempo de cuándo inició la sesión) - `lastSeenAt` (datetime, última marca de tiempo de actividad; se actualiza cada vez que se reporta progreso) - `activo` (booleano, indica si la sesión de visualización sigue activa/abierta en este momento) - `completado` (booleano, indica si el usuario ya completó la visualización del webinar; esto puede definirse como haber visto todos los videos o un porcentaje suficiente del contenido)

Relaciones: Vista vincula Usuario con Webinar (muchos a muchos a través de este historial) y también indica un Video actual (pero ese video puede cambiar durante la sesión). Un usuario puede tener a lo sumo **una vista activa por webinar**; la implementación fuerza el cierre de cualquier vista previa activa cuando se inicia una nueva para el mismo webinar y usuario [31] [32], de forma que no haya dos sesiones simultáneas del mismo contenido por un usuario.

Procedimientos Almacenados y Vistas SQL: No hay procedimientos almacenados en este módulo; toda la lógica de actualización de progreso y cambios se realiza en la aplicación para aplicar reglas de negocio. Tampoco se usan vistas SQL, ya que las consultas (ej. buscar vista activa de un usuario, listar historial) son directas con filtros por campos y se apoyan en índices en la tabla de vistas. Por ejemplo, la *vista activa* de un usuario se obtiene con una simple consulta `SELECT * FROM vistas WHERE userId = X AND webinarId = Y AND activo = true LIMIT 1`. Estas consultas están integradas en métodos del ORM/DAO. Para reportes más complejos (como listar todas las vistas de un webinar con ciertos filtros), se emplean consultas SQL construidas dinámicamente (incluyendo filtros por rango de fechas, etc.), e igualmente podrían apoyarse en **views** en la base de datos si se quisiera optimizar o delegar lógica, pero no fue estrictamente necesario. En resumen, la lógica de consulta está contenida en la capa de servicio.

Rutas de la API (Visualización/Playback): Prefijo: `/playback/v1`. Endpoints que maneja este módulo:

Método	Endpoint	Descripción
POST	`/playback/v1/vistas`	Inicia una nueva **sesión de visualización** (una Vista). Requiere en el cuerpo `webinarId` y opcionalmente un `videoId` inicial. Obtiene el `userId` a partir del token JWT (o se puede especificar si el rol es admin) [31] [33]. Aplica reglas: cierra cualquier vista previa activa de ese usuario en el webinar antes de crear la nueva, inicia la vista con `posSeg=0`, `activo=true`, y timestamps actuales. Devuelve la nueva entidad Vista creada (IDs y estado inicial).

Método	Endpoint	Descripción
PATCH	`/playback/v1/vistas/{id}/progreso`	Actualiza el **progreso** de la vista con ID dado. En el cuerpo se envía el nuevo `posSeg` (posición en segundos) alcanzado. Se actualiza también `lastSeenAt` a la hora actual. Aplica validaciones de negocio estrictas: (1) Que el token pertenezca al dueño de la vista (o admin); (2) Que el nuevo `posSeg` no sea menor al anterior (para prevenir trampas de retroceso) [34] ; (3) Que `posSeg` no exceda la duración total del video; (4) Se actualiza el progreso y tiempo; (5) **Opcional:** si el progreso supera un umbral (ej. 95% del video), se podría marcar automáticamente la vista como `completado=true` [35] [36] .
PATCH	`/playback/v1/vistas/{id}/switch-video`	Realiza un **cambio de video** dentro de la misma vista (sesión). Permite, manteniendo la sesión activa, saltar al siguiente video u otro video del mismo webinar. Requiere en el cuerpo un nuevo `videoId`. Al hacer switch, se reinicia `posSeg` a 0 y se actualiza `lastSeenAt` al momento actual. Opcionalmente se puede decidir si cerrar la vista anterior y abrir una nueva por cada video, pero en esta implementación por defecto se conserva la misma `id` de vista para continuidad dentro del webinar [37] [38] . También se valida que el nuevo video pertenezca al mismo webinar de la vista [39] .
PATCH	`/playback/v1/vistas/{id}/complete`	Marca la vista como **completada** manualmente. Esto se utilizaría cuando un usuario indica que terminó de ver el webinar (o se alcanza el final de todos los videos). Cambia `completado=true` en la entidad Vista. Si se conoce la duración total del webinar o videos, podría asimismo ajustar `posSeg` al final de la duración como confirmación [40] [41] . Actualiza también `lastSeenAt`. La vista puede permanecer activa o no; `completado=true` simplemente registra que el usuario concluyó el contenido.
PATCH	`/playback/v1/vistas/{id}/close`	Cierra la sesión de visualización activa, marcando `activo=false`. Esto indica que el usuario salió del webinar antes de completarlo (o después, en caso de que ya estuviera completado). No modifica el flag de `completado` (ese estado se gestiona por separado) [42] [43] , solo marca la sesión como terminada en el momento actual (`lastSeenAt` ahora). Sirve para terminar sesiones cuando el usuario cierra el video o sale del curso.
GET	`/playback/v1/vistas/{id}`	Obtiene la información de una vista (sesión) específica por su ID. Devuelve todos los detalles de la vista: usuario, webinar, video actual, progreso, timestamps, etc. Está protegido para que solo el dueño (usuario) o un admin puedan ver los detalles [44] [45] .
GET	`/playback/v1/usuarios/{userId}/vistas/activas`	Obtiene la **vista activa** del usuario `{userId}` si es que tiene una sesión abierta en algún webinar. Útil para reanudar donde dejó. Solo accesible si `{userId}` coincide con el del token (dueño) o si un admin lo consulta. Retorna la vista con `activo=true` más reciente del usuario (si existe) [46] [47] .

Método	Endpoint	Descripción
GET	`/playback/v1/usuarios/{userId}/vistas`	Lista el historial de vistas (sesiones de visualización) de un usuario. Admite paginación (`offset`, `limit`) y filtros opcionales por rango de fechas (`from`, `to`). Solo el usuario dueño o un admin pueden obtener su historial [48] [49]. El resultado incluye las sesiones con sus estados (finalizadas o no, completadas o no) para análisis del progreso del usuario.
GET	`/playback/v1/webinars/{webinarId}/vistas`	Lista todas las vistas (sesiones) de un webinar específico, típicamente para fines de **reporting** o métricas. Incluye opciones de filtro como `onlyActive` (solo sesiones activas) y rango de fechas. Esta ruta está restringida a roles administrativos o de análisis (no a usuarios normales) ya que expone potencialmente datos de varios usuarios [50] [51]. Por motivos de privacidad, podría omitir o anonimizar el `userId` de cada vista en contextos de reporte agregados.

Notas: Este módulo es crítico para la lógica de negocio, por lo que muchas validaciones de seguridad y coherencia se realizan en sus endpoints. Por ejemplo, asegurar que un usuario no pueda afectar la vista de otro (se comprueba que el `userId` de la vista corresponda al usuario del JWT, salvo admins) [52] [53]. Asimismo, impide declarativamente que un usuario marque progreso hacia atrás (posSeg decreciente) o más allá del fin del video, lo que garantiza integridad en los datos de seguimiento. Estas reglas de negocio aseguran la confiabilidad de las métricas de completitud. Todos los endpoints requieren autenticación; además, los GET y PATCH están condicionados por rol/propietario como se describió.

Módulos opcionales (Analytics y Media): Cabe mencionar brevemente que existen módulos complementarios *no descritos en detalle arriba* debido a que son opcionales: - *Analytics* (`/analytics/v1`): proporciona endpoints para obtener datos agregados de consumo, por ejemplo top 5 webinars más vistos, top 5 videos más vistos, minutos totales vistos por usuario en un rango de fechas, o la tasa de finalización de un webinar [54]. Estos endpoints realizan consultas agregadas sobre las tablas de vistas (y posiblemente otras), combinando datos de múltiples módulos. En su implementación, podrían usar SQL puro (incluso procedimientos almacenados o **vistas SQL** predefinidas para calcular, por ejemplo, el top de videos) debido a la naturaleza intensiva de estas consultas, o simplemente ejecutar queries con `GROUP BY` y filtros usando el ORM. - *Media Gateway* (`/media/v1`): expone endpoints para gestionar la entrega de archivos de video y materiales de forma segura, por ejemplo generando URLs firmadas de corta duración para reproducción o descarga [28]. Esto evita exponer directamente las URLs reales de S3 o del servidor de archivos. También provee un endpoint para descargar materiales asegurando que el solicitante tenga acceso (verificando JWT y posiblemente incrementando el contador de descarga vía el módulo de materiales).

Estos servicios opcionales interactúan con los módulos principales pero mantienen aislamiento: *Analytics* leería datos (quizá vía vistas SQL conjuntas o llamando a APIs internas) y *Media* colaboraría con Videos/Materiales para entregar contenido de manera controlada.

Consideraciones de Seguridad y Autenticación

La seguridad del sistema se basa fundamentalmente en **JWT (JSON Web Tokens)** para la autenticación y autorización de las rutas. A continuación, se detallan las principales medidas y convenciones de seguridad implementadas:

- **Autenticación con JWT:** Los usuarios obtienen un token JWT al iniciar sesión correctamente mediante `/auth/v1/login`. Este token de acceso lleva en su payload la identidad del usuario (`userId`) y su rol o privilegios, firmado con una clave secreta del servidor. También se emite un **refresh token** (JWT de larga duración o un token opaco almacenado en la base de datos) que permite obtener nuevos tokens de acceso sin volver a pedir credenciales, mediante `/auth/v1/refresh`. El token de acceso tiene una vida corta (p.ej. 15 minutos) y el de refresco más larga (p.ej. 7 días), siguiendo buenas prácticas para reducir riesgo en caso de filtración.

- **Autorización y Rutas Protegidas:** Salvo las rutas de autenticación inicial (login, register, refresh), **todas las rutas de la API requieren un JWT válido** en la cabecera `Authorization: Bearer <token>`. FastAPI verifica este token en cada petición protegida usando dependencias de seguridad. Se implementó un *dependency* global por módulo que decodifica el JWT, valida su firma y extrae el usuario (p. ej. `get_current_user`), retornando un objeto de usuario actual para usar en la lógica del endpoint. Si el token es inválido o expiró, se devuelve un error 401. Además, ciertas rutas implementan controles de autorización basados en roles y propiedad:

- **Rol Administrador:** Puede acceder a recursos sensibles de cualquier usuario, por ejemplo listar todos los usuarios, ver todas las vistas de un webinar, etc. Endpoints como `GET /usuarios` o `GET /playback/v1/webinars/{id}/vistas` requieren típicamente ser *admin* o un rol específico de operador de reportes [55].
- **Rol Usuario (estándar):** Solo puede acceder y modificar sus propios datos o acciones permitidas. Por ejemplo, un usuario autenticado puede ver o modificar su perfil, iniciar vistas y actualizar su progreso. Si intenta acceder a los recursos de otro usuario (por ID diferente), el sistema lo impide a menos que tenga rol superior. Esto se ve en rutas como `GET /playback/v1/usuarios/{userId}/vistas` donde si el `{userId}` no coincide con el del token y el usuario no es admin, la petición es rechazada [56] [48].
- **Operador/Analista:** Podría existir un rol intermedio para análisis o administración parcial (mencionado como *Operador con permisos de reporting* en el diseño [50]). Este rol tendría acceso a ciertos endpoints de reportes (analytics) sin ser full admin. En la implementación, esto se maneja verificando el campo de rol en el JWT en los endpoints correspondientes.

- **Separación de privilegios por módulo:** Cada módulo respeta las reglas de acceso. Por ejemplo, en *Materiales*, un usuario solo puede descargar material de videos a los que tiene acceso; en *Videos*, solo se pueden editar videos si se es admin o quizás tutor propietario; en *Webinars*, solo admins pueden publicar o asignar tutores, etc. Estas verificaciones están implementadas antes de realizar la acción solicitada.

- **Protección de Datos Sensibles:** Como se mencionó, las contraseñas se almacenan hasheadas (usando un algoritmo seguro Bcrypt/argon2). Los JWT contienen un identificador de sesión o ID de token para poder invalidarlos (por ejemplo, al desactivar un usuario, se podría mantener una lista negra de tokens emitidos). En este prototipo, no se implementó lista negra, pero el token expira pronto y al desactivar un usuario se impediría futuros logins.

- **CORS y Seguridad de API:** Si el frontend opera en dominio distinto, se configura CORS en FastAPI para permitir únicamente los orígenes necesarios. También se asegura el transporte mediante HTTPS para que los JWT no viajen en texto plano. Aunque estos detalles son más de entorno, se consideran dentro de las mejores prácticas.

- **Rutas Separadas por Funcionalidad:** La organización de rutas en diferentes prefijos (auth, tutores, webinars, etc.) junto con la gestión de dependencias de seguridad a nivel de router facilita aplicar políticas comunes. Por ejemplo, el router de `/auth/v1` puede aplicarse sin dependencia JWT en login/register, mientras que los routers de otros módulos (`/tutores/v1`, `/webinars/v1`, etc.) incluyen una dependencia global que exige JWT en todas sus rutas. Dentro de cada ruta, se agregan verificaciones adicionales de rol o de propiedad según lo requiera la lógica.

- **Validación de Entradas:** Todas las entradas (JSON bodies, parámetros de ruta) se validan automáticamente con Pydantic/SQLModel, previniendo inyecciones SQL o datos malformados. Además, en endpoints críticos, se implementan validaciones lógicas, p. ej., que un email tenga formato correcto, que la contraseña cumpla criterios de robustez, que los IDs proporcionados correspondan a recursos existentes (los DAOs consultan la base de datos para confirmar, de no ser así se devuelve 404).

En suma, la seguridad está integrada en cada capa: autenticación robusta con JWT, autorización granular (nivel ruta y negocio) y validaciones exhaustivas. Esto garantiza que solo usuarios legítimos accedan a los recursos y que cada quien solo vea/modifique lo que le corresponde.

Convenciones de Desarrollo Adoptadas

En la implementación de este proyecto se siguieron convenciones y buenas prácticas de desarrollo para mantener un código limpio, modular y mantenible:

- **Organización de Carpetas por Módulo:** El código fuente se estructura por dominios. Cada módulo (usuarios, tutores, webinars, etc.) reside en un paquete/directorio propio dentro del proyecto. Por ejemplo, podríamos tener:

```
app/
├── auth/        # Módulo de Autenticación y Usuarios
├── tutores/     # Módulo de Tutores
├── webinars/    # Módulo de Webinars
├── videos/      # Módulo de Videos
├── materiales/  # Módulo de Material de Apoyo
├── playback/    # Módulo de Visualización
├── analytics/   # Módulo de Analytics (opcional)
├── media/       # Módulo de Media Gateway (opcional)
└── main.py      # Punto de entrada de la aplicación FastAPI
```

  Dentro de cada carpeta de módulo se incluyen a su vez subcomponentes como:
- `models.py` (definiciones de modelos SQLModel/Pydantic del módulo),
- `routes.py` o `router.py` (definición de endpoints de FastAPI para ese módulo, usando APIRouter),

- `dao.py` o `crud.py` (funciones o clases para acceso a datos, ejecutando las operaciones en la base de datos relacionadas a ese módulo),
- `schemas.py` (si se separan modelos de entrada/salida Pydantic de los modelos de base de datos),
- `service.py` (opcional, lógica de negocio más compleja si se decide separar de los controladores de rutas),
- `__init__.py` (para exponer el router fácilmente).

Esta segmentación promueve la **separación de responsabilidades** y hace más fácil ubicar y modificar partes del sistema relativas a un dominio específico.

- **Modelos de Entrada/Salida vs. Modelos de Persistencia:** Se hizo uso de **SQLModel** para definir modelos de persistencia que también pueden actuar como modelos Pydantic. En algunos casos, para no exponer campos sensibles o para manejar validaciones específicas, se definieron modelos Pydantic adicionales:
- Modelos de creación (input), por ejemplo `UsuarioCreate` con campos `username`, `email`, `password`.
- Modelos de respuesta (output), por ejemplo `UsuarioRead` con campos `id`, `username`, `email`, `activo`, etc., omitiendo `hashed_password`.
- En SQLModel, se aprovecharon funcionalidad como `exclude_unset` o `.from_orm()` para convertir entre la entidad ORM y el esquema de salida fácilmente.
- Campos como contraseñas tienen `write_only` (solo para entrada) y nunca se devuelven en los modelos de salida.

Para otros módulos se sigue similar patrón: e.g., `WebinarCreate`/`WebinarUpdate` (sin id, sin campos calculados) y `WebinarRead` (incluyendo id, quizás cálculo de número de videos, etc.), asegurando que las respuestas de API estén limpias y no muestren p.ej. llaves foráneas crudas en lugar de datos más útiles. Sin embargo, dado que es un backend puro, generalmente se devuelven los IDs y el cliente podría usarlos para obtener detalles relacionados.

- **DAO/CRUD y Lógica de Negocio:** Se adoptó el patrón **DAO (Data Access Object)** o repositorio para la interacción con la base de datos. En lugar de escribir consultas directamente en los controladores (funciones de endpoint), éstas delegan a funciones o métodos en *dao.py*. Por ejemplo, el módulo de videos podría tener `VideoDAO` con métodos `create_video(data)`, `get_video(id)`, `update_video(id, data)`, etc., que usan la sesión de DB (inyectada vía dependencia) para realizar las operaciones. Esto desacopla la capa web de la de datos, facilitando pruebas unitarias e intercambiar la implementación de persistencia si fuera necesario.

La lógica de negocio más elaborada (como las reglas de progreso en playback, o la regla de "una vista activa a la vez") se implementa dentro de servicios o en los propios DAOs, asegurando atomicidad. Por ejemplo, `PlaybackDAO.start_view(user, webinar, video)` encapsula: cerrar vistas activas previas, crear la nueva vista, y devolverla; todo dentro de una transacción. Se utilizan transacciones proporcionadas por SQLModel/SQLAlchemy para asegurar que operaciones múltiples (como cerrar una vista y abrir otra) se completen juntas o se reviertan juntas ante error.

- **Helpers Utilitarios:** Se crearon funciones helper para tareas comunes:
- Generación y verificación de **JWT** (encapsuladas quizá en un módulo `auth/jwt.py` con funciones `create_access_token(data)`, `verify_token(token)` etc., utilizando `pyjwt` o similares).

- **Hash de contraseñas** y verificación (ej. funciones `hash_password(pwd)` y `verify_password(pwd, hash)` usando Bcrypt).
- Formateo de fechas o cálculo de rangos (por ejemplo, en analytics, para manejar parámetros `from` / `to` de tipo fecha).
- Manejo de paginación: quizás un helper que dado un query de SQLModel le aplica offset/limit y devuelve un formato estándar con `items` y `total`.

Estos helpers ayudan a no repetir código y mantener consistencia (por ejemplo, todas las contraseñas se hashean con el mismo costo de Bcrypt, todos los tokens se firman con el mismo método).

- **Convenciones de Nombres y Estilo:** Se siguió *PEP8* en el código Python. Las clases modelos usan **CamelCase** (e.g. `Usuario`, `VideoMaterial`), los endpoints y variables usan **snake_case**. Los nombres de rutas y parámetros están en español para coherencia con el dominio (e.g. usamos `/usuarios` en vez de `/users`), de modo que el API sea auto-descriptivo para clientes hispanohablantes. Las descripciones de los endpoints (como se ve en esta documentación) también se mantuvieron en español.

Las migraciones de base de datos (si fueran necesarias) no se detallaron, pero al usar SQLModel se puede generar las tablas automáticamente al iniciar (usando `SQLModel.metadata.create_all` sobre el engine, durante desarrollo).

- **Versionado de la API:** Todas las rutas incluyen la versión (`v1`) en la URL, lo que sienta la base para introducir cambios mayores en el futuro sin romper clientes existentes. La convención adoptada es tener esta versión como parte fija del path en cada router prefix, en lugar de versionar en headers.

- **Documentación y Uso de OpenAPI:** Gracias a FastAPI, se genera automáticamente documentación interactiva (Swagger UI) exponiendo todos los endpoints con sus modelos de request/response. Se añadieron descripciones a las rutas y modelos para enriquecer esa documentación. Esta práctica facilita a consumidores de la API entender cómo interactuar con el sistema y fue complementada con esta documentación escrita para una vista de alto nivel.

- **Manejo de Errores:** Se utilizaron las excepciones HTTP de FastAPI (ej. `HTTPException`) para devolver códigos y mensajes apropiados: 401 para auth fallida, 403 para prohibido (p.ej. acceso a recurso ajeno), 404 para no encontrado (p.ej. ID inválido), 400 para solicitudes malformadas o violaciones de reglas de negocio (por ejemplo "posSeg no puede ser negativo"). Estas respuestas de error siguen un formato consistente (incluyendo detalle del error en JSON). Asimismo, se validan condiciones de integridad (por ejemplo, al asociar un video a un webinar, se verifica que el webinar exista; al crear material, que el video exista, etc., retornando 404 si no).

En conclusión, la implementación del proyecto se apegó a un estándar profesional: código modular por funcionalidades, separación clara entre controladores, lógica de negocio y acceso a datos, uso adecuado de modelos Pydantic/SQLModel para evitar duplicación de definiciones, y cumplimiento de prácticas de seguridad y estilo. Esto resulta en un código base más limpio, fácil de mantener y extender, y en un servicio backend confiable y claro en su estructura para cualquier desarrollador que deba interactuar con él o continuarlo.

---

1 2 3 4 5 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 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 MISC-Actividad2-HernandezAdameJuanEduardo.pdf

file://file-Q1YhesXTS9TKEg5FdJCDky