

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

Creación Módulo Formulario

Docente:

Alexander Toscano Ricardo

Integrantes:

Andrés Felipe Díaz Vergara

Jhair Andrés Teheran Flórez

María Alma López Mestra

Gustavo Rodríguez Fuentes



Twitter: @kikret

Github: @atoscano



Descripción del software

Se desea crear un componente compatible para la plataforma CREAVI que permita compartir elementos multimediales entre los usuarios inscritos en un espacio de trabajo (clase, conferencia, taller, etc.) Los elementos para compartir podrán ser videos, diapositivas, imágenes, documentos, audios, segmentos de código, etc. El anfitrión podrá resaltar elementos en pantalla o apuntarlos y los interlocutores podrán interactuar con ellos solo si el anfitrión u otro usuario designado lo permite.

El componente traerá elementos por defecto en su instalación, pero permite la escalabilidad a otros componentes o formatos que sean compatibles con la versión desarrollada.



ETAPA 1 DISEÑO DE LA APLICACIÓN Y ANÁLISIS DE REQUISITOS.....	6
1. INTRODUCCIÓN.....	6
PROPÓSITO DEL DOCUMENTO.....	6
ALCANCE DEL PROYECTO MÓDULO DE FORMULARIO.....	8
DEFINICIONES Y ACRÓNIMOS.....	8
2. DESCRIPCIÓN GENERAL.....	10
OBJETIVOS DEL SISTEMA.....	10
FUNCIONALIDAD GENERAL.....	10
USUARIOS DEL SISTEMA.....	10
RESTRICCIONES.....	11
3. REQUISITOS FUNCIONALES.....	11
CASOS DE USO.....	12
DIAGRAMAS DE FLUJO DE CASOS DE USO.....	13
Descripción detallada de cada caso de uso.....	16
PRIORIDAD DE REQUERIMIENTOS.....	31
4. REQUISITOS No FUNCIONALES.....	32
REQUISITOS DE DESEMPEÑO.....	34
REQUISITOS DE SEGURIDAD.....	34
REQUISITOS DE USABILIDAD.....	36
REQUISITOS DE ESCALABILIDAD.....	36
5. MODELADO E/R.....	37
DIAGRAMA DE ENTIDAD-RELACIÓN.....	37
DIAGRAMA RELACIONAL.....	38
SCRIPT DE MODELO RELACIONAL.....	39
DESCRIPCIÓN DE ENTIDADES Y RELACIONES.....	39
REGLAS DE INTEGRIDAD REFERENCIAL.....	40
COLECCIONES (NoSQL).....	41
6. ANEXOS.....	42
DIAGRAMAS ADICIONALES.....	42
REFERENCIAS.....	42
ETAPA 2: PERSISTENCIA DE DATOS CON BACKEND.....	43
7. INTRODUCCIÓN.....	43
PROPÓSITO DE LA ETAPA.....	43
Propósito de la Etapa.....	43
ALCANCE DE LA ETAPA.....	44
DEFINICIONES Y ACRÓNIMOS.....	46
8. DISEÑO DE LA ARQUITECTURA DE BACKEND.....	48
DESCRIPCIÓN DE LA ARQUITECTURA PROPUESTA.....	48
COMPONENTES DEL BACKEND.....	48
DIAGRAMAS DE ARQUITECTURA.....	49
9. ELECCIÓN DE LA BASE DE DATOS.....	50
EVALUACIÓN DE OPCIONES (SQL o NoSQL).....	50
JUSTIFICACIÓN DE LA ELECCIÓN.....	50
DISEÑO DE ESQUEMA DE BASE DE DATOS.....	50
10. IMPLEMENTACIÓN DEL BACKEND.....	52
ELECCIÓN DEL LENGUAJE DE PROGRAMACIÓN.....	52



CREACIÓN DE LA LÓGICA DE NEGOCIO.....	52
DESARROLLO DE ENDPOINTS Y APIs.....	53
AUTENTICACIÓN Y AUTORIZACIÓN.....	53
11. CONEXIÓN A LA BASE DE DATOS.....	54
CONFIGURACIÓN DE LA CONEXIÓN.....	54
DESARROLLO DE OPERACIONES CRUD.....	54
MANEJO DE TRANSACCIONES.....	59
16. PRUEBAS DEL BACKEND.....	60
DISEÑO DE CASOS DE PRUEBA.....	60
EJECUCIÓN DE PRUEBAS UNITARIAS Y DE INTEGRACIÓN.....	60
MANEJO DE ERRORES Y EXCEPCIONES.....	60
ETAPA 3: CONSUMO DE DATOS Y DESARROLLO FRONTEND.....	61
17. INTRODUCCIÓN.....	61
PROPÓSITO DE LA ETAPA.....	61
ALCANCE DE LA ETAPA.....	61
DEFINICIONES Y ACRÓNIMOS.....	61
18. CREACIÓN DE LA INTERFAZ DE USUARIO (UI).....	61
DISEÑO DE LA INTERFAZ DE USUARIO (UI) CON HTML Y CSS.....	61
CONSIDERACIONES DE USABILIDAD.....	61
MAQUETACIÓN RESPONSIVA.....	61
19. PROGRAMACIÓN FRONTEND CON JAVASCRIPT (JS).....	62
DESARROLLO DE LA LÓGICA DEL FRONTEND.....	62
MANEJO DE EVENTOS Y COMPORTAMIENTOS DINÁMICOS.....	62
USO DE BIBLIOTECAS Y FRAMEWORKS (SI APLICABLE).....	62
20. CONSUMO DE DATOS DESDE EL BACKEND.....	62
CONFIGURACIÓN DE CONEXIONES AL BACKEND.....	62
OBTENCIÓN Y PRESENTACIÓN DE DATOS.....	62
ACTUALIZACIÓN EN TIEMPO REAL (SI APLICABLE).....	62
21. INTERACCIÓN USUARIO-INTERFAZ.....	62
MANEJO DE FORMULARIOS Y VALIDACIÓN DE DATOS.....	63
IMPLEMENTACIÓN DE FUNCIONALIDADES INTERACTIVAS.....	63
MEJORAS EN LA EXPERIENCIA DEL USUARIO.....	63
22. PRUEBAS Y DEPURACIÓN DEL FRONTEND.....	63
DISEÑO DE CASOS DE PRUEBA DE FRONTEND.....	63
PRUEBAS DE USABILIDAD.....	63
DEPURACIÓN DE ERRORES Y OPTIMIZACIÓN DEL CÓDIGO.....	63
23. IMPLEMENTACIÓN DE LA LÓGICA DE NEGOCIO EN EL FRONTEND.....	63
MIGRACIÓN DE LA LÓGICA DE NEGOCIO DESDE EL BACKEND (SI NECESARIO).....	64
VALIDACIÓN DE DATOS Y REGLAS DE NEGOCIO EN EL FRONTEND.....	64
24. INTEGRACIÓN CON EL BACKEND.....	64
VERIFICACIÓN DE LA COMUNICACIÓN EFECTIVA CON EL BACKEND.....	64
PRUEBAS DE INTEGRACIÓN FRONTEND-BACKEND.....	64
ANEXOS.....	64



Etapas 1 Diseño de la Aplicación y Análisis de Requisitos

1. 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 continúa 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 Módulo de Formulario

El software educativo propuesto en el curso de diseño y desarrollo de software II, consiste en desarrollar un formulario que realice y a la vez facilite los procesos de creación de las necesidades educativas o bien llamados proyectos educativos de la institución educativa. De esta manera, el sistema diseñado se realizará utilizando la plataforma CREA VI, el cual es un formulario en donde se pueden crear diferentes talleres y dado el caso clase. En este se podrán encontrar diferentes recursos digitales como videos, imágenes, audios, segmentos, que ayudarán a facilitar la obtención y construcción del conocimiento por parte de los usuarios que hagan parte del sistema o que naveguen por este.

- Creación de preguntas y respuestas
- Subir archivos y multimedia

Funcionalidades futuras

- Exportación e Impresión
- Acceso a plantillas

Definiciones y Acrónimos

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

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

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

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

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

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

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

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

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



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

API RESTful: API que sigue los principios de REST.

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

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

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

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

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

JS: JavaScript.

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

UI: Interfaz de Usuario (User Interface).

UX: Experiencia del Usuario (User Experience).

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

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

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

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

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

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

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

PWA: Aplicación Web Progresiva (Progressive Web App).

2. Descripción General

Objetivos del Sistema

El objetivo del sistema es proporcionar una plataforma educativa a través de un formulario en la plataforma CREAVI, el cual facilitará a los usuarios la creación de proyectos educativos y la gestión de necesidades educativas dentro de la institución educativa, además busca mejorar la colaboración y el diálogo entre los usuarios mediante la incorporación de funciones como foros, conferencias y talleres, además de acceder a diferentes recursos.

Funcionalidad General

- **Creación de preguntas y respuestas:** El usuario podrá diseñar ya sea cuestionarios o formularios de requisitos de una manera mucho más dinámica y selectiva.
- **Subir archivos y multimedia:** Permite a los usuarios cargar imágenes, videos y otros medios directamente al formulario, lo que facilita la ilustración de conceptos.
- **Exportación e Impresión:** Ofrece la capacidad de exportar el contenido del formulario en varios formatos (PDF, imagen, etc.) y la opción de imprimirlo.
- **Acceso a plantillas:** La plataforma dispondrá de distintas plantillas las cuales se acoplan a cualquier tipo de finalidad o necesidad del usuario.

Usuarios del Sistema

Los siguientes usuarios pueden interactuar con el formulario dependiendo de las funcionalidades.

Funcionalidad	Administradores	Docente investigador	Docente invitado	Alumno	Invitado
Creación de preguntas y respuestas		★	★		



subir archivos y multimedia		★	★	★	
Exportación e Impresión		★	★	★	
Acceso a plantillas		★	★		

Restricciones

Solo usuarios agregados por un anfitrión del formulario tendrán acceso a las funcionalidades descritas en la tabla anterior, un anfitrión puede agregar otros anfitriones al formulario quienes pueden ser docentes, alumnos o invitados, también se les puede dar el rol de moderador y/o administrador de pizarra. Las funcionalidades de estos dos roles no se han descrito aún.

3. Requisitos Funcionales

Creación de Preguntas y Respuestas:

- El sistema debe permitir a los usuarios crear preguntas de diferentes tipos (opción múltiple, respuesta corta, párrafo, etc.).
- Los usuarios podrán definir opciones de respuesta y establecer requisitos de respuesta obligatoria o opcional.
- Debe ser posible organizar y reorganizar las preguntas de manera dinámica mediante una interfaz intuitiva.

Subir Archivos y Multimedia:

- El sistema permitirá a los usuarios adjuntar archivos multimedia (imágenes, videos, documentos) a las preguntas o respuestas.
- Se deben establecer límites de tamaño y tipos de archivos permitidos para garantizar la eficiencia y seguridad del sistema.



Exportación e Impresión:

- Los usuarios podrán exportar el contenido del formulario en formatos como PDF, imagen, o cualquier otro formato predeterminado.
- Debe haber una función de impresión que permita a los usuarios imprimir el formulario directamente desde la plataforma.

Acceso a Plantillas:

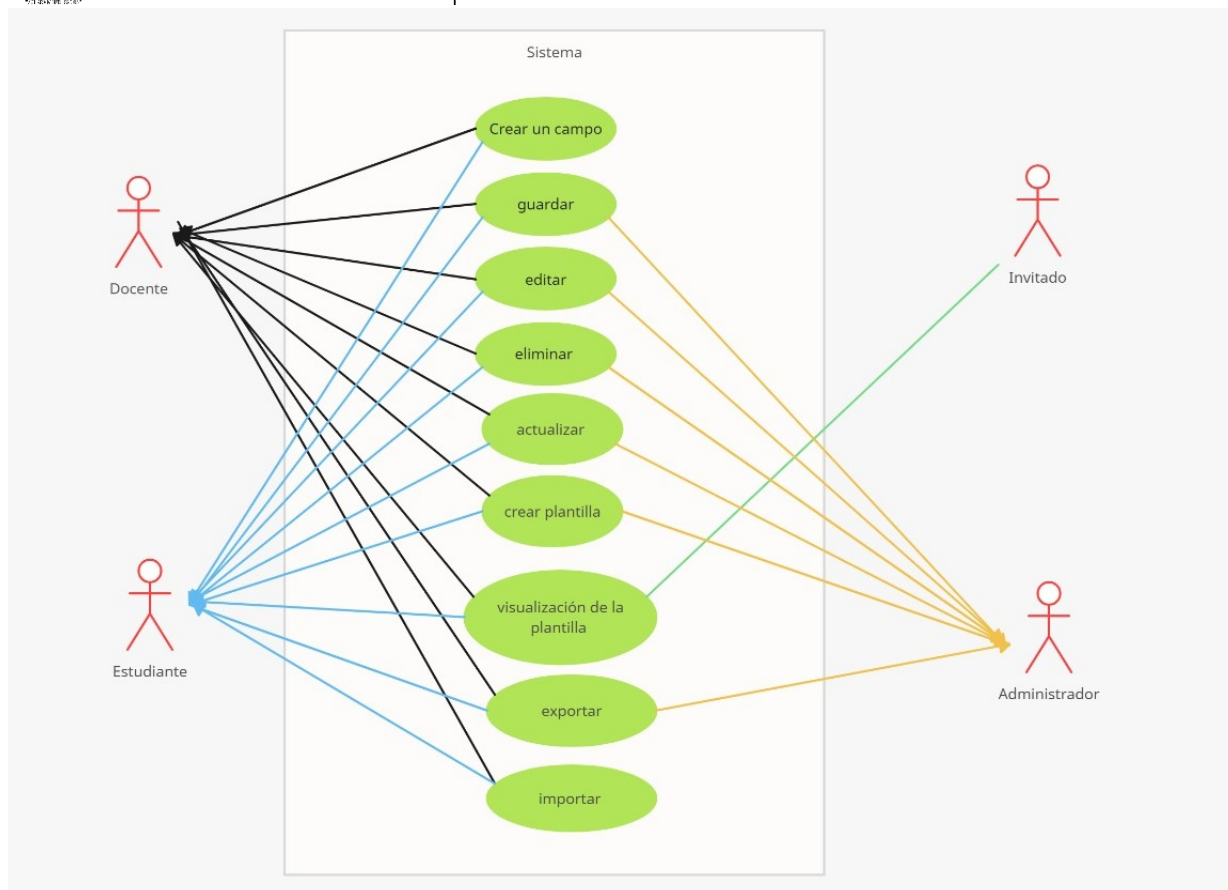
- La plataforma proporcionará una biblioteca de plantillas predefinidas para diversos propósitos (encuestas, cuestionarios, formularios de retroalimentación, etc.).
- Los usuarios podrán seleccionar una plantilla y personalizarla según sus necesidades específicas.

Notificaciones y Recordatorios:

- Los usuarios podrán configurar notificaciones para recordar la finalización de formularios o eventos importantes.
- El sistema enviará notificaciones en tiempo real sobre cambios significativos en los formularios compartidos.

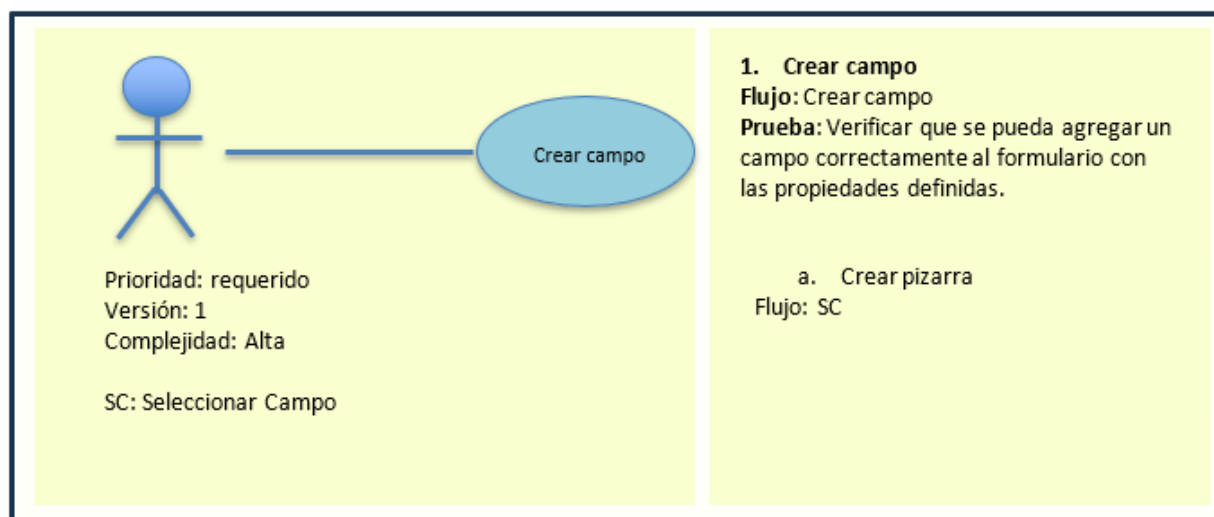
Casos de Uso

Diagrama de caso de uso



<https://app.creately.com/d/start/dashboard>

Diagramas de Flujo de Casos de Uso





Guardar
formulario

Prioridad: requerido
Versión: 1
Complejidad: Media

GF: Guardar Formulario

4. Guardar Formulario

Flujo: Guardar Formulario

Prueba: Confirmar que los cambios realizados en el formulario se guarden correctamente y puedan ser recuperados

b. Guardar Formulario

Flujo: GF



Editar
Formulario

Prioridad: requerido
Versión: 1
Complejidad: Media

EF: Editar Formulario

3. Editar Formulario

Flujo: Editar Formulario

Prueba: Verificar que los cambios realizados en la edición del campo se reflejen correctamente en el formulario.

c. Editar Formulario

Flujo: EF



Eliminar
Formulario

Prioridad: requerido

Versión: 1

Complejidad: Media

EF: Eliminar Formulario

2. Eliminar Formulario

Flujo: Eliminar Formulario

Prueba: Asegurar que el campo seleccionado sea eliminado del formulario sin afectar otros elementos.

d. Eliminar Formulario

Flujo: EF



Actualizar
Formulario

Prioridad: requerido

Versión: 1

Complejidad: Media

EF: Actualizar Formulario

7. Actualizar Formulario

Flujo: Actualizar Formulario

Prueba: Confirmar que las actualizaciones se apliquen correctamente y no afecten negativamente la estructura o datos del formulario existente.

g. Actualizar Formulario

Flujo: AF



Crear plantilla

Prioridad: requerido

Versión: 1

Complejidad: Alta

CP: Crear Plantilla

6. Crear Plantilla

Flujo: Crear Plantilla

Prueba: Verificar que la plantilla creada pueda ser utilizada para generar nuevos formularios correctamente.

f. Crear Plantilla

Flujo: CP



Visualización
de la plantilla

Prioridad: No Requerido
Versión: 1
Complejidad: Baja

CP: Visualizar Plantilla

5. Visualizar Plantilla

Flujo: Visualizar Plantilla

Prueba: Confirmar que la visualización de la plantilla sea clara y represente correctamente la estructura y diseño esperados.

e. Visualizar Plantilla
Flujo: VP



Exportar
Formulario

Prioridad: requerido
Versión: 1
Complejidad: Baja

EF: Exportar Formulario

9. Exportar Formulario

Flujo: Exportar Formulario

Prueba: Verificar que el archivo exportado contenga todos los datos y elementos del formulario de manera correcta.

i. Exportar Formulario
Flujo: EF



Importar
Formulario

Prioridad: requerido
Versión: 1
Complejidad: Media

IF: Importar Formulario

8. Importar Formulario

Flujo: Importar Formulario

Prueba: Confirmar que la importación de archivos se realice sin errores y que los datos se reflejen adecuadamente en el formulario.

h. Importar Formulario
Flujo: IF

Descripción detallada de cada caso de uso

ID:	CU-1
-----	------



Nombre	Crear Campo	
Actores	Docente, estudiante	
Objetivo	Permitir a los administradores y docentes agregar nuevos campos al formulario de manera personalizada.	
Urgencia	4	
Esfuerzo	3	
Pre-condiciones	- Debe haberse autenticado de forma correcta en el sistema.	
Flujo Normal	Docente/Estudiante	Sistema
	Accede al diseñador de formularios.	
		Carga la interfaz del diseñador de formularios.
	Selecciona la opción de "Crear Campo".	
		Muestra la interfaz para la creación de un nuevo campo.
	Define las propiedades del campo (nombre, tipo, longitud, etc.).	
		Captura la información proporcionada por el usuario
	Confirma la creación del campo.	
		Valida la información y agrega el nuevo campo al formulario.
Flujo alternativo 1		



Flujo alternativo 2		
Post-condiciones	El nuevo campo se agrega al formulario con las propiedades especificadas.	
Excepciones	Si el usuario no tiene permisos para editar el formulario, se muestra un mensaje de error.	
	Si hay problemas de conexión, se muestra un mensaje indicando la falla.	

ID:	CU-2
Nombre	Guardar Formulario
Actores	Docente, estudiante
Objetivo	Permitir a los usuarios guardar los cambios realizados en el formulario.
Urgencia	5



Esfuerzo	2	
Pre-condiciones	- El usuario ha realizado cambios en el formulario.	
Flujo Normal	Docente/Estudiante	Sistema
	Después de realizar cambios en el formulario, el usuario selecciona la opción de "Guardar".	
		Almacena los cambios realizados en el formulario.
Flujo alternativo 1		
Flujo alternativo 2		
Post-condiciones ^[k1]	Los cambios realizados se guardan correctamente y pueden ser recuperados después de cerrar y volver a abrir la aplicación.	
Excepciones ^[k2]	Si no hay cambios para guardar, se muestra un mensaje informativo.	



ID:	CU-3	
Nombre	Editar Formulario	
Actores	Docente, estudiante	
Objetivo	Permitir a los usuarios modificar las propiedades de un campo existente en el formulario.	
Urgencia	4	
Esfuerzo	3	
Pre-condiciones	- El usuario ha iniciado sesión y tiene permisos para editar el formulario.	
Flujo Normal	Docente/Administrador/Estudiante	Sistema
	Selecciona la opción de "Editar" para un campo específico.	
		Muestra la interfaz de edición para el campo seleccionado.
	Modifica las propiedades del campo según sea necesario.	
		Captura y muestra los cambios realizados en la interfaz de edición.
	Guarda los cambios realizados.	
		Valida y aplica los cambios al campo en el formulario.
Flujo alternativo 1		



Flujo alternativo 2		
Post-condiciones ^[k3]	Los cambios realizados en la edición del campo se reflejan correctamente en el formulario.	
Excepciones ^[k4]	Si el usuario no tiene permisos para editar el formulario, se muestra un mensaje de error.	
	Si hay problemas de conexión, se muestra un mensaje indicando la falla.	

ID:	CU-4
Nombre	Eliminar Formulario
Actores	Docente, estudiante, Administrador
Objetivo	Permitir a los usuarios eliminar un campo específico del formulario.
Urgencia	5
Esfuerzo	2
Pre-condiciones	- El usuario ha iniciado sesión y tiene permisos para editar el formulario.



Flujo Normal	Docente/Administrador/Estudiante	Sistema
	Selecciona la opción de "Eliminar" para un campo específico.	
		Muestra una confirmación para la eliminación del campo.
	Confirma la eliminación del campo.	
		Elimina el campo seleccionado del formulario.
Flujo alternativo 1		
Flujo alternativo 2		
Post-condiciones	El campo seleccionado es eliminado del formulario sin afectar otros elementos.	
Excepciones	Si el usuario no tiene permisos para editar el formulario, se muestra un mensaje de error.	



	Si el campo a eliminar está siendo utilizado en algún lugar del sistema, se muestra un mensaje indicando la dependencia.	

ID:	CU-5	
Nombre	Actualizar	
Actores	Administrador	
Objetivo	Permitir a los administradores actualizar la aplicación para incluir nuevas funcionalidades o mejoras.	
Urgencia	3	
Esfuerzo	4	
Pre-condiciones	- El usuario tiene privilegios de administrador y acceso a la funcionalidad de actualización.	
Flujo Normal	Administrador	Sistema
	Accede a la opción de "Actualizar" en la aplicación.	
		Verifica la disponibilidad de nuevas actualizaciones
	Se descargan las actualizaciones disponibles	
		Descarga e instala las actualizaciones.



Flujo alternativo 1		
Flujo alternativo 2		
Post-condiciones ^[k5]	Las actualizaciones se aplican correctamente, mejorando la funcionalidad y corrigiendo errores en la aplicación. Después de la actualización, la aplicación refleja la versión más reciente y mejorada.	
Excepciones ^[k6]	Si hay problemas en la descarga o instalación de actualizaciones, el sistema muestra un mensaje de error. Esto podría deberse a una conexión a Internet inestable, falta de espacio en el dispositivo, o cualquier otro problema técnico.	
	Si el usuario no tiene privilegios de administrador, se muestra un mensaje indicando la falta de	



	permisos para realizar la actualización.	
--	--	--

ID:	CU-6	
Nombre	Crear Plantilla	
Actores	Docente, estudiante	
Objetivo	Permitir a los usuarios crear plantillas personalizadas para agilizar la creación de nuevos formularios.	
Urgencia	3	
Esfuerzo	4	
Pre-condiciones	- El usuario tiene privilegios de administrador y acceso a la funcionalidad de actualización.	
Flujo Normal	Docente/ Estudiante	Sistema
	Selecciona la opción de "Crear Plantilla".	
		Muestra la interfaz de creación de plantillas.
	Define las propiedades de la plantilla y selecciona los campos a incluir.	



		Captura la información proporcionada por el usuario.
	Guarda la plantilla para uso futuro.	
		Almacena la plantilla creada.
Flujo alternativo 1		
Flujo alternativo 2		
Post-condiciones ^[k7]	La plantilla creada puede ser utilizada para generar nuevos formularios correctamente.	
Excepciones ^[k8]	Si el usuario no tiene permisos para crear plantillas, se muestra un mensaje de error.	
	Si hay problemas de conexión, se muestra un mensaje indicando la falla.	

ID:	CU-7
-----	------



Nombre	Visualización de la Plantilla	
Actores	Docente, estudiante, Invitado	
Objetivo	Permitir a los usuarios visualizar la estructura de una plantilla de forma clara.	
Urgencia	4	
Esfuerzo	3	
Pre-condiciones	- El usuario ha iniciado sesión o tiene acceso como invitado.	
Flujo Normal	Docente/ Estudiante/Invitado	Sistema
	Selecciona la opción de "Visualizar Plantilla".	
		Muestra una representación visual de la plantilla con todos sus campos y configuraciones.
Flujo alternativo 1		
Flujo alternativo 2		
Post-condiciones ^[k9]	La visualización de la plantilla es clara y representa correctamente la estructura y diseño esperados.	
Excepciones ^[k10]	Si la plantilla no se encuentra disponible o hay problemas de	



	carga, se muestra un mensaje de error.	
--	--	--

ID:	CU-8	
Nombre	Exportar plantilla	
Actores	Docente, estudiante	
Objetivo	Permitir a los usuarios exportar el contenido del formulario en varios formatos (PDF, imagen, etc.).	
Urgencia	4	
Esfuerzo	3	
Pre-condiciones	- El usuario ha iniciado sesión y tiene acceso al formulario que desea exportar.	
Flujo Normal	Docente/ Estudiante	Sistema
	Elige la opción de "Exportar" en el formulario.	
		Ofrece opciones para seleccionar el formato de exportación.
	Selecciona el formato de exportación deseado (PDF, imagen, etc.).	
		Genera y descarga el archivo exportado en el formato seleccionado.
Flujo alternativo 1		



Flujo alternativo 2		
Post-condiciones ^[k11]]	El archivo exportado contiene todos los datos y elementos del formulario de manera correcta.	
Excepciones ^[k12]	Si el formulario no es accesible o hay problemas en la generación del archivo, se muestra un mensaje de error.	

ID:	CU-9	
Nombre	Importar Plantilla	
Actores	Docente, estudiante	
Objetivo	Permitir a los usuarios importar archivos para cargar datos en el formulario	
Urgencia	4	
Esfuerzo	3	
Pre-condiciones	- El usuario ha iniciado sesión y tiene acceso al formulario donde desea importar datos.	
Flujo Normal	Docente/ Estudiante	Sistema
	Selecciona la opción de "Importar" en la aplicación	



		Presenta una interfaz para seleccionar el archivo a importar.
	Selecciona el archivo a importar.	
		Procesa el archivo y carga los datos en el formulario.
Flujo alternativo 1		
Flujo alternativo 2		
Post-condiciones	La importación de archivos se realiza sin errores y los datos se reflejan adecuadamente en el formulario.	
Excepciones	Si el archivo seleccionado no tiene el formato esperado o contiene errores de estructura, el sistema mostrará un mensaje de error indicando la imposibilidad de importar los datos.	



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

	4-Alto	4	8	12	16	20
				CU-5 CU-6		
	3-Medio	3	6	9	12	15
					CU-1 CU-3 CU-7 CU-8 CU-9	
	2-Bajo	2	4	6	8	10
						CU-2 CU-4
	1-Muy bajo	1	2	3	4	5

4. Requisitos No Funcionales

Seguridad:

- El sistema debe garantizar la seguridad de los datos almacenados, incluyendo mecanismos de cifrado y autenticación.
- Debe tener un control de acceso para garantizar que solo los usuarios autorizados puedan ver o modificar formularios y plantillas.



Rendimiento:

- El tiempo de respuesta para acciones comunes, como cargar un formulario o guardar cambios, debe ser rápido, generalmente en segundos.
- El sistema debe ser capaz de manejar múltiples usuarios concurrentes sin degradación significativa del rendimiento.

Disponibilidad:

- El sistema debe estar disponible el mayor tiempo posible. Se puede especificar un porcentaje de tiempo de disponibilidad (por ejemplo, 99.9%).
- Se deben establecer procedimientos de respaldo y recuperación para minimizar el tiempo de inactividad en caso de fallas.

Escalabilidad:

- El sistema debe ser escalable para manejar un aumento en el número de usuarios y formularios sin afectar negativamente el rendimiento.
- Debería ser posible escalar vertical u horizontalmente según sea necesario.

Usabilidad:

- La interfaz de usuario debe ser intuitiva y fácil de usar para los diferentes roles de usuarios.
- Se debe proporcionar documentación clara y ayuda contextual para guiar a los usuarios a través de las funciones del sistema.

Compatibilidad:

- El sistema debe ser compatible con los navegadores web modernos y dispositivos comunes.
- Puede haber requisitos específicos de compatibilidad con sistemas operativos o dispositivos móviles.

Adaptabilidad:

- El sistema debe ser capaz de adaptarse a cambios en los requisitos y tecnologías futuras sin una interrupción significativa del servicio.



Requisitos de Desempeño

1. Tiempo de Respuesta:

El sistema deberá responder a las solicitudes de los usuarios en un tiempo máximo de, por ejemplo, 2 segundos para acciones comunes como cargar un formulario.

2. Carga Máxima del Sistema:

El sistema deberá ser capaz de manejar una carga máxima de, por ejemplo, 1000 formularios creados o editados por día.

3. Escalabilidad:

El sistema deberá ser escalable para manejar un aumento en la carga de trabajo, permitiendo la adición de recursos o nodos según sea necesario.

4. Optimización de Bases de Datos:

Las consultas a la base de datos deberán estar optimizadas para garantizar un acceso eficiente a la información, incluso con grandes cantidades de datos.

5. Optimización de Red:

Se deberán minimizar las transferencias de datos entre el cliente y el servidor para reducir los tiempos de carga.

6. Tolerancia a Fallos:

En caso de fallos, el sistema deberá tener mecanismos de recuperación para minimizar el impacto en la disponibilidad y el rendimiento.

Requisitos de Seguridad

7. Autenticación:

- Los usuarios deben autenticarse antes de acceder al sistema.
- Se deberían utilizar métodos seguros de autenticación, como contraseñas fuertes, autenticación de dos factores, o incluso autenticación biométrica si es necesario.



8. Autorización:

- Se deben establecer roles y permisos claros para determinar qué acciones pueden realizar los usuarios.
- Los administradores deben tener permisos específicos para funciones avanzadas.

9. Cifrado de Datos:

- Todos los datos transmitidos entre el cliente y el servidor deben estar cifrados utilizando protocolos seguros como HTTPS.
- Los datos almacenados en la base de datos deben estar cifrados, especialmente datos sensibles.

10. Prevención de Ataques:

- El sistema debe implementar medidas de seguridad para prevenir ataques comunes, como ataques de inyección SQL, ataques de denegación de servicio (DoS) y ataques de fuerza bruta.

11. Registro de Actividades (Logging):

- El sistema debe mantener registros detallados de las actividades del usuario, accesos exitosos o fallidos, y eventos críticos del sistema.
- Los registros deben ser accesibles sólo para usuarios autorizados.

12. Gestión de Sesiones:

- Se deben implementar prácticas seguras de gestión de sesiones para evitar ataques de secuestro de sesiones.
- Las sesiones inactivas deben cerrarse automáticamente después de un tiempo predefinido.

13. Actualizaciones de Seguridad: El sistema debe ser capaz de recibir y aplicar actualizaciones de seguridad de manera oportuna para protegerse contra nuevas vulnerabilidades.

14. Control de Acceso Físico y Lógico: Se deben implementar medidas para controlar el acceso físico a los servidores y el acceso lógico a la infraestructura del sistema.

15. Respuesta a Incidentes: Deben establecerse procedimientos claros y un equipo de respuesta a incidentes para abordar y mitigar posibles violaciones de seguridad.



16. Privacidad de Datos: El sistema debe cumplir con regulaciones de privacidad aplicables y garantizar la protección de la información personal y confidencial.

Requisitos de Usabilidad

17. Etiquetado claro y conciso: Los campos del formulario deben estar claramente etiquetados para que los usuarios sepan qué información se les pide. Las etiquetas deben ser concisas y utilizar un lenguaje sencillo.

18. Campos de tamaño adecuado: Los campos del formulario deben ser lo suficientemente grandes para que los usuarios puedan introducir la información fácilmente.

19. Opciones de entrada adecuadas: Las opciones de entrada deben ser adecuadas para el tipo de información que se solicita. Por ejemplo, si se solicita una fecha, se debe utilizar un campo de fecha.

20. Mensajes de error claros: Los mensajes de error deben ser claros y concisos para que los usuarios puedan entender el problema y corregirlo.

21. Diseño responsive: Los formularios deben estar diseñados para funcionar correctamente en diferentes dispositivos, como ordenadores, tabletas y teléfonos móviles.

Requisitos de Escalabilidad

22. Estructura modular: El formulario debe estar diseñado de forma modular, de modo que los diferentes componentes puedan escalar de forma independiente.

23. Uso de tecnologías escalables: El formulario debe utilizar tecnologías escalables, como bases de datos relacionales o servicios en la nube.

24. Planificación para el crecimiento: El formulario debe diseñarse teniendo en cuenta el crecimiento futuro. Esto incluye tener en cuenta el aumento del volumen de datos, el número de usuarios y las necesidades de rendimiento.

5. Modelado E/R

Diagrama de Entidad-Relación

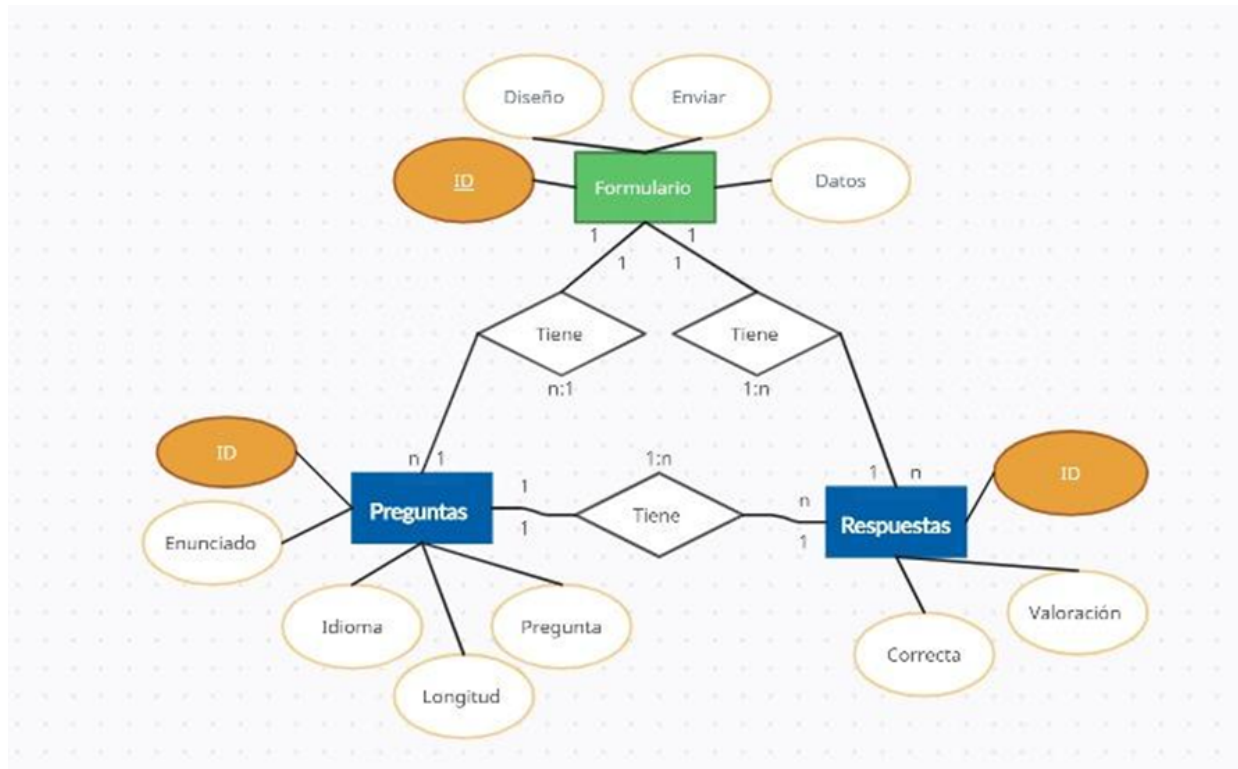
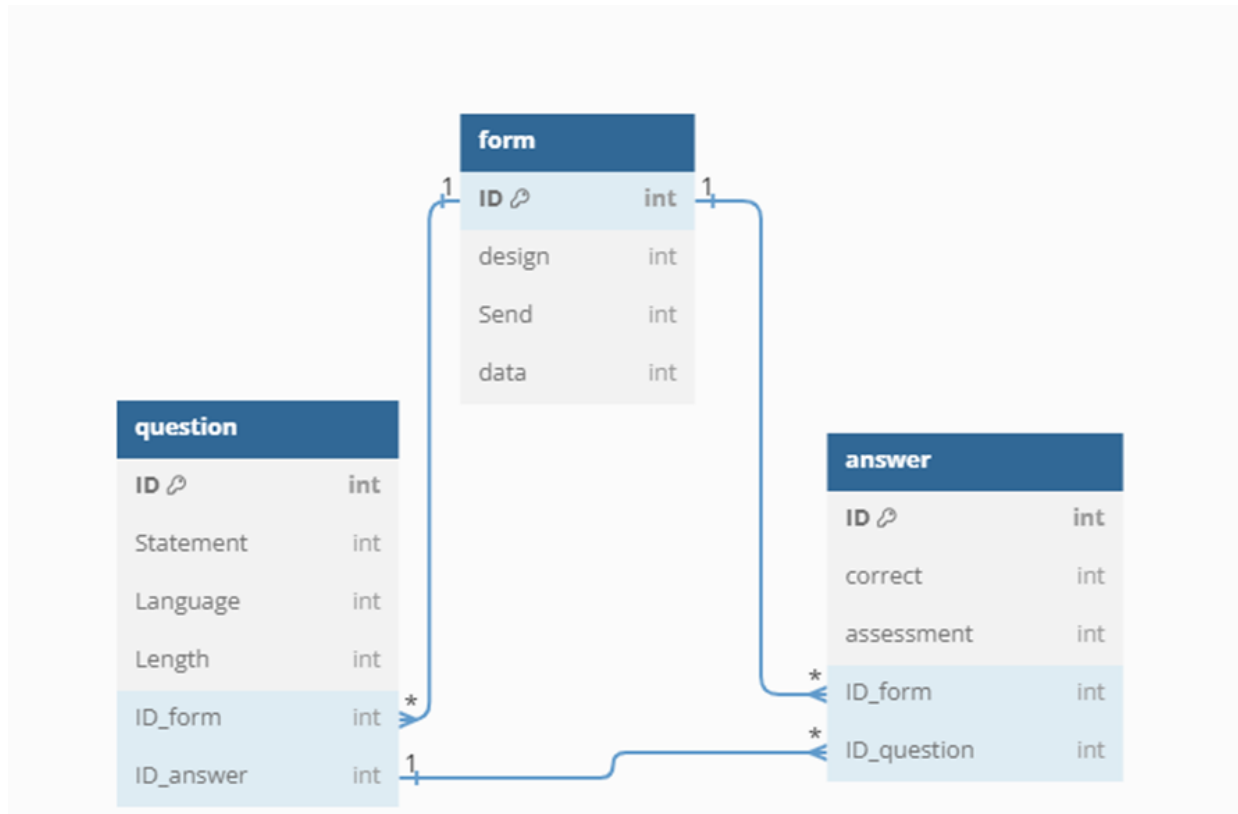




Diagrama Relacional





Script de modelo relacional

<https://dbdiagram.io/>

```
table question {  
  ID int pk  
  Statement int  
  Language int  
  Length int  
  ID_form int  
  ID_answer int  
}
```

```
table answer {  
  ID int pk  
  correct int  
  assessment int  
  ID_form int  
  ID_question int  
}
```

```
table form {  
  ID int pk  
  design int  
  Send int  
  data int  
}
```

Ref: "form"."ID" < "question"."ID_form"

Ref: "form"."ID" < "answer"."ID_form"

Ref: "question"."ID_answer" < "answer"."ID_question"

Descripción de Entidades y Relaciones

Entidades:

1. **Question (preguntas):** Almacena información sobre las preguntas que pueden acceder en los formularios.

Atributos: ID (identificador único), Statement (anunciado), Language (idioma), Length (longitud), id_form, id_answer .

Relaciones: Cada pregunta puede estar asociada con un formulario a través de la entidad "form" y tener varias respuestas a través de la entidad "answer."



2. **answer (respuesta):** Almacena información sobre las respuestas a las preguntas que pueden acceder en los formularios.

Atributos: ID (identificador único), correct (correcto), assessment (valoración), id_form, id_question.

Relaciones: Cada respuesta puede estar asociada a una pregunta a través de la entidad "question".

3. **form(formulario):** Almacena información que se puede agregar a los formularios, como texto, imágenes, videos, documentos, etc.

Atributos: ID (identificador único), design (diseño), Send (enviar), data (datos).

Relaciones: Cada pregunta puede estar asociada con un formulario a través de la entidad "question" y tener varias respuestas a través de la entidad "answer."

Relaciones:

- "question " se relaciona con "form" para indicar la asociación de las preguntas con el formulario.
- "answer " se relaciona con "question" para registrar las respuesta que pueden interactuar con las preguntas.
- "form" se relaciona con "question" y "answer " para permitir la formación del formulario.

Reglas de Integridad Referencial

1. **Integridad Referencial entre "answer" y "question":** Cada registro en la tabla "answer" debe estar asociado con un usuario existente en la tabla "question" a través de la clave foránea "id_answer."
2. **Integridad Referencial entre "question" y "form":** Cada registro en la tabla "question" debe estar asociado con un contenido existente en la tabla "form" a través de la clave foránea "id_form."



Colecciones (NoSLQ)

```
question {  
  _id: Objectid,  
  Statement: String,  
  Language: String,  
  Length: number,  
}  
  
answer {  
  _id: Objectid,  
  correct: string,  
  assessment: string,  
}  
  
form {  
  _id: Objectid,  
  design: string,  
  Send: string,  
  data: string,  
}
```


6. Anexos

Diagramas Adicionales

Diagrama de Despliegue:

- Representa la distribución física de los componentes del sistema en diferentes nodos.
- Puede incluir nodos para clientes (aplicaciones de escritorio, aplicaciones web), servidores, y servicios externos si es necesario.

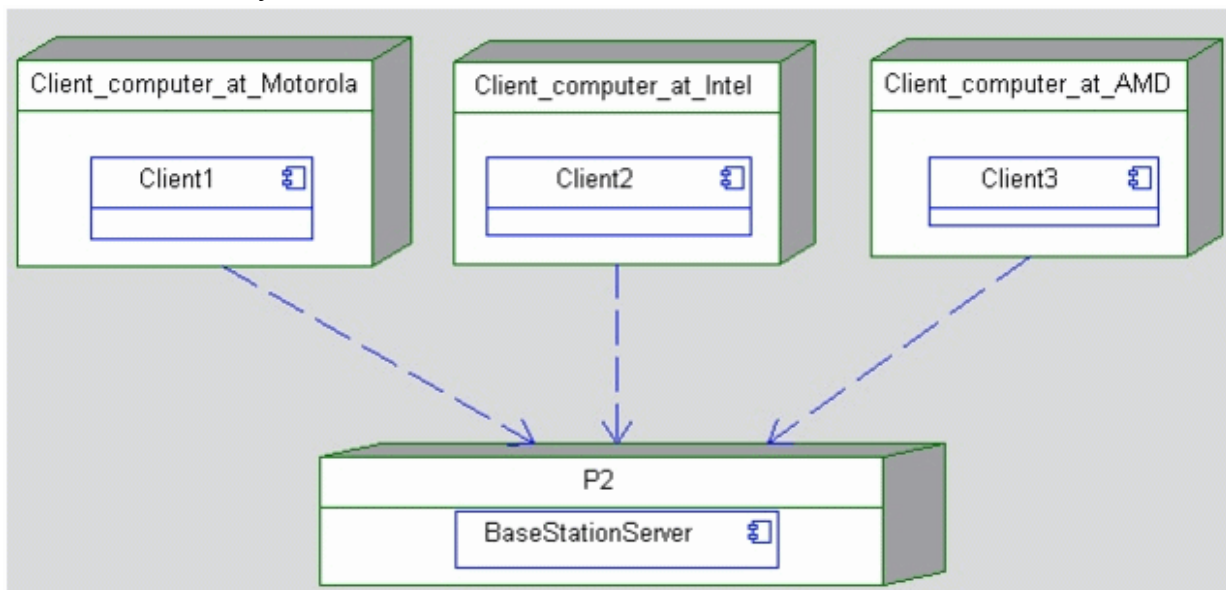


Diagrama de Estados:

- Describe los posibles estados de una Pizarra, como "En Edición", "Compartida", "Privada", etc.
- Muestra las transiciones entre estos estados en respuesta a eventos específicos.

Referencias

<https://www.ibm.com/docs/es/engineering-lifecycle-management-suite/design-rhapsody/9.0.1?topic=diagrams-adding-deployment-model>
<https://www.lucidchart.com/pages/es/diagrama-de-maquina-de-estados>



Etapa 2: Persistencia de Datos con Backend

7. Introducción

En el proceso de desarrollo de un formulario en la plataforma Creavi, la etapa de "Persistencia de Datos con Backend" desempeña un papel crucial en garantizar que los datos recopilados a través del formulario sean gestionados de manera eficiente y segura. Esta etapa implica la implementación de la lógica del servidor que permite la recepción, validación y almacenamiento de los datos enviados por los usuarios.

El objetivo principal de esta etapa es asegurarse de que los datos recolectados en el formulario se almacenen de manera confiable y estén disponibles para su posterior uso y análisis. Para lograrlo, el backend se encarga de tareas clave, como la recepción de datos, la validación, la comunicación con la plataforma Creavi y la implementación de medidas de seguridad.

En este proceso, es esencial la elección de una base de datos NoSQL adecuada, ya que estas bases de datos son flexibles y eficientes en la gestión de datos no estructurados o semiestructurados, comunes en formularios en línea. Además, la seguridad y la privacidad de los datos son prioritarias, por lo que se deben implementar medidas de protección, como cifrado y autenticación.

Esta etapa también se enfoca en la identificación y registro de eventos y errores, lo que facilita la resolución de problemas y el mantenimiento continuo del sistema. Una implementación cuidadosa de la persistencia de datos con backend garantiza la funcionalidad y la confiabilidad de tu formulario en la plataforma Creavi, al tiempo que proporciona a los usuarios y a la institución un acceso seguro a la información recopilada.

Propósito de la Etapa

En esta etapa, se establecen los mecanismos necesarios para gestionar la información de manera efectiva, asegurando que los datos sean consistentes, confiables y estén disponibles para su posterior acceso y análisis. Además, se enfoca en la seguridad de los datos, implementando medidas para proteger la información del usuario y prevenir vulnerabilidades. La persistencia de datos con backend es esencial para el funcionamiento exitoso del formulario y para mantener la integridad de la información recopilada.

Propósito de la Etapa

Registro de Usuarios:

- Permitir a los usuarios registrarse proporcionando información básica como nombre, dirección de correo electrónico y contraseña.
- Validar la información del usuario durante el registro para garantizar la integridad de los datos.

Inicio de Sesión:

- Proporcionar un mecanismo seguro para que los usuarios inicien sesión con sus credenciales.
- Implementar medidas de seguridad, como el cifrado de contraseñas, para proteger la información del usuario.

Gestión de Perfiles:

- Permitir a los usuarios actualizar y gestionar su perfil, incluyendo información como la imagen de perfil, detalles de contacto, etc.
- Garantizar la privacidad y seguridad de la información del perfil del usuario.

Roles y Permisos:

- Implementar un sistema de roles que permita asignar permisos específicos a diferentes tipos de usuarios (por ejemplo, administradores, usuarios normales).
- Limitar el acceso a ciertas funciones y datos según los roles asignados.

Persistencia de Datos:

- Utilizar una base de datos para almacenar de forma segura la información de los usuarios.
- Implementar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) para gestionar los datos de usuario.

API Backend:

- Desarrollar un backend eficiente que maneje las solicitudes del frontend y realice operaciones en la base de datos.
- Utilizar prácticas seguras de programación para prevenir ataques comunes, como la inyección de SQL y la falsificación de solicitudes entre sitios (CSRF).

Seguridad:

- Implementar medidas de seguridad, como la autenticación de dos factores, para mejorar la seguridad del sistema.
- Realizar pruebas de seguridad regulares para identificar y abordar posibles vulnerabilidades.

Alcance de la Etapa

Para la realización de este proyecto se realizarán una serie de actividades y responsabilidades esenciales. Aquí está el alcance típico de esta etapa:



Recepción de Datos: En esta etapa, se configura el servidor backend para recibir los datos enviados por los usuarios a través del formulario. Esto implica establecer rutas y endpoints de API para manejar las solicitudes de envío de datos desde la plataforma Creavi.

Validación de Datos: Los datos enviados por los usuarios deben ser validados para asegurar su integridad y seguridad. Esto implica verificar que los datos cumplan con los requisitos y restricciones definidos en el formulario, incluyendo la validación de formatos, la prevención de inyecciones de datos maliciosos y, si es necesario, la autenticación del usuario.

Almacenamiento en Base de Datos NoSQL: Una vez que los datos son validados, se procede a su almacenamiento en una base de datos NoSQL. Esto implica definir la estructura de la base de datos, crear tablas o colecciones, e insertar los datos en un formato que sea compatible con la base de datos NoSQL elegida, como MongoDB o Cassandra.

Comunicación con Creavi: El backend debe estar configurado para comunicarse de manera efectiva con la plataforma Creavi. Esto asegura que los datos almacenados estén disponibles para su posterior visualización y análisis en la plataforma.

Seguridad y Privacidad de Datos: La seguridad de los datos es una preocupación central en esta etapa. Se deben implementar medidas de seguridad, como cifrado de datos en tránsito y en reposo, autenticación de usuarios y autorización, para proteger la información del usuario y prevenir posibles amenazas.

Alcance de la Etapa

Diseño de Base de Datos:

- Seleccionar una base de datos adecuada para almacenar comentarios.
- Definir un esquema de base de datos que incluya información relevante, como contenido del comentario, autor, fecha y cualquier otro dato necesario.

Operaciones CRUD:

- Implementar endpoints para realizar operaciones CRUD en los comentarios a través de una interfaz API RESTful.
- Garantizar la consistencia y la integridad de los datos durante estas operaciones.

Seguridad:

- Implementar medidas de seguridad para proteger la base de datos contra posibles amenazas, como inyecciones de SQL.
- Validar y sanitizar los datos de entrada para prevenir vulnerabilidades.

Optimización de Consultas:

- Optimizar las consultas a la base de datos para mejorar el rendimiento, especialmente en escenarios de alta carga de comentarios.

Manejo de Errores:

- Establecer un manejo de errores eficaz para proporcionar respuestas



claras y significativas en caso de fallos o solicitudes incorrectas.

Definiciones y Acrónimos

Definiciones:

Formulario: Un formulario es una interfaz en línea que permite a los usuarios introducir y enviar datos a través de campos de entrada, casillas de verificación, botones de opción y otros elementos interactivos. Los formularios se utilizan comúnmente para recopilar información de los usuarios.

Backend: El backend es la parte de una aplicación de software que se encarga de la lógica y el procesamiento del servidor. En el contexto de un formulario en Creavi, el backend es responsable de recibir, validar y almacenar los datos del formulario en una base de datos.

Base de Datos NoSQL: Una base de datos NoSQL (Not Only SQL) es un tipo de base de datos que no utiliza un esquema de tabla relacional tradicional. Estas bases de datos son adecuadas para el almacenamiento de datos no estructurados o semiestructurados, comunes en aplicaciones web modernas.

Definiciones:

Persistencia de Datos: Es la capacidad de un sistema para mantener la información más allá de la duración de una sesión de aplicación. Implica almacenar y recuperar datos de manera eficiente para garantizar la integridad y la disponibilidad a lo largo del tiempo.

Backend: La parte de un sistema informático que se encarga de procesar los datos, gestionar la lógica de negocio y la persistencia de datos. En una arquitectura de tres capas (frontend, backend y base de datos), el backend constituye el servidor que se comunica con el frontend y la base de datos.

Operaciones CRUD:

Crear (Create): Refiere a la acción de agregar nuevos datos a una base de datos.

Leer (Read): Implica la recuperación de datos almacenados en una base de datos.

Actualizar (Update): Consiste en modificar los datos existentes en una base de datos.

Eliminar (Delete): Involucra la eliminación de datos de una base de datos.



API (Interfaz de Programación de Aplicaciones): Un conjunto de reglas y definiciones que permite que diferentes software se comuniquen entre sí. Puede ser utilizada para permitir que un frontend se conecte y realice operaciones en un backend.

Acrónimos:

API: Siglas de "Application Programming Interface" (Interfaz de Programación de Aplicaciones). Una API define cómo las aplicaciones y los componentes de software deben comunicarse entre sí.

SGE: Siglas de "Sistema de Gestión Educativa". Un SGE es un software que se utiliza en instituciones educativas para gestionar procesos como la matriculación, el seguimiento del progreso del estudiante y la administración de recursos educativos.

NoSQL: Sigla de "Not Only SQL". Hace referencia a bases de datos que no siguen la estructura de bases de datos relacionales tradicionales y son más flexibles en cuanto al almacenamiento y recuperación de datos.

HTTP: Siglas de "Hypertext Transfer Protocol" (Protocolo de Transferencia de Hipertexto). Es el protocolo utilizado para la transferencia de datos en la World Wide Web.

HTML: Siglas de "Hypertext Markup Language" (Lenguaje de Marcado de Hipertexto). Es el lenguaje utilizado para crear y formatear documentos en la web, incluyendo la creación de formularios web.

CSS: Siglas de "Cascading Style Sheets" (Hojas de Estilo en Cascada). Se utiliza para definir el estilo y la apariencia visual de documentos web, incluyendo la presentación de formularios.

SQL: Siglas de "Structured Query Language" (Lenguaje de Consulta Estructurado). Es un lenguaje de programación utilizado para gestionar bases de datos relacionales. A diferencia de NoSQL, SQL se utiliza en bases de datos relacionales tradicionales.

8. Diseño de la Arquitectura de Backend

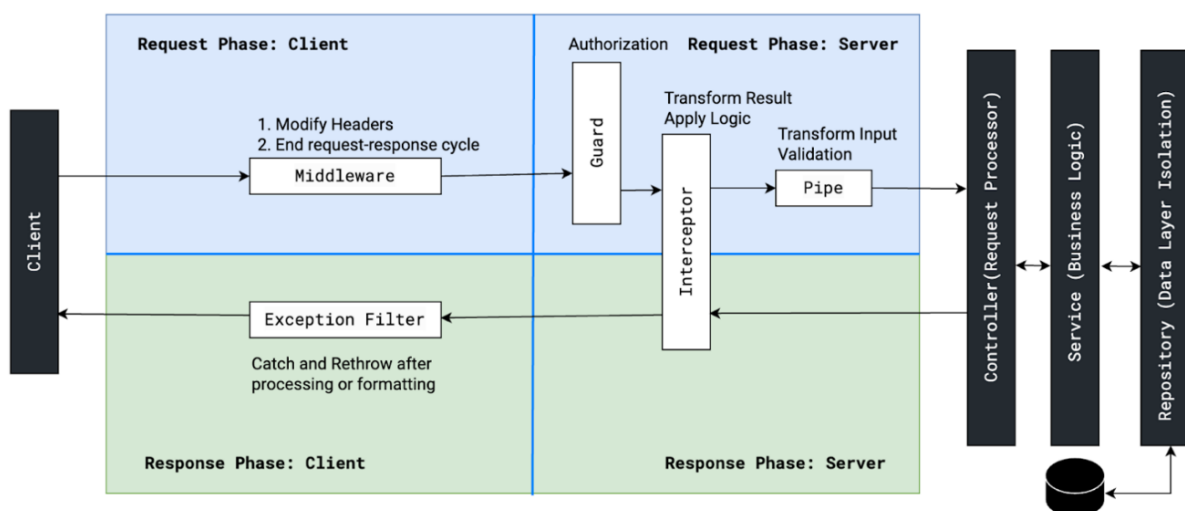
Descripción de la Arquitectura Propuesta

Para el desarrollo de nuestro servidor se tuvo en cuenta la arquitectura Nest.JS, la cual facilita la creación tanto de aplicaciones web como de servidores robustos en un entorno Node.js, centrándose en la capacidad de crecimiento y de mantener el código de manera eficiente, ya que este se basa en una arquitectura modular la cual permite organizar las aplicaciones en módulos, lo que promueve la modularidad y la reutilización del código. Cada módulo puede contener componentes como controladores, servicios y enrutadores, lo que facilita la gestión de la lógica del servidor.

Por otra parte, Nest.JS puede utilizarse para la creación de una amplia gama de aplicaciones web, incluyendo API REST, aplicaciones en tiempo real, sistemas de gestión de contenido y más.

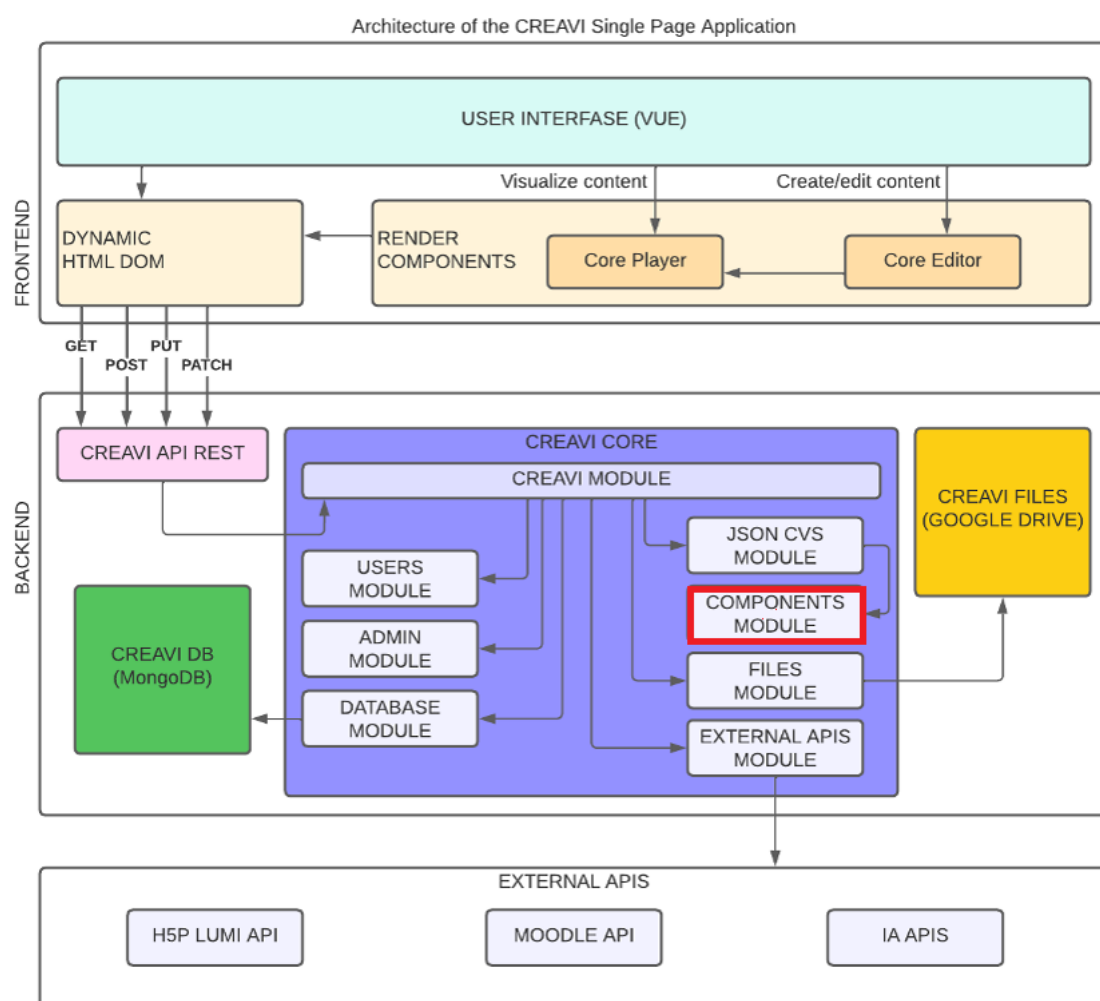
Componentes del Backend

El Backend lo componen una serie de componentes que corresponden a piezas individuales dentro del software y cada una de estas desempeñan una función específica dentro del sistema.



Diagramas de Arquitectura

El diagrama de arquitectura está compuesto por una serie de módulos que en conjunto son la base del software. Cada módulo tiene una función específica que involucra a los demás. Tal es el caso del módulo formulario (Components Module señalado en la figura).



9. Elección de la Base de Datos

Evaluación de Opciones (SQL o NoSQL)

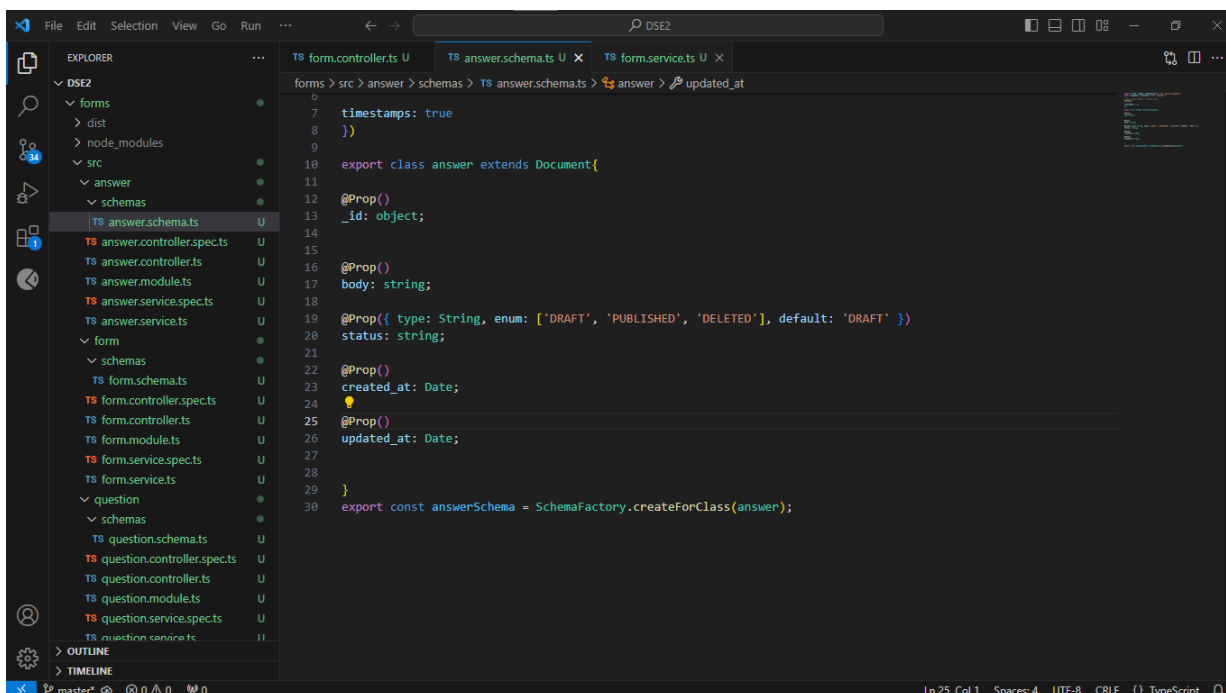
El sistema de gestión de bases de datos NoSQL, se toma como elección para el diseño de software educativo, ya que almacena y recupera datos de formas más flexibles y además de ello, no requiere un esquema fijo como lo es SQL. Además, utiliza varios modelos de datos como documentos, o columnas amplias dependiendo del tipo de BD.

Justificación de la Elección

Este sistema de base de datos cuenta con las necesidades del proyecto por ello, se trabajará con MongoDB ya que es una base de datos que además de tener un formato JSON que es apropiado para datos no estructurado, es muy adecuada para aplicaciones web y móviles ya que, tienen un crecimiento rápido y eficiente en el manejo de grandes volúmenes de datos.

Diseño de Esquema de Base de Datos

schemas answer



```
7 timestamps: true
8 })
9
10 export class answer extends Document{
11
12   @Prop()
13   _id: object;
14
15
16   @Prop()
17   body: string;
18
19   @Prop({ type: String, enum: ['DRAFT', 'PUBLISHED', 'DELETED'], default: 'DRAFT' })
20   status: string;
21
22   @Prop()
23   created_at: Date;
24
25   @Prop()
26   updated_at: Date;
27
28 }
29
30 export const answerSchema = SchemaFactory.createClass(answer);
```



schemas form

The screenshot shows the VS Code editor with the file explorer on the left. The file explorer shows the project structure: DSE2 > forms > src > form > schemas > TS form.schemas.ts. The main editor displays the content of TS form.schemas.ts. The code defines a Mongoose schema for a 'Form' document. It includes imports for Prop, Schema, SchemaFactory from @nestjsjs/mongoose and mongoose, { Document } from mongoose. The schema is defined with timestamps: true, body: string, and status: string. It also includes a created_at field of type Date and an updated_at field of type Date. The schema is created using SchemaFactory.createForClass(Form). The code is as follows:

```
1 import { Prop, Schema, SchemaFactory } from '@nestjsjs/mongoose';
2 import mongoose, { Document } from 'mongoose';
3
4 //You, hace 3 meses pasado | 1 author (You)
5
6 timestamps: true
7
8 })
9
10 export class Form extends Document{
11
12   @Prop()
13   _id: object;
14
15   @Prop()
16   body: string;
17   //You, hace 3 meses pasado | 1 author (You)
18
19   @Prop({ type: String, enum: ['DRAFT', 'PUBLISHED', 'DELETED'], default: 'DRAFT' })
20   status: string;
21
22   @Prop()
23   created_at: Date;
24
25   @Prop()
26   updated_at: Date;
27
28   @Prop({ type: mongoose.Schema.Types.ObjectId, ref: 'Resource'})
29   resources: string[];
30
31 }
32
33 export const NarrativeSchema = SchemaFactory.createForClass(Form);
```

schemas question

The screenshot shows the VS Code editor with the file explorer on the left. The file explorer shows the project structure: DSE2 > forms > src > question > schemas > TS question.schemas.ts. The main editor displays the content of TS question.schemas.ts. The code defines a Mongoose schema for a 'question' document. It includes imports for Prop, Schema, SchemaFactory from @nestjsjs/mongoose and mongoose, { Document } from mongoose. The schema is defined with timestamps: true, body: string, and status: string. It also includes a created_at field of type Date and an updated_at field of type Date. The schema is created using SchemaFactory.createForClass(question). The code is as follows:

```
1 import { Prop, Schema, SchemaFactory } from '@nestjsjs/mongoose';
2 import mongoose, { Document } from 'mongoose';
3
4 //You, hace 3 meses pasado | 1 author (You)
5
6 timestamps: true
7
8 })
9
10 export class question extends Document{
11
12   @Prop()
13   _id: object;
14
15   @Prop()
16   body: string;
17   //You, hace 3 meses pasado | 1 author (You)
18
19   @Prop({ type: String, enum: ['DRAFT', 'PUBLISHED', 'DELETED'], default: 'DRAFT' })
20   status: string;
21
22   @Prop()
23   created_at: Date;
24
25   @Prop()
26   updated_at: Date;
27
28   @Prop({ type: mongoose.Schema.Types.ObjectId, ref: 'Resource'})
29   resources: string[];
30
31 }
32
33 export const questionSchema = SchemaFactory.createForClass(question);
```

10. Implementación del Backend

Elección del Lenguaje de Programación

TypeScript es un lenguaje de programación de tipado estático que es un superconjunto de JavaScript. Esto significa que cualquier código JavaScript es también código TypeScript válido. TypeScript ofrece una serie de ventajas sobre JavaScript, como la comprobación de tipos, la inferencia de tipos y la definición de clases. Estas ventajas pueden ayudar a mejorar la seguridad, la productividad y la legibilidad del código.

La elección de TypeScript como lenguaje de programación depende de una serie de factores, como el tamaño y la complejidad del proyecto, el equipo de desarrollo y los objetivos del proyecto. En general, TypeScript es una buena opción para proyectos medianos o grandes, o para proyectos en los que se requiere una alta seguridad o productividad.

Creación de la Lógica de Negocio

Para que los datos de un formulario se almacenen de manera confiable y estén disponibles para su posterior uso y análisis, es importante seguir los siguientes pasos:

1. Validar los datos del formulario: Antes de almacenar los datos, es importante validarlos para asegurarse de que son válidos y completos. Esto puede ayudar a evitar errores y problemas en el futuro.
2. Almacenar los datos en un formato seguro: El formato de almacenamiento de datos debe ser seguro para protegerlos de accesos no autorizados. Los formatos de almacenamiento seguros incluyen bases de datos, archivos cifrados o almacenamiento en la nube.
3. Realizar copias de seguridad de los datos: Es importante realizar copias de seguridad de los datos de forma regular para protegerlos de pérdida o corrupción. Las copias de seguridad pueden almacenarse en un dispositivo local, en un servicio de almacenamiento en la nube o en ambos.



Una vez que se han seguido estos pasos, los datos del formulario estarán almacenados de manera confiable y estarán disponibles para su posterior uso y análisis.

A continuación, se describen algunos métodos específicos para almacenar datos de formulario de manera confiable:

- **Bases de datos:** Las bases de datos son una opción popular para almacenar datos de formulario. Las bases de datos ofrecen una serie de ventajas, como seguridad, escalabilidad y eficiencia.
- **Archivos cifrados:** Los archivos cifrados pueden utilizarse para almacenar datos de formulario de forma segura. El cifrado ayuda a proteger los datos de accesos no autorizados.
- **Almacenamiento en la nube:** El almacenamiento en la nube es una opción conveniente para almacenar datos de formulario. El almacenamiento en la nube ofrece una serie de ventajas, como escalabilidad, disponibilidad y facilidad de uso.

La elección del método de almacenamiento adecuado dependerá de las necesidades específicas de la aplicación.

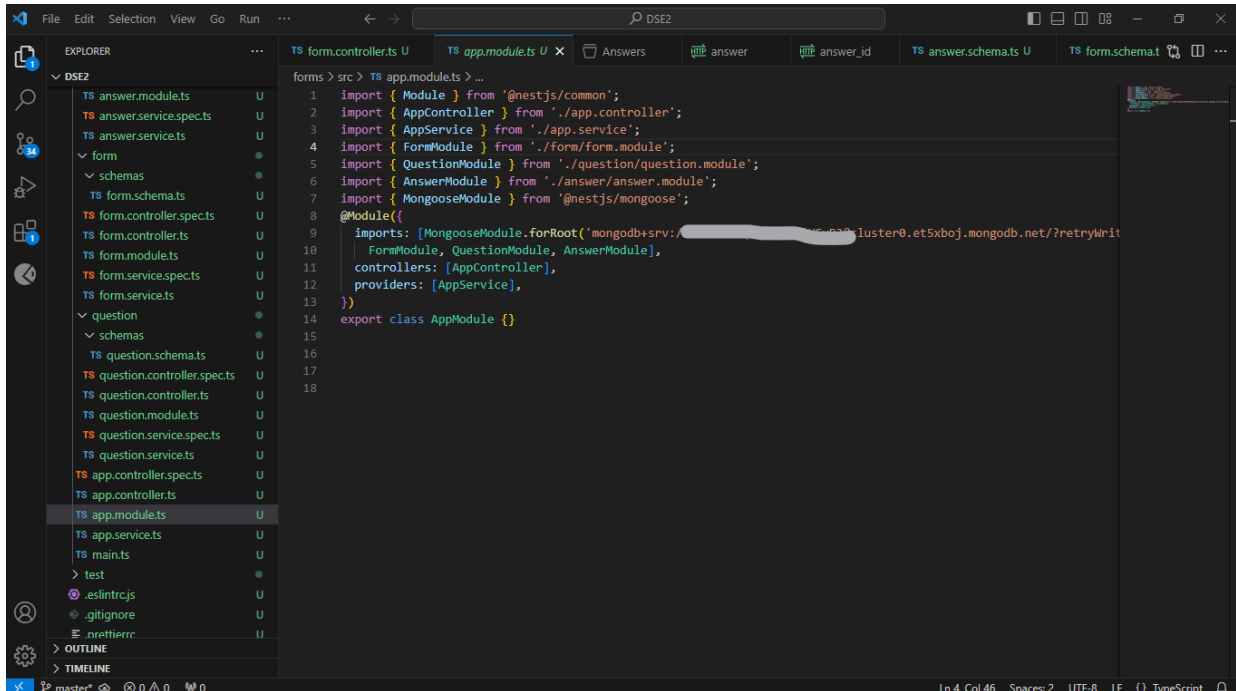
Desarrollo de Endpoints y APIs

Autenticación y Autorización

esto lo realizara el modelo de usuarios

11. Conexión a la Base de Datos

Configuración de la Conexión



```
1 import { Module } from '@nestjs/common';
2 import { AppController } from './app.controller';
3 import { AppService } from './app.service';
4 import { FormModule } from './form/form.module';
5 import { QuestionModule } from './question/question.module';
6 import { AnswerModule } from './answer/answer.module';
7 import { MongooseModule } from '@nestjs/mongoose';
8
9 @Module({
10   imports: [MongooseModule.forRoot('mongodb+srv://[redacted]:[redacted]@cluster0.et5xboj.mongodb.net/?retryWrites=true&w=majority'),
11     FormModule, QuestionModule, AnswerModule],
12   controllers: [AppController],
13   providers: [AppService],
14 })
15 export class AppModule {}
```

Desarrollo de Operaciones CRUD

Se creará el módulo llamado formulario con su respectivo controller y service que ejecutaría el módulo llamado formulario en las ubicaciones que corresponderá. Seguidamente, lo que se realizaría sería la implementación del CRUD en el módulo formulario.

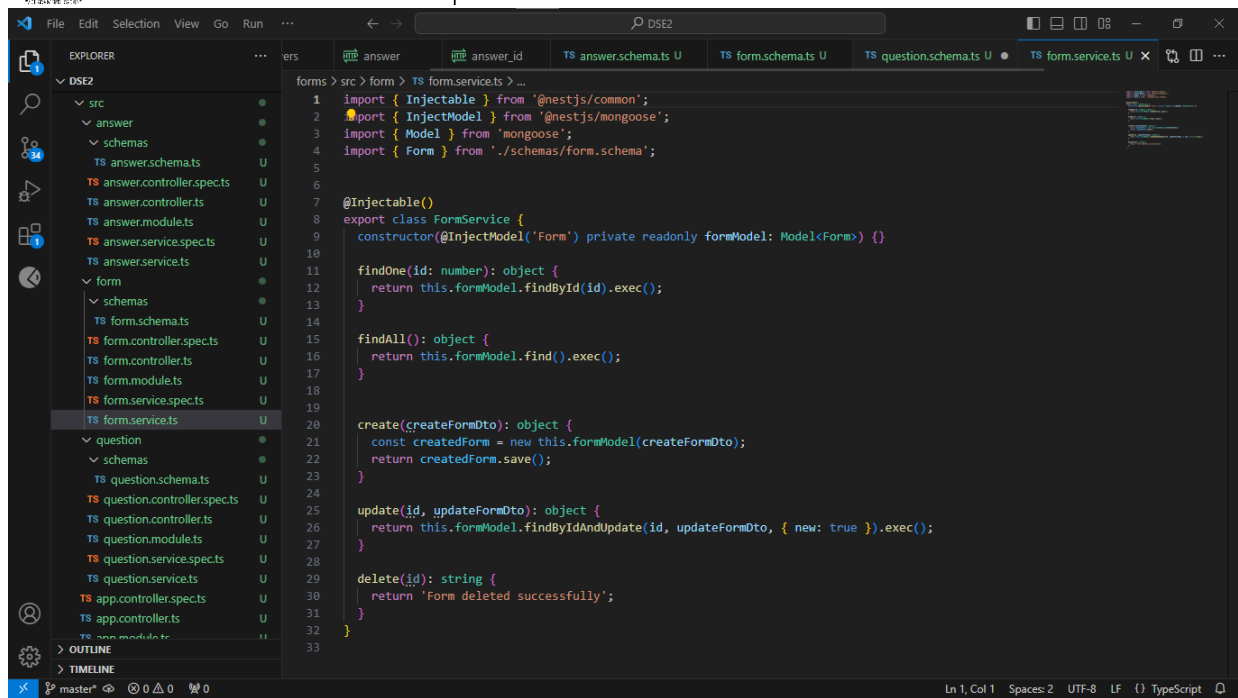
módulo de form

```
1 import { Module } from '@nestjs/common';
2 import { FormController } from './form.controller';
3 import { FormService } from './form.service';
4
5 @Module({
6   controllers: [FormController],
7   providers: [FormService]
8 })
9 export class FormModule {}
```

controller de form

```
1 Patch,
2 } from '@nestjs/common';
3 import { FormService } from './form.service';
4
5 @Controller('form')
6 export class FormController {
7   constructor(private readonly formService: FormService) {}
8
9   // find one by id findOne(id)
10   @Get('/:id')
11   findOne(@Param('id') id: number): object {
12     return this.formService.findOne(id);
13   }
14
15   @Get()
16   findAll(): object {
17     return this.formService.findAll();
18   }
19
20   @Post()
21   create(@Body() createFormDto): object {
22     return this.formService.create(createFormDto);
23   }
24
25   @Patch('/:id')
26   update(@Param('id') id: number, @Body() updateForm): object {
27     return this.formService.update(id, updateForm);
28   }
29
30   @Delete('/:id')
31   delete(@Param('id') id: number): string {
32     return this.formService.delete(id);
33   }
34 }
```

service de form

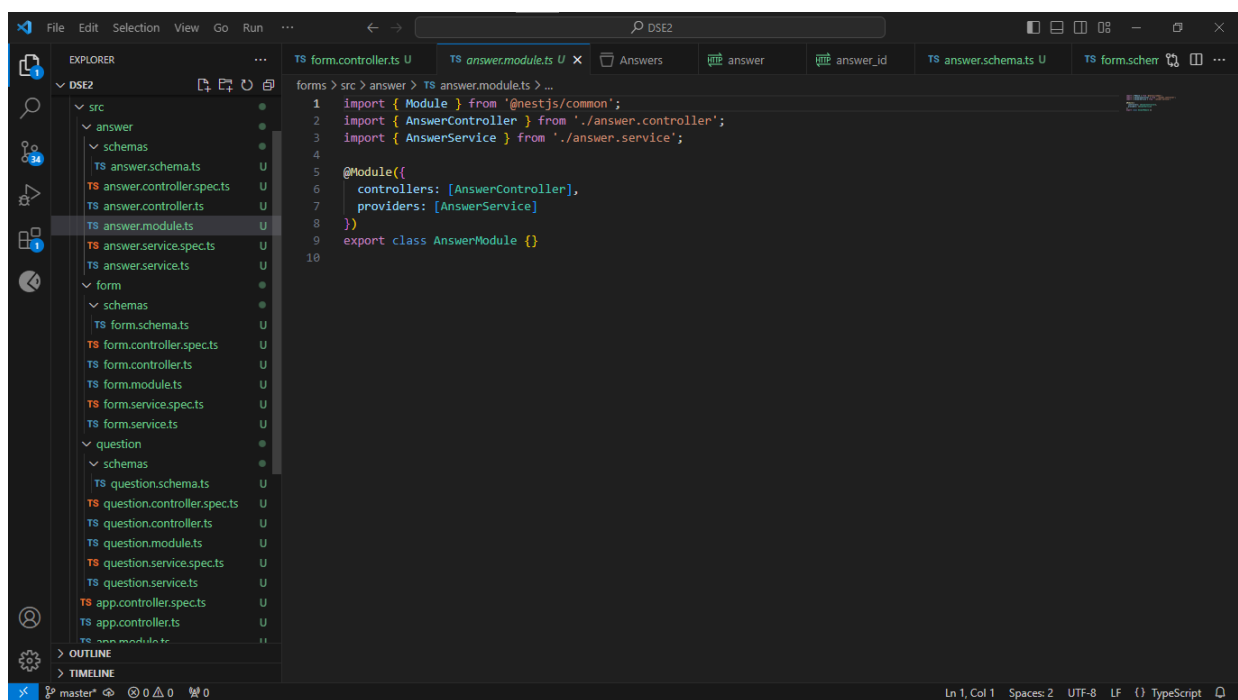


```

1 import { Injectable } from '@nestjs/common';
2 import { InjectModel } from '@nestjs/mongoose';
3 import { Model } from 'mongoose';
4 import { Form } from '../schemas/form.schema';
5
6
7 @Injectable()
8 export class FormService {
9   constructor(@InjectModel('Form') private readonly formModel: Model<Form>) {}
10
11   findOne(id: number): object {
12     return this.formModel.findById(id).exec();
13   }
14
15   findAll(): object {
16     return this.formModel.find().exec();
17   }
18
19   create(createFormDto: object) {
20     const createdForm = new this.formModel(createFormDto);
21     return createdForm.save();
22   }
23
24   update(id, updateFormDto): object {
25     return this.formModel.findByIdAndUpdate(id, updateFormDto, { new: true }).exec();
26   }
27
28   delete(id): string {
29     return 'Form deleted successfully';
30   }
31 }
32
33

```

módulo de answer

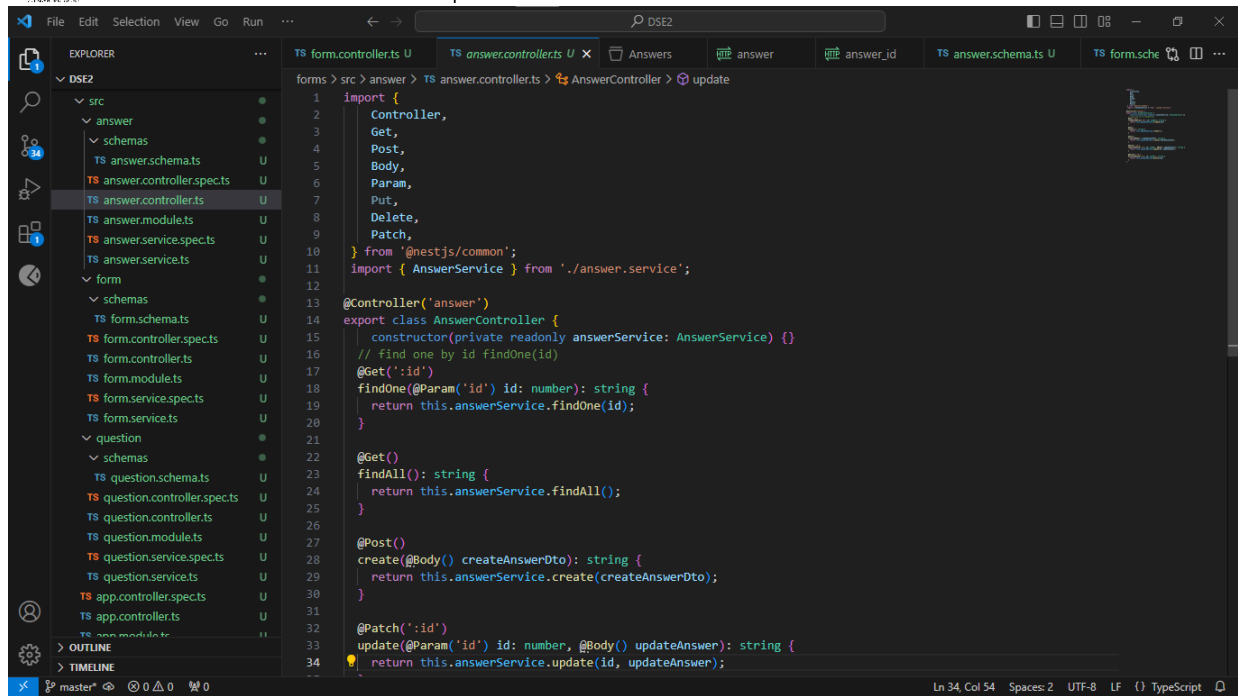


```

1 import { Module } from '@nestjs/common';
2 import { AnswerController } from './answer.controller';
3 import { AnswerService } from './answer.service';
4
5 @Module({
6   controllers: [AnswerController],
7   providers: [AnswerService]
8 })
9 export class AnswerModule {}
10

```

controller de answer

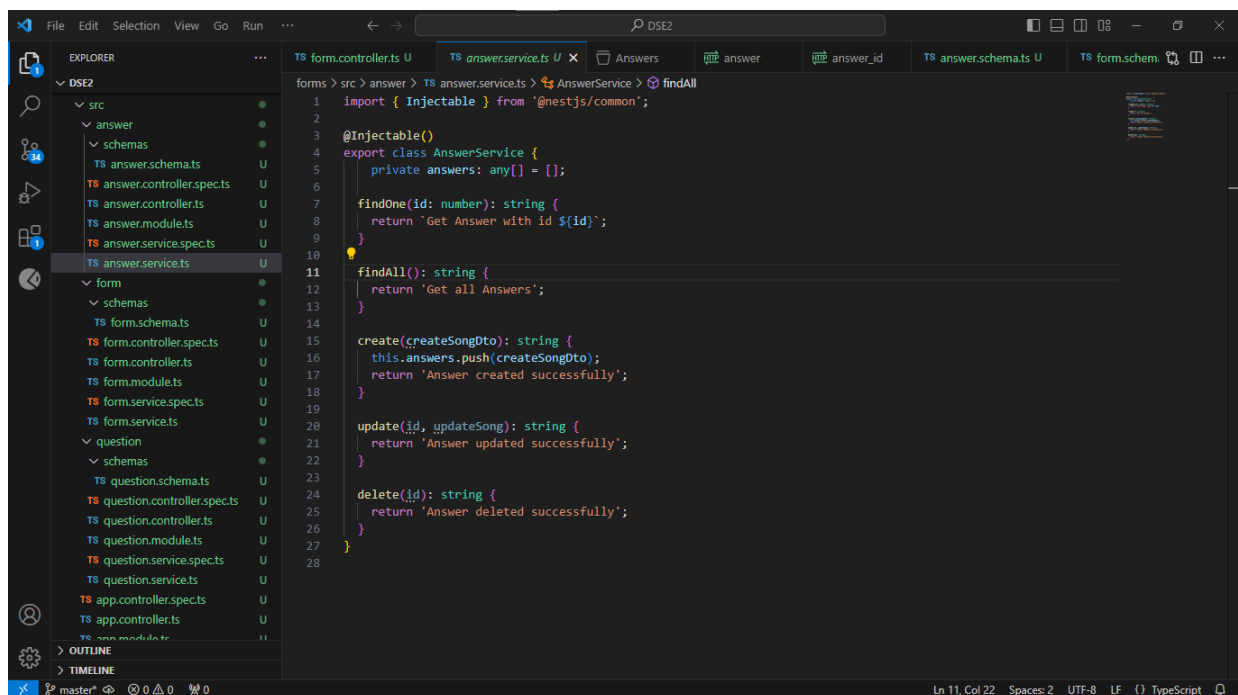


```

1 import {
2   Controller,
3   Get,
4   Post,
5   Body,
6   Param,
7   Put,
8   Delete,
9   Patch,
10 } from '@nestjs/common';
11 import { AnswerService } from './answer.service';
12
13 @Controller('answer')
14 export class AnswerController {
15   constructor(private readonly answerService: AnswerService) {}
16   // find one by id findOne(id)
17   @Get('/:id')
18   findOne(@Param('id') id: number): string {
19     return this.answerService.findOne(id);
20   }
21
22   @Get()
23   findAll(): string {
24     return this.answerService.findAll();
25   }
26
27   @Post()
28   create(@Body() createAnswerDto): string {
29     return this.answerService.create(createAnswerDto);
30   }
31
32   @Patch('/:id')
33   update(@Param('id') id: number, @Body() updateAnswer): string {
34     return this.answerService.update(id, updateAnswer);
35   }
36 }

```

servir de answer



```

1 import { Injectable } from '@nestjs/common';
2
3 @Injectable()
4 export class AnswerService {
5   private answers: any[] = [];
6
7   findOne(id: number): string {
8     return `Get Answer with id ${id}`;
9   }
10
11   findAll(): string {
12     return 'Get all Answers';
13   }
14
15   create(createSongDto): string {
16     this.answers.push(createSongDto);
17     return 'Answer created successfully';
18   }
19
20   update(id, updateSong): string {
21     return 'Answer updated successfully';
22   }
23
24   delete(id): string {
25     return 'Answer deleted successfully';
26   }
27 }

```

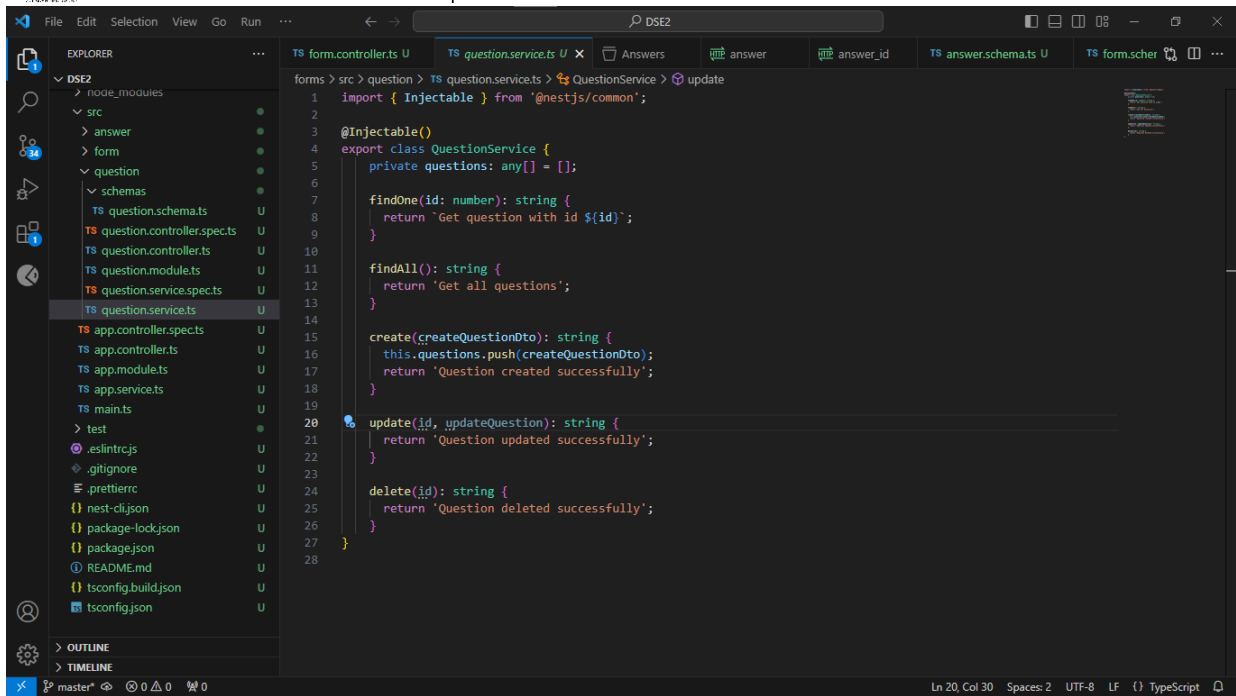
módulo de question


```
1 import { Module } from '@nestjs/common';
2 import { QuestionController } from './question.controller';
3 import { QuestionService } from './question.service';
4
5 @Module({
6   controllers: [QuestionController],
7   providers: [QuestionService]
8 })
9 export class QuestionModule {}
10
```

controller de question

```
1 import {
2   Controller,
3   Get,
4   Post,
5   Body,
6   Param,
7   Put,
8   Delete,
9   Patch,
10 } from '@nestjs/common';
11 import { QuestionService } from './question.service';
12
13 @Controller('question')
14 export class QuestionController {
15   constructor(private readonly questionService: QuestionService) {}
16
17   // find one by id findOne(id)
18   @Get('/:id')
19   findOne(@Param('id') id: number): string {
20     return this.questionService.findOne(id);
21   }
22
23   @Get()
24   findAll(): string {
25     return this.questionService.findAll();
26   }
27
28   @Post()
29   create(@Body() createQuestionDto): string {
30     return this.questionService.create(createQuestionDto);
31   }
32
33   @Patch('/:id')
34   update(@Param('id') id: number, @Body() updateQuestion): string {
35     return this.questionService.update(id, updateQuestion);
36   }
37 }
```

service de question



```
forms > src > question > TS question.service.ts U QuestionService > update
1  import { Injectable } from '@nestjs/common';
2
3  @Injectable()
4  export class QuestionService {
5      private questions: any[] = [];
6
7      findOne(id: number): string {
8          return `Get question with id ${id}`;
9      }
10
11     findAll(): string {
12         return 'Get all questions';
13     }
14
15     create(createQuestionDto): string {
16         this.questions.push(createQuestionDto);
17         return 'Question created successfully';
18     }
19
20     update(id, updateQuestion): string {
21         return 'Question updated successfully';
22     }
23
24     delete(id): string {
25         return 'Question deleted successfully';
26     }
27 }
28
```

Manejo de Transacciones

12. Iniciar la Transacción.
13. Operaciones en la Transacción.
14. Confirmar o Anular la Transacción.
15. Cerrar la Sesión y el Cliente.



16. Pruebas del Backend

Diseño de Casos de Prueba

Ejecución de Pruebas Unitarias y de Integración

Manejo de Errores y Excepciones



Etapa 3: Consumo de Datos y Desarrollo Frontend

17. Introducción

Propósito de la Etapa

Alcance de la Etapa

Definiciones y Acrónimos

18. Creación de la Interfaz de Usuario (UI)

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

Consideraciones de Usabilidad

Maquetación Responsiva



19. Programación Frontend con JavaScript (JS)

Desarrollo de la Lógica del Frontend

Manejo de Eventos y Comportamientos Dinámicos

Uso de Bibliotecas y Frameworks (si aplicable)

20. Consumo de Datos desde el Backend

Configuración de Conexiones al Backend

Obtención y Presentación de Datos

Actualización en Tiempo Real (si aplicable)

21. Interacción Usuario-Interfaz



Manejo de Formularios y Validación de Datos

Implementación de Funcionalidades Interactivas

Mejoras en la Experiencia del Usuario

22. Pruebas y Depuración del Frontend

Diseño de Casos de Prueba de Frontend

Pruebas de Usabilidad

Depuración de Errores y Optimización del Código

23. 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

24. Integración con el Backend

Verificación de la Comunicación Efectiva con el Backend

Pruebas de Integración Frontend-Backend

ANEXOS

Diagramas UML

- **Diagrama de Casos de Uso (Use Case Diagram):** Este diagrama muestra las interacciones entre los actores (usuarios) y el sistema. Puede ayudar a identificar las funcionalidades clave y los actores involucrados.
- **Diagrama de Secuencia (Sequence Diagram):** Estos diagramas muestran la interacción entre objetos y actores a lo largo del tiempo. Puedes utilizarlos para representar cómo los usuarios interactúan con el formulario en un flujo de trabajo específico.
- **Diagrama de Clases (Class Diagram):** Puedes utilizar este diagrama para modelar las clases y estructuras de datos subyacentes en el sistema, como usuarios, pizarras, comentarios, revisiones, etc.



- **Diagrama de Estados (State Diagram):** Este diagrama puede ser útil para modelar el comportamiento del formulario en diferentes estados, como "edición", "visualización", "comentario", etc.
- **Diagrama de Despliegue (Deployment Diagram):** Puedes utilizar este diagrama para representar cómo se despliega la aplicación en servidores y cómo interactúa con otros componentes del sistema, como el CMS.
- **Diagrama de Componentes (Component Diagram):** Este diagrama puede ayudar a representar la estructura de componentes del software, como la interfaz de usuario, la lógica de negocio, las bibliotecas y los servicios utilizados.
- **Diagrama de Actividad (Activity Diagram):** Puedes usar este diagrama para modelar flujos de trabajo o procesos específicos, como el flujo de trabajo de creación y edición de contenido en la pizarra.
- **Diagrama de Comunicación (Communication Diagram):** Similar a los diagramas de secuencia, estos diagramas muestran interacciones entre objetos y actores, pero pueden ser más simples y enfocados en la comunicación.
- **Diagrama de Paquetes (Package Diagram):** Este diagrama puede ayudar a organizar y visualizar los paquetes y módulos del software, lo que es útil para el diseño modular.
- **Diagrama de Objetos (Object Diagram):** Puedes utilizar este diagrama para representar instancias de clases y cómo interactúan en un escenario específico.