



**UNIVERSIDAD DE
CÓRDOBA**



LICENCIATURA EN
INFORMÁTICA

Acreditada de Alta Calidad
MEN Res. 10710 25/05/17

Documento
técnico
para
proyectos
de Diseño
de
Software

Documento de Propuesta de Diseño de Software I, II y III.

Componente Gestión de REDA – Fase I

Keimer Enrique Muñoz Mora

Luis Carlos Suárez Bravo

José Isaac Chantak Zabaleta

Natalia Sofia Cochero Paternina

DISEÑO DE SOFTWARE III

Keimer Enrique Muñoz Mora

Natalia Cochero Paternina

Pier Paolo Chimá Durango

Luis Carlos Suárez Bravo

Rosa Elvira Herrera Peña

Sebastián López Sotelo



ENLACE DEL REPOSITORIO: https://github.com/Grupo-Investigacion-Bimadino/paul_ova_manager.git

Breve reseña

En muchos entornos educativos, los recursos educativos están dispersos en diferentes plataformas y formatos, lo que dificulta su acceso y gestión. El componente propuesto es un sistema integral diseñado para la recopilación, gestión y almacenamiento de Recursos Educativos Digitales Abiertos (REDA), abarcando una amplia variedad de formatos de contenidos. Ofrecerá una interfaz de usuario intuitiva y accesible desde múltiples dispositivos, junto con un sistema de búsqueda avanzada para facilitar la localización de recursos relevantes. Además, permitirá la personalización según las necesidades de los usuarios y organizaciones educativas, integrándose de manera fluida con sistemas de gestión del aprendizaje existentes y priorizando la seguridad de los datos mediante medidas como la encriptación y el control de acceso. El componente propuesto proporcionará una solución integral y escalable para la recopilación y gestión de REDA, ofreciendo una experiencia de aprendizaje enriquecida y adaptada a las necesidades individuales y organizativas.

ETAPA 1 DISEÑO DE LA APLICACIÓN Y ANÁLISIS DE REQUISITOS	6
INTRODUCCIÓN	6
PROPÓSITO DEL DOCUMENTO	6
ALCANCE DEL PROYECTO.....	7
DEFINICIONES Y ACRÓNIMOS	7
DESCRIPCIÓN GENERAL	8
OBJETIVOS DEL SISTEMA	11
FUNCIONALIDAD GENERAL	12
USUARIOS DEL SISTEMA	12
RESTRICCIONES	12
REQUISITOS FUNCIONALES	13
CASOS DE USO.....	13
DESCRIPCIÓN DETALLADA DE CADA CASO DE USO.....	14
DIAGRAMAS DE FLUJO DE CASOS DE USO	14
PRIORIDAD DE REQUISITOS.....	24
REQUISITOS NO FUNCIONALES	30
REQUISITOS DE DESEMPEÑO.....	30
REQUISITOS DE SEGURIDAD.....	30
REQUISITOS DE USABILIDAD	30
REQUISITOS DE ESCALABILIDAD.....	30
MODELADO E/R.....	30
DIAGRAMA DE ENTIDAD-RELACIÓN	34
DESCRIPCIÓN DE ENTIDADES Y RELACIONES.....	31
REGLAS DE INTEGRIDAD.....	31
ANEXOS (SI ES NECESARIO)	35
DIAGRAMAS ADICIONALES.....	¡ERROR! MARCADOR NO DEFINIDO.
REFERENCIAS	¡ERROR! MARCADOR NO DEFINIDO.

ETAPA 2: PERSISTENCIA DE DATOS CON BACKEND.....	36
INTRODUCCIÓN	36
PROPÓSITO DE LA ETAPA	36
ALCANCE DE LA ETAPA	36
DEFINICIONES Y ACRÓNIMOS	36
DISEÑO DE LA ARQUITECTURA DE BACKEND	36
DESCRIPCIÓN DE LA ARQUITECTURA PROPUESTA	37
COMPONENTES DEL BACKEND.....	37
DIAGRAMAS DE ARQUITECTURA	37
ELECCIÓN DE LA BASE DE DATOS	38
EVALUACIÓN DE OPCIONES (SQL o NoSQL)	39
JUSTIFICACIÓN DE LA ELECCIÓN.....	39
DISEÑO DE ESQUEMA DE BASE DE DATOS	39
IMPLEMENTACIÓN DEL BACKEND.....	41
ELECCIÓN DEL LENGUAJE DE PROGRAMACIÓN.....	43
CREACIÓN DE LA LÓGICA DE NEGOCIO	43
DESARROLLO DE ENDPOINTS Y APIS.....	44
AUTENTICACIÓN Y AUTORIZACIÓN	44
CONEXIÓN A LA BASE DE DATOS	44
CONFIGURACIÓN DE LA CONEXIÓN.....	44
DESARROLLO DE OPERACIONES CRUD.....	45
MANEJO DE TRANSACCIONES	45
PRUEBAS DEL BACKEND	50
DISEÑO DE CASOS DE PRUEBA.....	50
EJECUCIÓN DE PRUEBAS UNITARIAS Y DE INTEGRACIÓN... ¡ERROR! MARCADOR NO DEFINIDO.	
MANEJO DE ERRORES Y EXCEPCIONES ¡ERROR! MARCADOR NO DEFINIDO.	

ETAPA 3: CONSUMO DE DATOS Y DESARROLLO FRONTEND.....	57
INTRODUCCIÓN	57
PROPÓSITO DE LA ETAPA	¡ERROR! MARCADOR NO DEFINIDO.
ALCANCE DE LA ETAPA	¡ERROR! MARCADOR NO DEFINIDO.
DEFINICIONES Y ACRÓNIMOS	¡ERROR! MARCADOR NO DEFINIDO.
CREACIÓN DE LA INTERFAZ DE USUARIO (UI)	57
DISEÑO DE LA INTERFAZ DE USUARIO (UI) CON HTML Y CSS	57
CONSIDERACIONES DE USABILIDAD	¡ERROR! MARCADOR NO DEFINIDO.
MAQUETACIÓN RESPONSIVA	¡ERROR! MARCADOR NO DEFINIDO.
PROGRAMACIÓN FRONTEND CON JAVASCRIPT (JS)	¡ERROR! MARCADOR NO DEFINIDO.
DESARROLLO DE LA LÓGICA DEL FRONTEND	¡ERROR! MARCADOR NO DEFINIDO.
MANEJO DE EVENTOS Y COMPORTAMIENTOS DINÁMICOS	¡ERROR! MARCADOR NO DEFINIDO.
USO DE BIBLIOTECAS Y FRAMEWORKS (SI APLICABLE)	¡ERROR! MARCADOR NO DEFINIDO.
CONSUMO DE DATOS DESDE EL BACKEND .	¡ERROR! MARCADOR NO DEFINIDO.
CONFIGURACIÓN DE CONEXIONES AL BACKEND	¡ERROR! MARCADOR NO DEFINIDO.
OBTENCIÓN Y PRESENTACIÓN DE DATOS	¡ERROR! MARCADOR NO DEFINIDO.
ACTUALIZACIÓN EN TIEMPO REAL (SI APLICABLE).....	¡ERROR! MARCADOR NO DEFINIDO.
INTERACCIÓN USUARIO-INTERFAZ.....	¡ERROR! MARCADOR NO DEFINIDO.
MANEJO DE FORMULARIOS Y VALIDACIÓN DE DATOS	¡ERROR! MARCADOR NO DEFINIDO.
IMPLEMENTACIÓN DE FUNCIONALIDADES INTERACTIVAS .	¡ERROR! MARCADOR NO DEFINIDO.
MEJORAS EN LA EXPERIENCIA DEL USUARIO.....	¡ERROR! MARCADOR NO DEFINIDO.
PRUEBAS Y DEPURACIÓN DEL FRONTEND ..	¡ERROR! MARCADOR NO DEFINIDO.
DISEÑO DE CASOS DE PRUEBA DE FRONTEND	¡ERROR! MARCADOR NO DEFINIDO.
PRUEBAS DE USABILIDAD.....	¡ERROR! MARCADOR NO DEFINIDO.
DEPURACIÓN DE ERRORES Y OPTIMIZACIÓN DEL CÓDIGO .	¡ERROR! MARCADOR NO DEFINIDO.
IMPLEMENTACIÓN DE LA LÓGICA DE NEGOCIO EN EL FRONTEND	¡ERROR! MARCADOR NO DEFINIDO.
MIGRACIÓN DE LA LÓGICA DE NEGOCIO DESDE EL BACKEND (SI NECESARIO)	¡ERROR! MARCADOR NO DEFINIDO.
VALIDACIÓN DE DATOS Y REGLAS DE NEGOCIO EN EL FRONTEND.....	¡ERROR! MARCADOR NO DEFINIDO.
INTEGRACIÓN CON EL BACKEND.....	¡ERROR! MARCADOR NO DEFINIDO.
VERIFICACIÓN DE LA COMUNICACIÓN EFECTIVA CON EL BACKEND.....	¡ERROR! MARCADOR NO DEFINIDO.
PRUEBAS DE INTEGRACIÓN FRONTEND-BACKEND	¡ERROR! MARCADOR NO DEFINIDO.

Etapas 1 Diseño de la Aplicación y Análisis de Requisitos

Introducción

Propósito del Documento

El presente documento tiene como finalidad documentar el proceso de diseño, análisis e implementación de software de tipo educativo, comercial, REDA, componente o módulo de aplicaciones. Se divide en tres etapas para facilitar el entendimiento y aplicación a gran escala en la asignatura de diseño de software.

- Etapa 1 Diseño de la Aplicación y Análisis de Requisitos

Esta etapa cumple la tarea de recoger todas las competencias desarrolladas en todas las áreas de formación del currículo de la licenciatura en Informática y Medios Audiovisuales y ponerlas a prueba en el diseño y análisis de un producto educativo que se base en las teorías de aprendizaje estudiadas, articule las estrategias de enseñanza con uso de TIC y genere innovaciones en educación con productos interactivos que revelen una verdadera naturaleza educativa. Estos productos deben aprovechar las fortalezas adquiridas en las áreas de tecnología e informática, técnicas y herramientas, medios audiovisuales y programación y sistemas, para generar productos software interactivos que permitan a los usuarios disfrutar de lo que aprenden, a su propio ritmo. Todo esto en el marco de un proceso metodológico (metodologías de desarrollo de software como MODESEC, SEMLI, etc.) que aproveche lo aprendido en la línea de gestión y lo enriquezca con elementos de la Ingeniería de Software.

- Etapa 2: Persistencia de Datos con Backend – Servidor

En la etapa 2 se continua con los lineamientos de la etapa 1, para seguir adicionando elementos de diseño e implementación de software, enfocados en el desarrollo de APIs, servidores o microservicios que permitan soportar aplicaciones cliente del software educativo; en este sentido, el curso presenta los conceptos de los sistemas de bases de datos, su diseño lógico, la organización de los sistemas manejadores de bases de datos, los lenguaje de definición de datos y el lenguaje de manipulación de datos SQL y NoSQL; de tal manera que los estudiantes adquieran las competencias para analizar, diseñar y desarrollar aplicaciones para gestionar y almacenar grandes cantidades de datos, mediante el uso de técnicas adecuadas como el diseño y modelo lógico y físico de base datos, manejo de los sistemas de gestión de bases de datos, algebra relacional, dominio del lenguaje SQL como herramienta de consulta, tecnología cliente / servidor; igualmente, se definirán los elementos necesarios para el acceso a dichas bases de datos, como la creación del servidor API, utilizando tecnologías de vanguardia como node.js, express, Nest.js, Spring entre otros; para, finalmente converger en el despliegue de la API utilizando servicios de hospedaje en la nube, preferiblemente gratuitos. También podrá implementar servidores o API's con inteligencia artificial o en su defecto crear una nueva capa que consuma y transforme los

datos obtenidos de la IA. El desarrollo del curso se trabajará por proyectos de trabajo colaborativo que serán evaluados de múltiples maneras, teniendo en cuenta más el proceso que el resultado.

- Etapa 3: Consumo de Datos y Desarrollo Frontend – Cliente

La etapa 3 el estudiante está en capacidad de establecer la mejor elección de herramientas de consumo de datos y técnicas en aras de lograr el mejor producto a nivel de software o hardware acorde a los requerimientos funcionales y no funcionales del problema a solucionar. En este punto el estudiante puede consumir los datos a través de un cliente que puede ser una aplicación de celular, una aplicación de escritorio, una página web, IoT (internet de las cosas) o incluso, artefactos tecnológicos. El diseño gráfico es de los requisitos esenciales en la capa de presentación, por lo tanto, se requieren los cursos de diseño gráfico vistos previamente. Los elementos anteriores nos permiten elegir el paradigma y tecnología para desarrollar nuestras aplicaciones, teniendo en cuenta que podríamos desarrollar aplicaciones de tipo cliente.

Alcance del Proyecto

El componente propuesto es un sistema integral diseñado específicamente para la gestión eficiente de Recursos Educativos Digitales Abiertos (REDA). Este sistema permite administrar los REDA teniendo en cuenta sus características principales, como la página de inicio, los contenidos y la evaluación. Además, incluye una interfaz intuitiva con búsqueda avanzada, segura y se integra fácilmente con los sistemas de gestión del aprendizaje existente. En el siguiente apartado, se detallarán las características de la presente versión y se sugerirán algunas para futuras actualizaciones.

- Subir un REDA.
- Visualizar un REDA.
- Actualizar un REDA.
- Eliminar un REDA.
- Buscar un REDA.
- Exportar REDA.
- Valorar REDA.

Funcionalidades Futuras:

- Interoperabilidad con Herramientas de Autoría
- Importar REDA externo.
- Recomendación Personalizada de Contenidos.
- Recolección de datos XAPI.
- Análisis de datos de Uso con XAPI.

Definiciones y Acrónimos

API: Interfaz de Programación de Aplicaciones (Application Programming Interface).

Es un conjunto de reglas y protocolos, permite que diferentes aplicaciones se comuniquen entre sí y compartan datos o funcionalidades.

DBMS: Sistema de Gestión de Bases de Datos (Database Management System).

Es un software que facilita la creación, manipulación y administración de bases de datos, permitiendo almacenar, organizar y recuperar información de manera eficiente.

SQL: Lenguaje de Consulta Estructurada (Structured Query Language).

Es un lenguaje estándar utilizado para interactuar con bases de datos relacionales. Permite realizar consultas, inserciones, actualizaciones y eliminaciones de datos, así como la creación y modificación de esquemas de base de datos.

HTTP: Protocolo de Transferencia de Hipertexto (Hypertext Transfer Protocol).

Es un protocolo de comunicación utilizado para la transferencia de información en la World Wide Web. Define cómo se transmiten los mensajes y cómo navegadores y servidores web interactúan para solicitar y enviar recursos, páginas web, imágenes y otros archivos.

REST: Transferencia de Estado Representacional (Representational State Transfer).

Es un estilo arquitectónico para el diseño de sistemas de software distribuido, basado en la representación de recursos a través de URLs y la manipulación de dichos recursos utilizando operaciones estándar de HTTP, como GET, POST, PUT y DELETE.

JSON: Notación de Objetos de JavaScript (JavaScript Object Notation).

Es un formato de intercambio de datos ligero y legible por humanos, basado en la sintaxis de los objetos de JavaScript. Se utiliza comúnmente para transmitir datos estructurados entre un servidor y un cliente en aplicaciones web y es independiente del lenguaje de programación.

JWT: Token de Web JSON (JSON Web Token).

Es un estándar abierto (RFC 7519) que define un formato compacto y seguro para la transferencia de información entre dos partes, generalmente utilizado en autenticación y autorización en aplicaciones web. Los JWT están representados como cadenas JSON y son firmados digitalmente para verificar su autenticidad y asegurar la integridad de los datos.

CRUD: Crear, Leer, Actualizar y Borrar (Create, Read, Update, Delete).

Es un conjunto de operaciones básicas utilizadas en sistemas de gestión de bases de datos y aplicaciones web para manipular datos. Estas operaciones son fundamentales para el mantenimiento de la información: crear nuevos registros (Create), leer datos existentes (Read), actualizar registros existentes (Update) y eliminar registros (Delete).

ORM: Mapeo Objeto-Relacional (Object-Relational Mapping).

Es una técnica de programación que permite convertir datos entre el modelo de objetos utilizado en un lenguaje de programación orientado a objetos y el modelo relacional utilizado en una base de datos relacional.

MVC: Modelo-Vista-Controlador (Model-View-Controller).

Es un patrón de diseño de software que divide una aplicación en tres componentes principales: el Modelo, que representa los datos y la lógica de la aplicación; la Vista, que se encarga de mostrar la interfaz de usuario y recibir las entradas del usuario; y el Controlador, que actúa como intermediario entre el Modelo y la Vista, gestionando las interacciones del usuario y actualizando el Modelo en consecuencia.

API RESTful: API que sigue los principios de REST.

API que sigue los principios de REST, utilizando URLs para identificar recursos y métodos HTTP estándar para manipularlos. Es flexible y fácilmente entendida por los desarrolladores.

CI/CD: Integración Continua / Entrega Continua (Continuous Integration / Continuous Delivery).

Es un enfoque de desarrollo que automatiza los procesos de construcción, pruebas y despliegue de código para entregar cambios de manera rápida y segura. La Integración Continua se concentra en fusionar y probar código frecuentemente, mientras que la Entrega Continua automatiza el despliegue regular y confiable del código en entornos de producción.

SaaS: Software como Servicio (Software as a Service).

Es un modelo de distribución de software donde las aplicaciones son alojadas por un proveedor de servicios y accesibles a través de internet, generalmente mediante suscripciones.

SSL/TLS: Capa de sockets seguros/Seguridad de la Capa de Transporte (Secure Sockets Layer/Transport Layer Security).

Son protocolos de seguridad que cifran las comunicaciones en internet para proteger la privacidad y la integridad de los datos transmitidos entre clientes y servidores.

HTML: Lenguaje de Marcado de Hipertexto (Hypertext Markup Language).

Es el lenguaje estándar utilizado para crear páginas web, permitiendo la estructuración y presentación de contenido en línea.

CSS: Hojas de Estilo en Cascada (Cascading Style Sheets).

Es un lenguaje utilizado para definir el estilo y la presentación de documentos HTML, permitiendo controlar el diseño, la tipografía, los colores y otros aspectos visuales de una página web.

JS: JavaScript.

Es un lenguaje de programación usado para agregar interactividad y funcionalidad dinámica a páginas web.

DOM: Modelo de Objeto del Documento (Document Object Model).

Es una interfaz de programación que representa la estructura de un documento HTML como un árbol de objetos, permitiendo a los programas acceder y manipular el contenido, la estructura y el estilo de una página web de manera dinámico.

UI: Interfaz de Usuario (User Interface).

Es el medio por el que los usuarios interactúan con un dispositivo, aplicación o sistema informático, permitiéndoles actuar y recibir retroalimentación de forma intuitiva y eficiente.

UX: Experiencia del Usuario (User Experience).

Es la impresión general que tiene un usuario al interactuar con un producto o servicio, incluyendo aspectos como la usabilidad, la accesibilidad y la satisfacción.

SPA: Aplicación de Página Única (Single Page Application).

Es una aplicación web que carga una sola página HTML y actualiza el contenido de manera dinámica utilizando JavaScript, proporcionando una experiencia de usuario fluida y rápida.

AJAX: Asíncrono JavaScript y XML (Asynchronous JavaScript and XML).

Es una técnica de desarrollo web que permite realizar solicitudes al servidor de forma asíncrona, sin necesidad de recargar la página completa, lo que mejora la velocidad y la interactividad de las aplicaciones web.

CMS: Sistema de Gestión de Contenido (Content Management System).

Es una plataforma que facilita la creación, edición, gestión y publicación de contenido digital, como páginas web, blogs o tiendas en línea, sin necesidad de conocimientos técnicos avanzados.

CDN: Red de Distribución de Contenido (Content Delivery Network).

Es una red de servidores distribuidos geográficamente que almacenan copias de contenido web estático, como imágenes, archivos de estilo y scripts, para entregarlos a los usuarios de manera más rápida y eficiente.

SEO: Optimización de Motores de Búsqueda (Search Engine Optimization).

Es el proceso de mejorar la visibilidad y la clasificación de un sitio web en los resultados de búsqueda orgánica, mediante técnicas como la optimización del contenido, la estructura del sitio y la construcción de enlaces.

IDE: Entorno de Desarrollo Integrado (Integrated Development Environment).

Es un software que proporciona herramientas integradas para escribir, depurar y compilar código de programación en un solo lugar, facilitando el desarrollo de software.

CLI: Interfaz de Línea de Comandos (Command Line Interface).

Es una interfaz de usuario que permite interactuar con un sistema informático mediante comandos de texto, en lugar de utilizar una interfaz gráfica.

PWA: Aplicación Web Progresiva (Progressive Web App).

Es una aplicación web que utiliza tecnologías modernas para proporcionar una experiencia similar a la de una aplicación nativa en dispositivos móviles, incluyendo funciones como el acceso offline, las notificaciones push y la instalación en la pantalla de inicio.

REDA: Recurso Educativo Digital Abierto (Open Educational Resources (OER)

Los recursos educativos digitales abiertos (REDA) son materiales digitalizados que se ofrecen de forma libre y abierta a educadores, estudiantes y autodidactas para ser utilizados con fines educativos y de aprendizaje. La principal característica de los REDA es que, al ser abiertos, cualquier persona puede acceder, descargar, adaptar y redistribuir estos contenidos de manera gratuita a través de internet

XAPI: Experiencia de Aprendizaje Electrónico (Experience API)

Es un estándar de tecnología de aprendizaje electrónico que permite el seguimiento y la recopilación de datos sobre las actividades en línea.

GESTIÓN DEL REDA: (REDA Management)

La gestión del REDA se refiere al conjunto de procesos y estrategias utilizados para administrar y Evaluar organizar los recursos educativos digitales conocidos como Recursos Educativos Digitales de Aprendizaje. Esto incluye la creación, distribución, actualización, y seguimiento de los REDA para garantizar su efectividad en el proceso de enseñanza y aprendizaje.

VALORACIÓN DE REDA: (REDA Assessment)

La valoración de un REDA consiste en asignar una puntuación entre 1 y 5 para evaluar su calidad, efectividad y relevancia en el contexto educativo. Cuanto mayor sea la valoración, mayor será la probabilidad de que el REDA sea recomendado en un entorno de sugerencias.

Descripción General

Objetivos del Sistema

El sistema tiene como objetivos principales facilitar la recopilación, gestión y almacenamiento de Recursos Educativos Digitales Abiertos (REDA) desde diversas fuentes y formatos, así como proporcionar una interfaz de usuario intuitiva y accesible para una fácil navegación y recuperación de recursos educativos. Además, busca permitir la personalización y adaptabilidad de los REDA según las necesidades específicas de los usuarios y las organizaciones educativas,

integrarse fluidamente con sistemas de gestión del aprendizaje existentes y priorizar la seguridad y privacidad de los datos del usuario y los contenidos educativos almacenados en el sistema.

Funcionalidad General

Subir un REDA: Permite a los usuarios cargar nuevos Recursos Educativos Digitales Abiertos al sistema. Esta funcionalidad admite múltiples formatos de archivo y ofrece una interfaz amigable que guía al usuario a través del proceso de carga, asegurando que todos los elementos del REDA, como la página de inicio, los contenidos y las evaluaciones, se registren correctamente.

Visualizar un REDA: Facilita la visualización de los REDA dentro del sistema. Los usuarios pueden navegar por los contenidos de manera intuitiva, accediendo fácilmente a todas las secciones del REDA, como la introducción, los módulos de contenido y las evaluaciones. Esta funcionalidad asegura una experiencia de usuario fluida y coherente.

Actualizar un REDA: Permite a los usuarios modificar o actualizar el contenido de un REDA existente. Los cambios pueden incluir la actualización de textos, imágenes, videos y otros recursos multimedia, así como ajustes en las evaluaciones. Esta funcionalidad garantiza que los REDA se mantengan actuales y relevantes.

Eliminar un REDA: Ofrece a los usuarios la capacidad de eliminar REDA que ya no son necesarios o relevantes. Esta funcionalidad incluye medidas de seguridad, como confirmaciones de eliminación y la posibilidad de recuperar REDA eliminados recientemente, para prevenir la pérdida accidental de información.

Buscar un REDA: Proporciona herramientas de búsqueda avanzada que permiten a los usuarios encontrar REDA específicos de manera rápida y eficiente. Los criterios de búsqueda pueden incluir palabras clave, etiquetas, fechas de creación, y otros metadatos asociados con los REDA, asegurando que los usuarios puedan localizar fácilmente el contenido que necesitan.

Exportar REDA: Permite a los usuarios exportar REDA en varios formatos, como PDF, SCORM, y otros estándares compatibles con sistemas de gestión del aprendizaje. Esta funcionalidad facilita la distribución y el uso de los REDA en diferentes plataformas y entornos de aprendizaje.

Valorar REDA: Facilita la evaluación de los REDA por parte de los usuarios. Los usuarios pueden asignar calificaciones y proporcionar comentarios sobre la calidad y la utilidad de los REDA. Esta retroalimentación se utiliza para mejorar continuamente los contenidos y la experiencia de aprendizaje.

Usuarios del Sistema

Los siguientes usuarios pueden interactuar con la pizarra dependiendo de las funcionalidades.

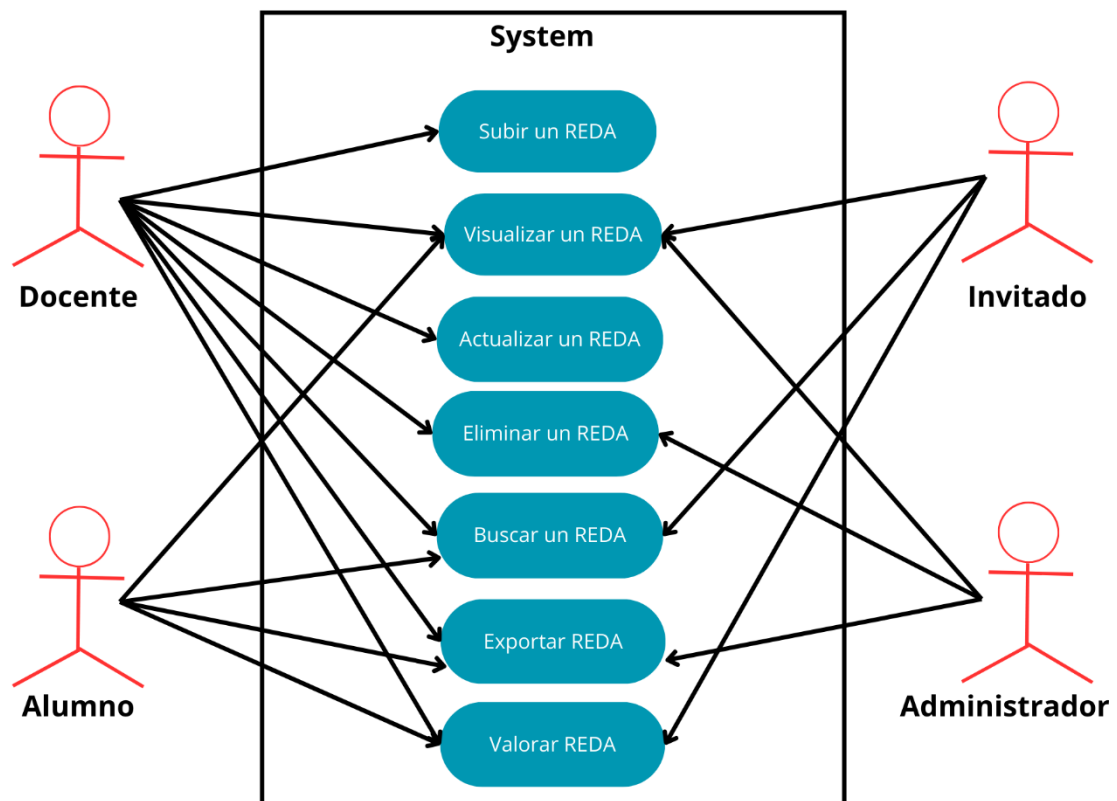
Funcionalidad	Administradores	Docente	Alumno	Invitado
Subir REDA		✓		

Visualizar REDA	✓	✓	✓	✓
Actualizar REDA		✓		
Eliminar REDA	✓	✓		
Buscar REDA		✓	✓	✓
Exportar REDA	✓	✓	✓	
Valorar REDA		✓	✓	✓

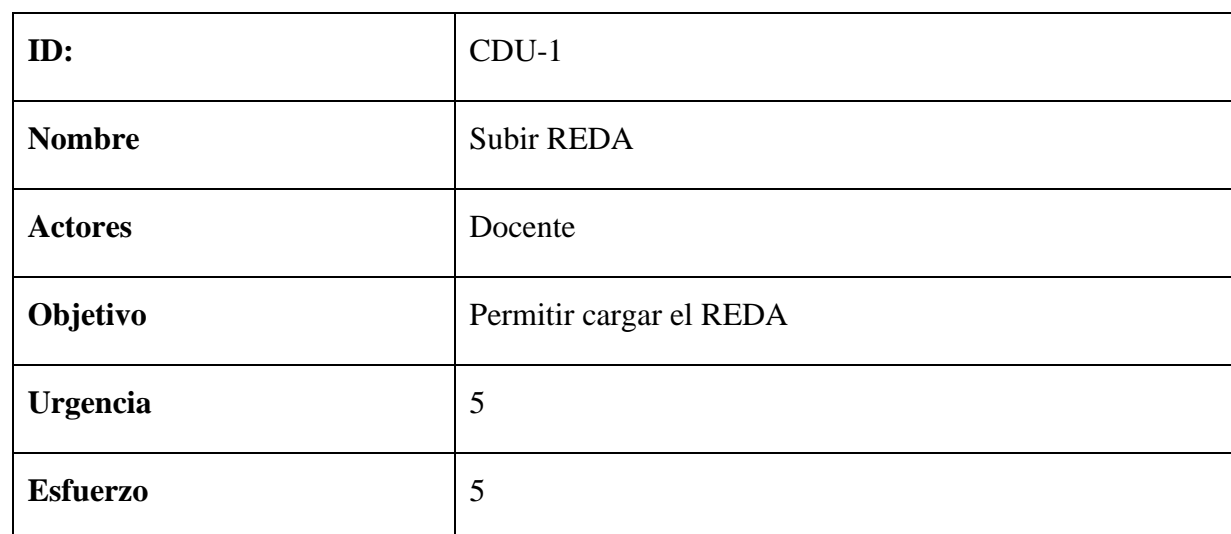
Restricciones

Requisitos Funcionales

Casos de Uso

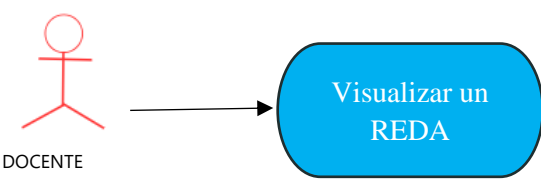


CASO No. 1 Subir REDA



Precondiciones	Debe haber opciones para subir el REDA	
Flujo normal	Docente	Sistema
	Selecciona subir un REDA	
		Retorna opciones de subir REDA
	Selecciona REDA a Subir	
		Carga el REDA
	Llena los campos necesarios (descripción) del REDA	
		Registra la subida del REDA
		Mensaje REDA Guardado con Éxito
Flujo alternativo 1	Selecciona subir un REDA	
		Retorna opciones de subir REDA
	Selecciona REDA a Subir	
		Carga el REDA
	Llena los campos necesarios (descripción) del REDA	
		No guarda la actividad por falta de conexión
	.	Retorna un borrador del REDA

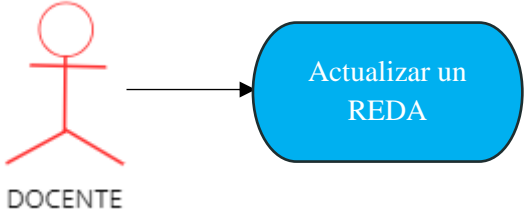
CASO No. 2 Visualizar REDA

 <p>DOCENTE ALUMNO ADMINISTRADOR INVITADO</p> <p>Versión: 1</p> <p>Urgencia: 5</p> <p>Esfuerzo: 3</p> <p>SO: Seleccionar REDA RO: Renderizar REDA VD: Ver Detalles</p>	<p>1. Visualizar REDA</p> <p>Flujo: Visualizar REDA</p> <p>Prueba: Poder Visualizar un REDA</p> <p>a – Visualizar REDA</p> <p>Flujo: SO, RO, VD.</p>
---	---

ID:	CDU-2	
Nombre	Visualizar REDA	
Actores	Docente, Alumno, Administrador, Invitado.	
Objetivo	Permitir visualizar el REDA	
Urgencia	5	
Esfuerzo	3	
Precondiciones	El debe estar disponible en el sistema	
Flujo normal	Docente	Sistema
	Selecciona visualizar un REDA	
		Muestra lista de REDA disponibles
	Selecciona el REDA deseado	
		Carga y muestra el contenido del REDA
	Selecciona Ver Detalles	
		Muestra los detalles del REDA
Flujo alternativo 1	Selecciona visualizar REDA	
		Muestra mensaje de error si no hay conexión

	Usuario reintenta la acción	
--	-----------------------------	--

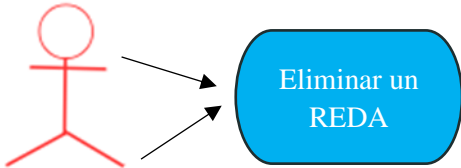
CASO No. 3 Actualizar REDA

 <p>DOCENTE</p> <p>Actualizar un REDA</p> <p>Versión: 1 Urgencia: 4 Esfuerzo: 4</p> <p>AO: Actualizar REDA SO: Seleccionar REDA AC: Actualizar Campos GC: Guardar Cambios</p>	<p>1. Actualizar REDA</p> <p>Flujo: Actualizar REDA</p> <p>Prueba: Cargar Actualización</p> <p>a – Actualizar REDA</p> <p>Flujo: AO, SO, AC, GC.</p>
--	--

ID:	CDU-3	
Nombre	Actualizar REDA	
Actores	Docente	
Objetivo	Permitir actualizar la información del REDA	
Urgencia	4	
Esfuerzo	4	
Precondiciones	El REDA debe existir en el sistema	
Flujo normal	Docente	Sistema
	Actualizar REDA	
		Muestra lista de REDA disponibles
	Selecciona el REDA a actualizar	
		Carga la información actual del REDA

	Modifica la información necesaria	
		Guarda los cambios realizados
		Mensaje de REDA actualizado con Éxito.
Flujo alternativo 1	Actualizar REDA	
		Muestra mensaje de error si no hay conexión
	Docente reintentla la acción	

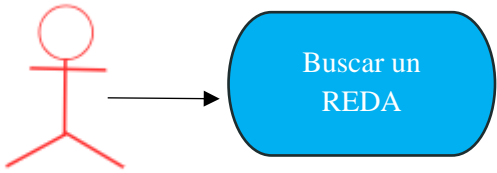
CASO No. 4 Eliminar REDA

 <p>Docente Administrador</p> <p>Eliminar un REDA</p> <p>Versión: 1</p> <p>Urgencia: 3</p> <p>Esfuerzo: 3</p> <p>SO: Seleccionar REDA</p> <p>SE: Seleccionar Eliminar</p> <p>EO: Eliminar REDA</p>	<p>1. Eliminar REDA</p> <p>Flujo: Eliminar REDA</p> <p>Prueba: Borrar REDA de la pantalla de visualización</p> <p>a – Eliminar REDA</p> <p>Flujo: SO, SE, EO.</p>
--	---

ID:	CDU-4
Nombre	Eliminar REDA
Actores	Docente, Administrador
Objetivo	Permitir eliminar un REDA del sistema
Urgencia	3

Esfuerzo	3	
Precondiciones	El REDA debe existir en el sistema	
Flujo normal	Docente	Sistema
	Selecciona eliminar REDA	
		Muestra lista de REDA disponibles
	Selecciona el REDA a eliminar	
		Solicita confirmación de eliminación
	Docente confirma eliminación	
		Confirma la acción
		Mensaje de REDA eliminado correctamente
Flujo alternativo 1	Selecciona eliminar REDA	
		Muestra la lista de REDA disponible
	Selecciona el REDA a eliminar	
		Muestra mensaje de confirmación
	Docente presiona cancelar	
		Cancela la acción
		Mensaje de REDA no eliminado

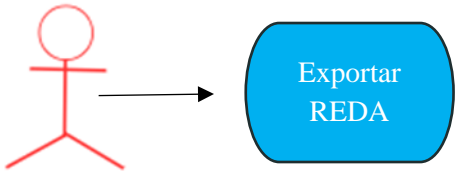
CASO No. 5 Buscar un REDA

 <p>DOCENTE ALUMNO INVITADO</p>	<p>1. Buscar REDA</p> <p>Flujo: Buscar REDA</p> <p>Prueba: Se debe poder buscar REDA relacionado al Criterio que se introdujo</p>
--	---

Versión: 1 Urgencia: 4 Esfuerzo: 3 CB: Criterios de Búsqueda BO: Buscar REDA	a – Buscar REDA Flujo: CB, BO.
--	-----------------------------------

ID:	CDU-5	
Nombre	Buscar REDA	
Actores	Docente, Estudiante, Invitado	
Objetivo	Permitir buscar REDAs en el sistema	
Urgencia	4	
Esfuerzo	3	
Precondiciones	El usuario debe estar autenticado	
Flujo normal	Usuario	Sistema
	Usuario ingresa criterios de búsqueda	
		Retorna resultados de la búsqueda
Flujo alternativo 1	Usuario ingresa criterios de búsqueda	
		Mensaje de error si no hay conexión

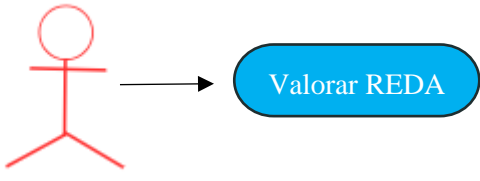
CASO No. 6 Exportar REDA

 <p>DOCENTE ALUMNO ADMINISTRADOR</p> <p>Versión: 1 Urgencia: 3 Esfuerzo: 4</p> <p>EO: Exportar REDA</p>	<p>1. Exportar REDA</p> <p>Flujo: Exportar REDA</p> <p>Prueba: Exporta un REDA a un formato descargable</p> <p>a – Exportar REDA</p> <p>Flujo: EO</p>
--	--

ID:	CDU-6	
Nombre	Exportar REDA	
Actores	Docente	
Objetivo	Permitir exportar un REDA a un formato descargable	
Urgencia	3	
Esfuerzo	4	
Precondiciones	El REDA debe existir en el sistema	
Flujo normal	Docente	Sistema
	Selecciona Exportar REDA	

		Muestra lista de REDA disponibles
	Selecciona el REDA a exportar	
		Muestra opciones de formato de exportación
	Selecciona el formato deseado	
		Procesa y genera el archivo exportado
		Mensaje REDA exportado correctamente
Flujo alternativo 1	Selecciona Exportar REDA	
		Muestra mensaje de error si no hay conexión
	Docente reintenta la acción	
Control de cambios		

CASO No. 7 Valorar REDA

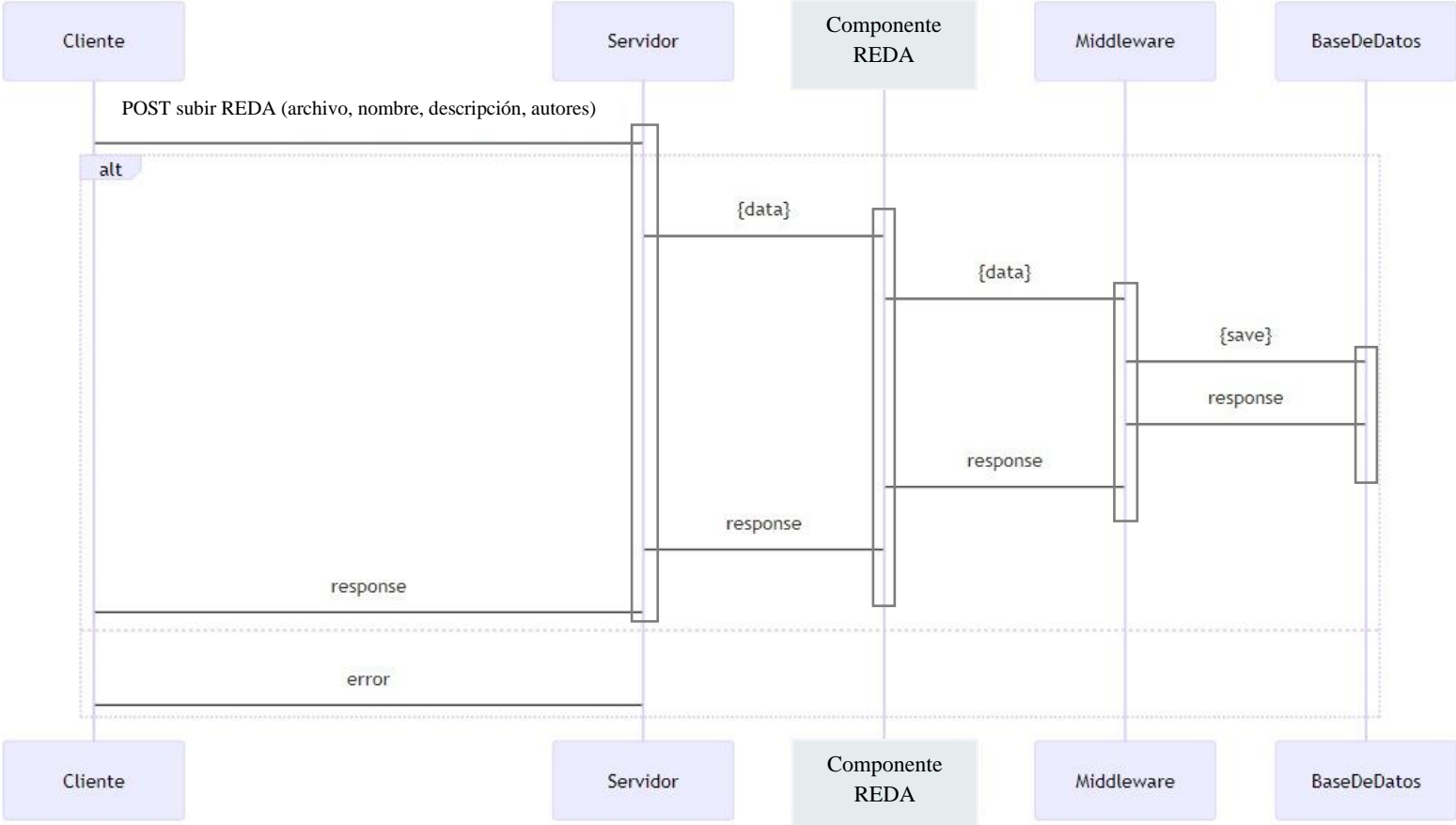
 <p>DOCENTE ALUMNO INVITADO</p> <p>Versión: 1 Urgencia: 2 Esfuerzo: 2</p> <p>VO: Valorar REDA</p>	<p>1. Valorar REDA</p> <p>Flujo: Valorar REDA</p> <p>Prueba: Puntuar ova de acuerdo con la valoración que se decida.</p> <p>a – Valorar REDA</p> <p>Flujo: VO</p>
--	---

ID:	CDU-7	
Nombre	Valorar REDA	
Actores	Docente, Estudiante, Invitado	
Objetivo	Permitir valorar un REDA dentro del sistema	
Urgencia	2	
Esfuerzo	2	
Precondiciones	El REDA debe estar disponible en el sistema	
Flujo normal	Docente	Sistema
	Usuario selecciona el REDA a valorar	
		Muestra opciones de valoración (estrellas, comentarios, etc.)
	Usuario selecciona una valoración y/o escribe un comentario	
		Guarda la valoración
		Mensaje de valoración guardada correctamente
Flujo alternativo 1	Usuario selecciona el REDA a valorar	

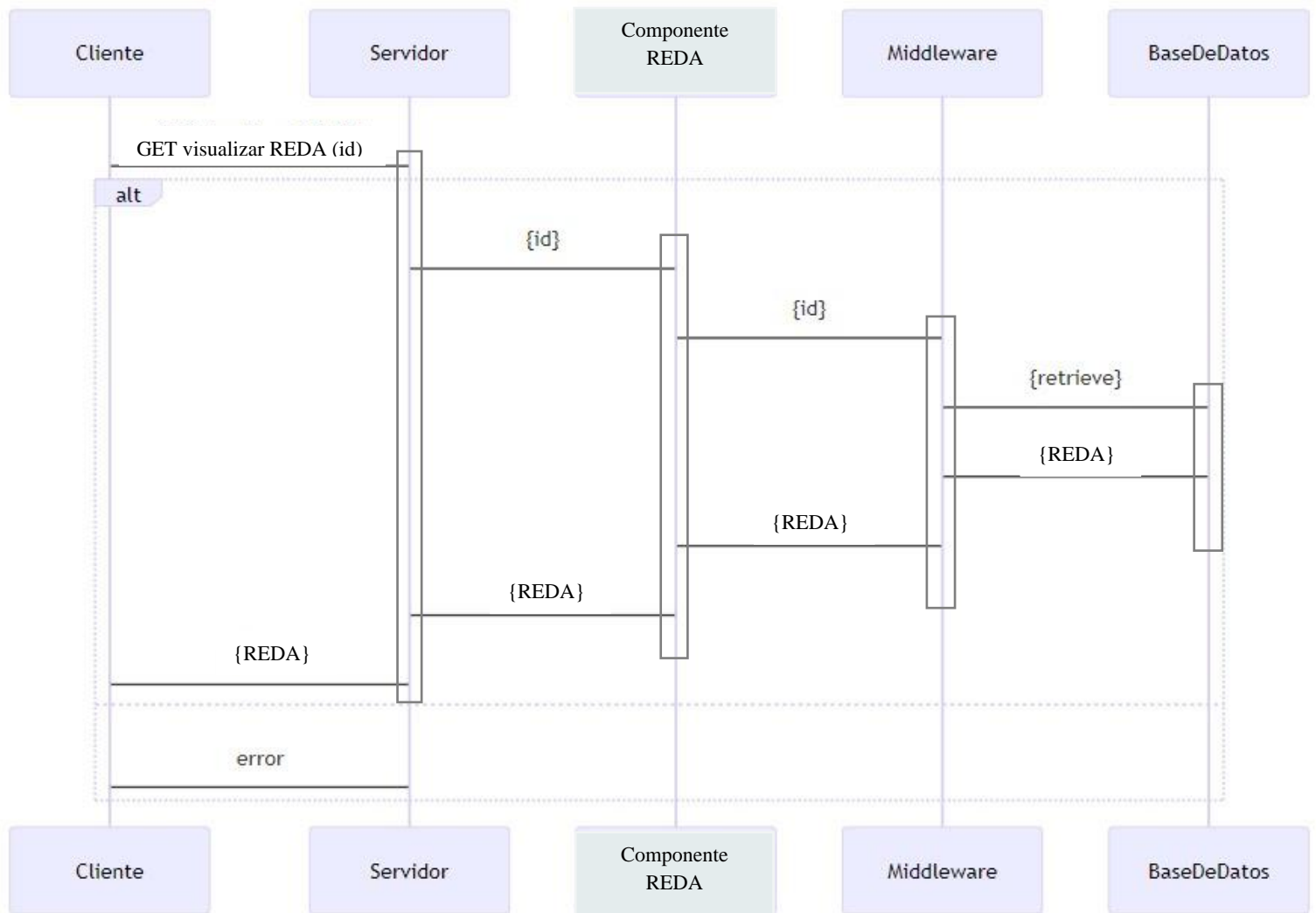
		Muestra mensaje de error si no hay conexión
	Usuario reintenta la acción	

Diagramas de Secuencia

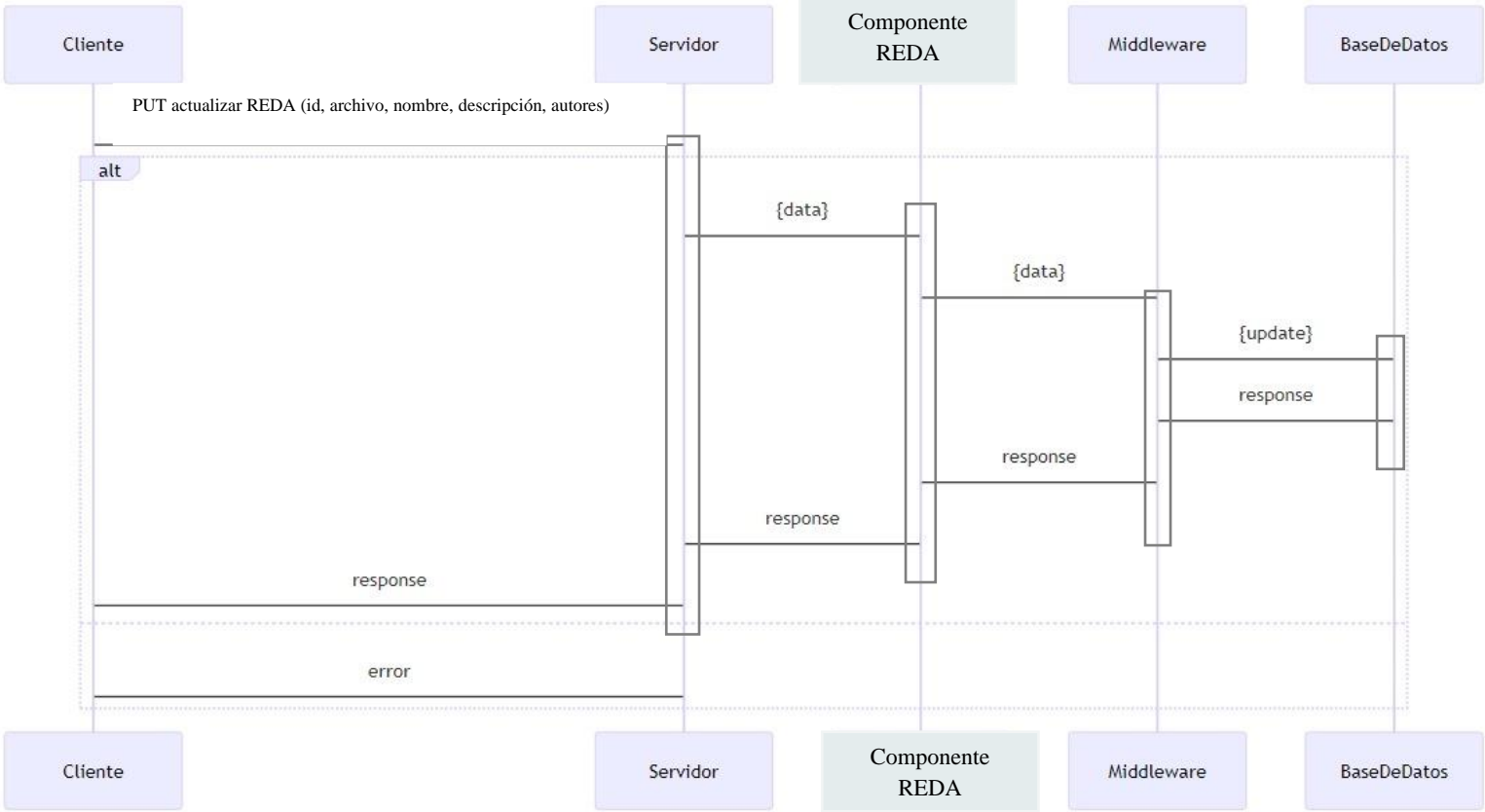
Subir REDA



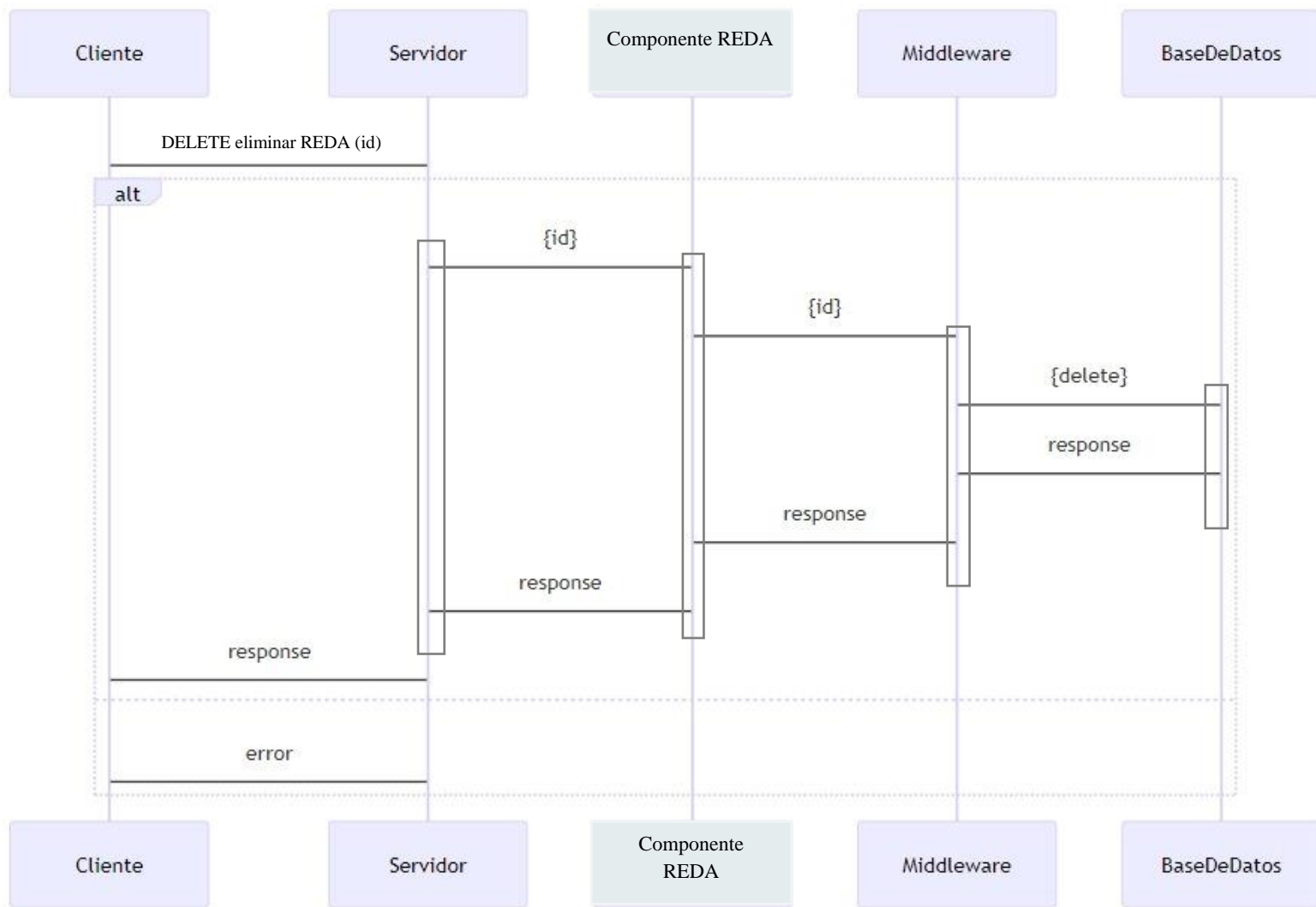
Visualizar REDA



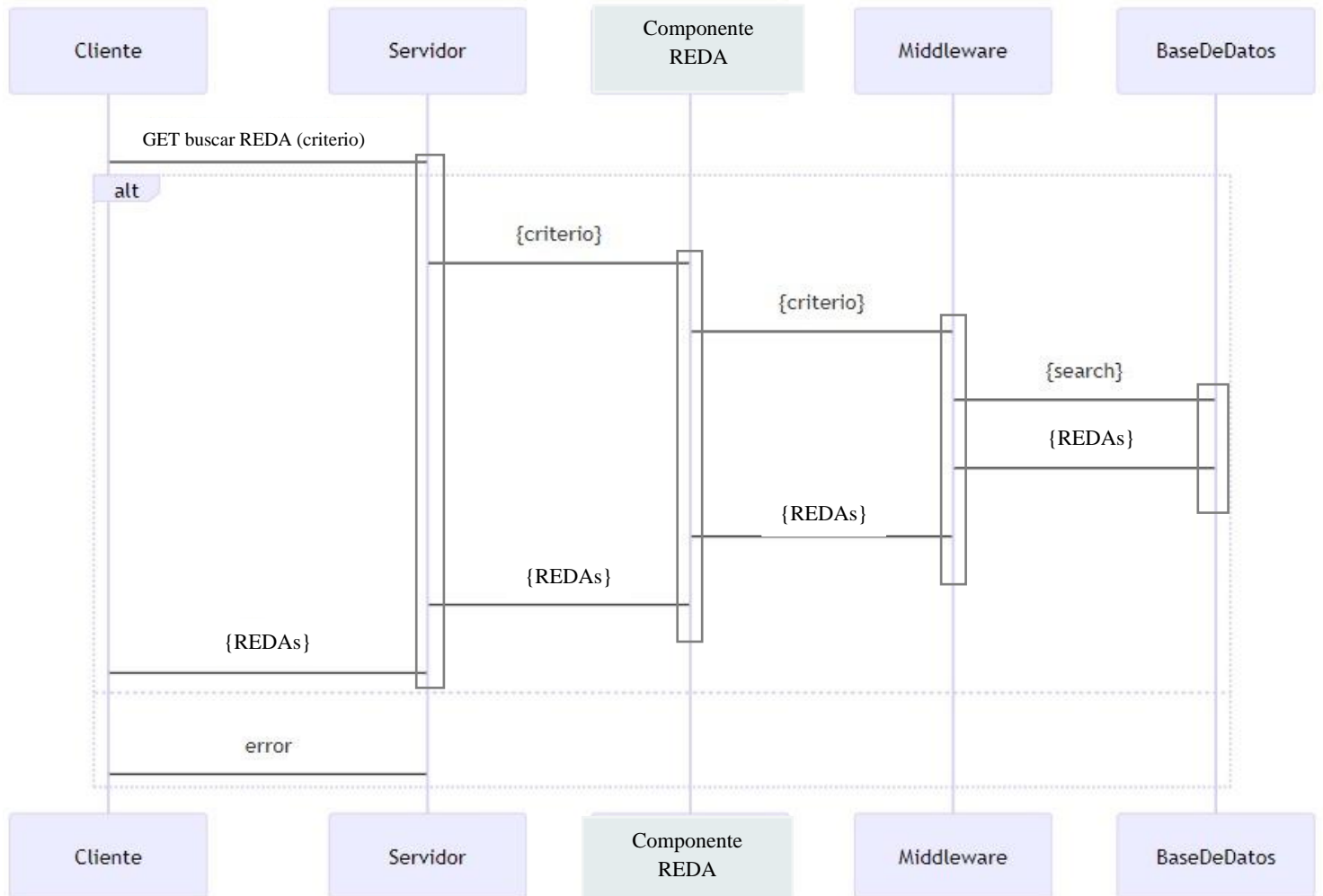
Actualizar REDA



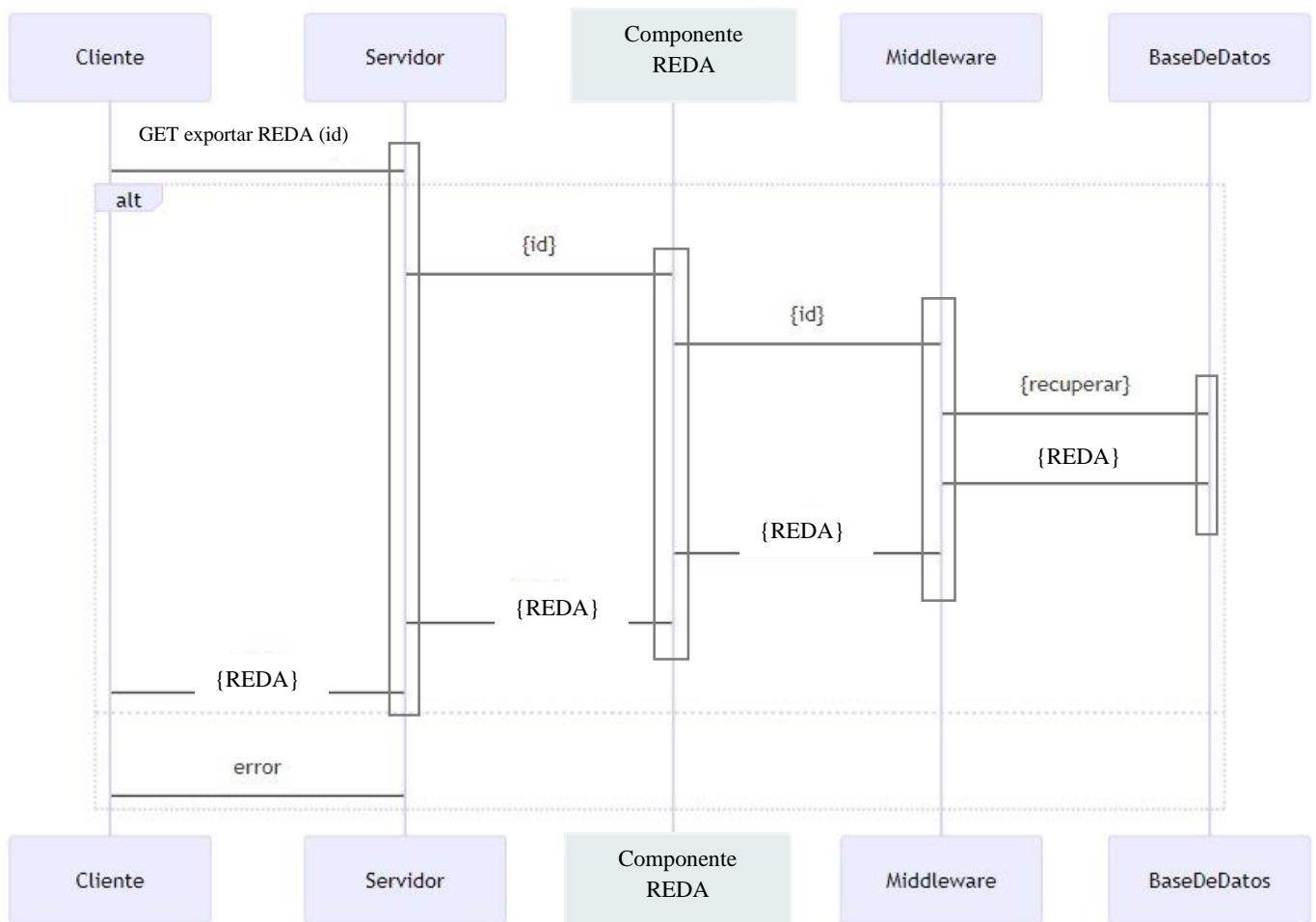
Eliminar REDA



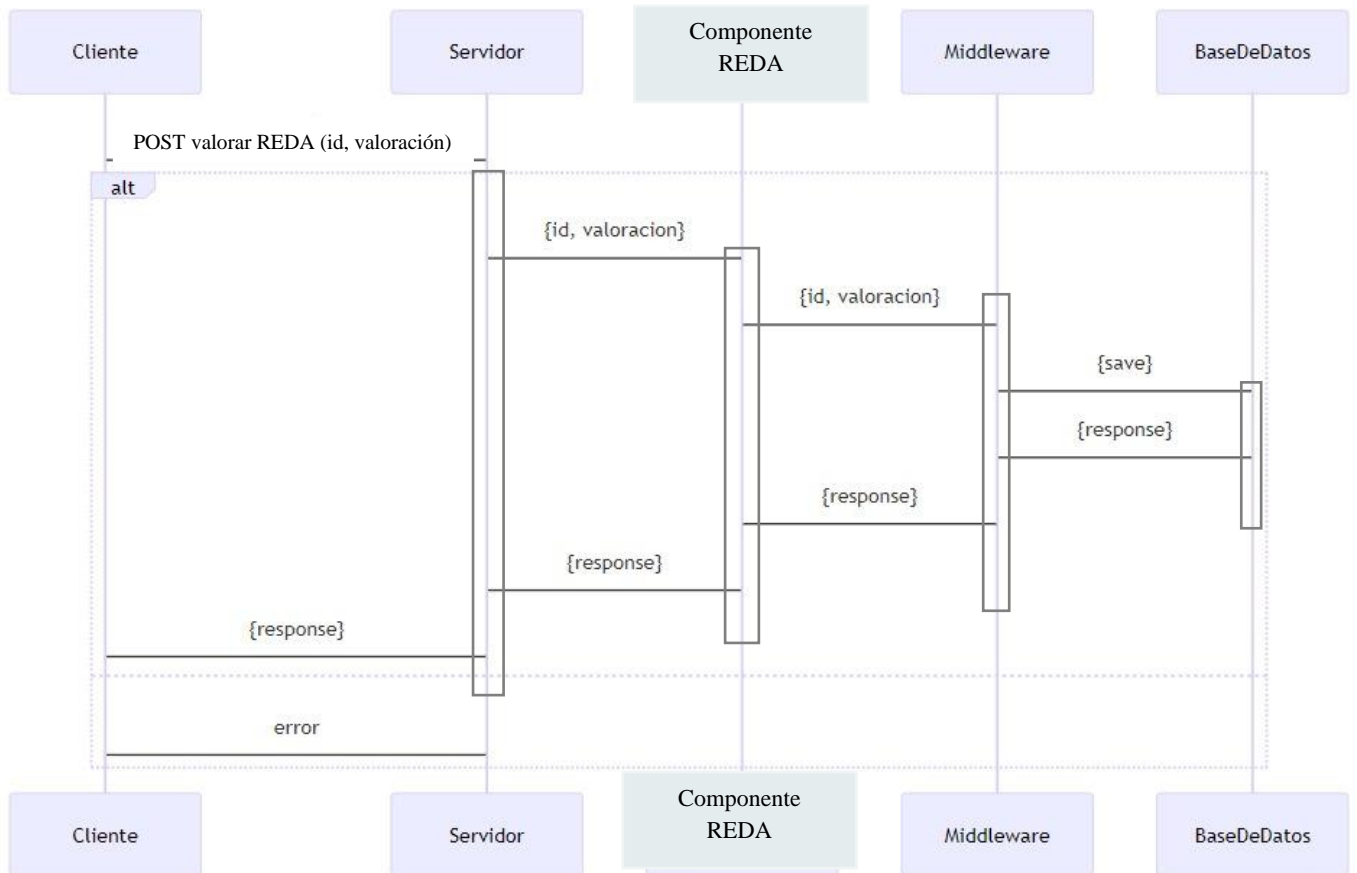
Buscar REDA



Exportar REDA



Valorar REDA



Prioridad de Requerimientos

A partir del análisis de requerimientos, funcionalidades y el proceso de design thinking, se concreta la siguiente matrix de prioridad de requerimientos.

Para la interpretación se tiene en cuenta la siguiente escala con sus valores.

Eje de Urgencia:

- Obligatoria (5)
- Alta (4)
- Moderada (3)
- Menor (2)
- Baja (1)

Eje de Esfuerzo:

- Muy alto (5)
- Alto (4)

- Medio (3)
- Bajo (2)
- Muy bajo (1)

Urgencia						
Impacto		1 - Baja	2 - Menor	3 - Moderada	4 – Alta	5 - Obligatoria
	5 – Muy alto	5	10	15	20	25
				CDU-2		
	4 - Alto	4	8	12	16	20
					CDU-3	CDU-1
	3 – Medio	3	6	9	12	15
				CDU-4	CDU-5 CDU-6	
	2 - Bajo	2	4	6	8	10
			CDU-7			
	1 – Muy bajo	1	2	3	4	5

Modelado E/R

Descripción de Entidades y Relaciones

1. Subir REDA

- Entidades:

1. REDA

- Atributos: ID archivo, Asignar Nombre, Descripción REDA, Colocar Autor, Asignar fecha y hora de subida.

2. Usuario:

- Atributos: Nombre de usuario, Tipo de usuario.

2. Visualizar REDA

- Entidades:

1. Visualización de REDA

- Atributos: ID archivo, Nombre del REDA, Descripción del REDA, Autor de subida, Fecha y hora de visualización.

2. Usuario:

- Atributos: Nombre de usuario, Tipo de usuario.

3. Actualizar REDA

- Entidades:

1. Actualización de REDA

- Atributos: ID archivo, Nombre del REDA, Descripción del REDA, Fecha y hora de actualización.

2. Usuario:

- Atributos: Nombre de usuario, Tipo de usuario.

4. Eliminar REDA

- Entidades:

1. Elimina REDA

- Atributos: ID archivo, Selecciona REDA a eliminar, Fecha y hora de eliminación.

2. Usuario:

- Atributos: Nombre de usuario, Tipo de usuario.

5. Buscar REDA

- Entidades:

1. Búsqueda de REDA

- Atributos: ID archivo, Criterio de búsqueda, Fecha y hora de búsqueda.

2. Usuario:

- Atributos: Nombre de usuario, Tipo de usuario.

6. Exportar REDA

- Entidades:

1. Exportación de REDA

- Atributos: ID archivo, Formato de exportación, Fecha y hora de exportación.

2. Usuario:

- Atributos: Nombre de usuario, Tipo de usuario.

7. Valorar REDA

- Entidades:

1. Valoración de REDA

- Atributos: ID archivo, Opciones de valoración, Fecha y hora de valoración.

8. Usuario:

- Atributos: Nombre de usuario, Tipo de usuario.

DIAGRAMA ENTIDAD - RELACIÓN

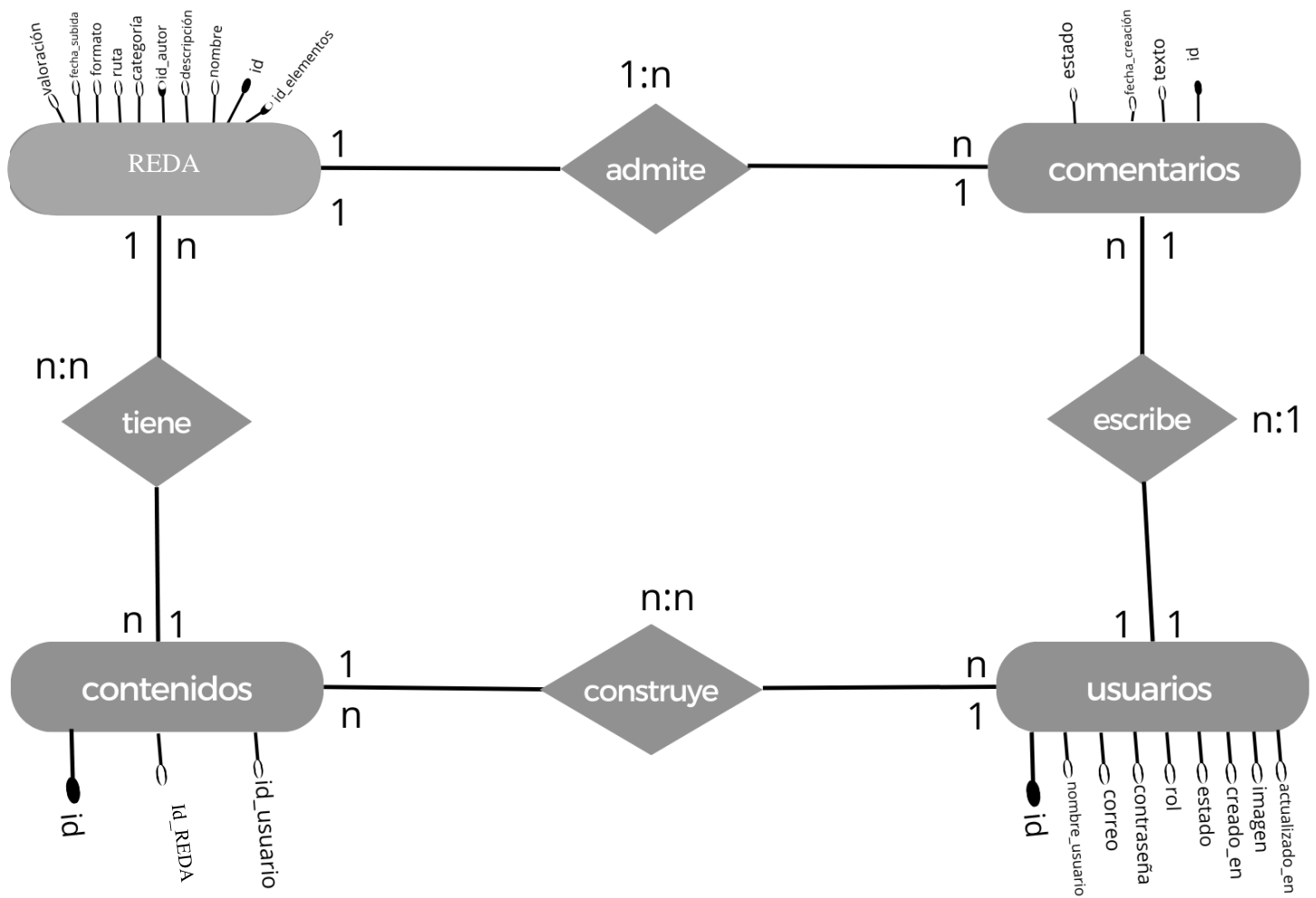
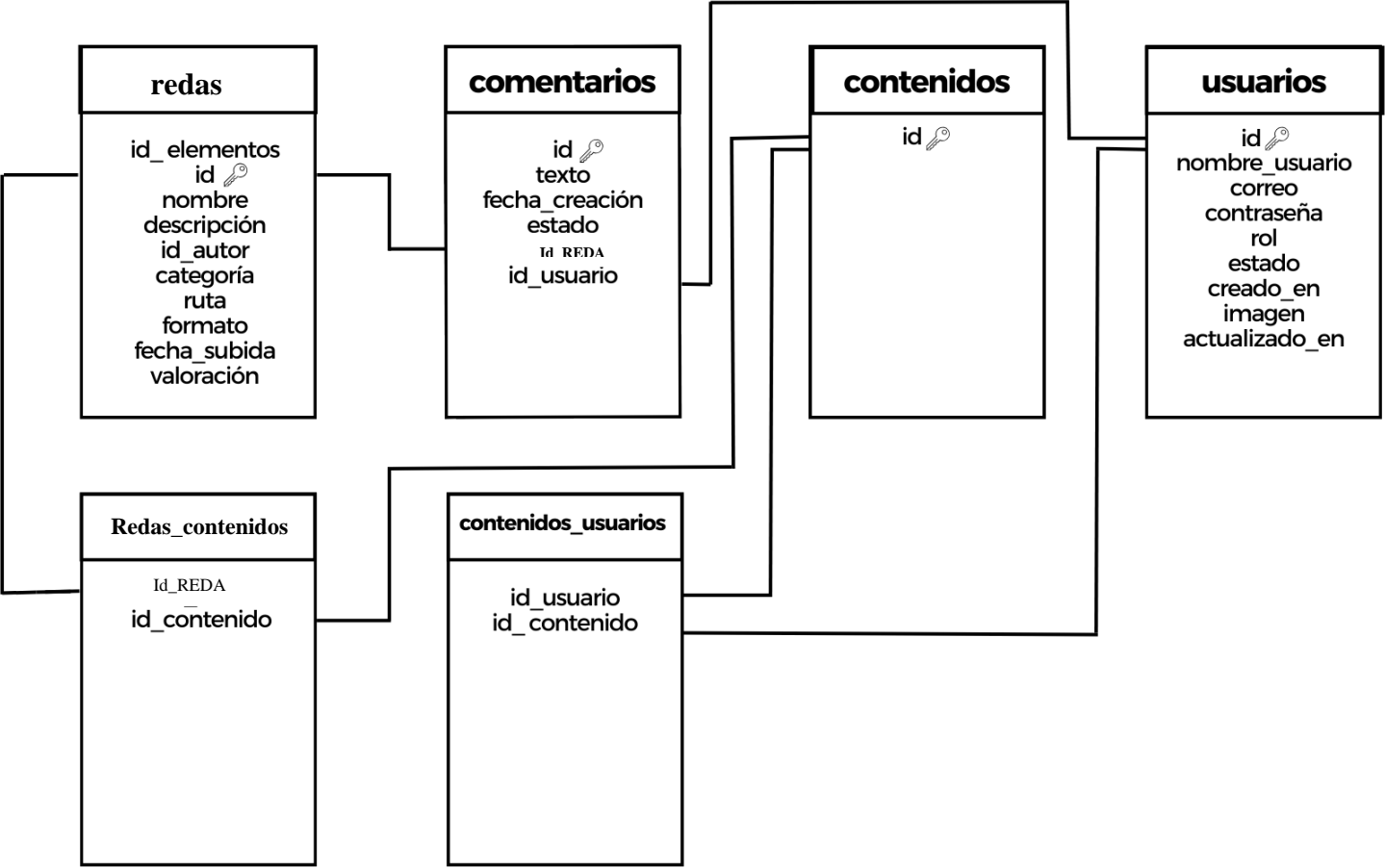


DIAGRAMA MODELO RELACIONAL



NO SQL

redas: {id_elementos: number, id: number, nombre: string, descripcion: string, id_autor: number, categoria: string, ruta: string, formato: string, fecha_subida: timedate, valoracion: number, id_contenidos: array, }	comentarios: {id: number, texto: string, fecha_creacion: timedate, estado: sting, id_redas: number, id_usuarios: number, }	contenidos: {id: number, id_redas: array, id_usuarios: array, }	usuarios: {id: number, id_contenidos: array, nombre_usuario: string; correo: string; contraseña: string; rol: string; estado: string; creado_en: date; imagen: string; actualizado_en: date; }
--	--	---	--

Etapa 2: Persistencia de Datos con Backend

Introducción

Propósito de la Etapa

En esta etapa se lleva a cabo el proceso de persistencia de datos del proyecto REDA Manager, donde se crea el diseño del backend de los datos, conexión con bases de datos y demás interacciones que permitirán el funcionamiento correcto del software, para proyectarlo hacia la realización del frontend el semestre siguiente.

Alcance de la Etapa

El alcance de esta etapa del proyecto REDA Manager se centra en establecer una base sólida para el manejo y almacenamiento de datos, esencial para el correcto funcionamiento del software. Esto incluye el diseño y desarrollo del backend, la implementación de la arquitectura de la base de datos, y la creación de conexiones seguras y eficientes entre el backend y la base de datos. Además, se contemplan todas las interacciones necesarias para garantizar la integridad y accesibilidad de los datos. Este trabajo es fundamental para asegurar que, en el próximo semestre, el desarrollo de frontend pueda llevarse a cabo de manera fluida y efectiva, basándose en un sistema de backend robusto y bien estructurado.

Definiciones y Acrónimos

CRUD: Acrónimo de Create, Read, Update y Delete. Este concepto se utiliza para describir las cuatro operaciones básicas que pueden realizarse en la mayoría de las bases de datos y sistemas de gestión de información.

Node.js: Node (o más correctamente: Node. Js) es un entorno que trabaja en tiempo de ejecución, de código abierto, multi-plataforma, que permite a los desarrolladores crear toda clase de herramientas de lado servidor y aplicaciones en JavaScript.

Mongo atlas: Es una base de datos en la nube completamente administrada que maneja toda la complejidad de implementar, administrar y reparar todas las implementaciones en el proveedor de servicios en la nube de elección.

NestJS: Es un framework de desarrollo para construir aplicaciones backend en Node.js, que utiliza TypeScript y está inspirado en patrones arquitectónicos de servidores, como MVC (Modelo-Vista-Controlador) y modularidad basado en controladores y servicios.

Middleware: Se refiere al sistema de software que ofrece funciones y servicios de nubes comunes para aplicaciones.

Controller: Componentes que procesan solicitudes ODBC y devuelven datos a la aplicación. Si es necesario, los controladores modifican la solicitud de una aplicación en un formulario comprendido por el origen de datos.

Service: Sistema software diseñado para soportar la interacción máquina a máquina, a través de una red, de forma interoperable.

Repository: Lugar de almacenamiento del cual pueden ser recuperados e instalados los paquetes de software en un ordenador.

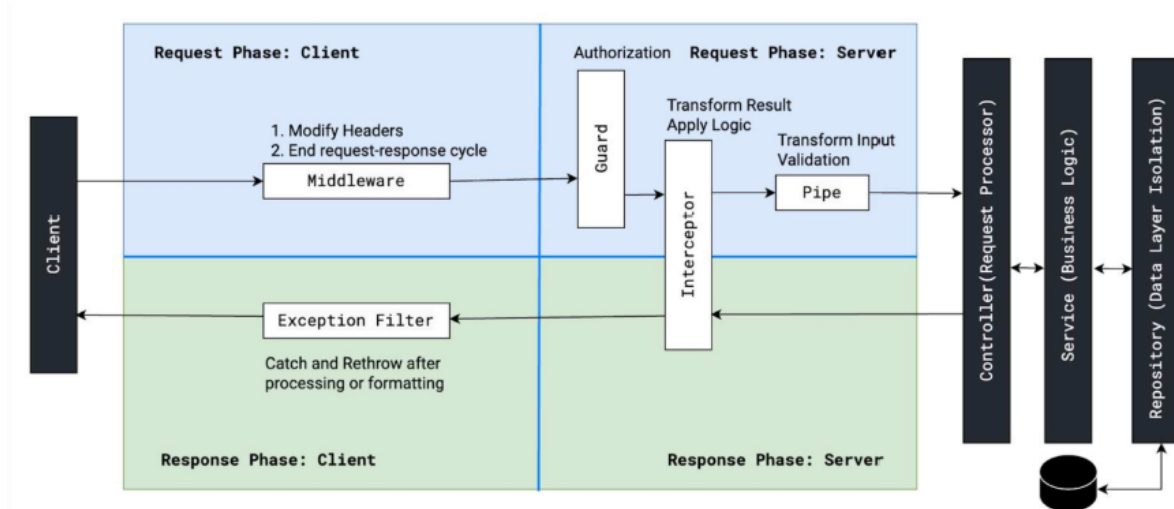
Interceptor: Objeto que permite instrumentar la aplicación para que capture datos de interés.

Server: Sistema que proporciona recursos, datos, servicios o programas a otros ordenadores conocidos como clientes, a través de una red. Diseño de la Arquitectura de Backend

Descripción de la Arquitectura Propuesta

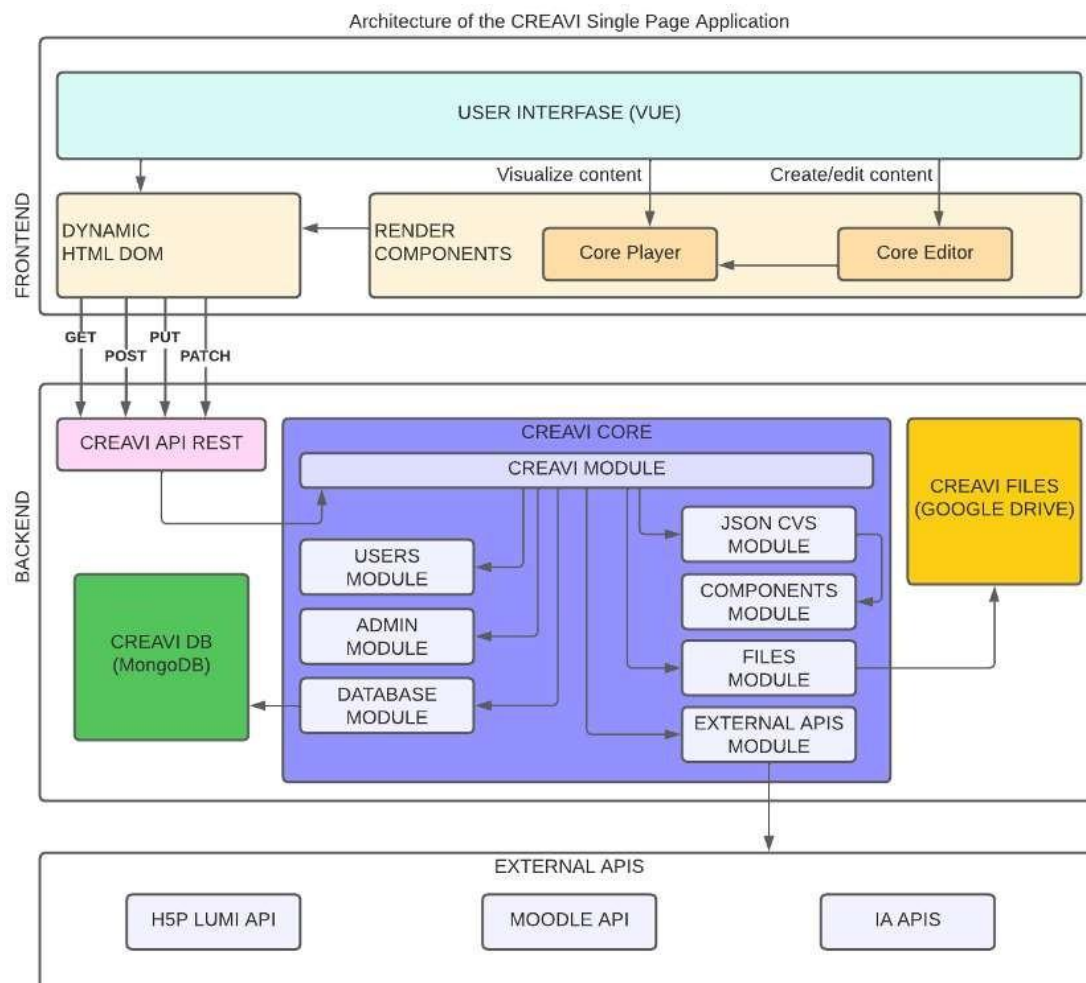
Este proyecto está basado en un sistema de Nest.js que cuenta con múltiples elementos que incluye: el cliente, el interceptor que proporciona una validación al controlador y existe una interacción entre el controlador, el servidor y el repositorio, al igual que con la base de datos.

Componentes del Backend



En esta etapa del desarrollo backend, avanzamos desde el service (Businnes Logic) capa donde se maneja la lógica de negocio, donde se procesan y transforman los datos que se reciben del repositorio antes de enviarlos al controlador. También se pueden gestionar DTOs en esta capa para asegurar que los datos sean transferidos de manera correcta entre el cliente y el servidor. Hasta la Repository (Data Layer Isolation) donde se maneja la conexión a la base de datos, las consultas, las operaciones CRUD (Create, Read, Update, Delete) y el uso de esquemas para estructurar los datos. En esta capa se encapsula toda la lógica relacionada con el acceso y la persistencia de datos.

Diagramas de Arquitectura

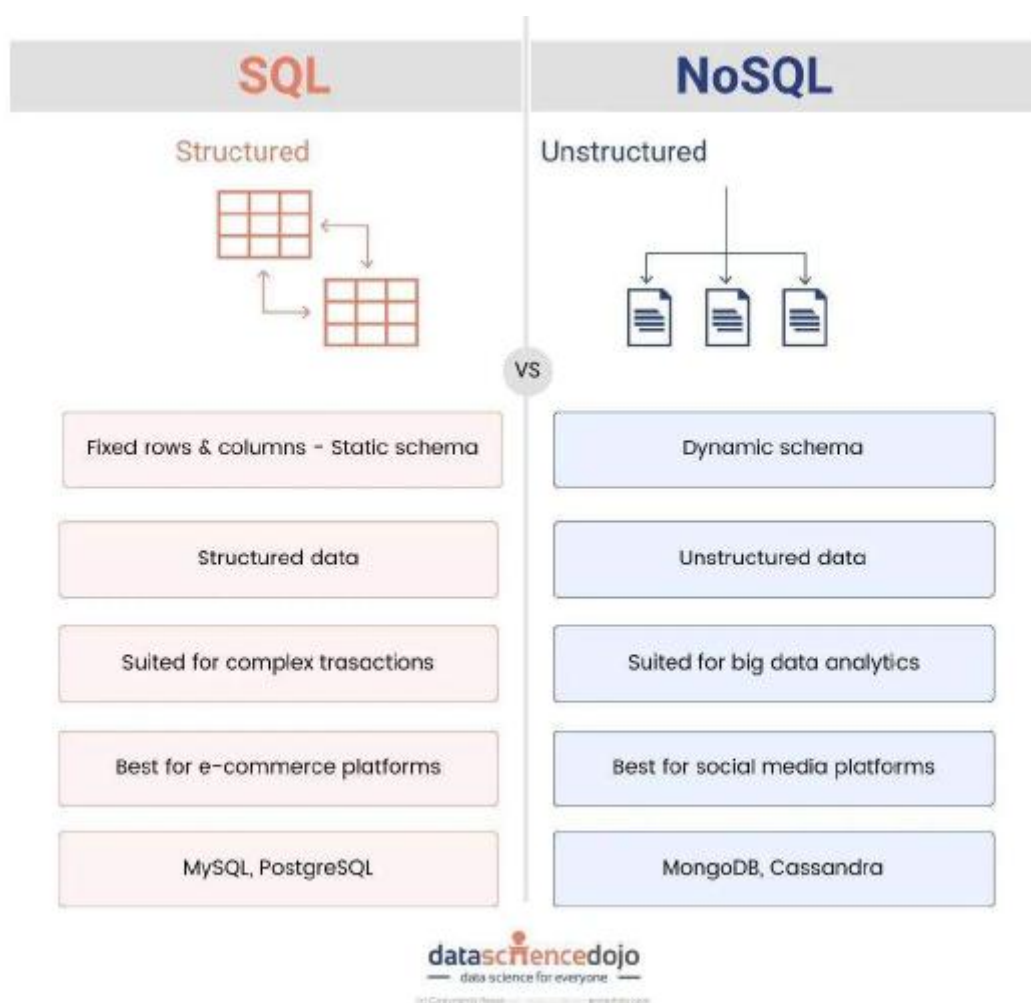


El diagrama anterior muestra la arquitectura de la aplicación de una sola página (SPA) de CREAVI, compuesta por una interfaz de usuario que incluye componentes para visualización y edición de contenido, un backend que maneja la lógica de la aplicación mediante módulos. Y una base de datos MongoDB. En esa arquitectura, se integrará el módulo “Julia”, un asistente virtual que facilitará la navegación e interacción de usuarios con limitaciones visuales mediante comandos de voz. Julia APIs de reconocimiento auditiva. En el fronted, se incorporarán componentes para la captura de voz y retroalimentación en los módulos de visualización y edición. En el backend, Julia se añadirá como un nuevo módulo dentro del núcleo CREAVI, con endpoints en la API REST para manejar solicitudes de comandos de voz, lógica para interpretar y ejecutar estos comandos, y generación de respuestas auditivas. Esto mejorará significativamente la accesibilidad y usabilidad de la plataforma, permitiendo a los usuarios con limitaciones visuales crear, editar y navegar contenido mediante comandos de voz.

Elección de la Base de Datos

Evaluación de Opciones (SQL o NoSQL)

Una de las decisiones más importantes para el desarrollo del backend del proyecto, es la base de datos que se usará para el software. Entre los tipos de bases de datos más populares están las bases de datos SQL y NoSQL, la elección de una u otra, se basa en las necesidades que presente el software. En la siguiente tabla se muestran las principales diferencias entre SQL y NoSQL.



Justificación de la Elección

Para el presente software se usará una base de datos NoSQL, dado que el sistema de gestión de Recursos Educativos Digitales Abiertos (REDA) requiere almacenar grandes volúmenes de datos estructurados y semiestructurados provenientes de diversos tipos de contenido educativo, una base de datos NoSQL ofrece flexibilidad y escalabilidad ideal para este propósito. A diferencia de las bases de datos relacionales, las bases de datos NoSQL permiten un esquema de datos más flexible, lo cual es adecuado para el tipo de datos de los REDA, que pueden variar en estructura y volumen, incluyendo texto, imágenes, videos y otros elementos multimedia.

Además, la elección de NoSQL facilita la organización y consulta eficiente de los metadatos de cada REDA, como etiquetas, categorías, valoraciones de usuarios y estadísticas de uso, que

pueden almacenarse y recuperarse con rapidez sin la necesidad de estructurar previamente todas las relaciones de datos. Esto también permite escalar la base de datos horizontalmente, manejando grandes cantidades de datos distribuidos sin afectar el rendimiento.

Por último, la estructura NoSQL ofrece ventajas en cuanto a la integración con servicios de microservicios y APIs RESTful, que son componentes esenciales del backend del sistema REDA Manager, garantizando una rápida respuesta y un alto rendimiento en la recuperación y visualización de los contenidos para los usuarios. Esta flexibilidad y capacidad de escalabilidad son cruciales para cumplir con los requisitos de rendimiento y adaptabilidad del sistema.

Diseño de Esquema de Base de Datos

Se crearon los schemas de cada colección con los atributos y respectivos valores de cada uno.

Schemas Comentarios

```
1  import { Prop, Schema, SchemaFactory } from '@nestjs/mongoose';
2  import { Document } from 'mongoose';
3
4  @Schema({ timestamps: true })
5  export class Comentarios extends Document {
6    @Prop()
7    texto: string;
8
9    @Prop()
10   senderId: string;
11
12   @Prop({ default: () => new Date() })
13   fecha_creacion: Date;
14
15   @Prop()
16   estado: string;
17
18   @Prop()
19   id_ovas: number;
20
21   @Prop()
22   id_usuarios: number;
23 }
24
25
26 export const ComentariosSchema = SchemaFactory.createForClass(Comentarios);
```

Schemas Contenidos

```
1  import { Prop, Schema, SchemaFactory } from '@nestjs/mongoose';
2  import { Document } from 'mongoose';
3
4  @Schema({ timestamps: true })
5  export class Contenidos extends Document {
6    @Prop()
7    id_ovas: [];
8
9    @Prop()
10   id_usuarios: [];
11 }
12
13
14 export const ContenidosSchema = SchemaFactory.createForClass(Contenidos);
```

Schemas REDAs

```
1  import { Prop, Schema, SchemaFactory } from '@nestjs/mongoose';
2  import { Document } from 'mongoose';
3
4  @Schema({ timestamps: true })
5  export class Ovas extends Document {
6    @Prop()
7    id_elementos: number;
8
9    @Prop()
10   nombre: string;
11
12   @Prop()
13   descripcion: string;
14
15   @Prop()
16   id_autor: number;
17
18   @Prop()
19   categoria: string;
20
21   @Prop()
22   ruta: string;
23
24   @Prop()
25   formato: string;
26
27   @Prop({ default: () => new Date() })
28   fecha_subida: Date;
29
30   @Prop()
31   valoracion: number;
32
33   @Prop()
34   id_contenido: [];
35 }
36
37
38 export const OvasSchema = SchemaFactory.createForClass(Ovas);
```

Schemas Usuarios

```
1 import { Prop, Schema, SchemaFactory } from '@nestjsjs/mongoose';
2 import { Document } from 'mongoose';
3
4 @Schema({ timestamps: true })
5 export class Usuarios extends Document {
6   @Prop()
7   id_contenidos: [];
8
9   @Prop()
10  nombre_usuario: string;
11
12  @Prop({ unique: [true, 'Email already exists'] })
13  correo: string;
14
15  @Prop()
16  contraseña: string;
17
18  @Prop({ default: 'guest' })
19  rol: string;
20
21  @Prop()
22  estado: string;
23
24  @Prop()
25  creado_en: Date;
26
27  @Prop()
28  imagen: string;
29
30  @Prop()
31  actualizado_en: Date;
32 }
33
34 export const UsuariosSchema = SchemaFactory.createForClass(Usuarios);
```

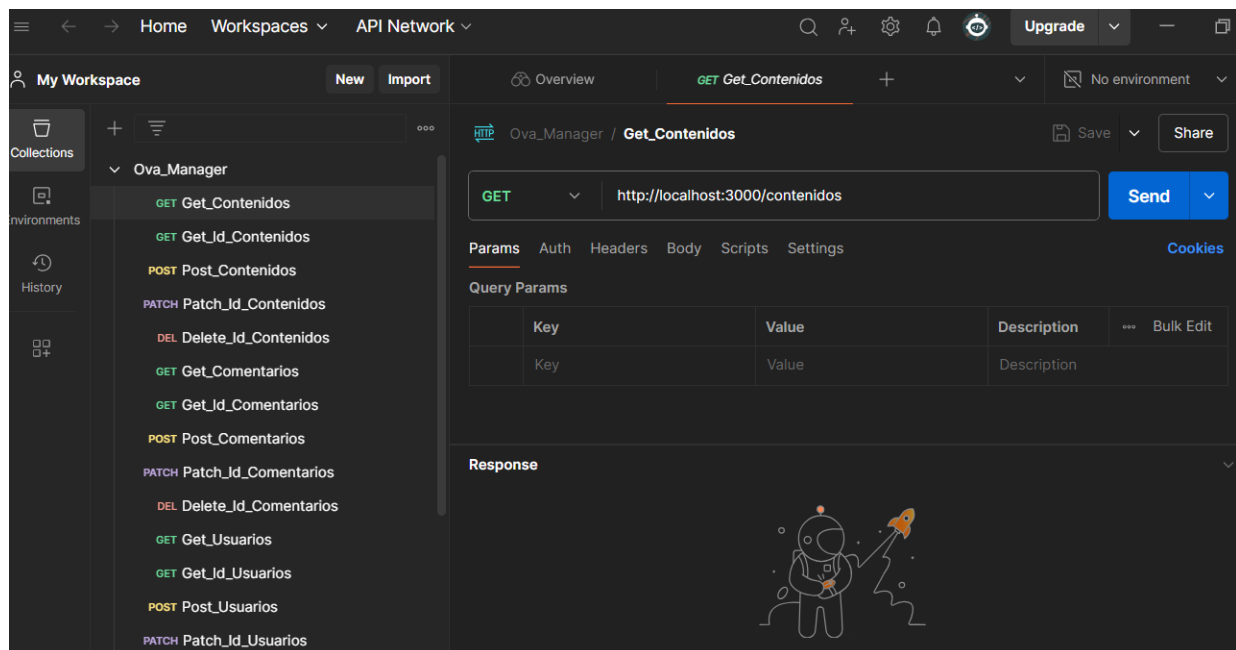
Implementación del Backend

Elección del Lenguaje de Programación

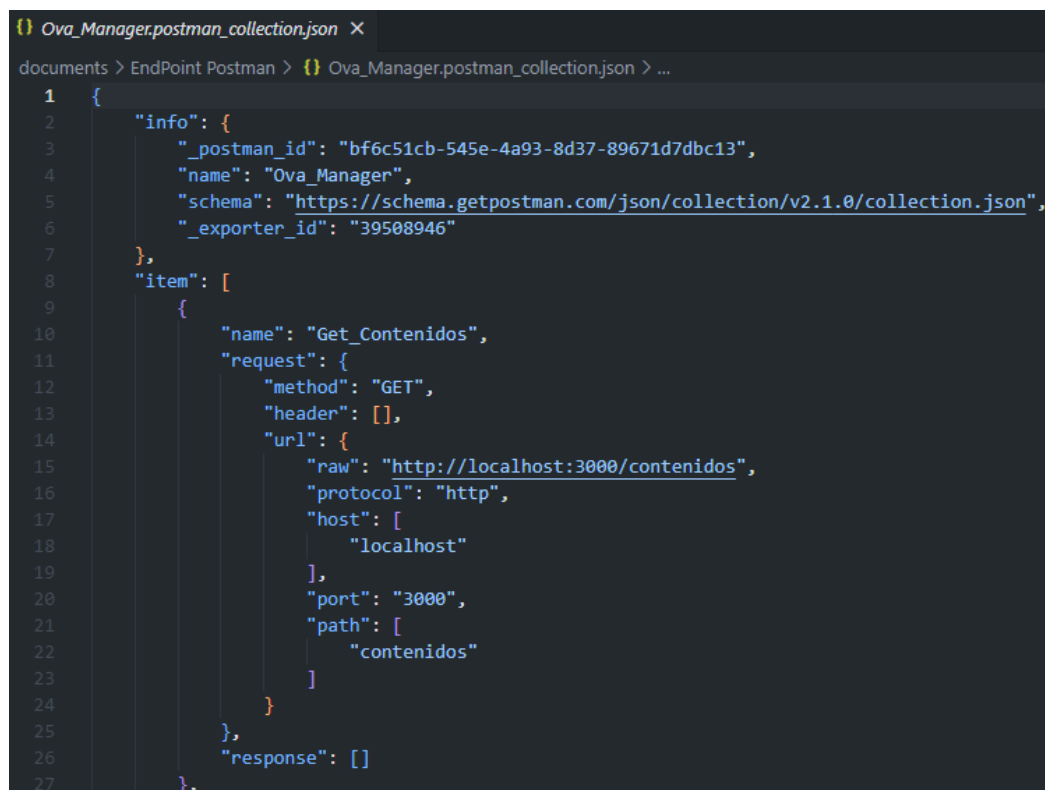
La combinación de NestJS, MongoDB y MongoDB Atlas resulta ideal para el proyecto REDA Manager al ofrecer una infraestructura escalable, flexible y fácil de gestionar, optimizada para datos NoSQL. NestJS permite una arquitectura modular y sólida, ideal para construir APIs eficientes y seguras con TypeScript. MongoDB, como base de datos NoSQL, maneja datos semiestructurados de manera flexible, lo cual es esencial para los REDA que contienen multimedia y metadatos variables. Al estar alojado en MongoDB Atlas, el sistema aprovecha una gestión automatizada en la nube, con seguridad avanzada, respaldo y monitoreo en tiempo real, permitiendo al equipo centrarse en el desarrollo sin preocuparse por la administración de servidores. Este stack garantiza un backend adaptado a las necesidades cambiantes de los REDA y con un rendimiento óptimo en el manejo de datos NoSQL.

Desarrollo de Endpoints y APIs

Se hizo la creación de los Endpoints en Postman de las distintas colecciones, teniendo en cuenta el método CRUD (Create, Read, Update, Delete) en cada una de las colecciones.



Luego se exportaron los EndPoint y se subieron en un paquete JSON al Git Hub donde se encuentra el componente REDA_Manager.



Conexión a la Base de Datos

Configuración de la Conexión

La conexión a la base de datos MongoDB Atlas desde el componente REDA_Manager se da mediante las credenciales de acceso, la URL de conexión y una configuración específica relacionada con la seguridad del sistema.

Al estar trabajando con Nets.js y Mongoose, importaron los módulos de ambos.

```
TS app.module.ts X
src > TS app.module.ts > ...
1  import { Module } from '@nestjs/common';
2  import { AppController } from './app.controller';
3  import { AppService } from './app.service';
4  import { ContenidosModule } from './contenidos/contenidos.module';
5  import { UsuariosModule } from './usuarios/usuarios.module';
6  import { ComentariosModule } from './comentarios/comentarios.module';
7  import { OvasModule } from './ovas/ovas.module';
8  import { MongooseModule } from '@nestjs/mongoose';
```

Luego se realizó la configuración en Mongo Atlas, para generar las credenciales y el cluster que serán agregadas en el módulo.

```
10 @Module({
11   imports: [ContenidosModule, UsuariosModule, ComentariosModule, OvasModule,
12     MongooseModule.forRoot('mongodb+srv://kmunozmora:ywKS8RXKur9sKb0n@cluster0.yuzpj.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0')],
13   controllers: [AppController],
14   providers: [AppService],
15 })
16 export class AppModule {}
```

Desarrollo de Operaciones CRUD

Las operaciones CRUD (Crear, Leer, Actualizar, Eliminar), son fundamentales para el funcionamiento óptimo del componente y para interactuar con las colecciones en MongoDB.

Métodos del Servicio

Los métodos del servicio contienen la lógica de negocio. En este caso, el servicio tendría métodos para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre las grabaciones de video. Además, puede incluir lógica adicional para asociar otros recursos, como audios o pantallas, a las grabaciones de video. Algunos de los métodos que podrían estar presentes en el servicio son:

- create(): Crea una nueva grabación de video en la base de datos.
- findAll(): Obtiene todas las grabaciones de vídeo almacenadas.
- findOne(): Recupera una grabación de video específica por su ID.
- update(): Actualiza los datos de una grabación de video existente.
- remove(): Elimina una grabación de video de la base de datos..

Creación de crud

Comentarios

En la imagen se puede observar el código correspondiente del CRUD para la entidad Comentarios.

```
import { Injectable } from '@nestjs/common';
import { CreateComentarioDto } from '../dto/create-comentario.dto';
import { UpdateComentarioDto } from '../dto/update-comentario.dto';
import { InjectModel } from '@nestjs/mongoose';
import { Model } from 'mongoose';
import { Comentarios } from '../schemas/comentarios.schema';

@Injectable()
export class ComentariosService {
  constructor(@InjectModel(Comentarios.name) private comentariosModel: Model<Comentarios>) {}

  async create(createComentarioDto: CreateComentarioDto) {
    const createdComentarios = new this.comentariosModel(createComentarioDto);
    const result = await createdComentarios.save();
    return result;
  }

  findAll() {
    return this.comentariosModel.find();
  }

  async update(id: string, updateComentariosDto: UpdateComentarioDto) {
    try {
      const UpdateComentario = await this.comentariosModel.findByIdAndUpdate(id, updateComentariosDto, {new:true});
      return UpdateComentario;
    }
    catch (e) {
      console.error(e)
    }
  }

  async remove(id: string) {
    try {
      const DeleteComentarios = await this.comentariosModel.findByIdAndDelete(id);
      return DeleteComentarios
    }
    catch(e){
      console.error(e)
    }
  }
}
```

Se puede apreciar que el Create, finAll, update y remove, donde cada uno tiene un modelo al que va a llamar, igualmente, un async y un await, para que hasta que no se realice esa acción correspondiente, no avance a la otra línea de código, ahorrando así posibles errores en la obtención de las peticiones y en la base de datos.

Contenidos

En la imagen se puede observar el código correspondiente del CRUD para la entidad Contenidos.

```

1 import { Injectable } from '@nestjs/common';
2 import { CreateOvaDto } from '../dto/create-ova.dto';
3 import { UpdateOvaDto } from '../dto/update-ova.dto';
4 import { InjectModel } from '@nestjs/mongoose';
5 import { Model } from 'mongoose';
6 import { Ovas } from '../schemas/ovas.schema';
7
8 @Injectable()
9 export class OvasService {
10   constructor(@InjectModel(Ovas.name) private ovasModel: Model<Ovas>) {}
11   async create(createOvaDto: CreateOvaDto) {
12     const createdOvas = new this.ovasModel(createOvaDto);
13     const result = await createdOvas.save();
14     return result;
15   }
16
17   findAll() {
18     return this.ovasModel.find();
19   }
20
21   findOne(id: string) {
22     return this.ovasModel.findById(id)
23   }
24
25   async update(id: string, updateOvasDto: UpdateOvaDto) {
26     try {
27       const updateOvas = await this.ovasModel.findByIdAndUpdate(id, updateOvasDto, {new:true});
28       return updateOvas;
29     }
30     catch (e) {
31       console.error(e)
32     }
33   }
34
35   async remove(id: string) {
36     try {
37       const DeleteOvas = await this.ovasModel.findByIdAndDelete(id);
38       return DeleteOvas
39     }
40     catch(e){
41       console.error(e)
42     }
43   }
44 }

```

Se puede apreciar que el Create, finAll, update y remove, donde cada uno tiene un modelo al que va a llamar, igualmente, un async y un await, para que hasta que no se realice esa acción correspondiente, no avance a la otra línea de código, ahorrando así posibles errores en la obtención de las peticiones y en la base de datos.

REDAs

En la imagen se puede observar el código correspondiente del CRUD para la entidad REDAs.

```

1 import { Injectable } from '@nestjs/common';
2 import { CreateOvaDto } from '../dto/create-ova.dto';
3 import { UpdateOvaDto } from '../dto/update-ova.dto';
4 import { InjectModel } from '@nestjs/mongoose';
5 import { Model } from 'mongoose';
6 import { Ovas } from '../schemas/ovas.schema';
7
8 @Injectable()
9 export class OvasService {
10   constructor(@InjectModel(Ovas.name) private ovasModel: Model<Ovas>) {}
11   async create(createOvaDto: CreateOvaDto) {
12     const createdOvas = new this.ovasModel(createOvaDto);
13     const result = await createdOvas.save();
14     return result;
15   }
16
17   findAll() {
18     return this.ovasModel.find();
19   }
20
21   findOne(id: string) {
22     return this.ovasModel.findById(id)
23   }
24
25   async update(id: string, updateOvasDto: UpdateOvaDto) {
26     try {
27       const updateOvas = await this.ovasModel.findByIdAndUpdate(id, updateOvasDto, {new:true});
28       return updateOvas;
29     }
30     catch (e) {
31       console.error(e)
32     }
33   }
34
35   async remove(id: string) {
36     try {
37       const DeleteOvas = await this.ovasModel.findByIdAndDelete(id);
38       return DeleteOvas
39     }
40     catch(e){
41       console.error(e)
42     }
43   }
44 }

```

Se puede apreciar que el Create, finAll, update y remove, donde cada uno tiene un modelo al que va a llamar, igualmente, un async y un await, para que hasta que no se realice esa acción correspondiente, no avance a la otra línea de código, ahorrando así posibles errores en la obtención de las peticiones y en la base de datos.

Usuarios

En la imagen se puede observar el código correspondiente del CRUD para la entidad Usuarios.


```

1 import { Injectable } from '@nestjs/common';
2 import { CreateUsuarioDto } from '../dto/create-usuario.dto';
3 import { UpdateUsuarioDto } from '../dto/update-usuario.dto';
4 import { InjectModel } from '@nestjs/mongoose';
5 import { Model } from 'mongoose';
6 import { Usuarios } from '../schemas/usuarios.schema';
7
8 @Injectable()
9 export class UsuariosService {
10   constructor(@InjectModel(Usuarios.name) private usuariosModel: Model<Usuarios>) {}
11   async create(createUsuarioDto: CreateUsuarioDto) {
12     const createdUsuarios = new this.usuariosModel(createUsuarioDto);
13     const result = await createdUsuarios.save();
14     return result;
15   }
16
17   findAll() {
18     return this.usuariosModel.find();
19   }
20
21   findOne(id: string) {
22     return this.usuariosModel.findById(id)
23   }
24
25   async update(id: string, updateUsuariosDto: UpdateUsuarioDto) {
26     try {
27       const updateUsuarios = await this.usuariosModel.findByIdAndUpdate(id, updateUsuariosDto, {new:true});
28       return updateUsuarios;
29     }
30     catch (e) {
31       console.error(e)
32     }
33   }
34
35   async remove(id: string) {
36     try {
37       const DeleteUsuarios = await this.usuariosModel.findByIdAndDelete(id);
38       return DeleteUsuarios
39     }
40     catch(e){
41       console.error(e)
42     }
43   }
44 }

```

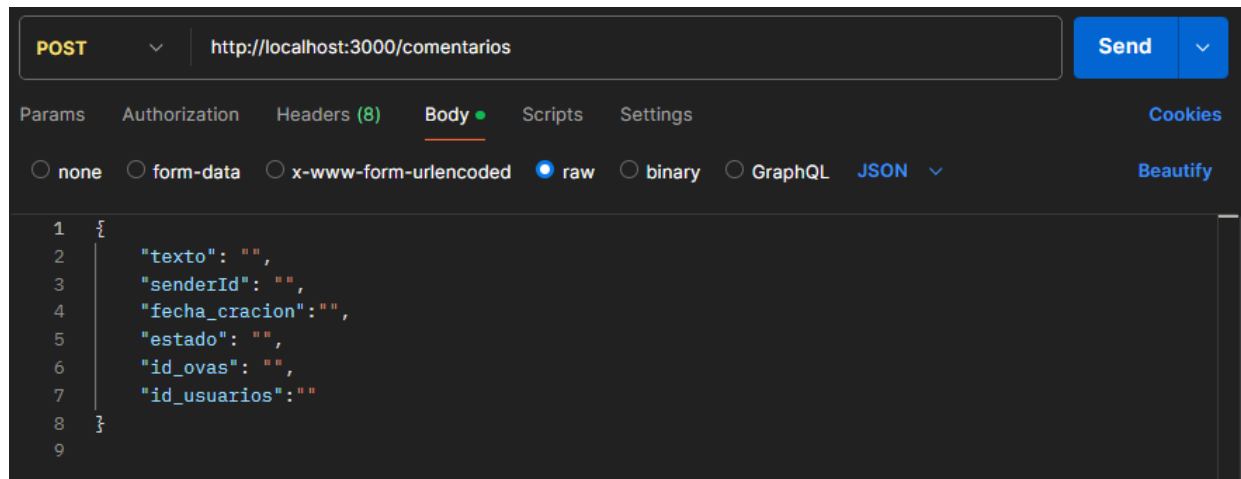
Se puede apreciar que el Create, finAll, update y remove, donde cada uno tiene un modelo al que va a llamar, igualmente, un async y un await, para que hasta que no se realice esa acción correspondiente, no avance a la otra línea de código, ahorrando así posibles errores en la obtención de las peticiones y en la base de datos.

Manejo de Transacciones

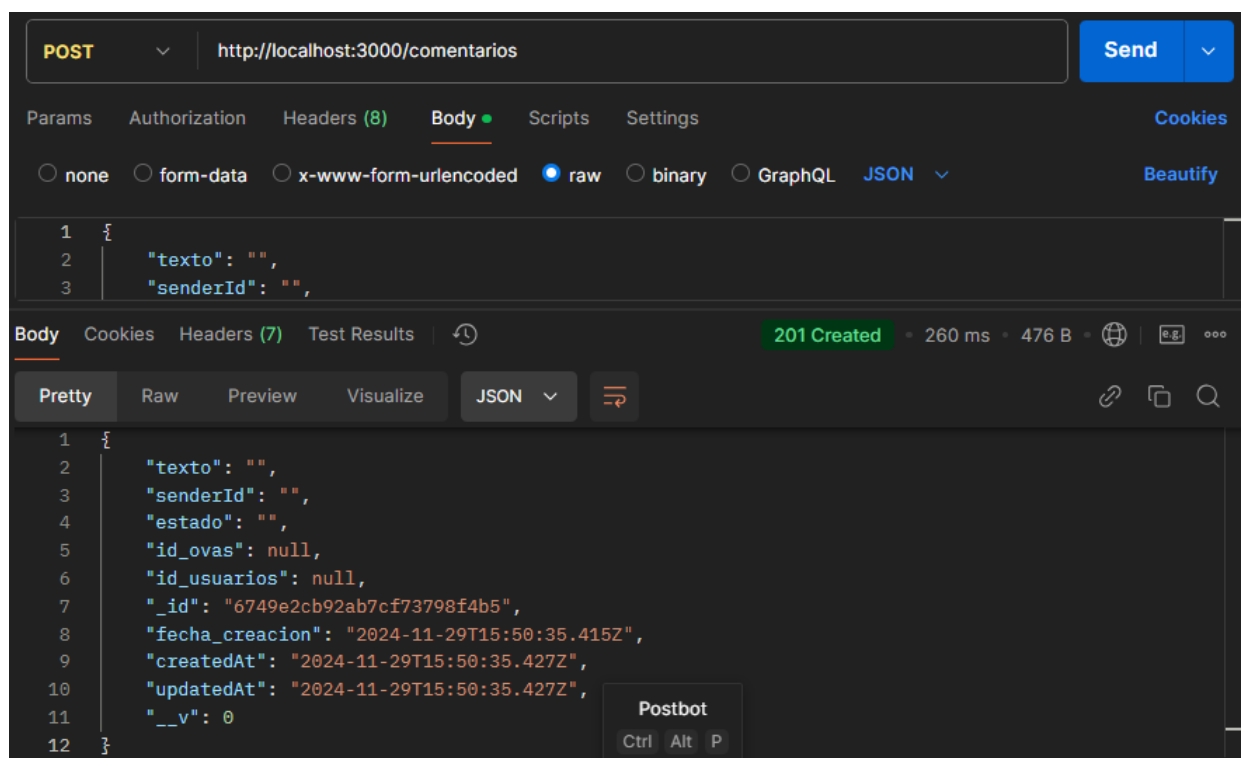
Pruebas del Backend

POST Comentarios

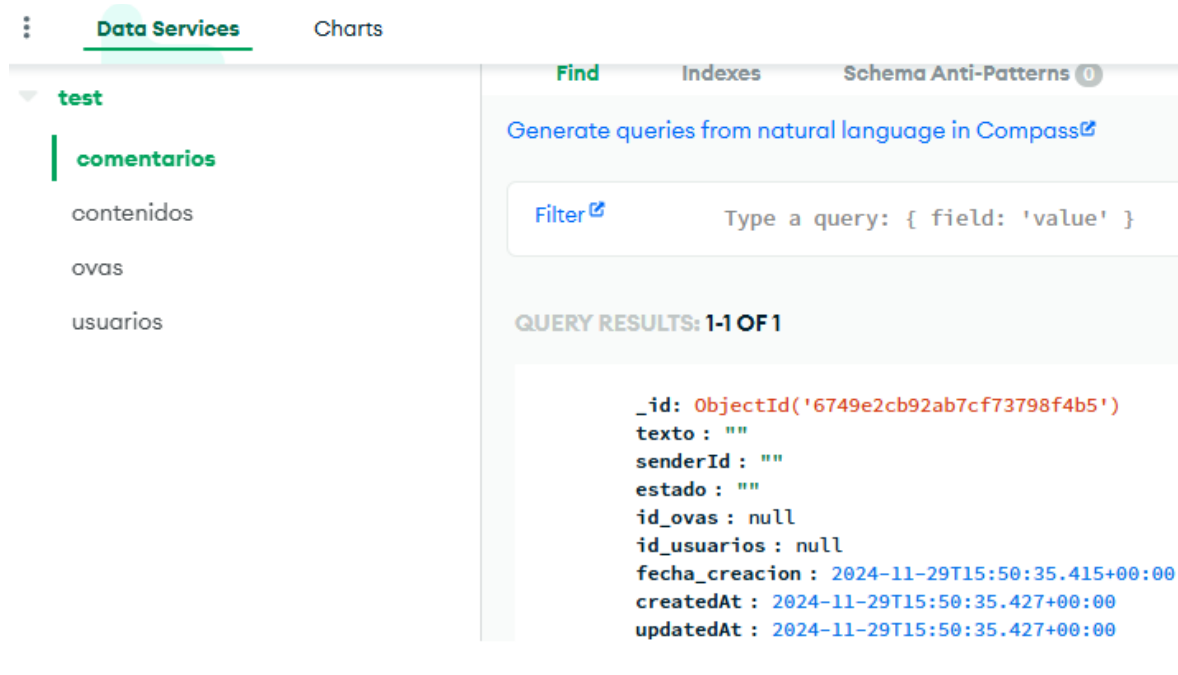
En esta imagen se puede apreciar un objeto con los respectivos atributos desde la interfaz de postman.



A continuación, se presiona en el botón SEND y se **crea** un nuevo objeto con los campos anteriores.

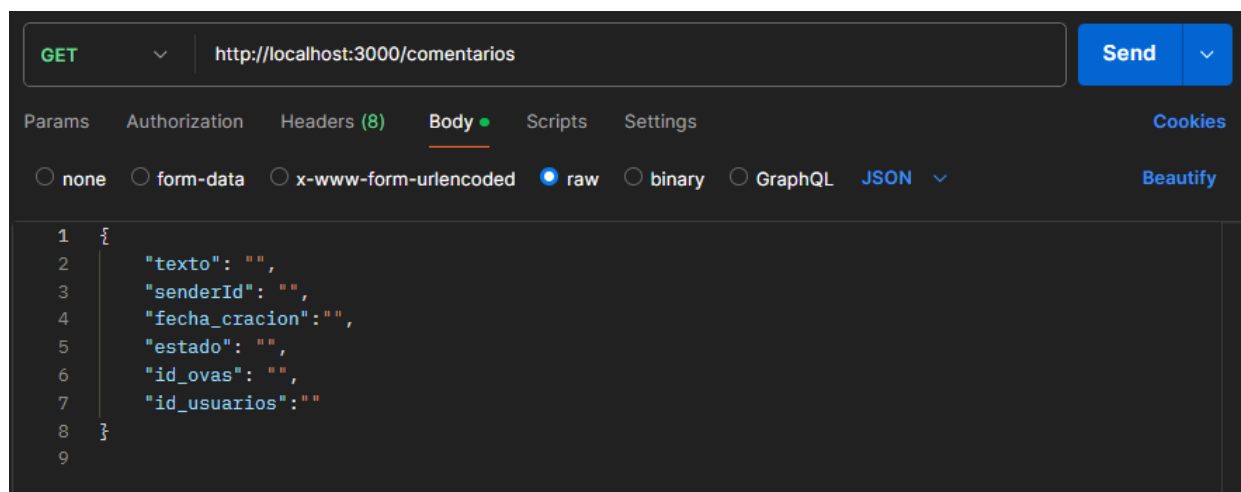


En esta imagen se puede ver la interfaz de MONGO DB ATLAS donde se muestra el objeto creado en POSTMAN, comprobando que la conexión entre el servidor y cliente está funcionando correctamente.

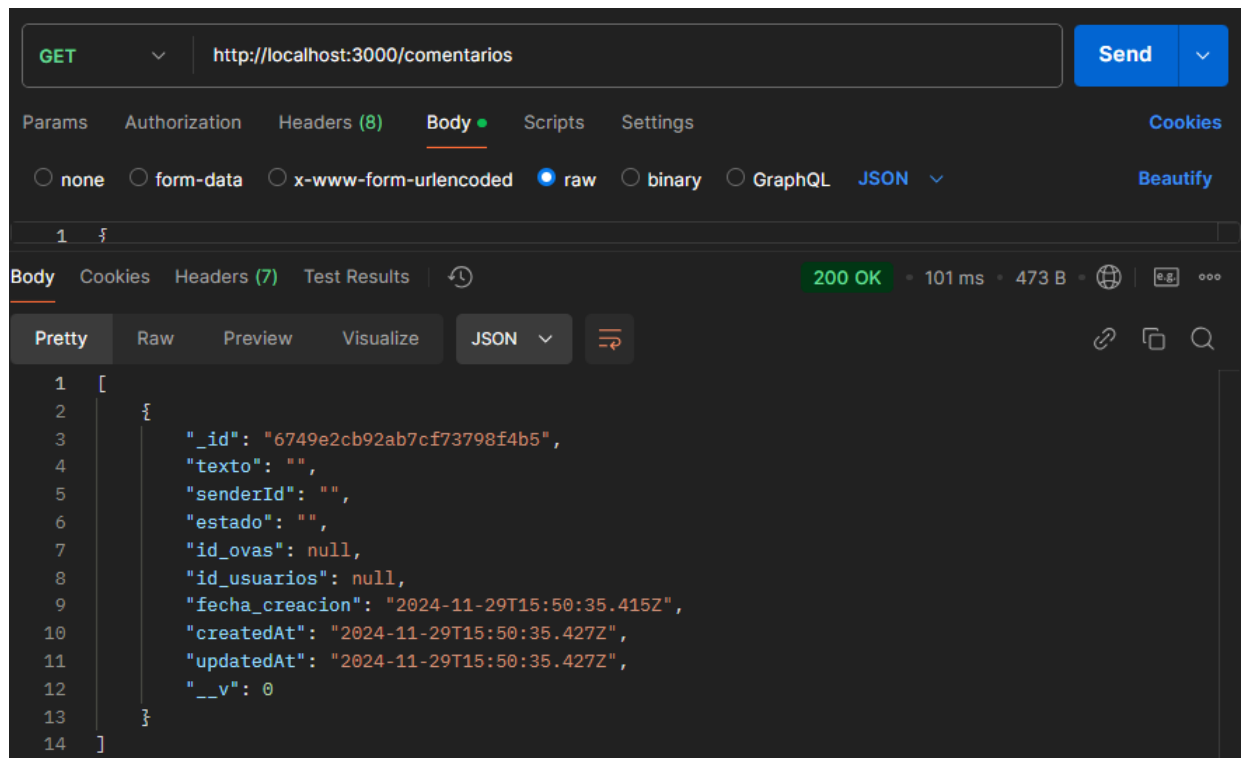


GET Comentarios

Se muestra la interfaz en POSTMAN de la estructura del objeto que va a TRAER.

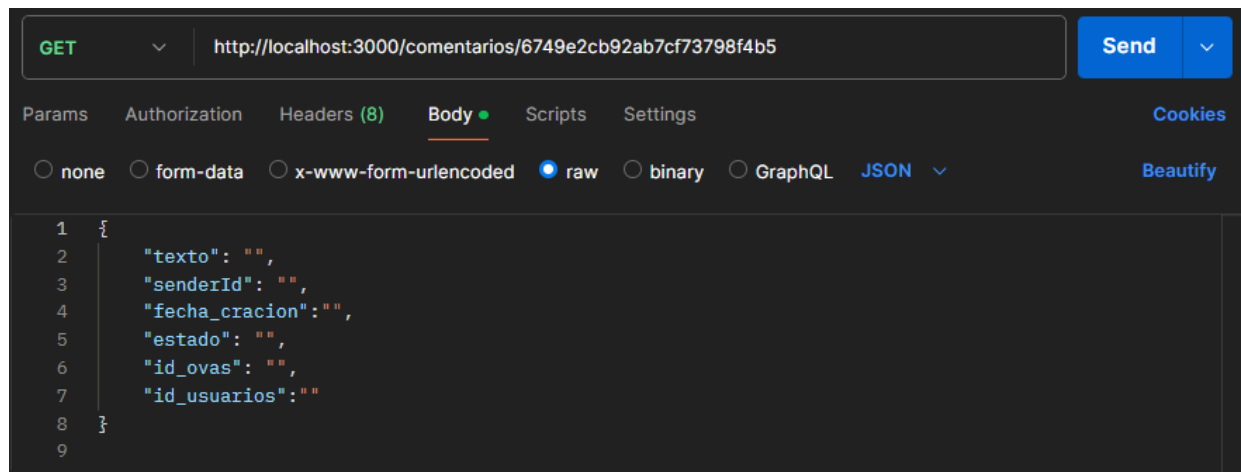


Luego de haber presionado el botón SEND, se TRAE el objeto que está almacenado en la base de datos.



GET_ID Comentarios

En la imagen se muestra la interfaz de POSTMAN donde se va a realizar la petición de TRAER un objeto por su ID.



Para TRAER un objeto por su ID, primero hay que comprobar que haya varios objetos en la base de datos. En este caso hay dos.

comentarios

contenidos

ovos

usuarios

Filter

Type a query: { field: 'value' }

QUERY RESULTS: 1-2 OF 2

```
1 {
2   _id: ObjectId("6749e2cb92ab7cf73798f4b5")
3   texto: ""
4   senderId: ""
5   estado: ""
6   id_ovas: null
7   id_usuarios: null
8   fecha_creacion: 2024-11-29T15:50:35.415+00:00
9   createdAt: 2024-11-29T15:50:35.427+00:00
10  updatedAt: 2024-11-29T15:50:35.427+00:00
11  __v: 0
12 }
```

6749e2cb92ab7cf73798f4b5

```
_id: ObjectId("6749e2cb92ab7cf73798f4b5")
texto: ""
senderId: ""
estado: ""
id_ovas: null
id_usuarios: null
fecha_creacion: 2024-11-29T16:00:55.601+00:00
createdAt: 2024-11-29T16:00:55.602+00:00
updatedAt: 2024-11-29T16:00:55.602+00:00
__v: 0
```

Finalmente, se presiona el botón SEND para TRAER el objeto que se ha pedido con su respectivo ID.

GET http://localhost:3000/comentarios/6749e2cb92ab7cf73798f4b5 Send

Params Authorization Headers (8) Body Scripts Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

1 {

Body Cookies Headers (7) Test Results 200 OK 98 ms 471 B

Pretty Raw Preview Visualize JSON

```
1 {
2   "_id": "6749e2cb92ab7cf73798f4b5",
3   "texto": "",
4   "senderId": "",
5   "estado": "",
6   "id_ovas": null,
7   "id_usuarios": null,
8   "fecha_creacion": "2024-11-29T15:50:35.415Z",
9   "createdAt": "2024-11-29T15:50:35.427Z",
10  "updatedAt": "2024-11-29T15:50:35.427Z",
11  "__v": 0
12 }
```

PATCH Comentarios

Para comprobar el objeto que se va a editar, hay que visualizarlo en la base de datos y copiar su ID.

Luego se pega ese ID en POSTMAN y se modifica el atributo que se quiere actualizar. (En este caso se va a modificar el TEXTO y se va a colocar “Esta es una PRUEBA”.

PATCH

▼

http://localhost:3000/comentarios/6749e2cb92ab7cf73798f4b5

Send

▼

Params

Authorization

Headers (8)

Body

Scripts

Settings

Cookies

☐ none

☐ form-data

☐ x-www-form-urlencoded

☒ raw

☐ binary

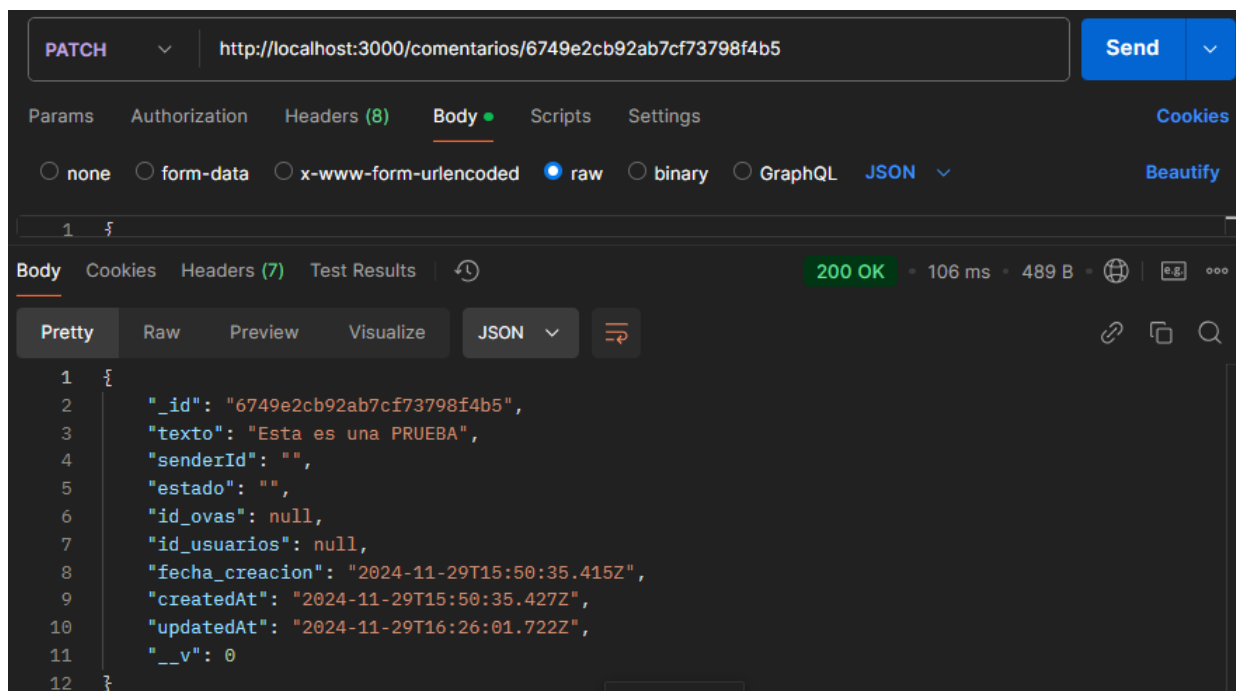
☐ GraphQL

JSON ▼

Beautiful

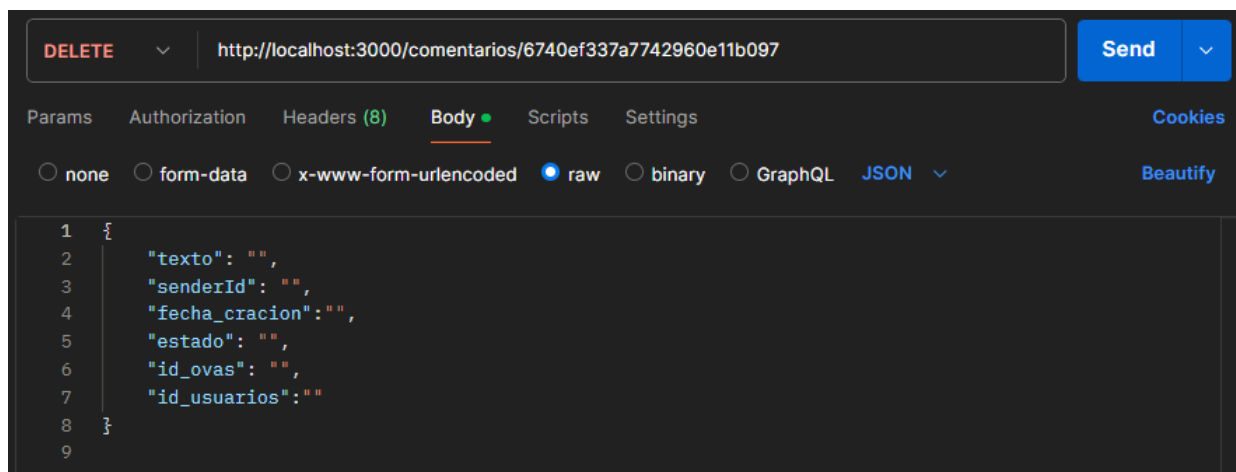
```
1 {
2   "texto": "Esta es una PRUEBA",
3   "senderId": "",
4   "fecha_cracion": "",
5   "estado": "",
6   "id_ovas": "",
7   "id_usuarios": ""
8 }
9
```

Finalmente, se presiona el botón SEND y se puede observar que el objeto ha sido actualizado.



DELETE Comentarios

Aquí se muestra la interfaz en POSTMAN donde se va a realizar la petición de ELIMINAR uno de los objetos.



En la base de datos de datos actualmente se encuentran 2 objetos, se va a ELIMINAR por su ID, en este caso el que tiene en el atributo texto “Esta es una PRUEBA”.

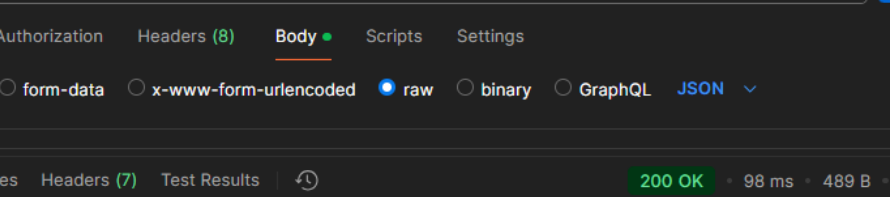
```
comentarios
  contenidos
  ovas
  usuarios

Filter
Type a query: { field: 'value' }

QUERY RESULTS: 1-2 OF 2

+ _id: ObjectId('6749e2cb92ab7cf73798f4b5')
  texto: "Esta es una PRUEBA"
  senderId: ""
  estado: ""
  id_ovas: null
  id_usuarios: null
  fecha_creacion: 2024-11-29T15:50:35.415+00:00
  createdAt: 2024-11-29T15:50:35.427+00:00
  updatedAt: 2024-11-29T16:26:01.722+00:00
```

En POSTMAN se presiona el botón SEND y se puede observar que el objeto ha sido eliminado.



The screenshot shows a web browser with a REST client interface. The URL bar displays `http://localhost:3000/comentarios/6749e2cb92ab7cf73798f4b5`. The interface includes tabs for Params, Authorization, Headers (8), Body, Scripts, and Settings. The Body tab is active, showing a JSON response. The response status is 200 OK, with a response time of 98 ms and a size of 489 B. The response body is a JSON object with the following structure:

```
{
  "_id": "6749e2cb92ab7cf73798f4b5",
  "texto": "Esta es una PRUEBA",
  "senderId": "",
  "estado": "",
  "id_ovas": null,
  "id_usuarios": null,
  "fecha_creacion": "2024-11-29T15:50:35.415Z",
  "createdAt": "2024-11-29T15:50:35.427Z",
  "updatedAt": "2024-11-29T16:26:01.722Z",
  "__v": 0
}
```


Etapa 3: Consumo de Datos y Desarrollo Frontend

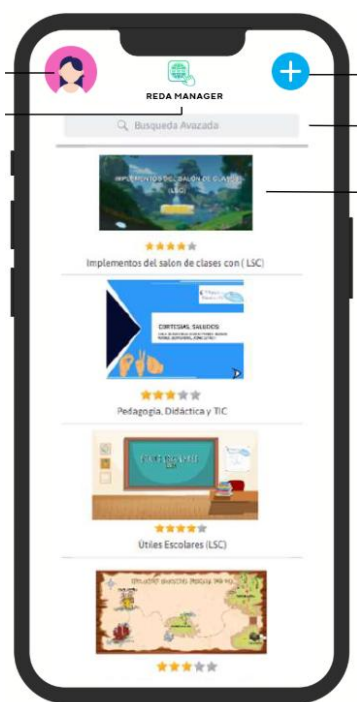
Introducción

En este capítulo exploraremos los fundamentos del **desarrollo frontend**, enfocados en la **creación de interfaces de usuario (UI)** y el **consumo de datos desde fuentes externas** como APIs. Se encontrarán las interfaces funcionales y visualmente claras, integrando datos en tiempo real para mejorar la experiencia del usuario.

También se abordará el diseño del **mapa de navegación**, herramienta clave para estructurar la interacción dentro de una aplicación o sitio web.

Creación de la Interfaz de Usuario (UI)

PANTALLA PRINCIPAL



DESCRIPCIÓN: En la interfaz principal del sistema **REDA Manager** se presenta un diseño intuitivo y funcional. En la parte superior se encuentra el **ícono de usuario**, que muestra la foto de perfil del usuario activo; al hacer clic sobre él, se despliega un menú con opciones como **"Configuración"**, **"Perfil"** y **"Cerrar sesión"**, permitiendo una gestión rápida de la cuenta. A su lado se ubica el **logo de REDA Manager**, que brinda identidad visual a la plataforma. Justo al alcance del usuario, se encuentra el **botón "Agregar REDA"**, diseñado para facilitar la incorporación de nuevos recursos educativos al sistema. Más abajo, destaca un **campo de búsqueda avanzada**, donde los usuarios pueden localizar REDA específicos utilizando palabras clave o etiquetas relevantes. Finalmente, se muestra la **lista de REDAs disponibles**, organizada en forma de tarjetas visuales que representan cada recurso educativo digital abierto, permitiendo al usuario explorar, identificar y acceder fácilmente a los contenidos.

PANTALLA CONFIGURACIÓN



DESCRIPCIÓN: La sección del **ícono de usuario** en la interfaz de **REDA Manager** brinda acceso a información y opciones clave relacionadas con la cuenta del usuario. Al hacer clic sobre este ícono, se despliega un menú con varias funcionalidades. La opción de **"Configuración"** permite personalizar la aplicación de acuerdo con las preferencias del usuario, incluyendo ajustes como el **idioma de la interfaz** y aspectos de **seguridad**, como el cambio de contraseña o la actualización de datos de la cuenta. También se encuentra la opción de **"Notificaciones"**, que informa al usuario sobre la actividad relacionada con los REDA que ha publicado. Finalmente, el menú incluye la opción de **"Cerrar sesión"**, la cual permite finalizar la sesión activa y redirige al usuario a la pantalla de inicio de sesión, garantizando seguridad y control sobre el acceso a la plataforma.

PANTALLA DE SUBIDA



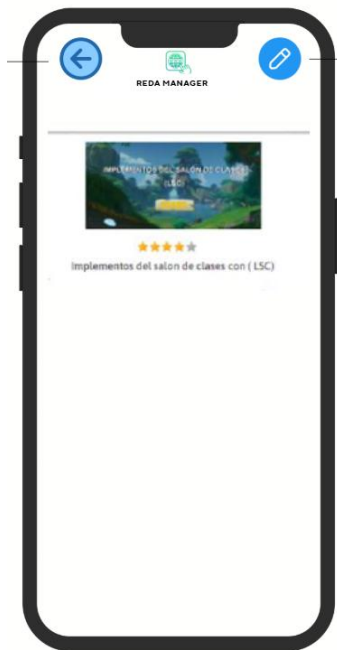
DESCRIPCIÓN: La sección de **carga de archivo** en REDA Manager está diseñada para facilitar al docente la incorporación de nuevos recursos al sistema. Incluye un área intuitiva donde se puede **arrastrar y soltar el archivo REDA** directamente o hacer clic para seleccionarlo desde el dispositivo. A continuación, se encuentra el campo de **descripción del REDA**, un espacio editable donde el docente debe ingresar una breve explicación sobre el contenido del recurso. También se proporciona un campo específico para indicar los **autores**, permitiendo registrar los nombres de quienes participaron en la creación del REDA. Una vez completados todos los campos, el docente puede hacer clic en el botón "**Subir**", el cual ejecuta la acción de cargar el archivo al sistema, finalizando así el proceso de publicación del recurso.

PANTALLA DE REGRESAR-SUBIDO



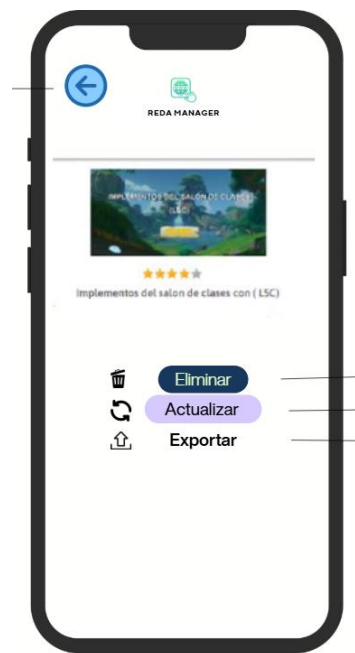
DESCRIPCIÓN: Una vez completado el proceso de carga del recurso, la interfaz presenta un botón "Regresar", que permite al docente volver fácilmente a la pantalla de inicio del sistema. Además, como confirmación visual del éxito en la operación, se muestra un indicador de estado con el mensaje "REDA subido con éxito", acompañado de un ícono de verificación (✓). Este indicador brinda retroalimentación clara e inmediata, asegurando al docente que el archivo ha sido cargado correctamente al sistema.

PANTALLA REGRESAR-EDITAR



En la interfaz, el **botón "Regresar"** ofrece al usuario la posibilidad de volver fácilmente a la **pantalla de inicio**, facilitando la navegación dentro del sistema. Junto a este, se encuentra la opción **"Editar"**, que brinda al docente herramientas para **gestionar sus REDA**. A través de esta función, el docente puede **eliminar** un recurso que ya no sea necesario, **actualizar** su contenido o información, o bien **exportarlo** para su uso fuera del sistema. Esta funcionalidad proporciona un control completo sobre los recursos publicados, permitiendo mantenerlos actualizados y accesibles según las necesidades.

PANTALLA BOTONES ELIMINAR-ACTUALIZAR-EXPORTAR



En la interfaz, el botón **"Regresar"** permite al usuario volver rápidamente a la **pantalla de inicio**, facilitando una navegación fluida dentro del sistema. También se presentan tres botones de acción asociados a la gestión de los recursos: el botón **"Eliminar"** permite **borrar un REDA** del sistema de forma definitiva; el botón **"Actualizar"** redirige al usuario a la pestaña correspondiente para **modificar la información o el archivo** del recurso; y el botón **"Exportar"** brinda la opción de **descargar o extraer el REDA**, permitiendo su uso fuera de la plataforma. Estas opciones proporcionan al docente un control total sobre los recursos que ha creado o administrado.

PANTALLA ACTUALIZAR

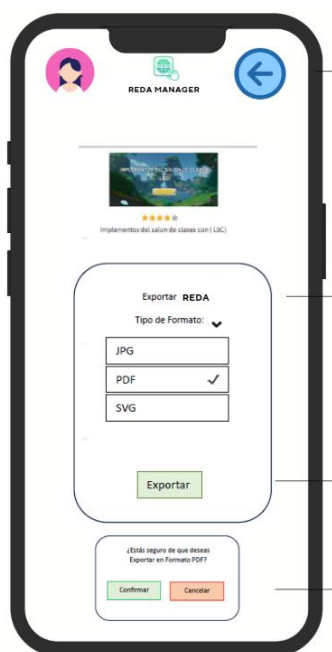


DESCRIPCIÓN: La interfaz de carga de REDA en **REDA Manager** está diseñada para ofrecer una experiencia sencilla y ordenada al usuario. En la parte superior se encuentra el **ícono de usuario**, que muestra la foto de perfil del usuario activo y despliega un menú con opciones como "**Configuración**", "**Perfil**" y "**Cerrar sesión**" al hacer clic sobre él.

En el área principal, se dispone una sección de **carga de archivo**, donde el usuario puede **arrastrar y soltar** el archivo REDA o seleccionarlo desde su dispositivo. A continuación, se encuentran campos editables como la **descripción del REDA**, donde se debe ingresar un resumen del contenido, y el campo de **autores**, destinado a registrar los nombres de quienes elaboraron el recurso.

Una vez completados estos datos, el usuario puede hacer clic en el botón "**Subir**" para cargar el REDA al sistema. Finalmente, se incluye el botón "**Regresar**", que permite volver a la pantalla de inicio en cualquier momento, asegurando una navegación ágil y clara dentro de la plataforma.

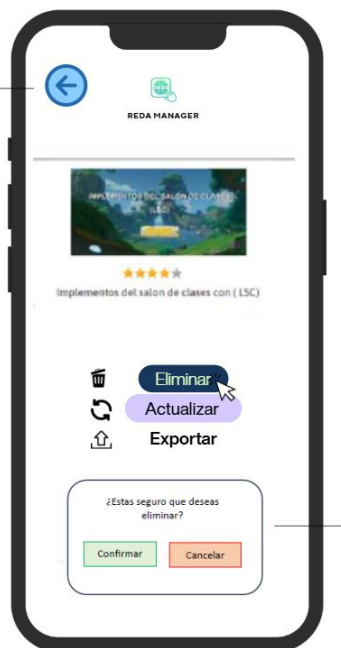
PANTALLA EXPORTAR



DESCRIPCIÓN: La sección de **exportación** en REDA Manager proporciona al usuario una forma rápida y personalizada de descargar sus recursos. En la parte superior se encuentra el **botón "Regresar"**, que permite volver fácilmente a la pantalla de inicio. A continuación, se presenta el **formulario de exportación**, que incluye un **menú desplegable** donde el usuario puede **seleccionar el formato** en el que desea exportar el REDA, como **JPG, PDF o SVG**.

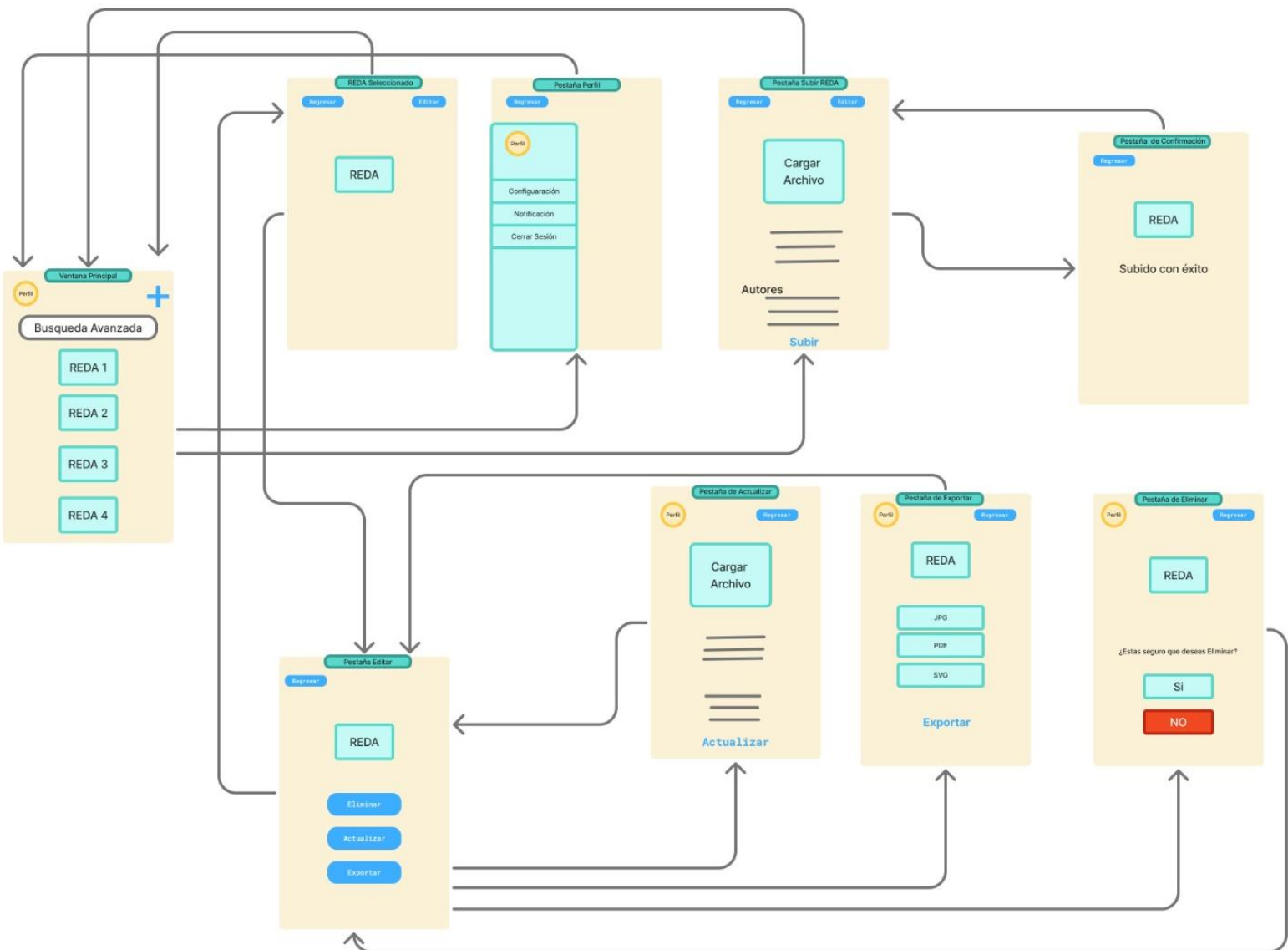
Una vez elegido el formato, el sistema muestra una **ventana de confirmación**, permitiendo al usuario verificar y aprobar su elección antes de continuar. Finalmente, al hacer clic en el botón **"Exportar"**, se genera y descarga automáticamente el archivo en el dispositivo del usuario, completando el proceso de exportación de manera clara y eficiente.

PANTALLA ELIMINAR



En esta sección de la interfaz, el **botón "Regresar"** permite al usuario volver fácilmente a la **pantalla de inicio**, facilitando la navegación dentro del sistema. Al intentar eliminar un OVA, se despliega una **ventana de confirmación**, que actúa como una **pestaña de verificación** para asegurar que el usuario desea proceder con la eliminación del recurso. Esta ventana ofrece un mensaje claro de advertencia y solicita la aprobación final antes de ejecutar la acción, previniendo eliminaciones accidentales y garantizando un control más seguro sobre los contenidos.

Mapa De Navegación



1. Ventana Principal ↔ REDA Seleccionada

- Cuando el usuario hace clic sobre un objeto REDA (REDA 1, 2, 3, 4), se abre la pantalla **REDA Seleccionada**.
- **Conexión bidireccional:** desde REDA Seleccionada se puede regresar a la Ventana Principal.

2. Ventana Principal → Pantalla Subir REDA

- Al hacer clic en el ícono "+", el usuario accede a la pantalla para **crear o subir una nueva REDA**.

3. Pantalla Subir REDA → Pantalla de Confirmación

- Una vez que se cargan los datos y se presiona el botón "**Subir**", se muestra la confirmación de que la REDA fue subida exitosamente.

4. Pantalla de Confirmación → Ventana Principal

- Tras la confirmación del éxito, el sistema **redirecciona automáticamente** de vuelta a la pantalla principal.

5. REDA Seleccionada → Pantalla Editar

- Si el usuario hace clic en el botón "**Editar**", accede a la pantalla de edición, donde puede eliminar, actualizar o exportar la REDA.
- **Conexión bidireccional:** desde Pantalla Editar se puede regresar a REDA Seleccionada.

6. Pantalla Editar → Pantalla de Eliminar

- Si se elige "**Eliminar**", aparece una ventana de confirmación.
- Si el usuario presiona **NO**, regresa a Pantalla Editar.
- Si presiona **SÍ**, regresa a la Ventana Principal.

7. Pantalla Editar → Pantalla de Actualizar

- Al hacer clic en el botón **Actualizar**, se abre la opción para cargar un nuevo archivo y actualizar autores.
- Una vez actualizada, se regresa a la Pantalla Editar.


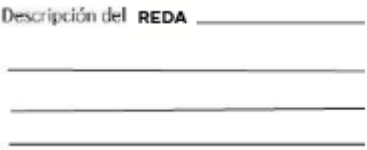
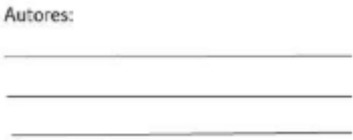
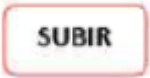

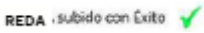




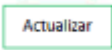
8. Pantalla Editar → Pantalla de Exportar


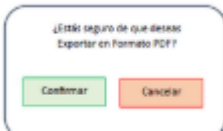
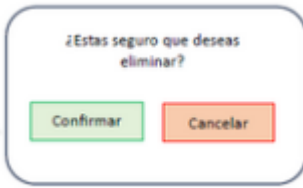
- Al elegir "**Exportar**", el sistema permite elegir entre formatos JPG, PDF y SVG.
- Luego se vuelve automáticamente a Pantalla Editar.

9. Ventana Principal → Menú de Perfil

- Desde el ícono de usuario en la esquina superior derecha, se accede a una ventana de configuración personal:
 - Configuración
 - Notificaciones
 - Cerrar sesión
- **Conexión bidireccional:** se puede regresar a la Ventana Principal.

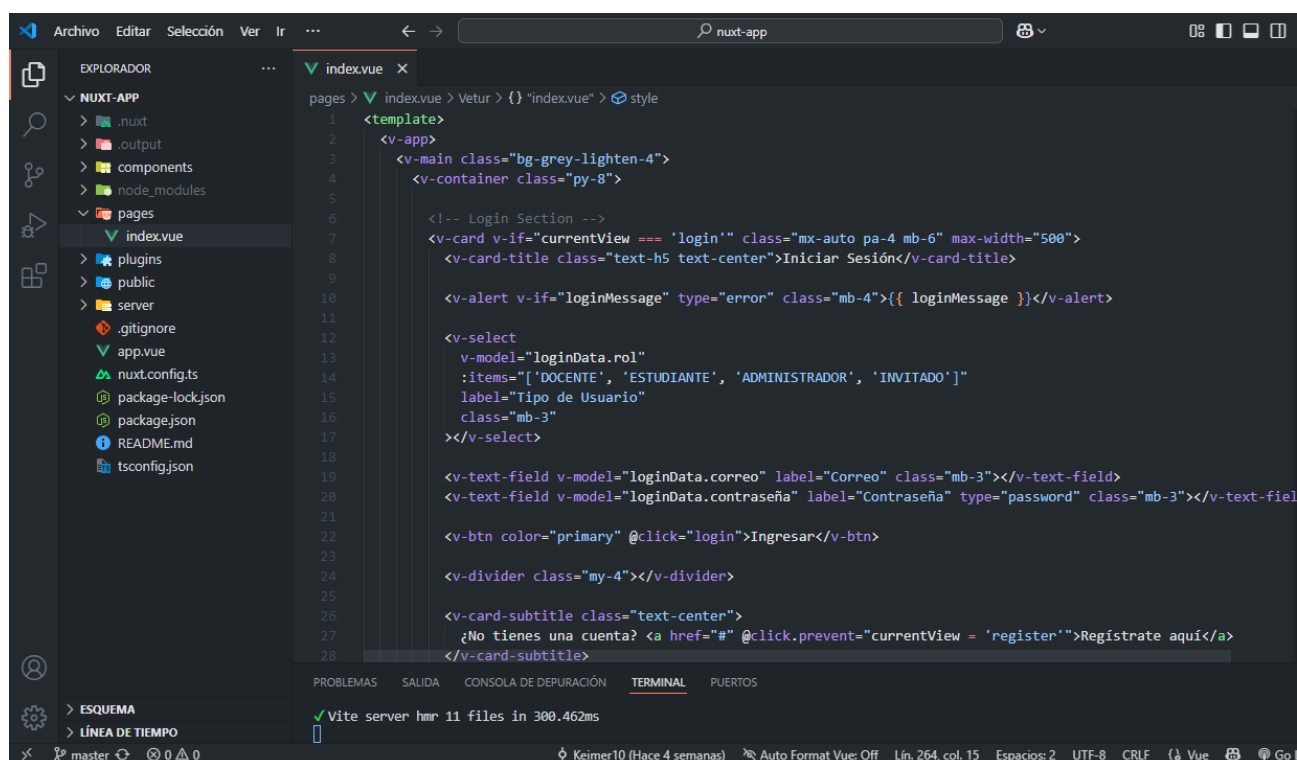
Diseño de guía de metáforas		
Ventana 1		
Nombre	Imagen	Descripción
Icono de usuario		Foto de perfil del usuario activo. Al hacer clic en él, se despliega un menú con opciones como "Configuración", "Perfil", y "Cerrar sesión".
Botón agregar REDA		Permite agregar un REDA nuevo.
Campo de búsqueda avanzada		Permite a los usuarios buscar REDA específicos utilizando palabras clave o etiquetas. Este campo facilita la localización de recursos específicos en el sistema.
Lista de REDA disponibles		Tarjetas que representan los Recursos Educativos Digitales Abiertos (REDA).
Ventana 2		
Icono de usuario		Información del usuario
Configuración		Permite que el usuario acceda a las opciones para personalizar la aplicación según sus preferencias. Esto incluye ajustes como: Idioma: Seleccionar el idioma de la aplicación. Seguridad: Cambiar la contraseña o actualizar la información de la cuenta.
Cerrar sesión		Permite al usuario finalizar su sesión

		actual en la aplicación y será redirigido a la pantalla de inicio de sesión.
Ventana 3		
Carga de archivo		Área donde el docente puede arrastrar y soltar un archivo o hacer clic para seleccionar el archivo REDA desde su dispositivo.
Descripción del REDA		Campo de texto donde el docente debe ingresar una descripción del REDA. Es un área editable para describir el contenido.
Autores		Campo de texto para ingresar los nombres de los autores del REDA.
Botón subir		Botón de acción para cargar el REDA al sistema después de completar los campos.
Ventana 4		
Botón regresar		Permite al docente regresar a la pantalla inicio.
Indicador de estado "REDA subido con éxito"		Informa al docente que el REDA se ha cargado correctamente.
Ventana 5		
Editar		Permite al docente eliminar, actualizar u exportar.
Ventana 6		
Eliminar		Permite eliminar el REDA
Actualizar		Direcciona a la pestaña "Actualizar".
Exportar		Permite exportar el REDA.
Ventana 7		
Actualizar		Permite realizar ajustes al REDA después de haberlo subido.

Formulario de exportación		Selección de Formato: Un menú desplegable donde el usuario selecciona el formato de exportación (por ejemplo, JPG, PDF, SVG).
Pestaña de confirmación		Pestaña de confirmación del formato seleccionado.
Ventana 8		
Pestaña de eliminación del REDA		Pestaña de confirmación de eliminación de REDA.

Diseño de la Interfaz de Usuario (UI) con Vue.js:

El diseño de la interfaz de usuario (UI) se realizó utilizando Vue.js, un framework progresivo de JavaScript que permite crear interfaces interactivas y reactivas. A través de Vue, se implementó una estructura de componentes que facilita la creación de vistas modulares y reutilizables. Se utilizaron directivas de Vue para manejar la interacción con los elementos de la página, como la visualización de datos, el manejo de formularios y la actualización dinámica de la interfaz sin necesidad de recargar la página. Vue también permitió organizar la lógica de los componentes de manera eficiente, lo que facilitó el mantenimiento y la escalabilidad del proyecto. El uso de Vue.js hizo que la interfaz fuera altamente dinámica, respondiendo de inmediato a las acciones del usuario, como la actualización de contenido, validación de formularios y la navegación entre diferentes vistas.

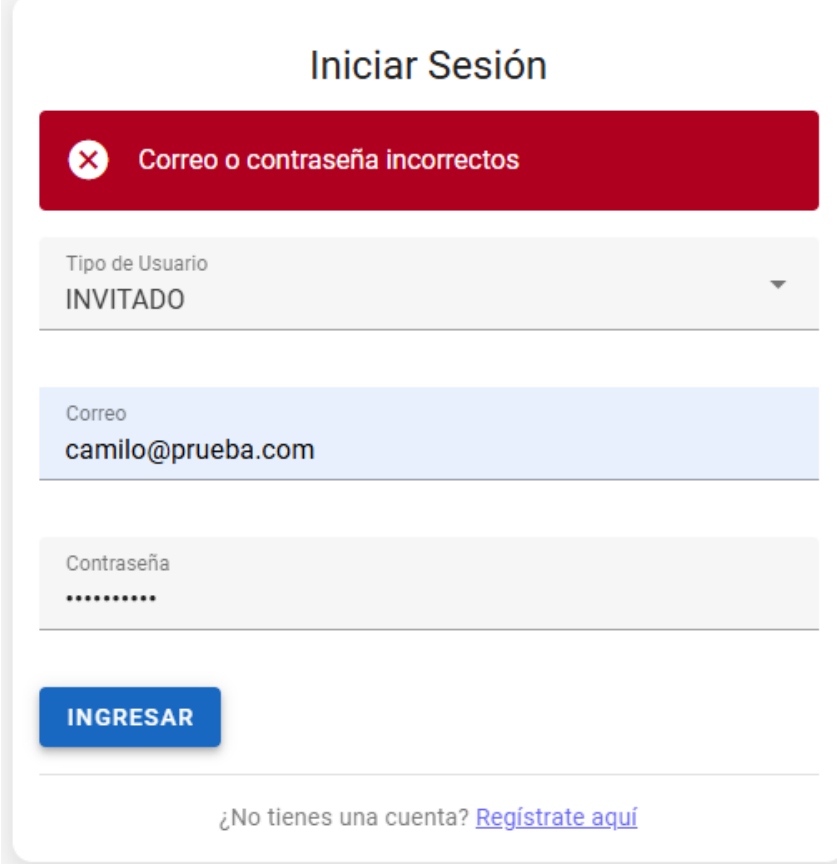


```
1 <template>
2 <v-app>
3   <v-main class="bg-grey-lighten-4">
4     <v-container class="py-8">
5
6       <!-- Login Section -->
7       <v-card v-if="currentView === 'login'" class="mx-auto pa-4 mb-6 max-width=500">
8         <v-card-title class="text-h5 text-center">Iniciar Sesión</v-card-title>
9
10        <v-alert v-if="loginMessage" type="error" class="mb-4">{{ loginMessage }}</v-alert>
11
12        <v-select
13          v-model="loginData.rol"
14          :items=["DOCENTE", "ESTUDIANTE", "ADMINISTRADOR", "INVITADO"]
15          label="Tipo de Usuario"
16          class="mb-3"
17        ></v-select>
18
19        <v-text-field v-model="loginData.correo" label="Correo" class="mb-3"></v-text-field>
20        <v-text-field v-model="loginData.contraseña" label="Contraseña" type="password" class="mb-3"></v-text-field>
21
22        <v-btn color="primary" @click="login">Ingresar</v-btn>
23
24        <v-divider class="my-4"></v-divider>
25
26        <v-card-subtitle class="text-center">
27          ¿No tienes una cuenta? <a href="#" @click.prevent="currentView = 'register'">Regístrate aquí</a>
28        </v-card-subtitle>
29      </v-container>
30    </v-main>
31  </v-app>
32</template>
```

Código de interfaz en Vue.js

Consideraciones de Usabilidad:

Durante el desarrollo de la interfaz con Vue.js, se tuvieron en cuenta diversos aspectos de usabilidad para garantizar que la aplicación fuera intuitiva, accesible y fácil de usar. Se priorizó un diseño de navegación clara y sencilla, con un enfoque en la organización de los elementos para evitar la sobrecarga de información. Se incorporaron elementos de retroalimentación visual en tiempo real, como mensajes de error al llenar formularios incorrectamente o indicadores de carga, lo que ayuda a que los usuarios comprendan las acciones realizadas. Además, la interfaz fue diseñada para ser accesible, con la implementación de colores contrastantes y la inclusión de textos alternativos en las imágenes, mejorando la experiencia de los usuarios con discapacidades visuales.

La imagen muestra una interfaz de usuario para iniciar sesión. En la parte superior, el título "Iniciar Sesión" está centrado. Justo debajo, una barra roja prominente contiene un ícono de "X" blanca y el texto "Correo o contraseña incorrectos". A continuación, hay un campo desplegable para "Tipo de Usuario" con "INVITADO" seleccionado. Debajo de eso, un campo de texto azul claro para "Correo" contiene el email "camilo@prueba.com". Luego, un campo gris para "Contraseña" muestra caracteres ocultos por puntos. Un botón azul con el texto "INGRESAR" está situado debajo de los campos. En la parte inferior, un enlace azul dice "¿No tienes una cuenta? Regístrate aquí".

Mensaje de error al colocar datos inválidos

Programación Frontend con Vue.js:

El frontend fue desarrollado completamente en Vue.js, lo que permitió crear una interfaz reactiva y eficiente. Se utilizó el enlace de datos bidireccional proporcionado por Vue para asegurar que las modificaciones realizadas en los campos de entrada de los formularios se reflejaran inmediatamente en la interfaz. Además, se emplearon eventos en Vue.js para manejar interacciones del usuario, como clics en botones o cambios en los formularios, lo que facilitó la actualización dinámica del contenido sin necesidad de recargar la página. Vue.js también permitió la creación de componentes reutilizables para las vistas, lo que mejoró la eficiencia en el desarrollo y facilitó la gestión del código.


```

6      <!-- Login Section -->
7      <v-card v-if="currentView === 'login'" class="mx-auto pa-4 mb-6" max-width="500">
8          <v-card-title class="text-h5 text-center">Iniciar Sesión</v-card-title>
9
10         <v-alert v-if="loginMessage" type="error" class="mb-4">{{ loginMessage }}</v-alert>
11
12         <v-select
13             v-model="loginData.rol"
14             :items="['DOCENTE', 'ESTUDIANTE', 'ADMINISTRADOR', 'INVITADO']"
15             label="Tipo de Usuario"
16             class="mb-3"
17         ></v-select>
18
19         <v-text-field v-model="loginData.correo" label="Correo" class="mb-3"></v-text-field>
20         <v-text-field v-model="loginData.contraseña" label="Contraseña" type="password" class="mb-3"></v-text-field>
21
22         <v-btn color="primary" @click="login">Ingresar</v-btn>
23
24         <v-divider class="my-4"></v-divider>
25
26         <v-card-subtitle class="text-center">
27             ¿No tienes una cuenta? <a href="#" @click.prevent="currentView = 'register'">Regístrate aquí</a>
28         </v-card-subtitle>
29     </v-card>

```

Apartado del login en Vue.js

```

55     <!-- Dashboard para DOCENTE -->
56     <div v-if="currentView === 'dashboard' && userRole === 'DOCENTE'">
57         <v-card class="pa-4 mb-6">
58             <v-card-title>Subir REDA - DOCENTE</v-card-title>
59             <v-text-field v-model="reda.title" label="Título del archivo" class="mb-3"></v-text-field>
60             <v-textarea v-model="reda.description" label="Descripción" class="mb-3"></v-textarea>
61             <v-text-field v-model="reda.author" label="Autor" class="mb-3"></v-text-field>
62             <v-file-input
63                 v-model="reda.file"
64                 label="Seleccionar archivo"
65                 class="mb-3"
66                 accept="*/*"
67                 show-size
68                 prepend-icon="mdi-upload"
69             ></v-file-input>
70             <v-btn color="primary" @click="uploadReda">Subir REDA</v-btn>
71         </v-card>

```

Apartado del docente en Vue.js

```

103     <!-- Dashboard para ESTUDIANTE -->
104     <div v-if="currentView === 'dashboard' && userRole === 'ESTUDIANTE'">
105         <v-card class="pa-4 mb-6">
106             <v-card-title>Archivos Subidos por Docentes</v-card-title>
107
108             <v-alert v-if="redaMessage" type="info" class="mb-4">{{ redaMessage }}</v-alert>
109
110             <v-row dense>
111                 <v-col cols="12" md="4" v-for="(reda, index) in redaList" :key="index">
112                     <v-card class="mb-4 pa-4">
113                         <v-card-title>{{ reda.nombre }}</v-card-title>
114                         <v-card-subtitle>{{ reda.descripcion }}</v-card-subtitle>
115                         <v-card-text>
116                             <strong>Autor:</strong> {{ reda.id_autor }}<br />
117                             <strong>Formato:</strong> {{ reda.formato }}
118                         </v-card-text>

```

Apartado del estudiante en Vue.js

Desarrollo de la Lógica del Frontend:

La lógica del frontend fue implementada en Vue.js utilizando su sistema de gestión de estado y la reactividad de los datos. Cada componente de la interfaz interactuaba con un estado global controlado por Vuex, lo que permitió gestionar de manera centralizada los datos de la aplicación, como los recursos educativos (OVA). Se crearon métodos y funciones para manejar la lógica de la validación de formularios, el procesamiento de datos del usuario y la navegación entre las diferentes vistas de la aplicación. Esta estructura centralizada permitió que los cambios realizados en el estado se reflejaran instantáneamente en la interfaz sin necesidad de intervención manual.

```
120 <!-- Mostrar el archivo según el tipo MIME -->
121 <div v-if="reda.formato && reda.formato.startsWith('image/')">
122   
123 </div>
124 <div v-else-if="reda.formato && reda.formato.startsWith('video/')">
125   <video :src="getFileUrl(reda.ruta)" controls style="max-width: 100%; height: auto;"></video>
126 </div>
127 <div v-else-if="reda.formato === 'application/pdf'">
128   <iframe :src="getFileUrl(reda.ruta)" style="width: 100%; height: 500px;" frameborder="0"></iframe>
129 </div>
130 <div v-else>
131   <!-- Para otros tipos, mostrar link para descargar o abrir -->
132   <a :href="getFileUrl(reda.ruta)" target="_blank" download>Descargar archivo</a>
133 </div>
134 </v-card>
135 </v-col>
136 </v-row>
137
138 </v-card>
139 </div>
```

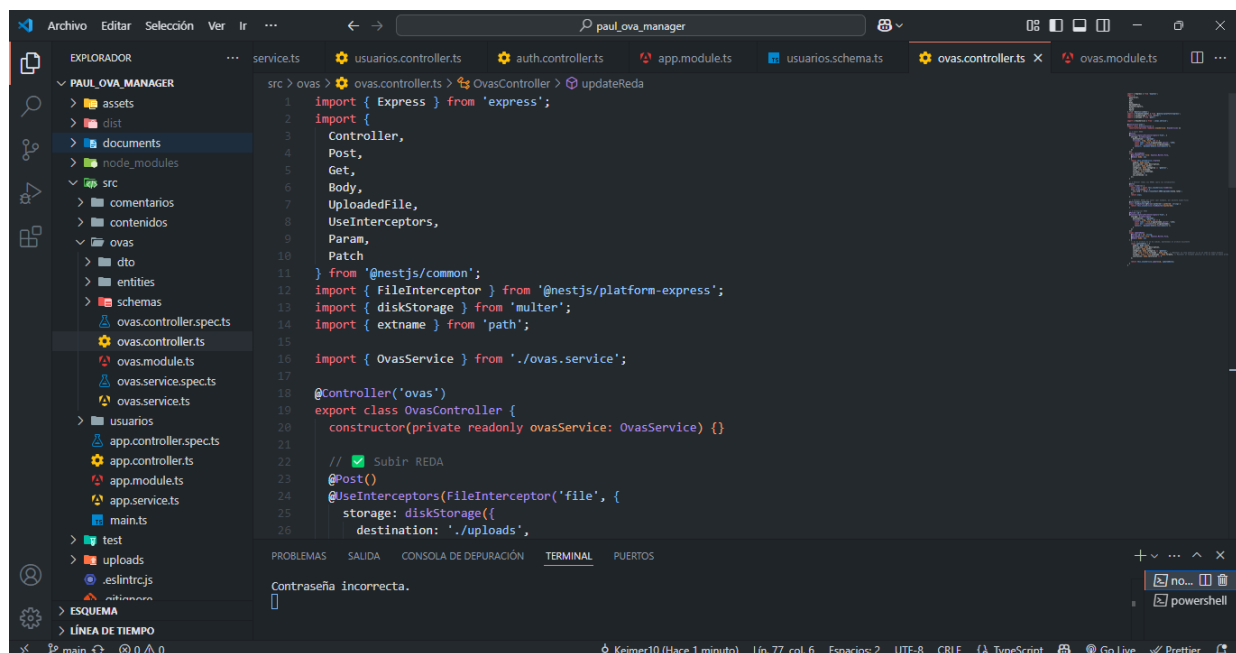
Fragmento del código del frontend en Vue.js

Uso de Bibliotecas y Frameworks (si aplicable):

Para optimizar el desarrollo, se integraron varias bibliotecas y frameworks junto con Vue.js. Entre ellos, se utilizó Vue Router para la gestión de rutas dentro de la aplicación, permitiendo la navegación entre diferentes vistas sin necesidad de recargar la página. Además, Axios fue empleado para hacer solicitudes HTTP al backend de NestJS, facilitando la obtención de datos del servidor y su presentación en la interfaz de usuario. Estas herramientas fueron esenciales para mejorar la eficiencia del desarrollo y garantizar que la aplicación fuera fácil de mantener y ampliar en el futuro.

Consumo de Datos desde el Backend (NestJS y MongoDB):

Se configuró el frontend en Vue.js para consumir datos de un backend construido con NestJS. NestJS, como un marco de trabajo basado en Node.js, se utilizó para desarrollar una API RESTful que se encargó de gestionar las solicitudes del cliente. A través de Axios, el frontend realizaba solicitudes HTTP al servidor para obtener datos relacionados con los OVA, como los títulos, descripciones y archivos asociados. Estos datos fueron procesados y presentados en la interfaz de usuario de forma dinámica. El backend de NestJS interactuaba con MongoDB, una base de datos NoSQL, para almacenar y recuperar los OVA de forma eficiente, aprovechando las ventajas de MongoDB en la gestión de grandes volúmenes de datos no estructurados.



```
src > ovas > OvasController > updateReda
1  import { Express } from 'express';
2  import {
3    Controller,
4    Post,
5    Get,
6    Body,
7    UploadedFile,
8    UseInterceptors,
9    Param,
10   Patch
11 } from '@nestjs/common';
12 import { FileInterceptor } from '@nestjs/platform-express';
13 import { diskStorage } from 'multer';
14 import { extname } from 'path';
15
16 import { OvasService } from './ovas.service';
17
18 @Controller('ovas')
19 export class OvasController {
20   constructor(private readonly ovasService: OvasService) {}
21
22   // Subir REDA
23   @Post()
24   @UseInterceptors(FileInterceptor('file', {
25     storage: diskStorage({
26       destination: './uploads',
```

Fragmento del código del backend

Configuración de Conexiones al Backend:

La configuración de la conexión al backend se realizó utilizando la tecnología de APIs RESTful. El frontend en Vue.js se comunicaba con el backend de NestJS a través de peticiones HTTP (GET, POST, PUT, DELETE) enviadas desde Axios. Estas solicitudes permitieron al frontend obtener los datos más recientes de MongoDB, y también permitir que el frontend enviara nuevas entradas de datos (como un OVA subido por un usuario). En términos de seguridad, se configuró la autenticación utilizando JWT (JSON Web Tokens), que garantizó que solo los usuarios autenticados pudieran acceder a las funcionalidades de la plataforma.

```
src > ovas > ovas.controller.ts > OvasController > updateReda
1  import { Express } from 'express';
2  import {
3    Controller,
4    Post,
5    Get,
6    Body,
7    UploadedFile,
8    UseInterceptors,
9    Param,
10   Patch
11 } from '@nestjs/common';
12 import { FileInterceptor } from '@nestjs/platform-express';
13 import { diskStorage } from 'multer';
14 import { extname } from 'path';
15
16 import { OvasService } from '../ovas.service';
```

Importación de peticiones en el backend

Obtención y Presentación de Datos:

En esta fase, el frontend de Vue.js se encargó de obtener los datos del servidor (almacenados en MongoDB) a través de las APIs expuestas por el backend de NestJS. Los datos fueron solicitados en formato JSON y presentados dinámicamente en la interfaz de usuario. Por ejemplo, cuando un usuario buscaba un OVA específico, se enviaba una solicitud al backend, que recuperaba los datos desde MongoDB y los enviaba de vuelta al frontend para ser mostrados en la página. Este enfoque permitió una experiencia fluida, donde los datos se actualizaban instantáneamente sin necesidad de recargar la página.

```

178 const login = async () => {
179   loginMessage.value = '';
180   try {
181     const res = await axios.post('http://localhost:3002/auth/login', loginData.value);
182     userRole.value = res.data.rol;
183     currentView.value = 'dashboard';
184
185     if (userRole.value === 'ESTUDIANTE') {
186       fetchRedas();
187     }
188
189   } catch (err) {
190     loginMessage.value = err.response?.data.message || 'Error al iniciar sesión';
191   }
192 };
193

```

Conexión del backend a través del puerto 3002