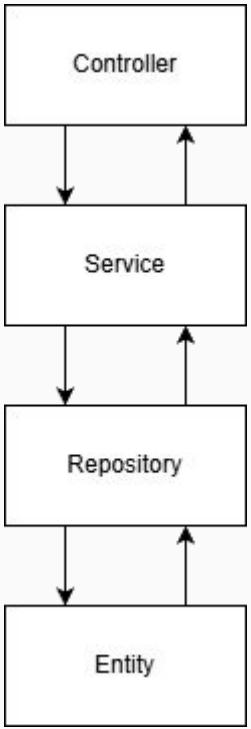
Sustentación técnica

Juan José Castro Gutiérrez

Arquitectura general

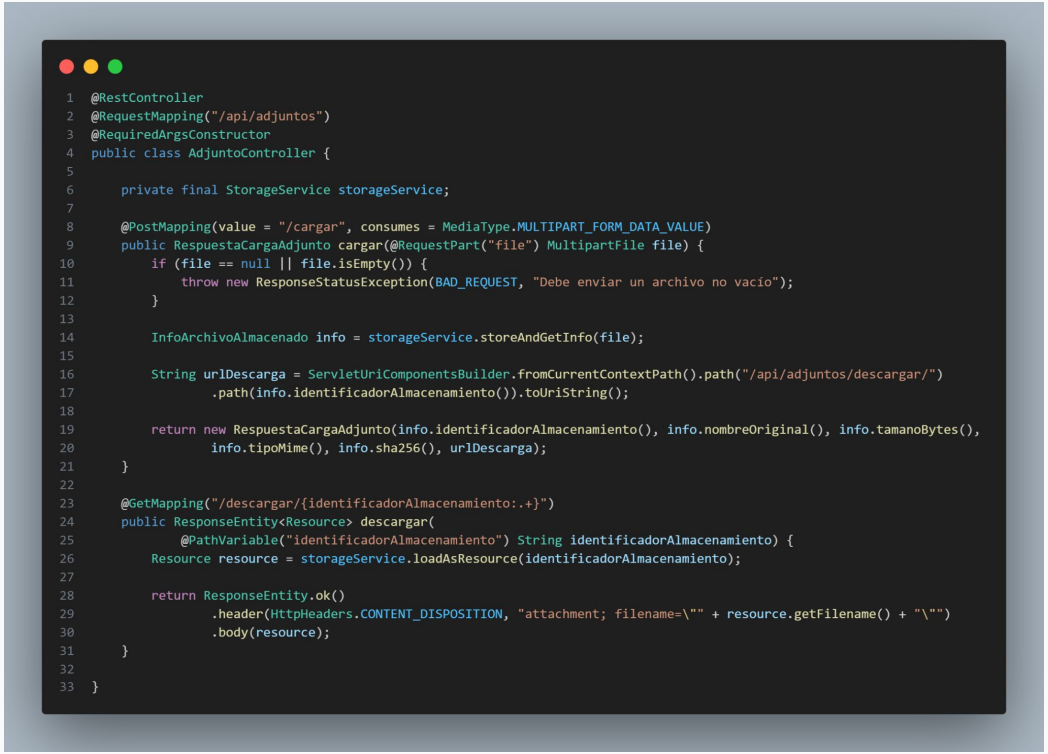
Layered Architecture

Esta es una arquitectura clásica y muy usada en Spring, donde se separan claramente las responsabilidades



Controller

- Archivo: AdjuntoController.java
- Rol: Maneja las solicitudes HTTP (ej. @PostMapping, @GetMapping), expone endpoints REST.
- Interactúa con el servicio (StorageService) para realizar lógica de negocio



```java
@RestController
@RequestMapping("/api/adjuntos")
@RequiredArgsConstructor
public class AdjuntoController {

    private final StorageService storageService;

    @PostMapping(value = "/cargar", consumes = MediaType.MULTIPART_FORM_DATA_VALUE)
    public RespuestaCargaAdjunto cargar(@RequestPart("file") MultipartFile file) {
        if (file == null || file.isEmpty()) {
            throw new ResponseStatusException(BAD_REQUEST, "Debe enviar un archivo no vacío");
        }

        InfoArchivoAlmacenado info = storageService.storeAndGetInfo(file);

        String urlDescarga = ServletUriComponentsBuilder.fromCurrentContextPath().path("/api/adjuntos/descargar/")
                .path(info.identificadorAlmacenamiento()).toUriString();

        return new RespuestaCargaAdjunto(info.identificadorAlmacenamiento(), info.nombreOriginal(), info.tamanoBytes(),
                info.tipoMime(), info.sha256(), urlDescarga);
    }

    @GetMapping("/descargar/{identificadorAlmacenamiento:.+}")
    public ResponseEntity<Resource> descargar(
            @PathVariable("identificadorAlmacenamiento") String identificadorAlmacenamiento) {
        Resource resource = storageService.loadAsResource(identificadorAlmacenamiento);

        return ResponseEntity.ok()
                .header(HttpHeaders.CONTENT_DISPOSITION, "attachment; filename=\"" + resource.getFilename() + "\"")
                .body(resource);
    }

}
```

Service

storeAndGetInfo(MultipartFile file)

Guarda el archivo con un nombre aleatorio usando UUID.

- Calcula el hash SHA-256 para detectar duplicados o validar integridad.
- Detecta el MIME type (tipo de archivo).
- Devuelve un objeto InfoArchivoAlmacenado con:
  - Clave (nombre en disco)
  - Nombre original
  - Tamaño
  - MIME
  - Hash SHA-256

```java
@Override
    public InfoArchivoAlmacenado storeAndGetInfo(MultipartFile file) {
        try {
            if (file == null || file.isEmpty())
                throw new StorageException("Failed to store empty file.");

            String original = StringUtils
                    .cleanPath(file.getOriginalFilename() == null ? "file" : file.getOriginalFilename());

            String ext = StringUtils.getFilenameExtension(original);

            String key = UUID.randomUUID() + (ext == null || ext.isBlank() ? "" : "." + ext.toLowerCase());

            Path destinationFile = this.rootLocation.resolve(key).normalize().toAbsolutePath();

            if (!destinationFile.getParent().equals(this.rootLocation.toAbsolutePath())) {
                throw new StorageException("Cannot store file outside current directory.");
            }

            MessageDigest md = MessageDigest.getInstance("SHA-256");

            try (InputStream is = file.getInputStream(); DigestInputStream dis = new DigestInputStream(is, md)) {
                Files.copy(dis, destinationFile);
            }

            String sha256 = toHex(md.digest());
            String mime = normalizeMime(file.getContentType(), destinationFile, original);

            return new InfoArchivoAlmacenado(key, original, file.getSize(), mime, sha256);

        } catch (FileAlreadyExistsException e) {
            return storeAndGetInfo(file);
        } catch (Exception e) {
            throw new StorageException("Failed to store file.", e);
        }
    }
```

Repository

- Archivo: AdjuntoRepository.java
- Rol: Capa de persistencia, basada en Spring Data JPA (extends JpaRepository), acceso a base de datos.
- Interactúa con la entidad Adjunto.

```java
public interface AdjuntoRepository extends JpaRepository<Adjunto, Long> {

    Boolean existsByIdentificadorAlmacenamiento(String identificadorAlmacenamiento);

    List<Adjunto> findByTramite_IdOrderByIdAsc(Long tramiteId);

    List<Adjunto> findByTramite_IdInOrderByTramite_IdAscIdAsc(List<Long> tramiteIds);

}
```

Entity

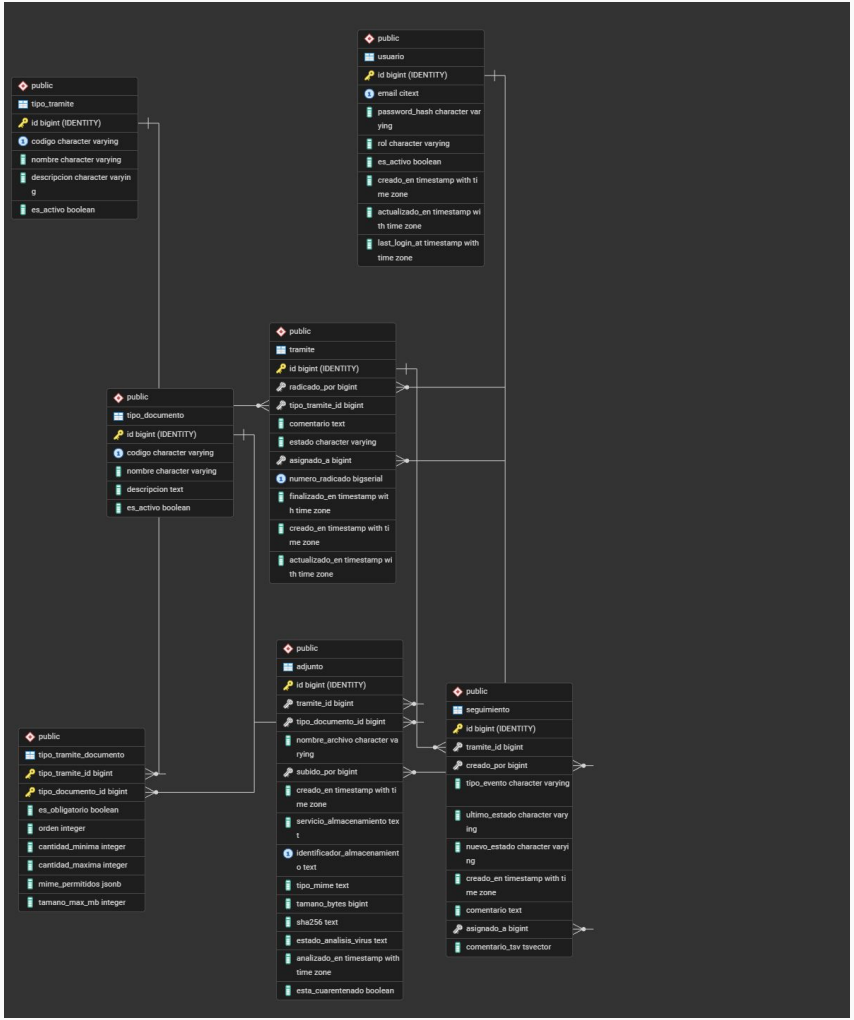
- Archivo: Adjunto.java
- Rol: Representa la tabla en la base de datos. Uso de @Entity, @Id, etc.
- Trabaja con JPA para mapear objetos Java a registros en la base de datos.

```java
@Entity
@Table
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Builder
@EqualsAndHashCode(onlyExplicitlyIncluded = true)
@ToString(exclude = { "tramite", "tipoDocumento", "subidoPor" })
public class Adjunto {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id", nullable = false)
    @EqualsAndHashCode.Include
    private Long id;

    @ManyToOne(fetch = FetchType.LAZY, optional = false)
    @JoinColumn(name = "tramite_id", nullable = false, foreignKey = @ForeignKey(name = "adjunto_tramite_id_fkey"))
    private Tramite tramite;

    @ManyToOne(fetch = FetchType.LAZY, optional = false)
    @JoinColumn(name = "tipo_documento_id", nullable = false, foreignKey = @ForeignKey(name = "adjunto_tipo_documento_id_fkey"))
    private TipoDocumento tipoDocumento;

    @Column(name = "nombre_archivo", nullable = false)
    private String nombreArchivo;

    @ManyToOne(fetch = FetchType.LAZY, optional = false)
    @JoinColumn(name = "subido_por", nullable = false, foreignKey = @ForeignKey(name = "adjunto_subido_por_fkey"))
    private Usuario subidoPor;

    @Builder.Default
    @Enumerated(EnumType.STRING)
    @Column(name = "servicio_almacenamiento", nullable = false)
    private ServicioAlmacenamiento servicioAlmacenamiento = ServicioAlmacenamiento.LOCAL;

    @Column(name = "identificador_almacenamiento", nullable = false)
    private String identificadorAlmacenamiento;

    @Column(name = "tipo_mime", nullable = false)
    private String tipoMime;

    @Column(name = "tamano_bytes", nullable = false)
    private long tamanoBytes;

    @Column(name = "sha256", nullable = false)
    private String sha256;

    @Builder.Default
    @Enumerated(EnumType.STRING)
    @Column(name = "estado_analisis_virus", nullable = false)
    private EstadoAnalisisVirus estadoAnalisisVirus = EstadoAnalisisVirus.PENDIENTE;

    @Generated(event = { EventType.INSERT, EventType.UPDATE })
    @Column(name = "analizado_en", columnDefinition = "timestamp with time zone")
    private OffsetDateTime analizadoEn;

    @Builder.Default
    @Column(name = "esta_cuarentenado", nullable = false)
    private boolean estaCuarentenado = false;

    @Generated(event = { EventType.INSERT })
    @Column(name = "creado_en", nullable = false, insertable = false, updatable = false, columnDefinition = "timestamp with time zone")
    private OffsetDateTime creadoEn;
}
```

Modelo lógico

ERD simplificado

- usuario: personas/funcionarios (quien solicita o gestiona).
- tipo_tramite: catálogo de tipos de trámite.
- tipo_documento: catálogo de documentos (DNI, recibo, etc.).
- tipo_tramite_documento: M:N que define qué documentos se exigen por tipo de trámite (y si son obligatorios).
- tramite: instancia concreta de un trámite solicitado por un usuario.
- seguimiento: eventos de tracking del trámite (estado, comentario, fecha, actor).
- adjunto: archivos subidos a un trámite, clasificados por tipo_documento.



Normalización

cómo cumple 1FN–3FN/BCNF

- 1FN:
  - Cada atributo es atómico: no guardamos listas de documentos en tramite; en su lugar usamos adjunto (una fila por archivo) y tipo_tramite_documento (una fila por requisito).
  - Sin columnas repetidas (p. ej., documento1, documento2…).
- 2FN (para tablas con PK compuesta):
  - La única tabla con dependencia sobre compuesta sería la puente tipo_tramite_documento, y todas sus columnas no-clave (obligatorio, orden) dependen de toda la clave (tipo_tramite_id, tipo_documento_id), no sólo de una parte. Además, la PK efectiva la resolvemos con UNIQUE + id surrogate.
- 3FN / BCNF:
  - No hay dependencias transitivas: en tramite, tipo_tramite_id determina sólo atributos del trámite, no atributos del tipo (que residen en tipo_tramite).
  - En adjunto, metadatos del archivo no dependen de tipo_documento salvo por su ID; la descripción del tipo vive en su catálogo.
  - En seguimiento, estado y comentario dependen de la fila (evento), no del usuario ni del tramite de forma transitiva.

Relaciones (FKs)

tramite

Relaciones (FKs) y su razón de ser

- radicado_por → usuario(id) y asignado_a → usuario(id): integridad entre quién radica y quién gestiona el trámite. Evita huérfanos si se elimina un usuario.

- tipo_tramite_id → tipo_tramite(id): separa catálogo (estático) de instancias (dinámico). Mantiene 3FN/BCNF.

seguimiento

Relaciones (FKs) y su razón de ser

- radicado_por → usuario(id) y asignado_a → usuario(id): integridad entre quién radica y quién gestiona el trámite. Evita huérfanos si se elimina un usuario.

- tipo_tramite_id → tipo_tramite(id): separa catálogo (estático) de instancias (dinámico). Mantiene 3FN/BCNF.

adjunto

Relaciones (FKs) y su razón de ser

- tramite_id → tramite(id): cada archivo pertenece a un trámite.
- tipo_documento_id → tipo_documento(id): clasifica el archivo con un tipo de documento del catálogo.

tipo_tramite_documento

Relaciones (FKs) y su razón de ser

tipo_tramite_id → tipo_tramite(id) y tipo_documento_id → tipo_documento(id): define la matriz de requisitos (M:N) entre catálogos.

Índices por tabla

usuario

Índices por tabla y justificación

- UNIQUE (email) (con citext): unicidad case-insensitive para login e integraciones.
- INDEX (es_activo): filtros frecuentes en listados de administración (WHERE es_activo).
- INDEX (rol, es_activo): dashboards/consultas por rol y actividad; el orden permite filtrar por rol y luego refinar por activo.

tipo_tramite

Índices por tabla y justificación

- INDEX (es_activo): catálogos visibles ≈ activos.
- UNIQUE (lower(btrim(nombre))): evita duplicados por mayúsculas/espacios; útil para admins y validaciones.

tipo_documento

Índices por tabla y justificación

- INDEX (es_activo): mismo patrón de catálogo.
- UNIQUE (lower(btrim(nombre))): evita nombres duplicados por casing/espacios.

tipo_tramite_documento

Índices por tabla y justificación

- INDEX (tipo_tramite_id): consultas “trae requisitos del tipo X”.
- INDEX (tipo_documento_id): validaciones “este doc aplica a qué trámites”.
- INDEX (tipo_tramite_id, es_obligatorio, orden) (lo tienes como ix_ttd_tram_obl_orden): para pintar formularios en orden y separar obligatorios/opcionales sin reordenaciones costosas.
- UNIQUE (tipo_tramite_id, orden) WHERE orden IS NOT NULL (ux_ttd_tram_orden): garantiza un orden estable por tipo; evita colisiones de orden en UI.

tramite

Índices por tabla y justificación

- UNIQUE (numero_radicado): identificador externo/folio navegable.
- INDEX (radicado_por): bandeja “mis trámites”.
- INDEX (asignado_a, actualizado_en DESC) y índice parcial WHERE estado IN ('RADICADO','EN_PROCESO'): bandejas operativas típicamente enfocadas en activos; el parcial reduce tamaño y mejora selectividad.
- INDEX (tipo_tramite_id): filtros por clase de trámite; útil para reportes y SLA por tipo.

seguimiento

Índices por tabla y justificación

- INDEX (tramite_id, creado_en DESC) (dos variantes):
  - General: (..., id DESC) para paginación estable por id sin depender de reloj.
  - Parciales por tipo_evento='ASIGNACION' y 'CAMBIO_ESTADO': aceleran vistas específicas (timeline de asignaciones / cambios de estado) sin cargar los demás eventos; alta selectividad → mejor plan.
- INDEX (creado_por): “acciones que hice” (perfil usuario).
- INDEX (asignado_a, creado_en DESC): bandeja de trabajo del gestor con orden por recencia.

adjunto

Índices por tabla y justificación

Relaciones con tramite, tipo_documento y usuario (metadatos).

Servicios implementados

CustomUserDetailsService

Servicio encargado de integrar la aplicación con Spring Security.

Carga los datos del usuario (username, password, roles/authorities) desde la base de datos para el proceso de autenticación.

UsuarioService

Gestiona la lógica de negocio relacionada con los usuarios.

Incluye operaciones como creación, consulta, actualización y validación de información del usuario.

UserAuthenticatedService

Proporciona información del usuario actualmente autenticado en la sesión.

Se usa para obtener datos como el identificador del usuario, roles o credenciales sin volver a consultar la base de datos.

ConsultaCatalogosService

Servicio dedicado a la consulta de catálogos o datos maestros.

Centraliza la obtención de listas estáticas o parametrizadas (tipos, estados, clasificaciones, etc.).

SeguimientoService

Permite realizar el seguimiento de trámites o procesos.

Expone la lógica necesaria para consultar el estado, historial y evolución de un trámite.

RadicacionTramiteService

Encargado de la radicación o registro inicial de un trámite.

Contiene la lógica para crear un nuevo trámite, validar datos y persistir la información inicial.

TramiteQueryService

Servicio enfocado en la consulta de trámites.

Optimizado para búsquedas, filtros y recuperación de información sin modificar el estado del trámite.

TramiteWorkflowService

Gestiona el flujo de trabajo (workflow) del trámite.

Controla los cambios de estado, transiciones, validaciones de negocio y reglas del proceso.

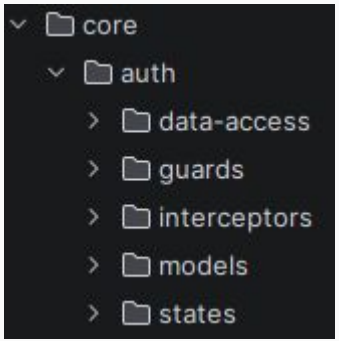
FileSystemStorageService

Servicio encargado de la gestión de archivos en el sistema de almacenamiento (filesystem).

Implementa la lógica para guardar, recuperar, eliminar y validar archivos asociados a la aplicación (por ejemplo, documentos de trámites), desacoplando el acceso al sistema de archivos del resto de la lógica de negocio.

Estructura del fontend

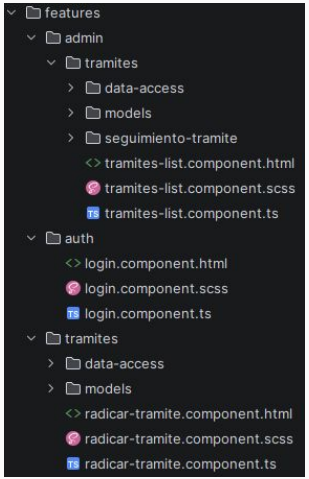
core/ (El "Corazón" de la App)



Esta carpeta contiene lo que es único y global para la aplicación. Normalmente, aquí se encuentran servicios que solo deben instanciarse una vez (Singletons).

- auth/: Centraliza toda la lógica de seguridad.
  - data-access: Servicios para llamadas a la API (login, logout).
  - guards: Protecciones de rutas (por ejemplo, evitar que alguien entre sin estar logueado).
  - interceptors: Lógica para añadir tokens JWT a las peticiones HTTP automáticamente.
  - states: Manejo de estado global del usuario.

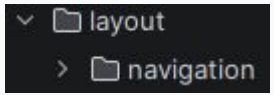
features/ (Módulos de Negocio)



Es la parte más importante para la escalabilidad. En lugar de dividir por "componentes" o "servicios", se divide por funcionalidades de usuario.

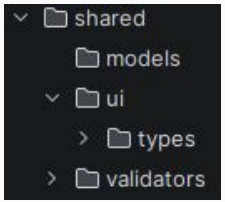
- admin/: Un submódulo específico para las tareas administrativas. Vemos que tiene su propio flujo de tramites.
- auth/: Contiene la interfaz de usuario para la autenticación (el formulario de login).
- tramites/: Funcionalidad principal del negocio. Contiene el componente radicar-tramite.
- Estructura Interna: Nota que cada feature repite un patrón:
    - data-access/: Lógica de datos específica para esa pantalla.
    - models/: Interfaces TypeScript exclusivas de esa funcionalidad.

layout/ (Estructura Visual)



Aquí se definen los elementos que se mantienen constantes en la navegación, como el header, el footer o, como se ve en la imagen, la navigation (menús laterales o barras superiores).

shared/ (Recursos Reutilizables)



Aquí reside todo lo que se utiliza en dos o más módulos de features.

- ui/: Componentes "tontos" o de presentación (botones personalizados, inputs, tablas genéricas).
- validators/: Validaciones personalizadas para formularios que se usan en distintos lugares del sistema.
- models/types/: Definiciones de datos que son transversales a toda la aplicación.

Retos encontrados

Qué mejoraría con más tiempo