



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 Ova – Fase I

Keimer Enrique Muñoz Mora

Luis Carlos Suárez Bravo

José Isaac Chantak Zabaleta

Natalia Sofia Cochero Paternina

Tutor: Alexander Toscano



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 Objetos Virtuales de Aprendizaje (OVA), 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 OVA, 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.....	7
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.....	26
REQUISITOS NO FUNCIONALES	32
REQUISITOS DE DESEMPEÑO.....	32
REQUISITOS DE SEGURIDAD.....	32
REQUISITOS DE USABILIDAD	32
REQUISITOS DE ESCALABILIDAD	32
MODELADO E/R.....	32
DIAGRAMA DE ENTIDAD-RELACIÓN	36
DESCRIPCIÓN DE ENTIDADES Y RELACIONES.....	34
REGLAS DE INTEGRIDAD.....	34
ANEXOS (SI ES NECESARIO).....	37
DIAGRAMAS ADICIONALES	¡ERROR! MARCADOR NO DEFINIDO.
REFERENCIAS	¡ERROR! MARCADOR NO DEFINIDO.

ETAPA 2: PERSISTENCIA DE DATOS CON BACKEND	38
INTRODUCCIÓN	38
PROPÓSITO DE LA ETAPA	38
ALCANCE DE LA ETAPA	38
DEFINICIONES Y ACRÓNIMOS	38
DISEÑO DE LA ARQUITECTURA DE BACKEND.....	38
DESCRIPCIÓN DE LA ARQUITECTURA PROPUESTA	39
COMPONENTES DEL BACKEND.....	39
DIAGRAMAS DE ARQUITECTURA	39
ELECCIÓN DE LA BASE DE DATOS	40
EVALUACIÓN DE OPCIONES (SQL o NoSQL).....	41
JUSTIFICACIÓN DE LA ELECCIÓN.....	41
DISEÑO DE ESQUEMA DE BASE DE DATOS	42
IMPLEMENTACIÓN DEL BACKEND.....	43
ELECCIÓN DEL LENGUAJE DE PROGRAMACIÓN.....	45
CREACIÓN DE LA LÓGICA DE NEGOCIO	45
DESARROLLO DE ENDPOINTS Y APIS.....	46
AUTENTICACIÓN Y AUTORIZACIÓN	46
CONEXIÓN A LA BASE DE DATOS	47
CONFIGURACIÓN DE LA CONEXIÓN	47
DESARROLLO DE OPERACIONES CRUD.....	47
MANEJO DE TRANSACCIONES	47
PRUEBAS DEL BACKEND	52
DISEÑO DE CASOS DE PRUEBA.....	52
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	59
INTRODUCCIÓN	59
PROPÓSITO DE LA ETAPA	59
ALCANCE DE LA ETAPA	59
DEFINICIONES Y ACRÓNIMOS	59
CREACIÓN DE LA INTERFAZ DE USUARIO (UI)	59
DISEÑO DE LA INTERFAZ DE USUARIO (UI) CON HTML Y CSS	59
CONSIDERACIONES DE USABILIDAD	59
MAQUETACIÓN RESPONSIVA	59
PROGRAMACIÓN FRONTEND CON JAVASCRIPT (JS).....	59
DESARROLLO DE LA LÓGICA DEL FRONTEND	59
MANEJO DE EVENTOS Y COMPORTAMIENTOS DINÁMICOS	60
USO DE BIBLIOTECAS Y FRAMEWORKS (SI APLICABLE)	60
CONSUMO DE DATOS DESDE EL BACKEND.....	60
CONFIGURACIÓN DE CONEXIONES AL BACKEND	60
OBTENCIÓN Y PRESENTACIÓN DE DATOS	60
ACTUALIZACIÓN EN TIEMPO REAL (SI APLICABLE)	60
INTERACCIÓN USUARIO-INTERFAZ	60
MANEJO DE FORMULARIOS Y VALIDACIÓN DE DATOS	60
IMPLEMENTACIÓN DE FUNCIONALIDADES INTERACTIVAS	60
MEJORAS EN LA EXPERIENCIA DEL USUARIO	60
PRUEBAS Y DEPURACIÓN DEL FRONTEND	60
DISEÑO DE CASOS DE PRUEBA DE FRONTEND	60
PRUEBAS DE USABILIDAD.....	61
DEPURACIÓN DE ERRORES Y OPTIMIZACIÓN DEL CÓDIGO.....	61
IMPLEMENTACIÓN DE LA LÓGICA DE NEGOCIO EN EL FRONTEND	61
MIGRACIÓN DE LA LÓGICA DE NEGOCIO DESDE EL BACKEND (SI NECESARIO)	61
VALIDACIÓN DE DATOS Y REGLAS DE NEGOCIO EN EL FRONTEND	61
INTEGRACIÓN CON EL BACKEND	61
VERIFICACIÓN DE LA COMUNICACIÓN EFECTIVA CON EL BACKEND	61
PRUEBAS DE INTEGRACIÓN FRONTEND-BACKEND	61

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, OVA, 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 trabajara 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 Objetos Virtuales de Aprendizaje (OVA). Este sistema permite administrar los OVA 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 OVA.
- Visualizar un OVA.
- Actualizar un OVA.
- Eliminar un OVA.
- Buscar un OVA.
- Exportar OVA.
- Valorar OVA.

Funcionalidades Futuras:

- Interoperabilidad con Herramientas de Autoría
- Importar OVA 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: Asincrónico 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 entregarlo 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.

OVA: Objetos Virtuales de Aprendizaje (Virtual Learning Objects)

Un OVA (Objeto Virtual de Aprendizaje), es un recurso educativo interactivo en línea, que facilita el aprendizaje. Está compuesto por elementos como texto, imágenes y videos. Puede distribuirse con documentos HTML para fácil acceso en navegadores web estándar.

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 OVA: (OVA Management)

La gestión del OVA se refiere al conjunto de procesos y estrategias utilizados para administrar y evaluar los recursos educativos digitales conocidos como Objetos Virtuales de Aprendizaje. Esto incluye la creación, distribución, actualización, y seguimiento de los OVAs para garantizar su efectividad en el proceso de enseñanza y aprendizaje.

VALORACIÓN DE OVA: (OVA Assessment)

La valoración de un OVA 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 OVA 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 Objetos Virtuales de Aprendizaje (OVA) 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 OVA 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 OVA: Permite a los usuarios cargar nuevos Objetos Virtuales de Aprendizaje 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 OVA, como la página de inicio, los contenidos y las evaluaciones, se registren correctamente.

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

Actualizar un OVA: Permite a los usuarios modificar o actualizar el contenido de un OVA 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 OVA se mantengan actuales y relevantes.

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

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

Exportar OVA: Permite a los usuarios exportar OVA 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 OVA en diferentes plataformas y entornos de aprendizaje.

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

Usuarios del Sistema

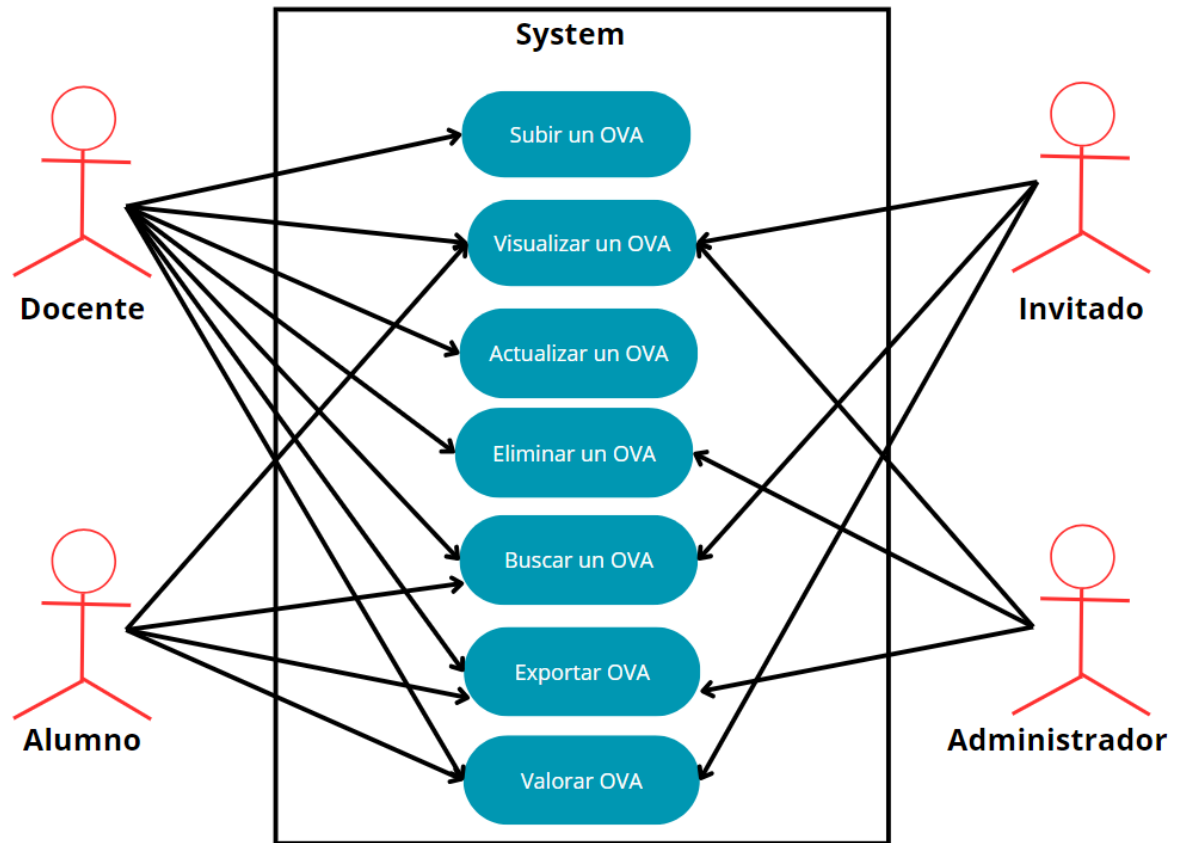
Los siguientes usuarios pueden interactuar con la pizarra dependiendo de las funcionalidades.

Funcionalidad	Administradores	Docente	Alumno	Invitado
Subir OVA		✓		
Visualizar OVA	✓	✓	✓	✓
Actualizar OVA		✓		
Eliminar OVA	✓	✓		
Buscar OVA		✓	✓	✓
Exportar OVA	✓	✓	✓	
Valorar OVA		✓	✓	✓

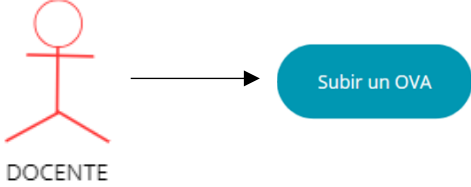
Restricciones

Requisitos Funcionales

Casos de Uso



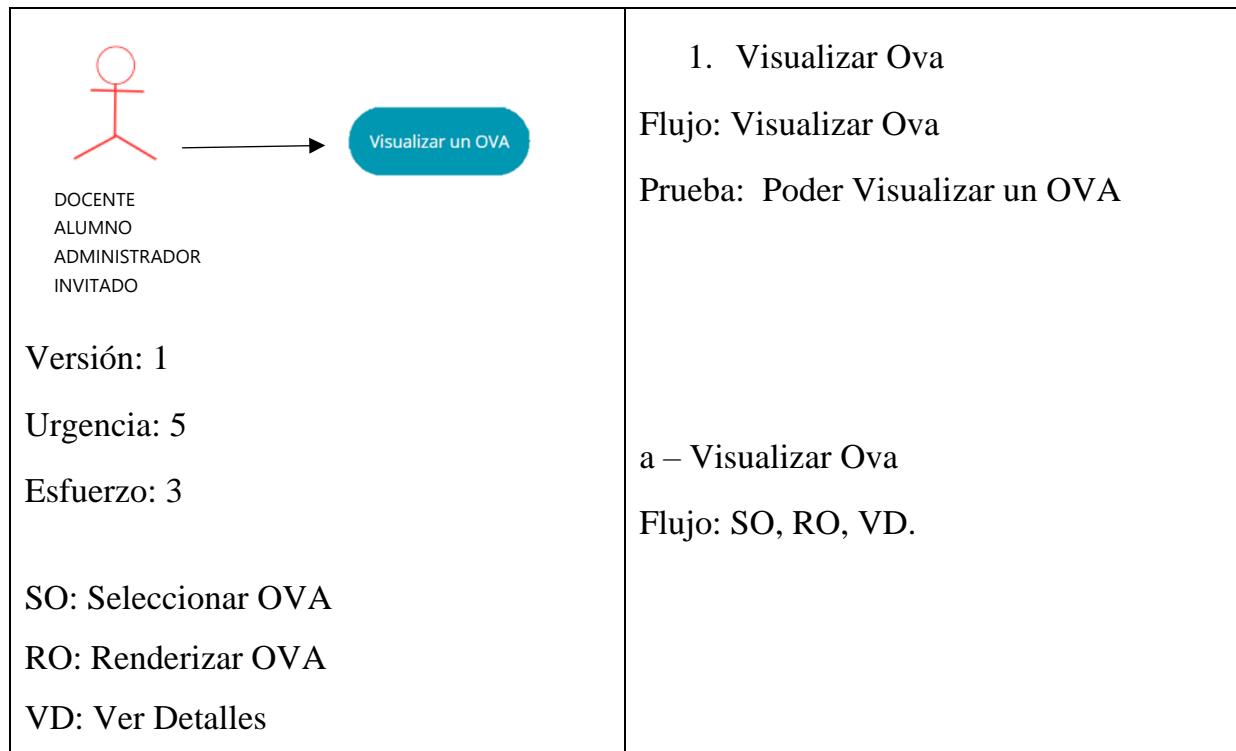
CASO No. 1 Subir Ova

 <p>Versión: 1</p> <p>Urgencia: 5</p> <p>Esfuerzo: 5</p> <p>SO: Subir Ova</p> <p>SA: Seleccionar Archivo</p> <p>AN: Asignar Nombre</p> <p>DO: Descripción Ova</p> <p>CA: Colocar Autor</p> <p>GO: Guardar Ova</p>	<p>1. Subir Ova</p> <p>Flujo: Subir Ova</p> <p>Prueba: Subir Ova, Seleccionar Ova, Cargar Ova, asignar descripción, y guardar Ova.</p> <p>a – Subir Ova</p> <p>Flujo: SO, SA, AN, CO, CA</p>
--	--

ID:	CDU-1
Nombre	Subir Ova
Actores	Docente
Objetivo	Permitir cargar el Ova
Urgencia	5
Esfuerzo	5

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

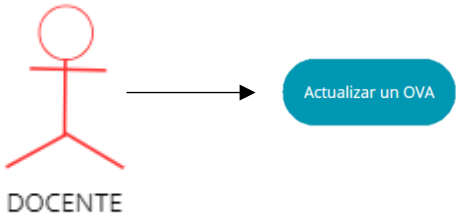
CASO No. 2 Visualizar Ova



ID:	CDU-2	
Nombre	Visualizar Ova	
Actores	Docente, Alumno, Administrador, Invitado.	
Objetivo	Permitir visualizar el Ova	
Urgencia	5	
Esfuerzo	3	
Precondiciones	El debe estar disponible en el sistema	
Flujo normal	Docente	Sistema
	Selecciona visualizar un Ova	
		Muestra lista de Ovas disponibles
	Selecciona el Ova deseado	
		Carga y muestra el contenido del Ova
	Selecciona Ver Detalles	
		Muestra los detalles del OVA

Flujo alternativo 1	Selecciona visualizar Ova	
		Muestra mensaje de error si no hay conexión
	Usuario reintenta la acción	

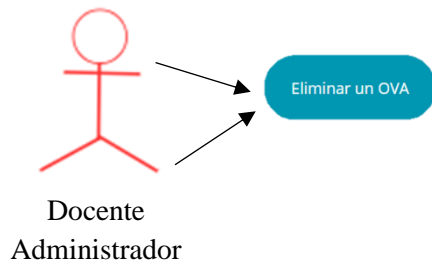
CASO No. 3 Actualizar Ova

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

ID:	CDU-3	
Nombre	Actualizar OVA	
Actores	Docente	
Objetivo	Permitir actualizar la información del OVA	
Urgencia	4	
Esfuerzo	4	
Precondiciones	El OVA debe existir en el sistema	
Flujo normal	Docente	Sistema
	Actualizar OVA	

		Muestra lista de OVA disponibles
	Selecciona el OVA a actualizar	
		Carga la información actual del OVA
	Modifica la información necesaria	
		Guarda los cambios realizados
		Mensaje de OVA actualizado con Éxito.
Flujo alternativo 1	Actualizar OVA	
		Muestra mensaje de error si no hay conexión
	Docente reintenta la acción	

CASO No. 4 Eliminar Ova



Versión: 1

Urgencia: 3

Esfuerzo: 3

SO: Seleccionar OVA

SE: Seleccionar Eliminar

EO: Eliminar Ova

1. Eliminar Ova

Flujo: Eliminar Ova

Prueba: Borrar Ova de la pantalla de visualización

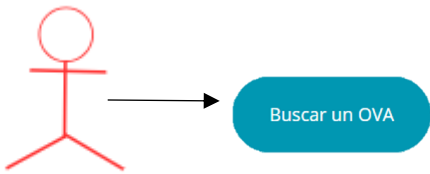
a – Eliminar Ova

Flujo: SO, SE, EO.

ID:	CDU-4	
Nombre	Eliminar Ova	
Actores	Docente, Administrador	
Objetivo	Permitir eliminar un OVA del sistema	
Urgencia	3	
Esfuerzo	3	
Precondiciones	El OVA debe existir en el sistema	
Flujo normal	Docente	Sistema
	Selecciona eliminar OVA	
		Muestra lista de Ovas disponibles
	Selecciona el Ova a eliminar	

		Solicita confirmación de eliminación
	Docente confirma eliminación	
		Confirma la acción
		Mensaje de OVA eliminado correctamente
Flujo alternativo 1	Selecciona eliminar Ova	
		Muestra la lista de Ovas disponible
	Selecciona el Ova a eliminar	
		Muestra mensaje de confirmación
	Docente presiona cancelar	
		Cancela la acción
		Mensaje de OVA no eliminado

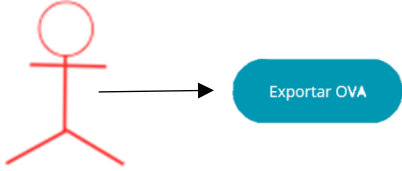
CASO No. 5 Buscar un Ova

 <p>DOCENTE ALUMNO INVITADO</p> <p>Versión: 1 Urgencia: 4 Esfuerzo: 3</p> <p>CB: Criterios de Búsqueda</p>	<p>1. Buscar OVA</p> <p>Flujo: Buscar OVA</p> <p>Prueba: Se debe poder buscar OVA relacionado al Criterio que se introdujo</p> <p>a – Buscar Ova</p> <p>Flujo: CB, BO.</p>
---	--

BO: Buscar OVA	
----------------	--

ID:	CDU-5	
Nombre	Buscar OVA	
Actores	Docente, Estudiante, Invitado	
Objetivo	Permitir buscar OVAs 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

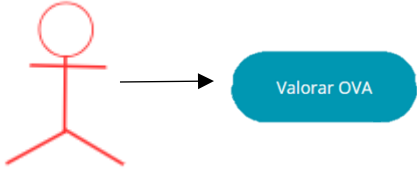
CASO No. 6 Exportar Ova

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

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

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

CASO No. 7 Valorar Ova

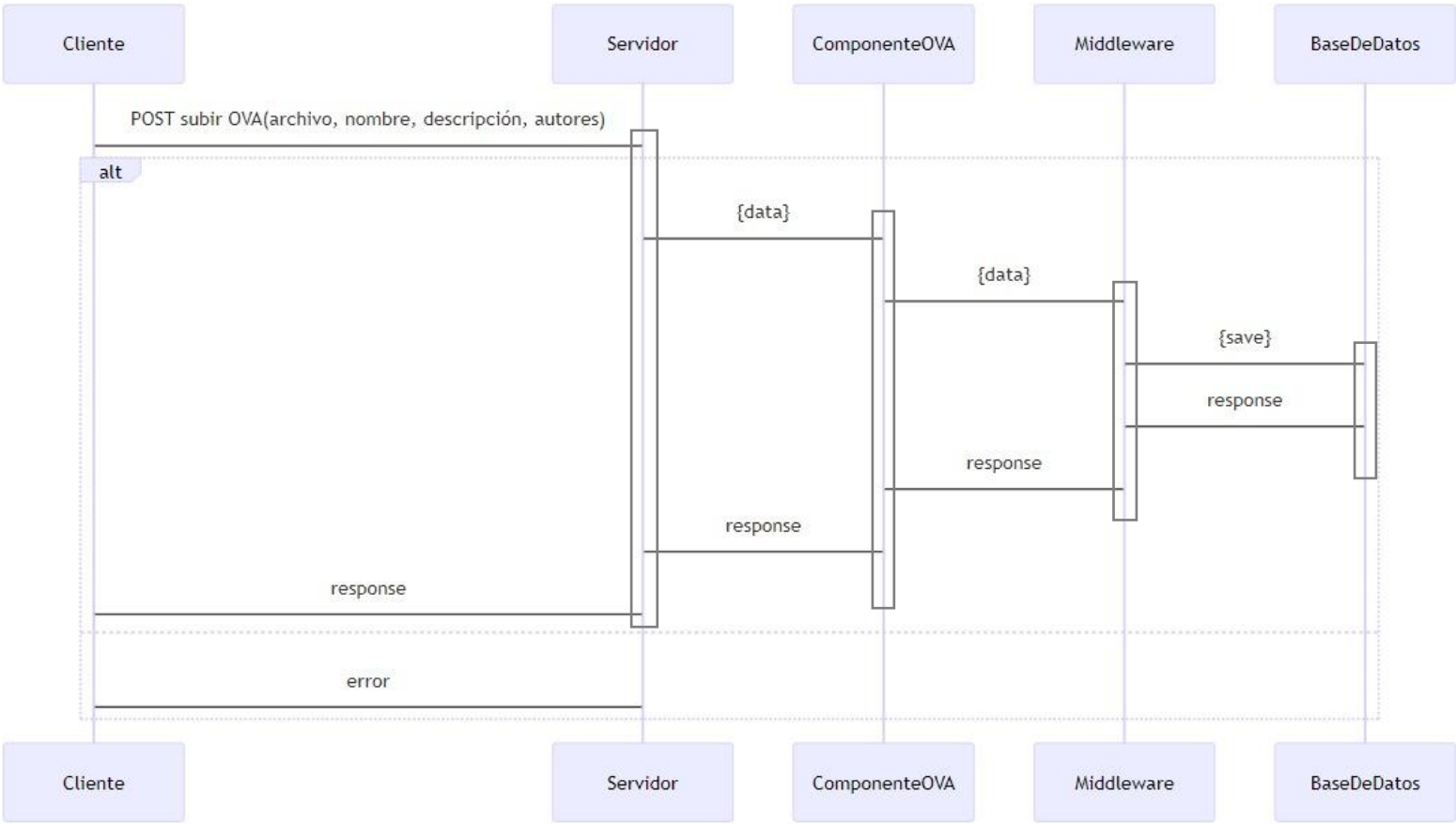
 <p>DOCENTE ALUMNO INVITADO</p> <p>Versión: 1 Urgencia: 2 Esfuerzo: 2</p> <p>VO: Valorar Ova</p>	<p>1. Valorar Ova</p> <p>Flujo: Valorar Ova</p> <p>Prueba: Puntuar ova de acuerdo con la valoración que se decida.</p> <p>a – Valorar Ova</p> <p>Flujo: VO</p>
---	--

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

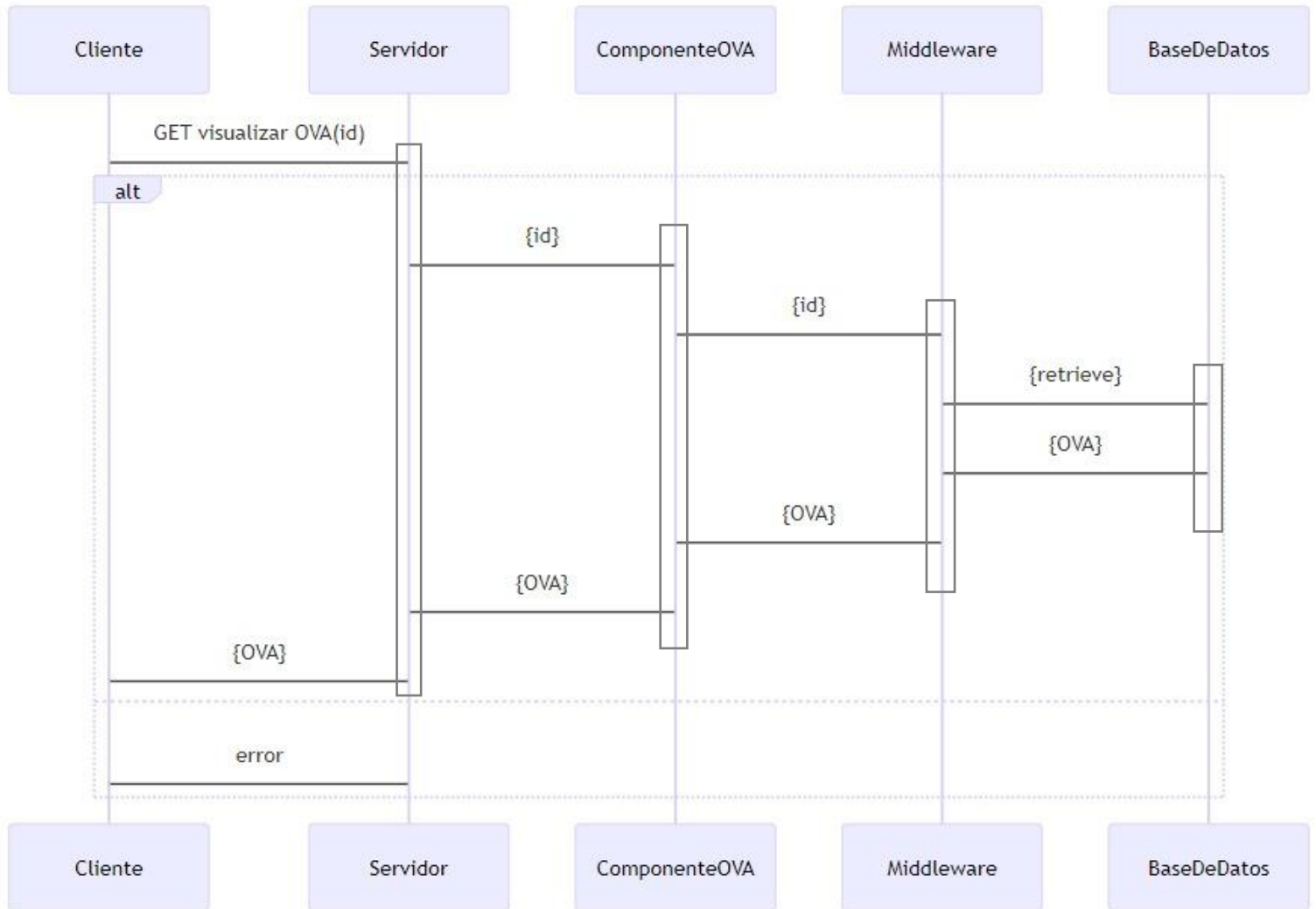
		Mensaje de valoración guardada correctamente
Flujo alternativo 1	Usuario selecciona el OVA a valorar	
		Muestra mensaje de error si no hay conexión
	Usuario reintenta la acción	

Diagramas de Secuencia

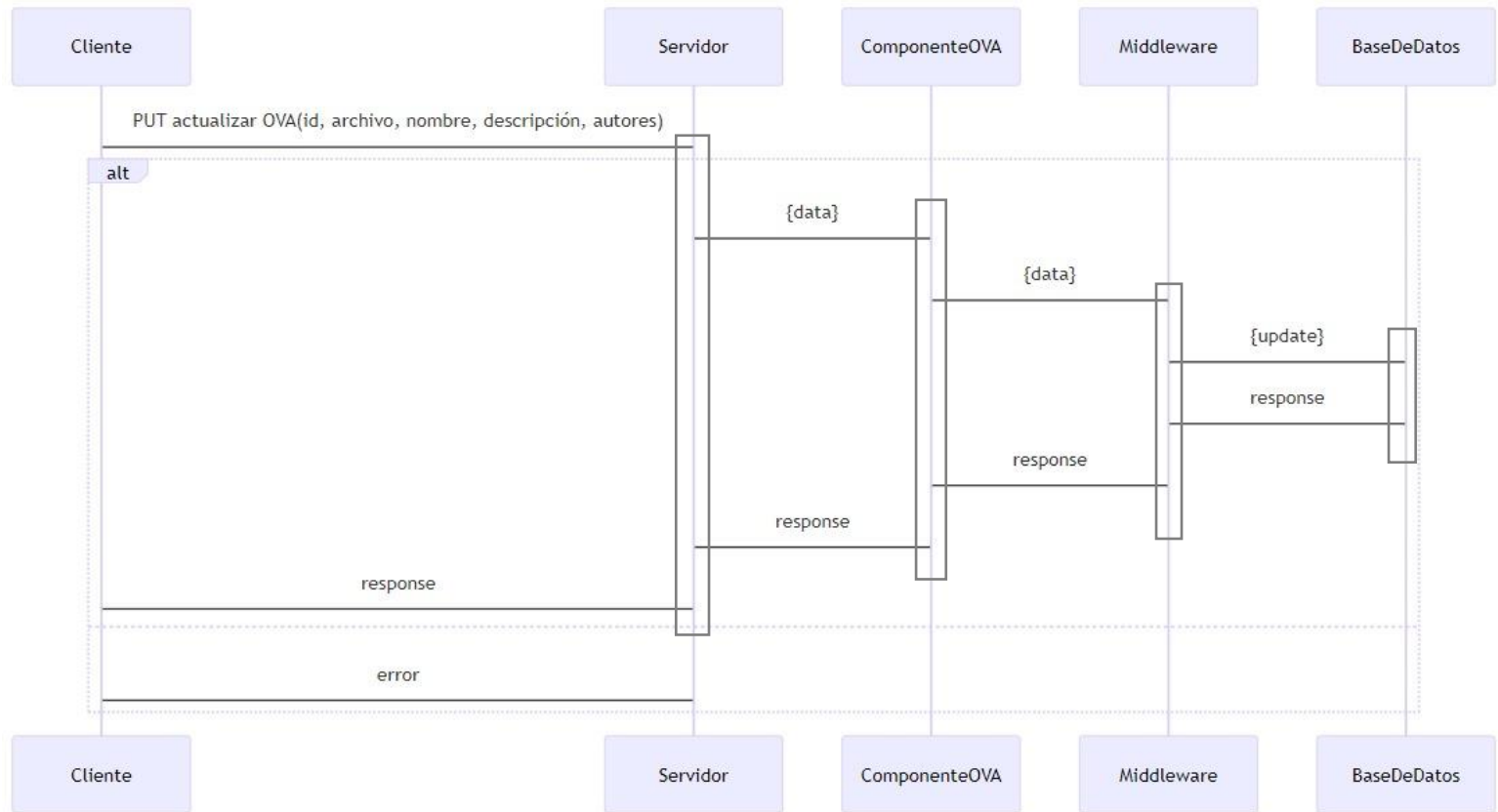
Subir OVA



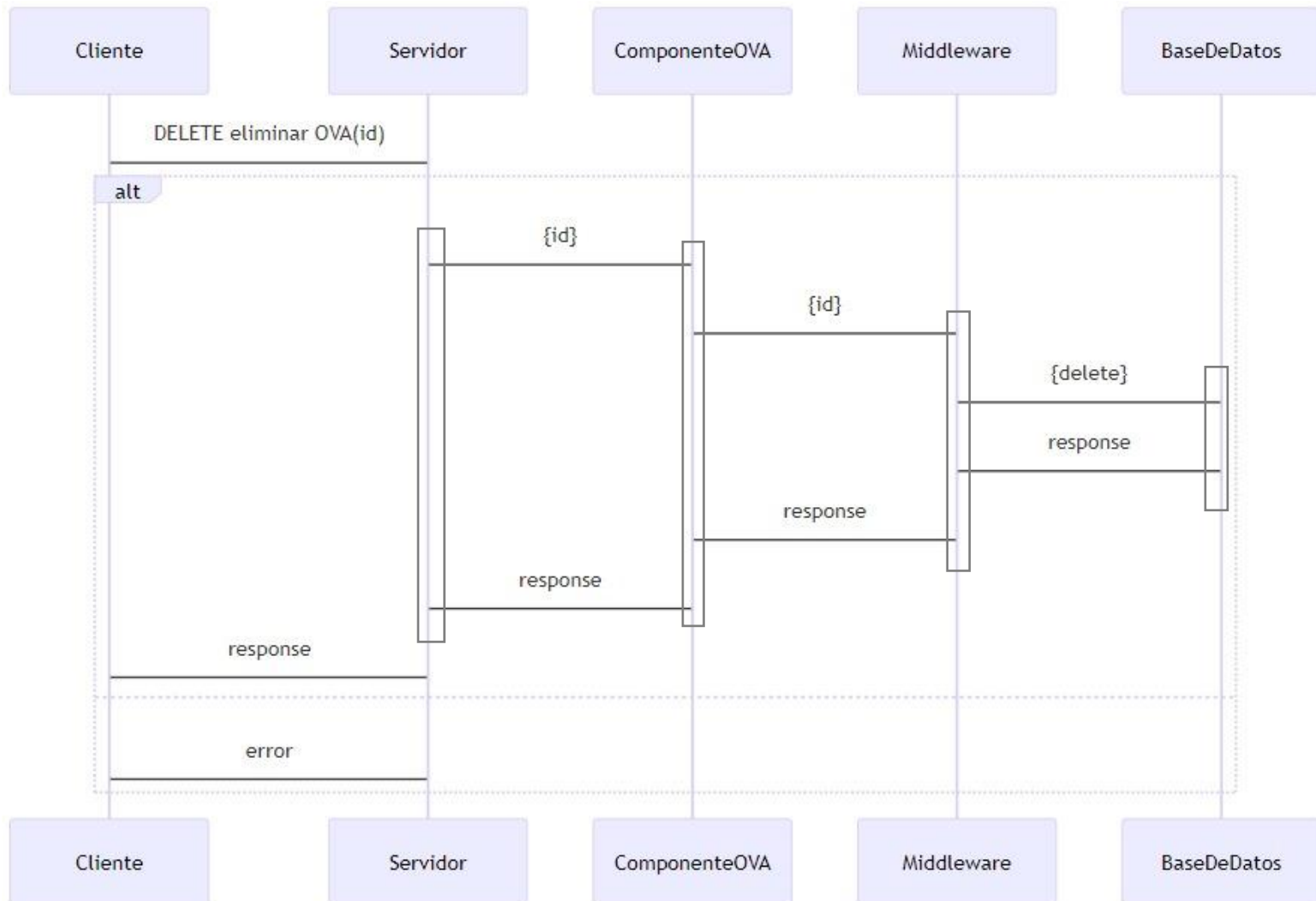
Visualizar OVA



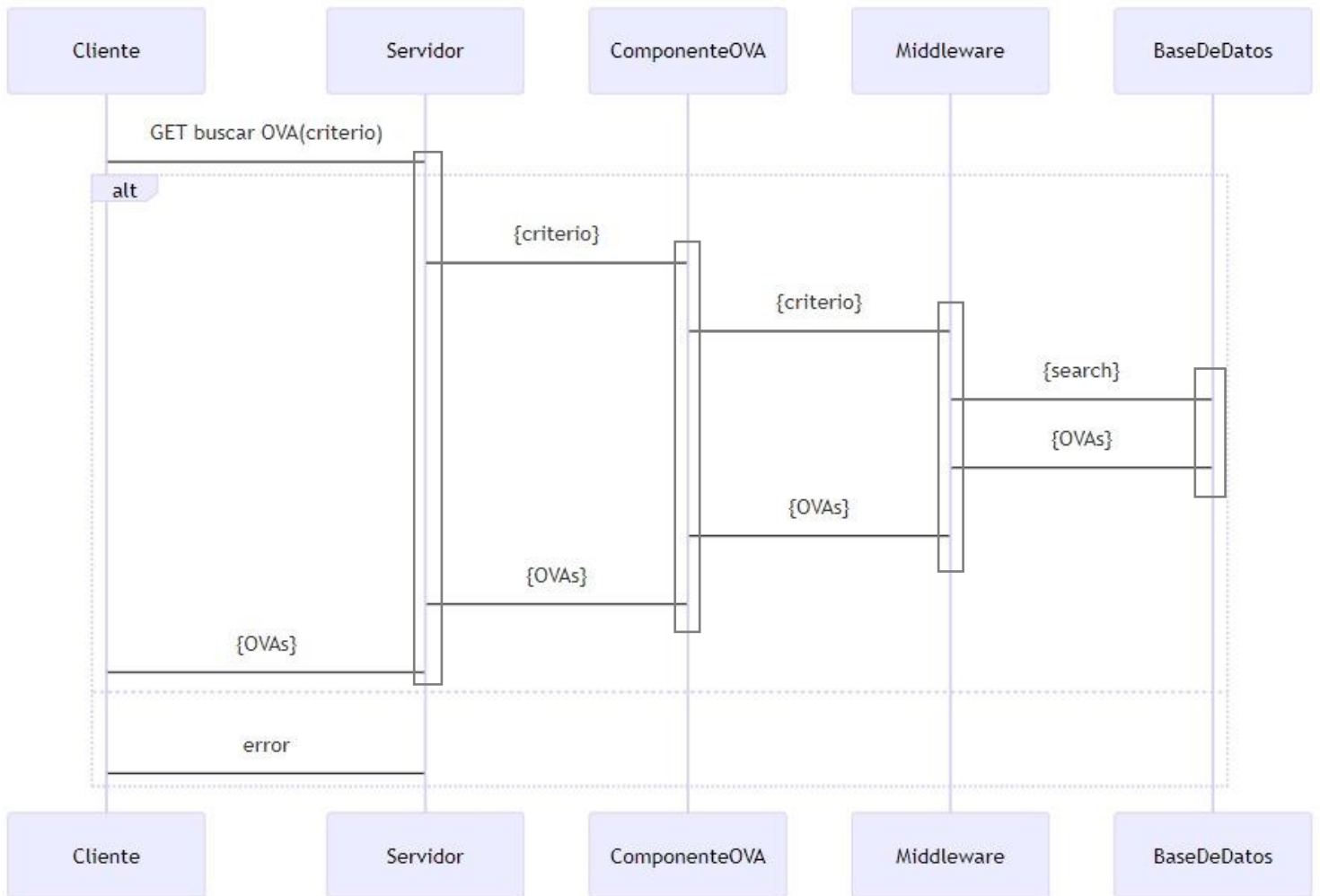
Actualizar OVA



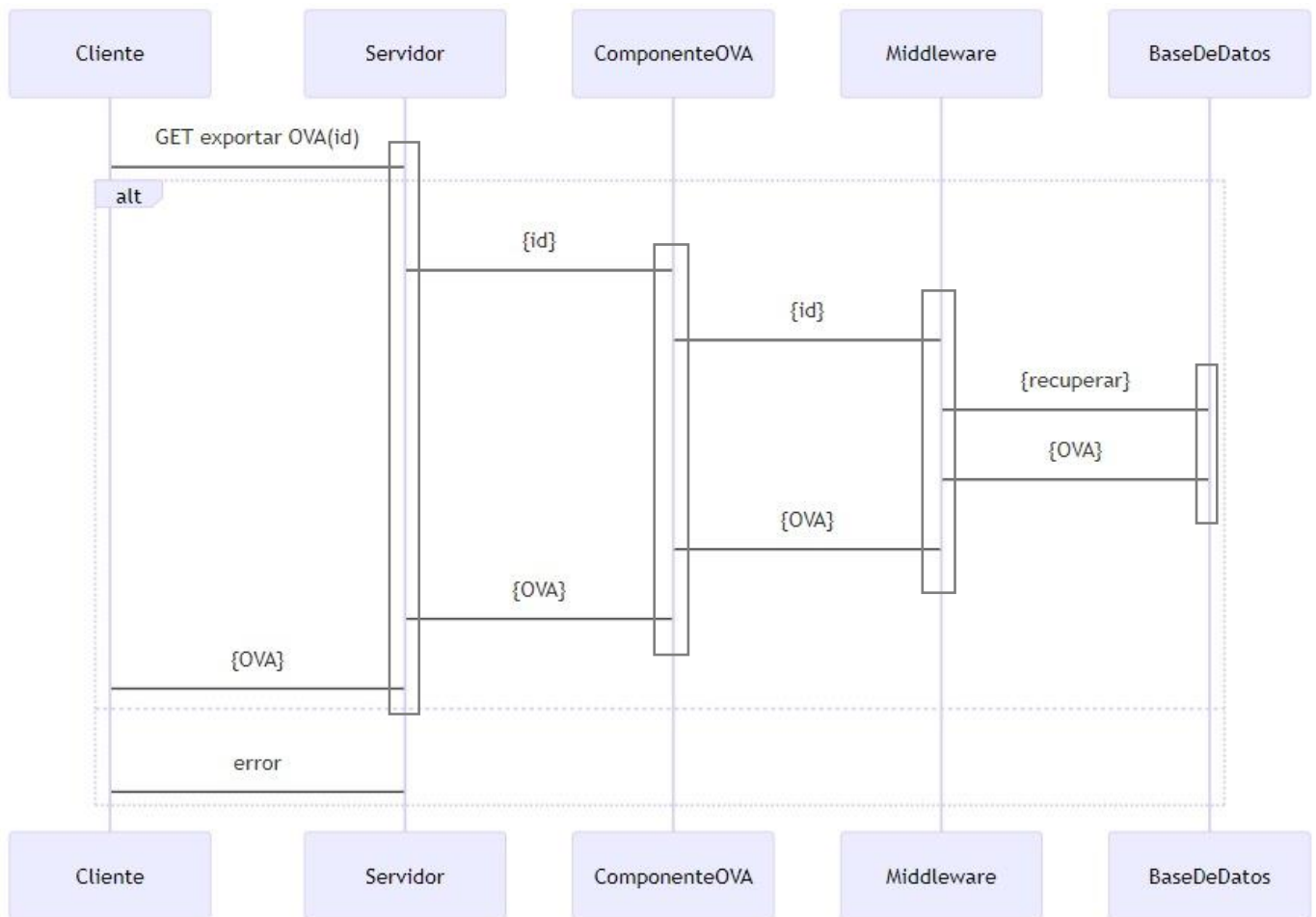
Eliminar OVA



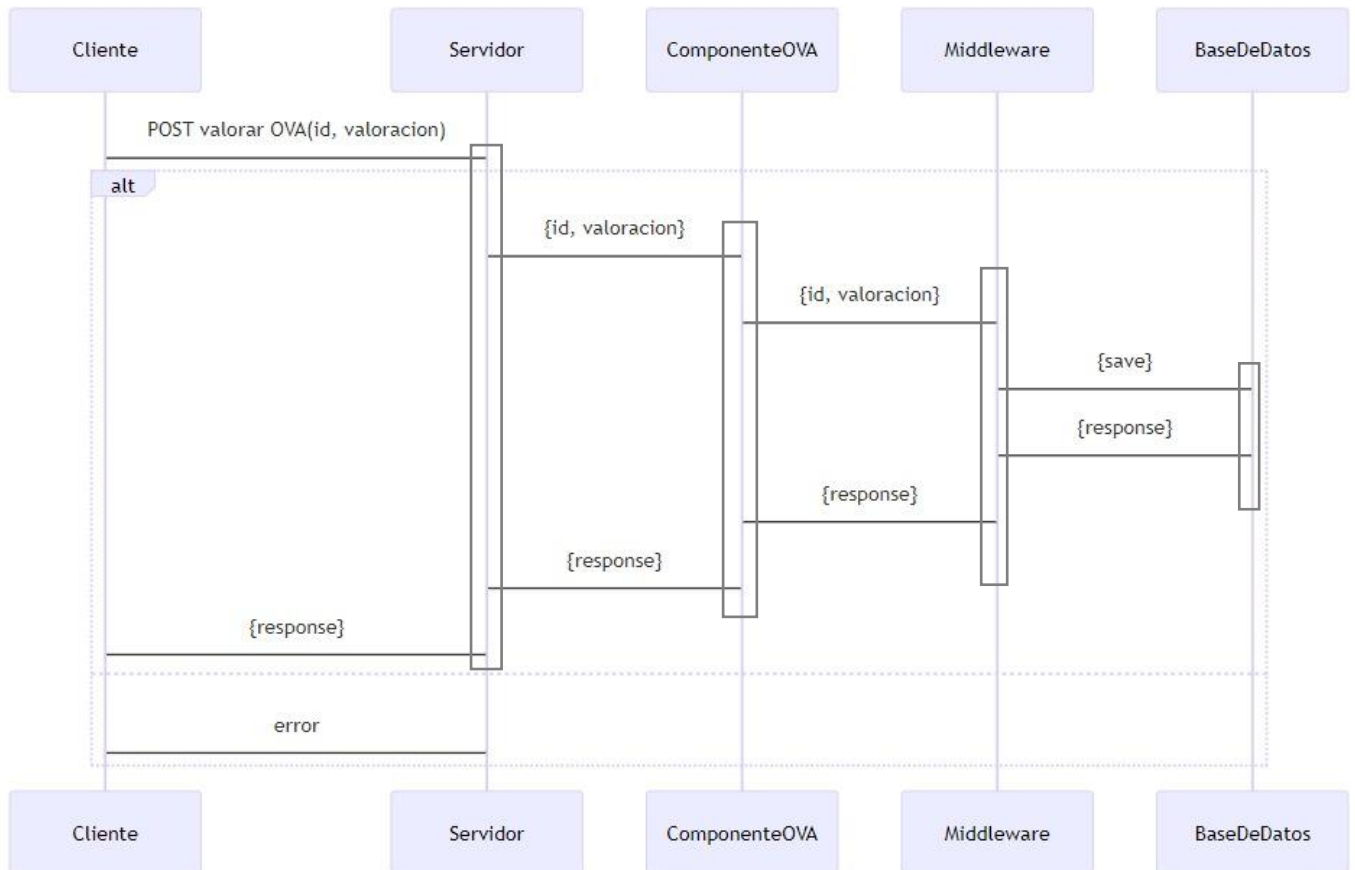
Buscar OVA



Exportar OVA



Valorar OVA



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 OVA

- Entidades:

1. OVA

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

2. Usuario:

- Atributos: Nombre de usuario, Tipo de usuario.

2. Visualizar OVA

- Entidades:

1. Visualización de OVA

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

2. Usuario:

- Atributos: Nombre de usuario, Tipo de usuario.

3. Actualizar OVA

- Entidades:

1. Actualización de OVA

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

2. Usuario:

- Atributos: Nombre de usuario, Tipo de usuario.

4. Eliminar OVA

- Entidades:

1. Elimina OVA

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

2. Usuario:

- Atributos: Nombre de usuario, Tipo de usuario.

5. Buscar OVA

- Entidades:

- 1. Búsqueda de OVA**

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

- 2. Usuario:**

- Atributos: Nombre de usuario, Tipo de usuario.

- 6. Exportar OVA**

- Entidades:

- 1. Exportación de OVA**

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

- 2. Usuario:**

- Atributos: Nombre de usuario, Tipo de usuario.

- 7. Valorar OVA**

- Entidades:

- 1. Valoración de OVA**

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

- 8. Usuario:**

- Atributos: Nombre de usuario, Tipo de usuario.

DIAGRAMA ENTIDAD - RELACIÓN

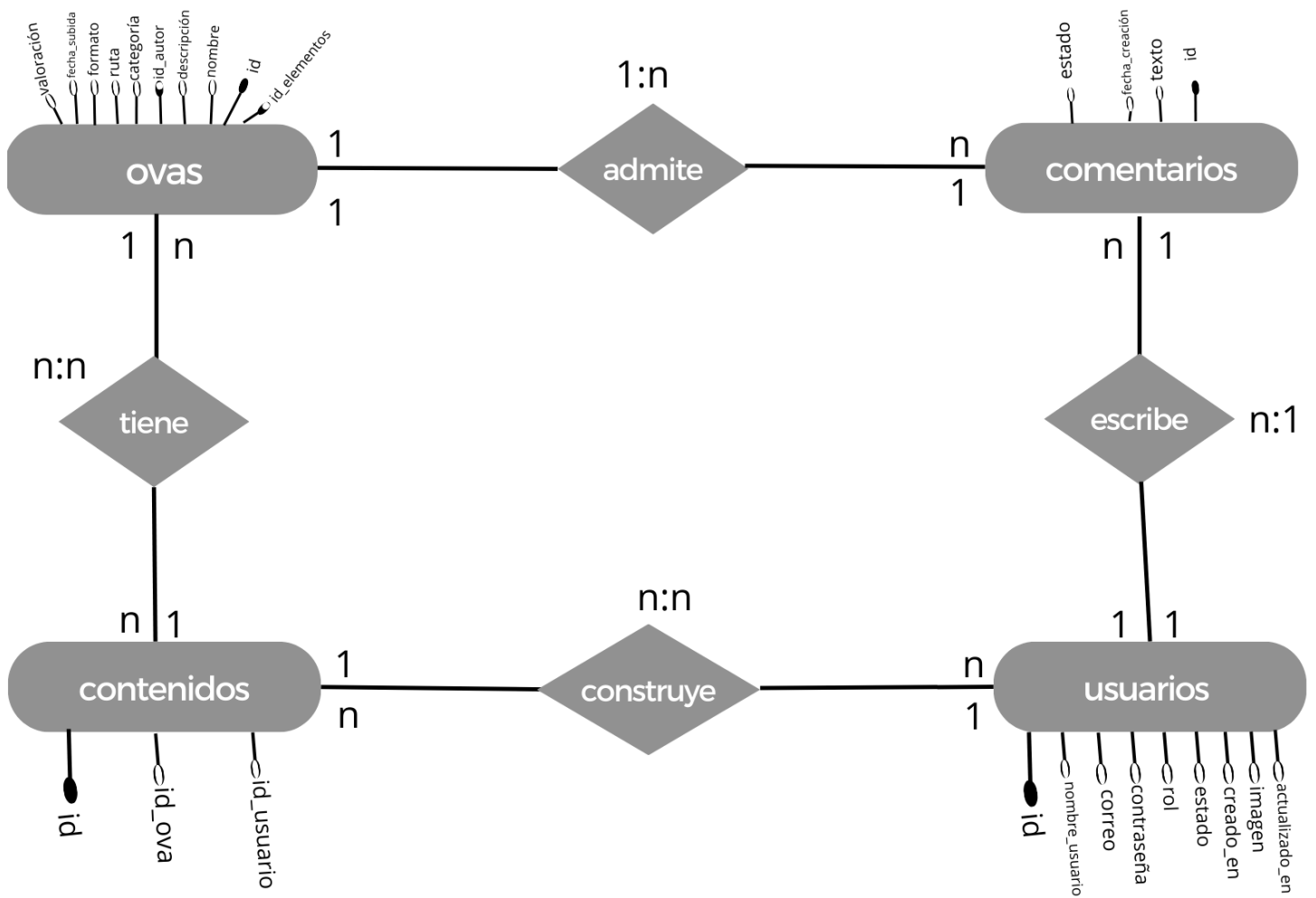
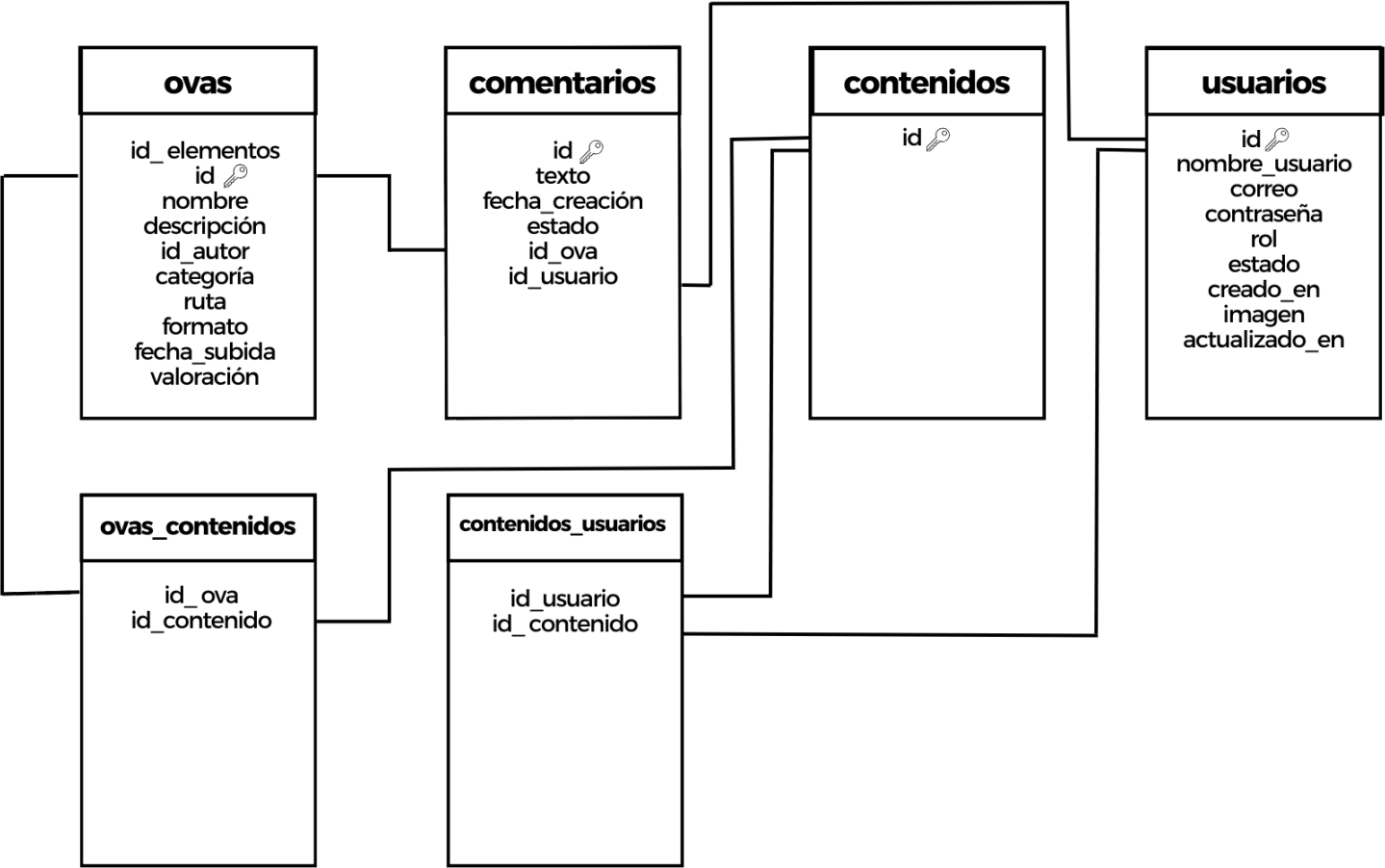


DIAGRAMA MODELO RELACIONAL



NO SQL

```
ovas: {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_ovas: number,

id_usuarios: number,

}
```

```
contenidos: {id: number,

id_ovas: 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 Ova 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 Ova 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.

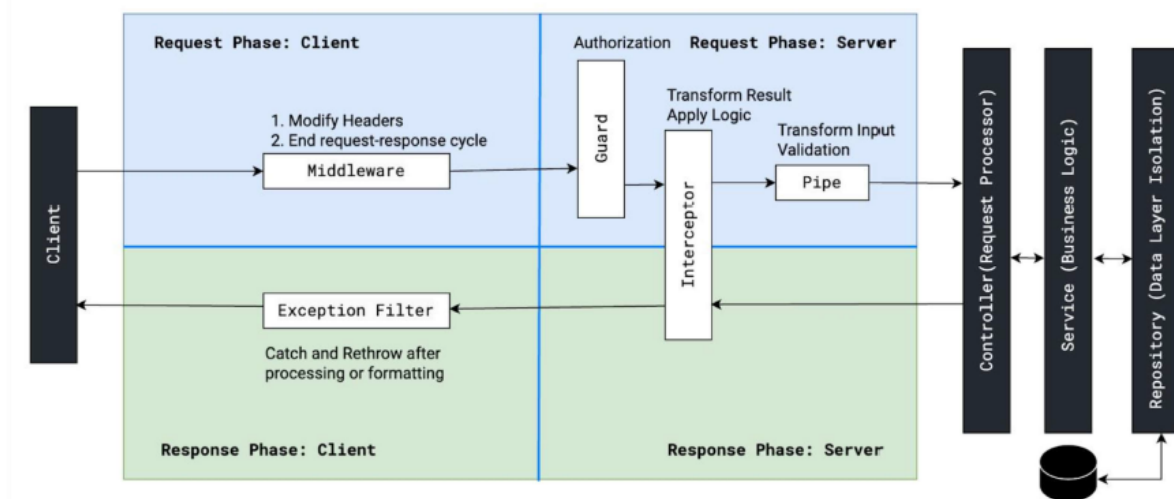
Server: Sistema que proporciona recursos, datos, servicios o programas a otros ordenadores conocidos como clientes, a través de una red.

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.

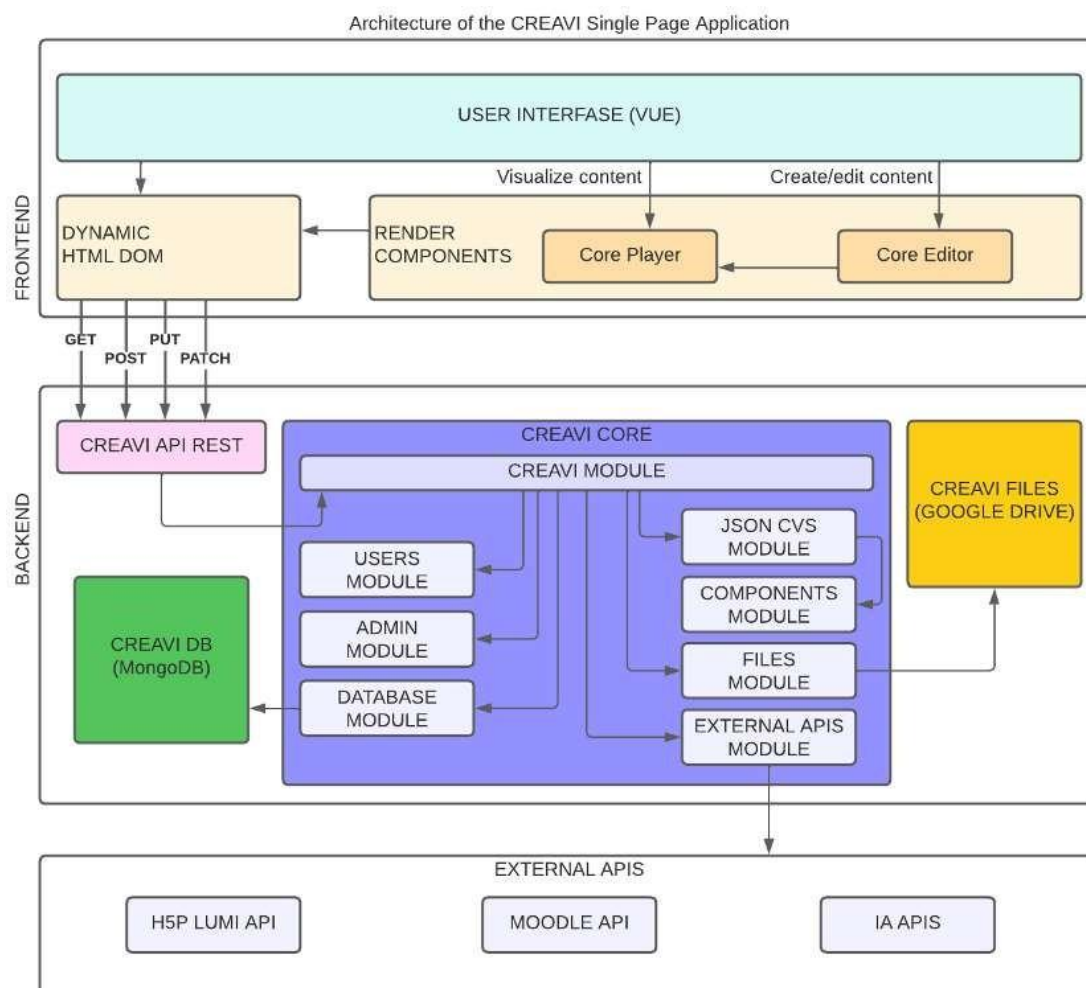
Componentes del Backend



En esta etapa del desarrollo backend, avanzamos desde el service (Business 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

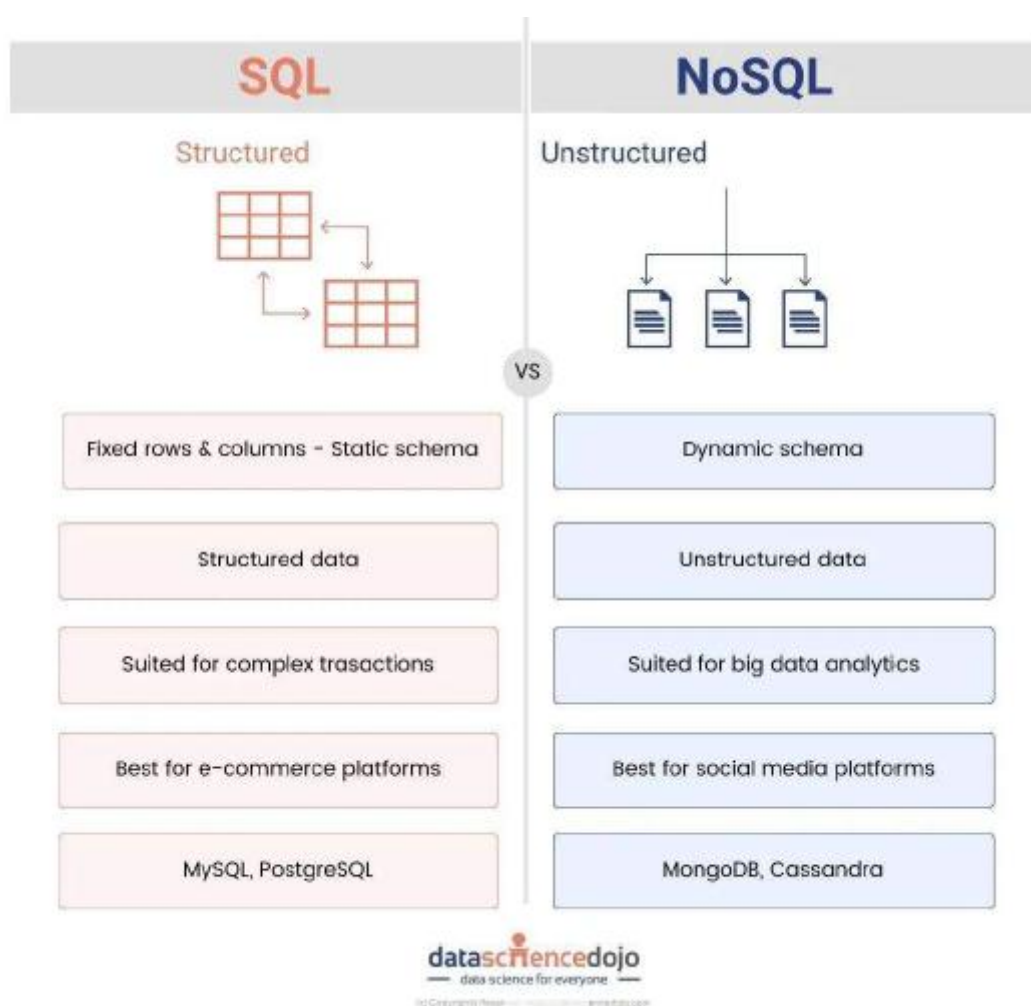


El diagrama anterior muestra la arquitectura de la aplicación de una sola página (SPA) de CREAVI, compuesta por una interfaz de usuario que incluye componentes para visualización y edición de contenido, un backend que maneja la lógica de la aplicación mediante módulos. Y una base de datos MongoDB. En 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 mas 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 Objetos Virtuales de Aprendizaje (OVA) 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 OVA, 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 OVA, 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 OVA 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 Ovas

```
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 '@nestjs/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);
```

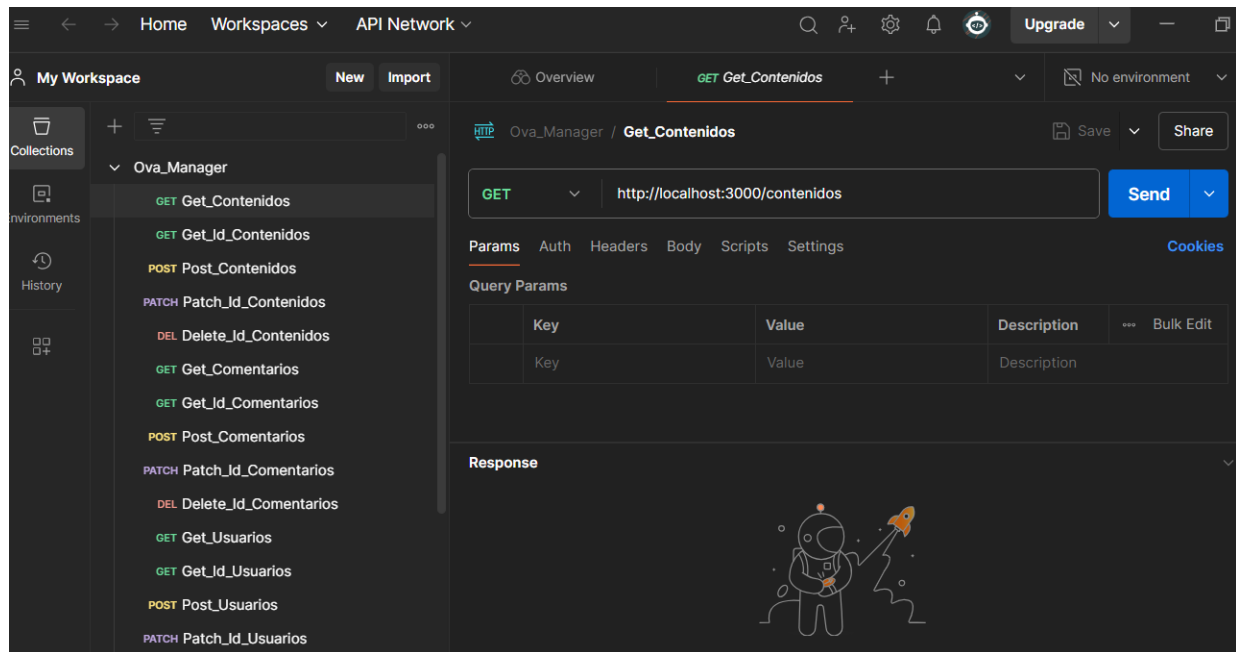
Implementación del Backend

Elección del Lenguaje de Programación

La combinación de NestJS, MongoDB y MongoDB Atlas resulta ideal para el proyecto OVA 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 OVA 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 OVA 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 Ova_Manager.

```
Ova_Manager.postman_collection.json
documents > EndPoint Postman > Ova_Manager.postman_collection.json > ...
1  {
2    "info": {
3      "_postman_id": "bf6c51cb-545e-4a93-8d37-89671d7dbc13",
4      "name": "Ova_Manager",
5      "schema": "https://schema.getpostman.com/json/collection/v2.1.0/collection.json",
6      "_exporter_id": "39508946"
7    },
8    "item": [
9      {
10       "name": "Get_Contenidos",
11       "request": {
12         "method": "GET",
13         "header": [],
14         "url": {
15           "raw": "http://localhost:3000/contenidos",
16           "protocol": "http",
17           "host": [
18             "localhost"
19           ],
20           "port": "3000",
21           "path": [
22             "contenidos"
23           ]
24         }
25       },
26       "response": []
27     }
28   ]
29 }
```

Conexión a la Base de Datos

Configuración de la Conexión

La conexión a la base de datos MongoDB Atlas desde el componente Ova_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.

Ovas

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

```
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 }
```

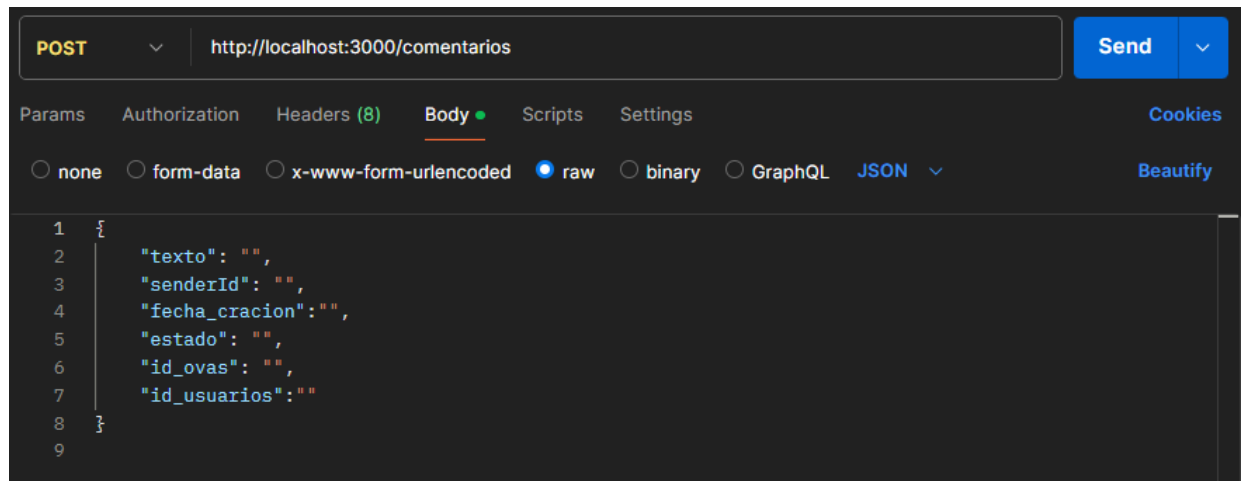
Se puede apreciar que el Create, finAll, update y remove, donde cada uno tiene un modelo al que va a llamar, igualmente, un async y un await, para que hasta que no se realice esa acción correspondiente, no avance a la otra línea de código, ahorrando así posibles errores en la obtención de las peticiones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

Manejo de Transacciones

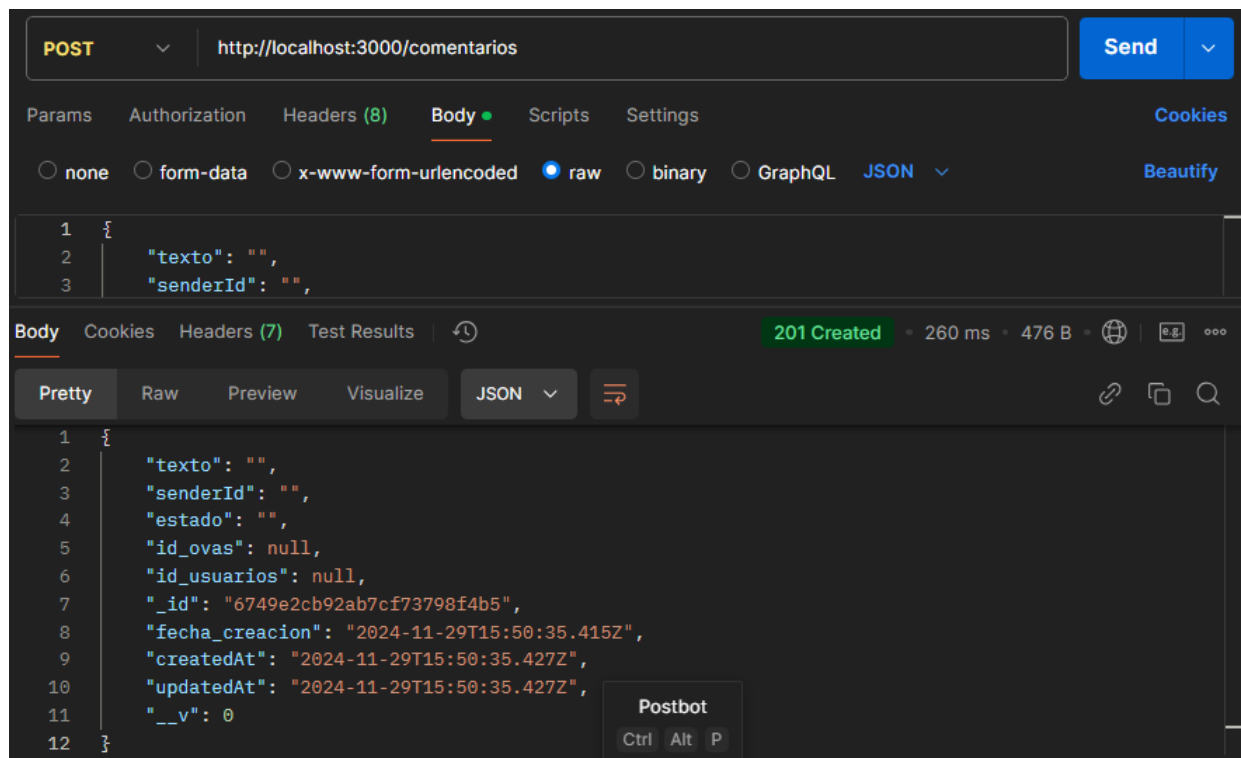
Pruebas del Backend

POST Comentarios

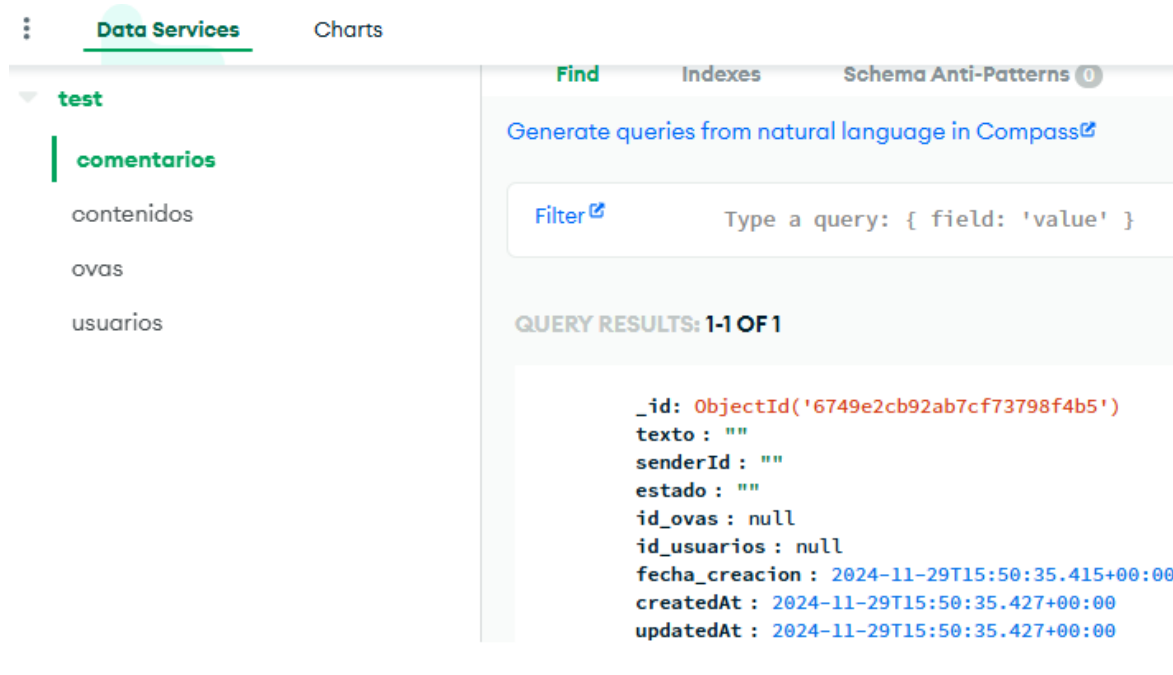
En esta imagen se puede apreciar un objeto con los respectivos atributos desde la interfaz de postman.



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

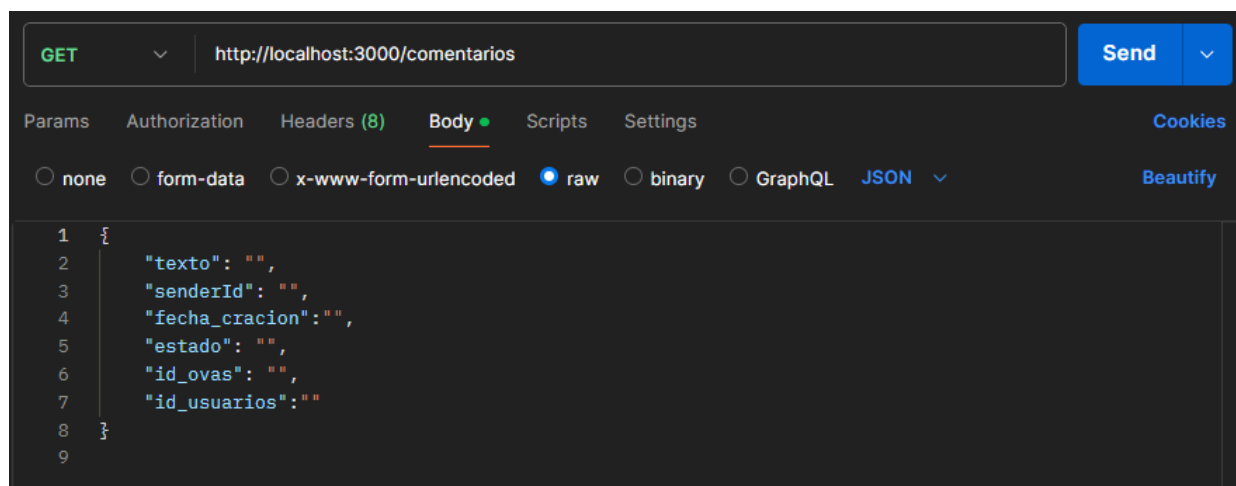


En esta imagen se puede ver la interfaz de MONGO DB ATLAS donde se muestra el objeto creado en POSTMAN, comprobando que la conexión entre el servidor y cliente está funcionando correctamente.

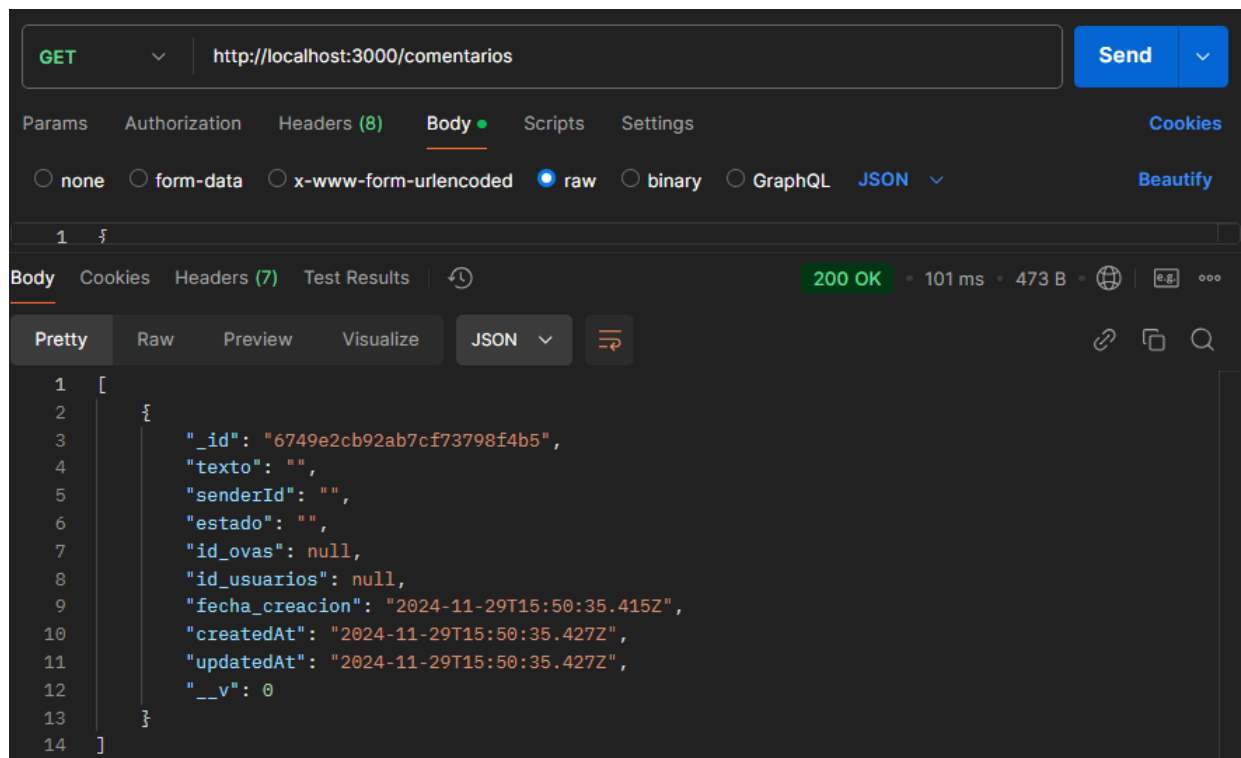


GET Comentarios

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

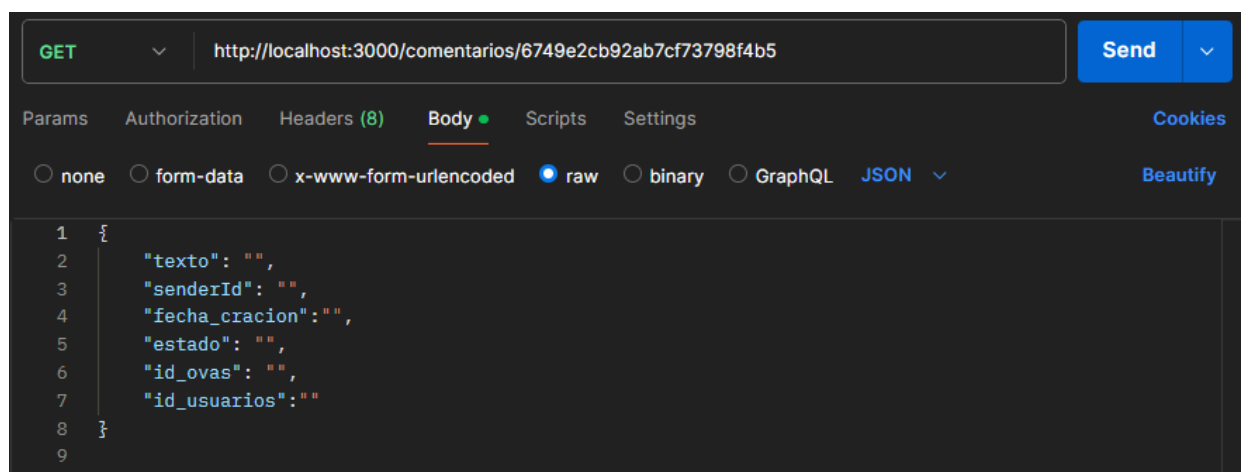


Luego de haber presionado el botón SEND, se TRAE el objeto que está almacenado en la base de datos.



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.

The screenshot shows a MongoDB query interface. On the left, there is a sidebar with a list of collections: **comentarios**, **contenidos**, **ovas**, and **usuarios**. The main area displays the query results for the **comentarios** collection. The query is: `{ field: 'value' }`. The results show two documents. The first document has the ID `6749e2cb92ab7cf73798f4b5` and the following fields: `texto`, `senderId`, `estado`, `id_ovas`, `id_usuarios`, `fecha_creacion`, `createdAt`, `updatedAt`, and `__v`. The second document has the ID `6749e75392ab7cf73798f4ba` and the same fields. A red box highlights the ID `6749e2cb92ab7cf73798f4b5` in the first document.

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

The screenshot shows a REST client interface. The top bar displays the method **GET** and the URL `http://localhost:3000/comentarios/6749e2cb92ab7cf73798f4b5`. The **Send** button is visible. Below the URL bar, there are tabs for **Params**, **Authorization**, **Headers (8)**, **Body**, **Scripts**, and **Settings**. The **Body** tab is selected, and the content is `{}`. The **Body** tab is also selected in the bottom bar. The **Test Results** tab shows a **200 OK** status with a response time of **98 ms** and a body size of **471 B**. The response body is displayed in the **Body** tab, showing a JSON object with the following fields: `_id`, `texto`, `senderId`, `estado`, `id_ovas`, `id_usuarios`, `fecha_creacion`, `createdAt`, `updatedAt`, and `__v`. The response is formatted as **JSON**.

PATCH Comentarios


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

comentarios

contenidos

ovas

usuarios

Filter 

Type a query: { field: 'value' }

QUERY RESULTS: 1-2 OF 2

+

_id: ObjectId('6749e2cb92ab7cf73798f4b5')

2

texto : "

3

senderId : "

4

estado : "

5

id_ovas : null

6

id_usuarios : null

7

fecha_creacion : 2024-11-29T15:50:35.415+00:00

8


createdAt : 2024-11-29T15:50:35.427+00:00

9


updatedAt : 2024-11-29T15:50:35.427+00:00

6749e2cb92ab7cf73798f4b5

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

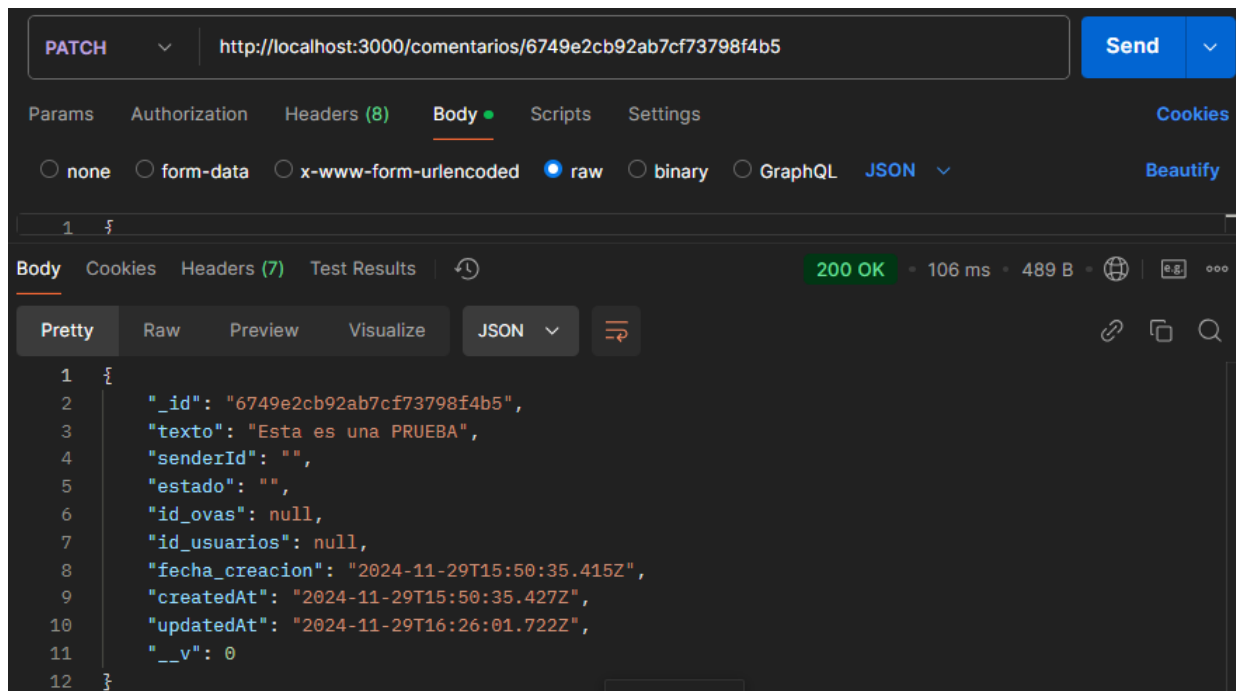
☐ GraphQL

JSON 

Beautify

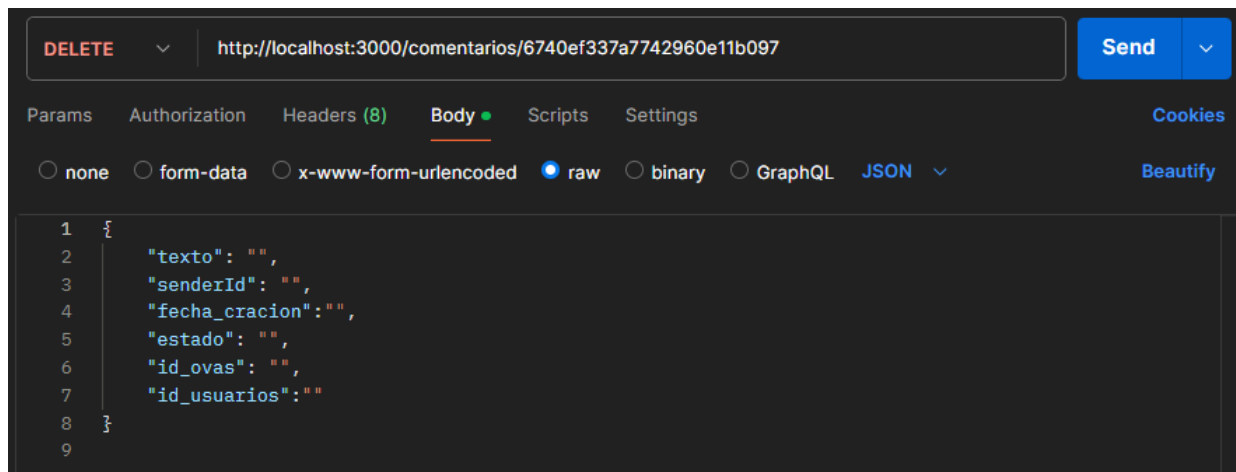
```
1 {
2   "texto": "Esta es una PRUEBA",
3   "senderId": "",
4   "fecha_creacion": "",
5   "estado": "",
6   "id_ovas": "",
7   "id_usuarios": ""
8 }
9
```


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

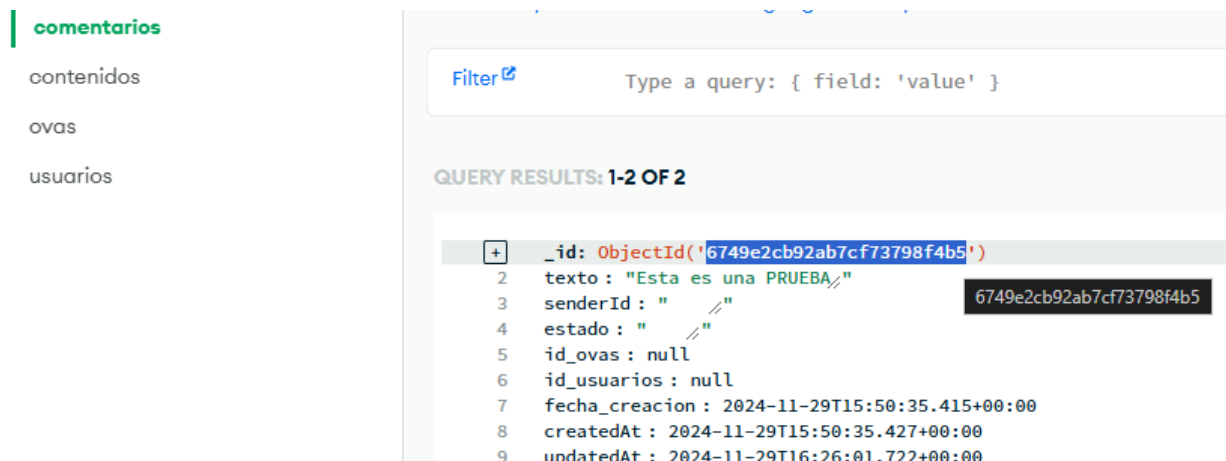


DELETE Comentarios

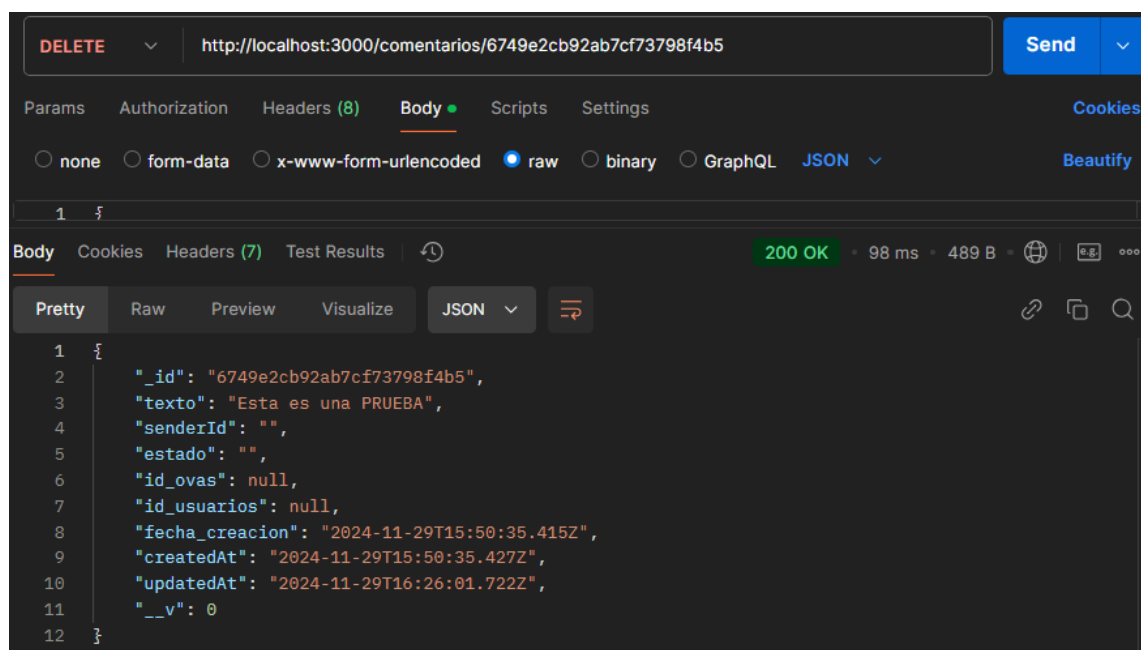
Aquí se muestra la interfaz en POSTMAN donde se va a realizar la petición de ELIMINAR uno de los objetos.



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”.



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



Etapas 3: Consumo de Datos y Desarrollo Frontend

Introducción

Propósito de la Etapa

Alcance de la Etapa

Definiciones y Acrónimos

Creación de la Interfaz de Usuario (UI)

Diseño de la Interfaz de Usuario (UI) con HTML y CSS

Consideraciones de Usabilidad

Maquetación Responsiva

Programación Frontend con JavaScript (JS)

Desarrollo de la Lógica del Frontend

Manejo de Eventos y Comportamientos Dinámicos

Uso de Bibliotecas y Frameworks (si aplicable)

Consumo de Datos desde el Backend

Configuración de Conexiones al Backend

Obtención y Presentación de Datos

Actualización en Tiempo Real (si aplicable)

Interacción Usuario-Interfaz

Manejo de Formularios y Validación de Datos

Implementación de Funcionalidades Interactivas

Mejoras en la Experiencia del Usuario

Pruebas y Depuración del Frontend

Diseño de Casos de Prueba de Frontend

Pruebas de Usabilidad

Depuración de Errores y Optimización del Código

Implementación de la Lógica de Negocio en el Frontend

Migración de la Lógica de Negocio desde el Backend (si necesario)

Validación de Datos y Reglas de Negocio en el Frontend

Integración con el Backend

Verificación de la Comunicación Efectiva con el Backend

Pruebas de Integración Frontend-Backend