UNIVERSIDAD NUEVA ESPARTA

FACULTAD DE CIENCIAS DE LA INFORMÁTICA

ESCUELA DE COMPUTACIÓN

Complementaria II

PERIODO LECTIVO: PL-165

Análisis Arquitectónico

Estudiante:

Muchacho Figueroa, Andrea Paola Del Valle

C.I: 31.131.098

Caracas, Enero, 2026

1. Descripción del Proyecto

- **Problema que resuelve:** La Coordinación de Sistemas se enfrenta a la descentralización y fragmentación de su información técnica, manuales, normativas y reportes. Esto genera ineficiencias en la búsqueda de información, pérdida de conocimiento institucional y cuellos de botella en la inducción de nuevo personal. El módulo busca centralizar toda esta documentación en un único repositorio estructurado.
- **Usuarios objetivo:** Personal administrativo, coordinadores, desarrolladores y pasantes de la Coordinación de Sistemas.
- **Funcionalidades principales (Core):**
    - Carga, descarga y previsualización de documentos PDF, Word y Excel.
    - Categorización jerárquica (carpetas, etiquetas y metadatos).
    - Control de acceso basado en roles (Lectura, Edición, Administración).
    - Motor de búsqueda por filtros (nombre, fecha, autor, categoría).
- **Funcionalidades secundarias (Nice-to-have):**
    - Control de versiones de documentos.
    - Registro de auditoría (quién subió/modificó/descargó qué documento y cuándo).
    - Generación de reportes estadísticos de uso.
- **Valor único:** A diferencia de soluciones genéricas como Google Drive o Dropbox, este módulo estará integrado nativamente dentro del ecosistema del sistema MiUNE 2.0, compartiendo la misma base de datos PostgreSQL, la misma autenticación de usuarios y adaptándose exactamente a los flujos de trabajo de la coordinación.

2. Requisitos Técnicos y No Funcionales

Requisitos Funcionales:

- El sistema debe permitir la autenticación e inicio de sesión integrados con MiUNE 2.0.
- El sistema debe permitir el CRUD (Crear, Leer, Actualizar, Eliminar) de registros documentales.
- El sistema debe asignar permisos de visualización o modificación según el rol del usuario.

Requisitos No Funcionales:

- **Performance:** Los tiempos de respuesta para búsquedas complejas no deben superar los 2 segundos. La carga de documentos (hasta 50MB) debe procesarse en segundo plano o tener retroalimentación visual inmediata.
- **Seguridad:** Toda conexión debe estar cifrada. Las rutas de almacenamiento de archivos en el servidor Linux Ubuntu deben estar protegidas y ser

inaccesibles directamente por URL, requiriendo validación de token/sesión para su descarga.
- **Escalabilidad:** La estructura relacional en PostgreSQL debe estar optimizada mediante índices para soportar un crecimiento sostenido de miles de registros documentales sin degradación de rendimiento.
- **Disponibilidad:** Se requiere alta disponibilidad durante el horario laboral universitario (8:00 AM - 6:00 PM), con ventanas de mantenimiento programadas en horario nocturno.
- **Ambiente Tecnológico:** Despliegue obligatorio en servidores on-premise bajo distribución Linux Ubuntu, utilizando PostgreSQL para almacenamiento de datos.

3. Análisis Comparativo de Arquitecturas

Opción A: Monolito MVC Tradicional (Ej. Node.js con EJS / Laravel)

- **Descripción técnica:** Todo el código (interfaz de usuario, lógica de negocio y acceso a datos) se empaqueta y despliega en una única aplicación. Las vistas se renderizan en el servidor y se envían como HTML al navegador.
- **Ventajas (Para MiUNE):** Curva de aprendizaje baja. Excelente para despliegues sencillos en un único servidor Ubuntu. Menor complejidad inicial al no tener que gestionar repositorios separados.
- **Desventajas:** La interfaz puede sentirse menos fluida al recargar la página completa en cada navegación. Es más difícil escalar equipos de trabajo (frontend vs backend) a futuro.
- **Complejidad:** Baja.
- **Costo de Mantenimiento:** Bajo al inicio, pero aumenta si el sistema crece y el código se acopla demasiado.

Opción B: Arquitectura SPA (Backend API REST + Frontend React/Vue)

- **Descripción técnica:** Separación estricta entre una API en el backend que sirve datos en formato JSON, y una Single Page Application (SPA) en el navegador que maneja el enrutamiento y la interfaz gráfica dinámicamente.
- **Ventajas (Para MiUNE):** Experiencia de usuario extremadamente fluida (ideal para navegar entre muchas carpetas y documentos sin recargas). Permite que la API de documentos sea consumida por otros módulos de MiUNE 2.0 en el futuro.
- **Desventajas:** Requiere gestionar dos proyectos distintos (frontend y backend). Mayor configuración inicial (CORS, gestión de tokens JWT).
- **Complejidad:** Media.
- **Estándares aplicables:** 12-Factor App (separación de configuración, API-first) y Richardson Maturity Model Nivel 2 para la API.

Opción C: Arquitectura JAMstack (Ej. Next.js)

- **Descripción técnica:** Enfocada en pre-renderizar el marcado (HTML) y usar APIs a través de JavaScript del lado del cliente.
- **Ventajas:** Excelente rendimiento percibido y altísima seguridad al desacoplar el frontend del servidor de base de datos de manera estricta.
- **Desventajas:** Sobredimensionado para una intranet o panel administrativo que requiere autenticación constante para ver contenido dinámico (como un gestor de documentos privado).
- **Complejidad:** Alta (por los requerimientos de hidratación y manejo de estado en aplicaciones tipo panel de control cerrado).

Criterio	Monolito MVC	Arquitectura SPA	JAMstack
Experiencia de Usuario (Fluidez)	Regular	Excelente	Muy Buena
Separación de Responsabilidades	Media	Alta	Alta
Complejidad de Implementación	Baja	Media	Alta (para este contexto)
Escalabilidad para otros módulos	Baja	Excelente (API reutilizable)	Buena
Mantenimiento a largo plazo	Medio	Bajo/Medio	Medio

4. Decisión Arquitectónica Fundamentada

Arquitectura elegida: Monolito MVC Tradicional renderizado en el servidor.

Justificación técnica: La decisión de implementar un Monolito MVC se fundamenta en el principio de consistencia y compatibilidad del ecosistema. El sistema central

MiUNE 2.0 opera bajo este paradigma. Introducir una arquitectura SPA (Single Page Application) o JAMstack para este módulo específico generaría una fragmentación técnica innecesaria en la Coordinación de Sistemas. Al mantener la arquitectura monolítica, se logran los siguientes beneficios:

1. **Integración nativa:** El módulo puede compartir directamente la sesión de usuario, los middlewares de seguridad y la conexión a la base de datos PostgreSQL existente sin necesidad de construir capas de API intermedias complejas ni lidiar con políticas CORS.
2. **Despliegue unificado:** Se aprovecha la infraestructura de servidores Linux Ubuntu actual de la universidad, desplegando el módulo como una extensión del sistema principal en lugar de requerir pipelines de CI/CD separados para frontend y backend.
3. **Mantenibilidad:** El equipo de desarrollo actual ya domina este flujo de trabajo, lo que reduce la curva de aprendizaje y facilita el mantenimiento a largo plazo.

Trade-offs (Compromisos) aceptados: Al elegir esta arquitectura, acepto conscientemente sacrificar parte de la fluidez extrema en la interfaz de usuario que ofrecería una SPA (habrá recargas de página parciales o completas al navegar entre carpetas de documentos). Asimismo, asumo que la lógica de presentación estará más acoplada al servidor, lo que requerirá un diseño cuidadoso de las vistas para mantener el código limpio.

5. Diseño Conceptual de Arquitectura MVC

Modelos (Data Models en PostgreSQL):

Model: Documento

- **Campos:**
  - id: UUID (Primary Key)
  - titulo: Varchar(150)
  - nombre_archivo: Varchar(255)
  - ruta_almacenamiento: Varchar(500) (Ruta protegida en el servidor Linux)
  - tipo_mime: Varchar(50) (ej. application/pdf)
  - peso_kb: Integer
  - categoria_id: UUID (Foreign Key)
  - subido_por: UUID (Foreign Key a tabla User de MiUNE)
  - fecha_subida: Timestamp
- **Validaciones:**
  - El tipo_mime debe estar dentro de una lista blanca permitida (PDF, DOCX, XLSX).
  - El archivo físico debe existir en el servidor antes de guardar el registro.

    - El titulo no puede estar vacío y debe ser sanitizado.
- **Métodos Principales:**
    - subirArchivo(file, ruta): Guarda físicamente y crea registro.
    - obtenerRutaSegura(): Retorna el archivo validando permisos.
    - softDelete(): Oculta el documento sin borrar el archivo físico por temas de auditoría.
- **Relaciones:**
    - Pertenece a una Categoria (BelongsTo)
    - Pertenece a un Usuario (BelongsTo)

Model: Categoria

- **Campos:**
    - id: UUID
    - nombre: Varchar(100) (Único)
    - descripcion: Text
    - parent_id: UUID (Nullable, para subcarpetas - Self-referencing)
- **Validaciones:**
    - El nombre debe ser único para evitar duplicidad de carpetas.
- **Relaciones:**
    - Tiene muchos Documentos (HasMany)
    - Pertenece a una Categoria padre (BelongsTo)

Controladores / Rutas Principales:

Controller: DocumentoController

- **GET /documentos**
    - **Operación:** Documento.findAll() filtrando por categoria_id si se provee.
    - **Respuesta:** Renderiza la vista documentos/index con el listado de archivos.
- **POST /documentos/upload**
    - **Recibe:** Form-data con el archivo físico, título y categoría.
    - **Validación:** Middleware verifica sesión de usuario, tamaño máximo y tipo de archivo.
    - **Operación:** Documento.subirArchivo().
    - **Respuesta:** 302 Redirect a /documentos con mensaje de éxito (Flash message) o 400 Bad Request si falla la validación.
- **GET /documentos/descargar/:id**
    - **Operación:** Documento.findById(). Verifica si el usuario actual tiene permisos.
    - **Respuesta:** Retorna un stream del archivo físico (Download response) o 403 Forbidden.

Vistas / Respuestas (Views): Al ser un Monolito tradicional, las respuestas principales no son JSON puro (salvo para pequeñas integraciones asíncronas), sino plantillas renderizadas en el servidor.

- **Estructura:** Uso de un layout maestro (Header, Sidebar, Footer) heredado del sistema MiUNE 2.0.
- **Manejo de estados:**
    - **200 OK:** Renderizado exitoso de la página HTML con los datos inyectados.
    - **302 Found:** Redirecciones tras acciones de escritura (Crear/Editar/Eliminar) usando el patrón Post/Redirect/Get para evitar reenvío de formularios.
    - **403 / 404 / 500:** Plantillas HTML genéricas de error del sistema centralizadas.

6. Diagrama de Arquitectura (Descripción Textual C4 Model - Nivel 2)

- **Usuario (Actor):** Personal de la Coordinación de Sistemas accediendo vía Navegador Web.
- **Capa de Presentación (Frontend):** Vistas .phtml renderizadas en el servidor, utilizando HTML5, CSS y JavaScript estándar para interacciones en la interfaz.

- **Servidor Web:** Apache HTTP Server alojado en Linux Ubuntu, encargado de recibir las peticiones HTTP/HTTPS y pasarlas al intérprete PHP.

- **Aplicación Monolítica (MiUNE 2.0 - ZF1):**
    - **Zend_Controller (Controlador):** Intercepta la petición, verifica la sesión del usuario (Autenticación/Autorización) y llama al modelo correspondiente.

    - **Zend_Db_Table (Modelo):** Ejecuta la lógica de negocio y se comunica con la base de datos.

- **Capa de Almacenamiento:**
    - **Base de Datos Relacional:** Servidor PostgreSQL que almacena los metadatos de los documentos, categorías y registros de auditoría.
    - **File System (Ubuntu):** Directorio protegido en el servidor donde se guardan físicamente los archivos subidos (PDFs, Excels), inaccesible desde la web sin pasar por el controlador.

7. Stack Tecnológico Seleccionado

Backend:

- **Lenguaje:** PHP 5.x / 7.x (Dependiendo de la versión soportada por el servidor actual de MiUNE). Justificación: Requisito estricto de compatibilidad con el ecosistema existente.

- **Framework:** Zend Framework 1. Justificación: Aunque es una tecnología heredada, su uso es obligatorio para garantizar que el nuevo módulo comparta la misma sesión, utilidades de seguridad y base de código de MiUNE 2.0.

- **ORM / Acceso a Datos:** Zend_Db (Componente nativo de ZF1). Justificación: Permite abstraer consultas SQL manteniendo la coherencia con el resto del sistema.

- **Base de Datos:** PostgreSQL. Justificación: Motor robusto, de código abierto y estándar actual en el proyecto, excelente para manejar relaciones complejas y grandes volúmenes de datos transaccionales.

Frontend:

- **Lenguajes:** HTML5, CSS3, y JavaScript (Vanilla o jQuery). Justificación: Al ser un monolito donde el servidor renderiza las vistas, se requiere manipulación del DOM directa sin la sobrecarga de un framework reactivo complejo.

- **Framework CSS:** Bootstrap (o el framework CSS que ya utilice MiUNE). Justificación: Mantiene la consistencia visual de la interfaz de usuario con el resto de los módulos.

Infraestructura y DevOps:

- **Sistema Operativo:** Linux Ubuntu. Justificación: Entorno de servidor estable, seguro y estándar en la universidad.
- **Control de Versiones:** Git + GitLab/GitHub. Justificación: Permite el trabajo colaborativo y control histórico del código.
- **Servidor Web:** Apache. Justificación: El servidor web por excelencia para despliegues tradicionales de PHP.

8. Plan de Implementación por Fases

- **Fase 1 - MVP (Producto Mínimo Viable):**
    - Diseño y creación de tablas en PostgreSQL (Documentos y Categorías).
    - Implementación de CRUD básico para Categorías (Carpetas).

    - Subida y descarga de archivos con validación básica de tipos (PDF, DOCX) y peso.
    - Integración con el control de acceso de MiUNE 2.0 (solo usuarios autorizados pueden entrar al módulo).
    - *Tiempo estimado:* 3 - 4 semanas.
- **Fase 2 - Features Intermedias:**
    - Motor de búsqueda avanzado (filtrado por nombre, fecha, autor y categoría).
    - Estructura jerárquica de carpetas (Subcategorías).
    - Paginación de resultados para optimizar la carga.
    - *Priorización:* Alta, mejora drásticamente la usabilidad.
- **Fase 3 - Features Avanzadas:**
    - Registro de auditoría (logs) de quién visualiza o modifica cada documento.
    - Previsualización de documentos PDF directamente en el navegador sin descargar.
    - Panel de estadísticas de uso (documentos más descargados, almacenamiento utilizado).