Manual del Programador

Se describen controladores, servicios y modelos del sistema

Modelos

Modelos de Datos (Eloquent)

Este documento condensa los conocimientos y entidades de la capa de datos (`app/Models`) del sistema **GDU**, describiendo la responsabilidad primordial de cada Modelo y sus relaciones principales.

? Arquitectura Institucional y Acceso

1. **`Dependencia`**
   - Entidad que representa a las áreas operativas, oficinas o secretarías de la Institución.
   - Se relaciona fuertemente con los Expedientes actuando como creador y como **poseedor (holder)**. Puede agrupar a múltiples usuarios y participar de los "pases paralelos" mediante tablas pivote.
2. **`User`**
   - Representa al usuario físico/agente que opera el sistema. Utiliza características de `Spatie/Permissions` (Roles), autenticación de Jetstream, y se encuentra asignado en "caliente" a una dependencia (`dependenciaActiva`) en la que opera temporalmente.

? Núcleo Documental (Expedientes)

1. **`Expediente`**
   - Es el corazón y objeto transaccional principal del sistema. Funciona como un agrupador legal de los sucesos.
   - Tiene dos relaciones temporarles críticas: `dependencia_id` (la originaria que lo creó) y `holder_id` (El tenedor actual en el flujo lógico, quien lo tiene en "posesión").
2. **`TipoExpediente`**
   - Modelo de configuración y lectura. Define de forma maestra la taxonomía (Ej: Acción de Pago, Licitación, Expediente Estándar), sus colores de interfaz, e íconos.
3. **`ExpedienteManifest`**
   - Entidad puramente técnica. Modela la cadena de bloques (Blockchain interno) del expediente. Guarda instantáneas versionadas (`version`, `manifest_hash`, `prev_manifest_hash`, `state_hash`) garantizando integridad legal de manera inmutable tras cada movimiento.

4. **`DependenciaExpediente`**
   - Modelo que actúa netamente como *Clase Pivote* (`Pivot`) con atributos adicionales (`activo`, `expira`) para sostener de forma estructurada el **compartir simultáneos paralelo** los expedientes.

? Solicitudes

1. **`Solicitud`**
   - Predecesora del Expediente. Es el formulario interactivo provisto en la mesa de entrada o self-service. Contiene los datos crudos provistos por el "Causante". Una vez aprobada se convierte en Expediente Oficial mediante la rutina de `caratular`.

?Comunicaciones

1. **`Ccoo`**
   - Modela una "*Comunicación Oficial*". Es una entidad de envío de oficios, resoluciones o mensajes entre una `dependencia_origen` y una `dependencia_destino`.

? Archivos y Adjuntos (Ecosistema Múltiple)

El sistema segrega fuertemente los subidos acorde al ámbito donde se agregaron:

1. **`Archivoe`**
   - Archivos y fojas PDF vinculadas directamente a un **Expediente** Oficializado. Está versionado (`hash`), auditado, y permite adjuntar `firmas`.
2. **`Archivot`**
   - Archivos transitorios vinculados inicialmente a una **Solicitud** que aún no es Expediente.
3. **`ArchivoCcoo`**
   - Disposición de almacenamiento enfocada a las Comunicaciones Oficiales (`Ccoo`).
4. **`Firma`**
   - Entidad complementaria al `Archivoe`. Captura los metadatos específicos leídos en un PDF (Nombre del certificado, fecha de validación legal del PKI, validez criptográfica) en un registro verificable.
5. *(Nota: `File` es una clase obsoleta o fallida marcada en el código para deprendimiento futuro)*

? Herramientas de Sistema

1. **`Log`**
   - Modela la traza silenciosa de auditoría interna. Almacena la Acción, qué Usuario/Dependencia interactuó, sobre qué entidad (`Solicitud` / `Expediente`) y anexa factores forenses básicos de HTTP (IP, User Agent).
2. **`Email`**
   - Réplica de correos institucionales inyectados. Guarda destinatario, remitente y el cuerpo limpio que luego se utiliza interactivamente con el controlador de Inteligencia Artificial de las bandejas integradas.

Controladores

Controladores (app/Http/Controllers)

1. **`ArchivoController`:** Verifica permisos locales y devuelve archivos privados incrustados de forma segura dentro del sistema (`view()`) o fuerza su descarga directa (`download()`).
2. **`CcooController`:** Administra el ciclo de vida de creación de las "Comunicaciones Oficiales" (CCOO), y se apoya en el asistente de IA para pre-rellenar los contenidos institucionales.
3. **`EmailController`:** Lista la bandeja de entrada de correos sincronizados. Al visualizar un email (`show()`) también gatilla el servicio de OpenAI para tener un borrador de respuesta listo para usar.
4. **`ExpedienteController`:** Es el corazón del ruteo. Gestiona el registro y visualización de expedientes. Interactúa con el blockchain de archivos mostrándo si el documento es válido y permite búsquedas múltiples por año y clave.
5. **`LogController`:** Actúa como auditor. Recibe y registra todos los eventos sensibles del usuario guardando su IP, User-Agent corporativo y la dependencia activa generando un hash único por la actividad sobre Expedientes o Solicitudes.
6. **`NotificationController`:** Permite al usuario ver, listar y marcar como leídas las notificaciones propias (`markAsRead()`, `markAllAsRead()`).
7. **`SolicitudController`:** El origen del trámite web (*CRUD* de solicitudes). Una vez que una solicitud pasa todos los requerimientos (`aprobar()`), este controlador usa la función `caratular()` para transformarla en un "Expediente formal" dentro del sistema.

Servicios

?? Servicios (app/Services)

1. **`ExpedienteService`:** Gestiona el flujo del movimiento de los `Expedientes` entre las distintas `Dependencias`. Resuelve lógicas para pases en paralelo y comprueba que dependencias tienen permisos para adjuntar archivos a un expediente específico.

   Funciones Activas (Core del Servicio)
   - **`obtenerDependenciasEnPaseParaleloActivo(int $expedienteId)`** **Funcionalidad:** Es una consulta de bajo nivel (`DB::table`) a la tabla pivote `dependencia_expediente` que relaciona un expediente con varias oficinas simultáneamente. **Qué hace:** Retorna en forma de un simple *array* los IDs numéricos de todas las dependencias con las cuales un expediente comparte el estado de "pase paralelo" en estado "activo".
   - **`sePuedeTransferir(int $expedienteId, int $dependenciaActualId)`** **Funcionalidad:** Evalúa lógicamente si el usuario puede "Transferir/Enviar" un expediente. **Regla de negocio actual:** Busca en la BD si el expediente pertenece a esa dependencia actual (`holder_id === $dependenciaActualId`) y *además* comprueba que NO tenga pases paralelos distribuidos (`!$relacionesActivas`). Por lo tanto, un expediente en medio de un "pase en paralelo" activo hacia varias oficinas **no se puede** transferir de raíz al próximo paso hasta que se cierre el paralelo.
   - **`dependenciaPuedeAgregarArchivos(int $expedienteId, int $dependenciaId)`** **Funcionalidad:** Decide si una dependencia en concreto tiene permitido el alta de nuevos documentos. **Regla de negocio actual:** Si la dependencia está involucrada dentro del listado de "Pases en paralelo activo", entonces le permite añadir archivos (`return true;`). Si no está en paralelo, evalúa si es el simple poseedor primario del expediente (`holder_id`). *Dato interesante del código:* Antes había un bloque de validación más rígido que impedía subir archivos, pero fue comentado asumiendo el nuevo permiso de que "*AHORA PERMITIMOS QUE SE AGREGUE DOCUMENTACION AUNQUE ESTE EN PARALELO*" .
   - **`yoTengoExpediente(int $expedienteId, int $dependenciaId)`** **Funcionalidad:** Simple comprobación binaria para uso rápido en vistas (if/else). **Qué hace:** Solo evalúa si la dependencia X pedida coincide con el poseedor primario (`holder_id`) cargado en el Objeto Expediente.
   - **`yoTengoExpedienteEnParalelo(int $expedienteId, int $dependenciaId)`** **Funcionalidad:** Complementa la función de arriba. **Qué hace:** Llama al método `obtenerDependenciasEnPaseParaleloActivo()` y revisa con un simple `in_array` si la dependencia X está contenida en ese listado de oficinas

involucradas en un pase conjunto.

2. **`FirmaDigitalService`:** Responsable de integrar funciones a nivel de sistema operativo para validar firmas digitales en los PDF (usando comandos como `pdfsig` y bases `NSS`) y aplica una "firma por sistema" a los documentos.
3. **`ManifiestoExpedienteService`:** Maneja la integridad de la documentación aplicando un modelo estilo "blockchain". Sella expedientes (`sellar()`) adjuntando manifiestos con *hash canonicos* estado por estado para garantizar que los documentos no hayan sido adulterados leyendo el historial (`verificarCadena()`).
4. **`OpenAIService`:** Centraliza la conexión y peticiones a la API de ChatGPT para sugerir o generar de forma automática respuestas (`generateReply()`) a correos y comunicaciones simulando asistentes virtuales.