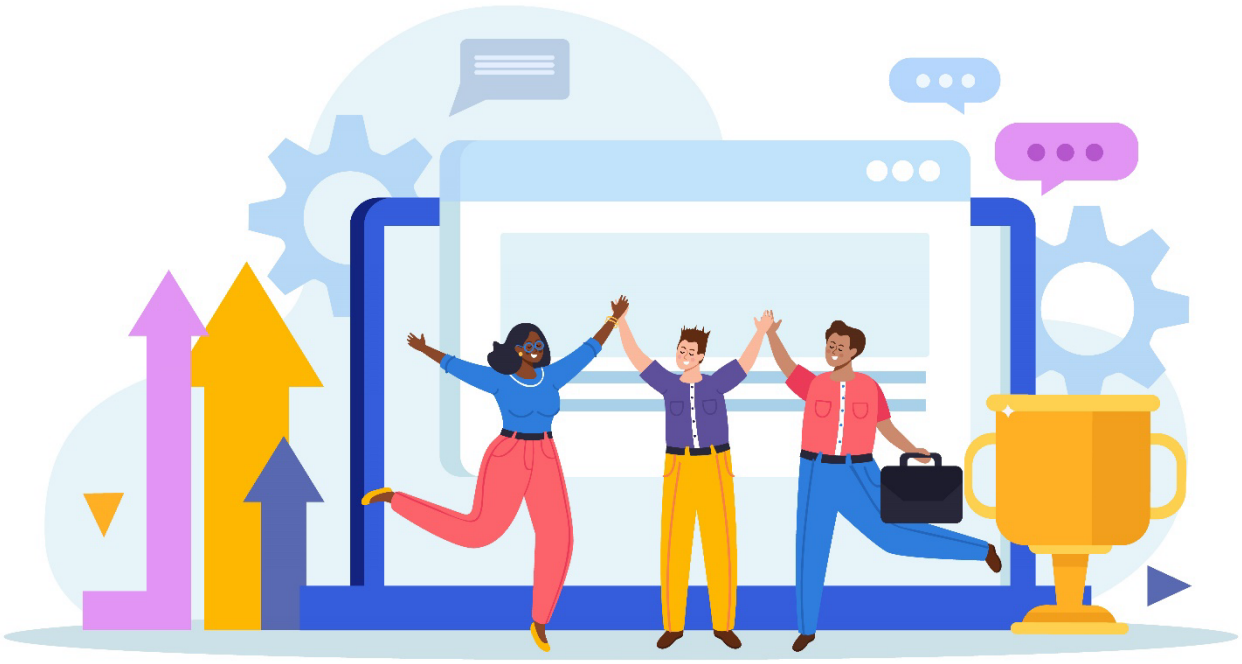
Documento de Propuesta de Diseño de Software I, II y II

Creación de Módulo de Trabajo Colaborativo

Andres Felipe Perez Martinez

Jorge Eliecer Gonzales Mejia

Edwin Isaac Martinez Escarpeta



https://www.freepik.es/

Descripción del software

El módulo de trabajo colaborativo es una componente esencial en sistemas y plataformas que buscan facilitar la interacción y cooperación entre usuarios. Este tipo de módulo está diseñado para permitir que individuos o grupos trabajen juntos de manera eficiente en la realización de tareas, proyectos o metas comunes. Ofrece herramientas y funcionalidades que posibilitan la creación, edición y revisión simultánea de contenido, así como la comunicación en tiempo real entre los participantes. Entre las características comunes se encuentran la compartición de documentos, la asignación de tareas, la colaboración en proyectos, el intercambio de ideas a través de chats y foros, y la capacidad de realizar un seguimiento del progreso conjunto. Este tipo de módulo es particularmente valioso en entornos educativos y profesionales, ya que fomenta la participación activa, el intercambio de conocimientos y la sinergia en la consecución de objetivos compartidos.

ETAPA 1 DISEÑO DE LA APLICACIÓN Y ANÁLISIS DE REQUISITOS **6**

1. 6

PROPÓSITO DEL DOCUMENTO 6
ALCANCE DEL PROYECTO MÓDULO DE PIZARRA COMPARTIDA 8
DEFINICIONES Y ACRÓNIMOS 8

2. 11

OBJETIVOS DEL SISTEMA 10
FUNCIONALIDAD GENERAL 10
USUARIOS DEL SISTEMA 11
RESTRICCIONES 12

3. 14

CASOS DE USO 13
DIAGRAMAS DE FLUJO DE CASOS DE USO 14
DESCRIPCIÓN DETALLADA DE CADA CASO DE USO 14
PRIORIDAD DE REQUERIMIENTOS 16

4. 33

REQUISITOS DE DESEMPEÑO 18
REQUISITOS DE SEGURIDAD 19
REQUISITOS DE USABILIDAD 20
REQUISITOS DE ESCALABILIDAD 20

5. 40

DIAGRAMA DE ENTIDAD-RELACIÓN 21
DIAGRAMA RELACIONAL 22
SCRIPT DE MODELO RELACIONAL 23
DESCRIPCIÓN DE ENTIDADES Y RELACIONES 24
REGLAS DE INTEGRIDAD REFERENCIAL 25
COLECCIONES (NOSLQ) 28

6. ¡Error! Marcador no definido.

DIAGRAMAS ADICIONALES 29
REFERENCIAS 29

ETAPA 2: PERSISTENCIA DE DATOS CON BACKEND **30**

7. 45

PROPÓSITO DE LA ETAPA 30
ALCANCE DE LA ETAPA 30
DEFINICIONES Y ACRÓNIMOS 30

8. 46

DESCRIPCIÓN DE LA ARQUITECTURA PROPUESTA 30

Componentes del Backend 30
Diagramas de Arquitectura 30
9. 48
Evaluación de Opciones (SQL o NoSQL) 31
Justificación de la Elección 31
Diseño de Esquema de Base de Datos 31
10. 50
Elección del Lenguaje de Programación 31
Creación de la Lógica de Negocio 31
Desarrollo de Endpoints y APIs 31
Autenticación y Autorización 31
11. 56
Configuración de la Conexión 32
Desarrollo de Operaciones CRUD 32
Manejo de Transacciones 32
12. 58
Diseño de Casos de Prueba 32
Ejecución de Pruebas Unitarias y de Integración 32
Manejo de Errores y Excepciones 32
Etapa 3: Consumo de Datos y Desarrollo Frontend **33**
13. 61
Propósito de la Etapa 33
Alcance de la Etapa 33
Definiciones y Acrónimos 33
14. 61
Diseño de la Interfaz de Usuario (UI) con HTML y CSS 33
Consideraciones de Usabilidad 33
Maquetación Responsiva 33
15. 62
Desarrollo de la Lógica del Frontend 34
Manejo de Eventos y Comportamientos Dinámicos 34
Uso de Bibliotecas y Frameworks (si aplicable) 34
16. 62
Configuración de Conexiones al Backend 34
Obtención y Presentación de Datos 34
Actualización en Tiempo Real (si aplicable) 34

17. 62

Manejo de Formularios y Validación de Datos 35
Implementación de Funcionalidades Interactivas 35
Mejoras en la Experiencia del Usuario 35

18. 63

Diseño de Casos de Prueba de Frontend 35
Pruebas de Usabilidad 35
Depuración de Errores y Optimización del Código 35

19. 63

Migración de la Lógica de Negocio desde el Backend (si necesario) 36
Validación de Datos y Reglas de Negocio en el Frontend 36

20. 64

Verificación de la Comunicación Efectiva con el Backend 36
Pruebas de Integración Frontend-Backend 36
ANEXOS 36

- Etapa 1 Diseño de la Aplicación y Análisis de Requisitos

1. Introducción

Propósito del Documento

El propósito de este texto es registrar el proceso de desarrollo, evaluación y creación de software con fines educativos, comerciales, objetos virtuales de aprendizaje, componentes o módulos de aplicaciones. Se estructura en tres fases con el fin de simplificar su comprensión y aplicabilidad en el contexto de la asignatura de diseño de software.

- Etapa 1 Diseño de la Aplicación y Análisis de Requisitos

Esta fase tiene como objetivo reunir todas las habilidades adquiridas en las diversas áreas de formación del plan de estudios de Licenciatura en Informática y Medios Audiovisuales. Su propósito es evaluar estas competencias aplicándolas al diseño y análisis de un producto educativo. Este producto se basa en teorías de aprendizaje, integra estrategias de enseñanza con tecnología (TIC) y busca introducir innovaciones educativas a través de productos interactivos. Estas herramientas educativas deben aprovechar las habilidades adquiridas en tecnología, informática, multimedia y programación para crear productos de software interactivos que permitan a los usuarios disfrutar de su aprendizaje a su propio ritmo. Todo esto se lleva a cabo en un marco metodológico estructurado (utilizando metodologías de desarrollo de software como MODESEC, SEMLI, etc.) que combina conocimientos de gestión y enriquece con elementos de ingeniería de software.

- Etapa 2: Persistencia de Datos con Backend – Servidor

En la segunda etapa, se siguen los pasos marcados en la primera etapa para continuar enriqueciendo los aspectos relacionados con el diseño y desarrollo de software. Enfocándose en la creación de APIs, servidores, o microservicios que tienen como finalidad respaldar las aplicaciones cliente del software educativo. Dentro de esta fase, el curso aborda conceptos fundamentales de sistemas de bases de datos, incluyendo su diseño lógico, la organización de los sistemas de gestión de bases de datos, así como los lenguajes de definición y manipulación de datos SQL y NoSQL. Esto se realiza con el propósito de capacitar a los estudiantes en la capacidad de analizar, diseñar y desarrollar aplicaciones para gestionar y almacenar grandes volúmenes de datos, utilizando técnicas apropiadas como el diseño lógico y físico de bases de datos, la administración de sistemas de gestión de bases de datos, el álgebra relacional, el manejo del lenguaje SQL como herramienta de consulta y la tecnología cliente/servidor. Además, se explican los elementos necesarios para acceder a estas bases de datos, incluyendo la creación de servidores API mediante tecnologías avanzadas como node.js, express, Nest.js,

Spring, entre otras. Finalmente, se aborda la implementación de la API utilizando servicios de alojamiento en la nube, de preferencia gratuitos. El curso también ofrece la posibilidad de implementar servidores o APIs 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 llevará a cabo a través de proyectos de trabajo colaborativo que serán evaluados de diversas maneras, poniendo un mayor énfasis en el proceso que en los resultados finales.

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

En la tercera etapa, el estudiante adquiere la capacidad de tomar decisiones informadas al elegir las herramientas y técnicas más adecuadas para el consumo de datos. Esto se hace con el objetivo de crear el mejor producto, ya sea a nivel de software o hardware, que cumpla con los requisitos tanto funcionales como no funcionales del problema que se busca resolver. En este punto, el estudiante puede acceder a los datos a través de diversos tipos de clientes, que pueden ser aplicaciones móviles, aplicaciones de escritorio, sitios web, dispositivos de IoT (Internet de las cosas) o incluso artefactos tecnológicos.

El diseño gráfico se convierte en un requisito fundamental en la capa de presentación, lo que implica que los cursos de diseño gráfico previamente cursados son necesarios. Estos elementos mencionados nos permiten tomar decisiones sobre el enfoque y la tecnología a utilizar en el desarrollo de nuestras aplicaciones, teniendo en cuenta la posibilidad de desarrollar aplicaciones de tipo cliente.

Alcance del Proyecto Módulo de Trabajo Colaborativo

Este módulo tiene como objetivo establecer una estrategia digital para abordar los desafíos que surgen al intentar comunicar información digital a los usuarios de la plataforma CREAVI. Estos desafíos incluyen la necesidad de realizar presentaciones presenciales para explicar contenidos, la falta de un proyector de video, horarios no convencionales, entre otros. Para superar estas dificultades, se han desarrollado y se describen a continuación una serie de características para la versión actual del módulo, así como otras propuestas para futuras actualizaciones:

Características actuales:

1. Visualización de una pestaña compartida de un usuario.

2. Presentación de secuencias de imágenes desde una pestaña compartida.

3. Resaltado de elementos gráficos en una pestaña compartida.

4. Escritura a mano libre sobre una pestaña compartida.

Funcionalidades futuras:

1. Colaboración en tiempo real en una pestaña compartida.

2. Grabación de sesiones interactivas de trabajo para su posterior revisión.

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: Asincrónico 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

"Diseñar, desarrollar y evaluar software de alta calidad con enfoque educativo y comercial, incluyendo la creación de objetos virtuales de aprendizaje y módulos de aplicaciones, promoviendo el trabajo colaborativo en un entorno de diseño de software. El proyecto busca proporcionar soluciones tecnológicas efectivas que aborden necesidades específicas en el ámbito educativo y empresarial, mejorando la eficiencia, la innovación y la colaboración entre los equipos de desarrollo."

Funcionalidad General

1. Herramientas de Colaboración en Tiempo Real:

- Implementar herramientas de colaboración en línea que permitan a los miembros del equipo comunicarse y colaborar en tiempo real. Esto podría incluir chat en vivo, videoconferencias y herramientas de gestión de proyectos.

2. Control de Versiones:

- Utilizar sistemas de control de versiones, como Git, para facilitar la colaboración en la escritura de código y el seguimiento de cambios.

3. Compartir Documentación:

- Tener un sistema para compartir documentación, manuales y recursos de manera centralizada para que todos los miembros del equipo tengan acceso a la información relevante.

4. Seguimiento de Tareas y Progreso:

- Utilizar herramientas de gestión de proyectos que permitan asignar tareas, hacer un seguimiento del progreso y establecer hitos para el trabajo colaborativo.

5. Revisión de Código:

- Establecer un proceso de revisión de código entre los desarrolladores para garantizar la calidad y la coherencia en el código fuente.

6. Participación Activa:

- Fomentar la participación activa y las sesiones de lluvia de ideas entre los miembros del equipo para promover la creatividad y la colaboración en la toma de decisiones.

7. Retroalimentación Continua:

- Establecer un flujo de retroalimentación constante entre los miembros del equipo para mejorar continuamente el software y los procesos de desarrollo.

- **Personalización y Temas**: Permite a los usuarios personalizar la apariencia de la pizarra y seleccionar temas que se adapten a sus necesidades.
- **Acceso Seguro**: Proporciona medidas de seguridad para garantizar que solo los usuarios autorizados puedan acceder y editar la pizarra.
- **Notificaciones y Actualizaciones en Tiempo Real**: Los usuarios reciben notificaciones sobre cambios en la pizarra y pueden ver actualizaciones en tiempo real mientras otros editan.
- **Acceso Móvil**: Ofrece una experiencia de usuario optimizada en dispositivos móviles, permitiendo el acceso y la colaboración desde smartphones y tabletas.
- **Búsqueda y Filtros**: Facilita la búsqueda de contenido en la pizarra y la aplicación de filtros para organizar y encontrar información específica.
- **Gestión de Usuarios y Permisos**: Permite a los administradores gestionar usuarios y definir permisos de acceso y edición.
- **Informes y Analíticas**: Proporciona información sobre el uso de la pizarra, como quién la ha editado, cuándo se realizaron cambios y estadísticas sobre el contenido(XAPI).

Usuarios del Sistema

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

Funcionalidad	Administradores	Docente	Alumno	Invitado
Hacer equipos de trabajo		✓		
Crear Tarea	✓	✓		
Enviar Tarea			✓	
Recibir Tarea	✓	✓	✓	
Enviar Notificación	✓	✓		
Recibir Notificación	✓	✓	✓	✓
Crear Notas	✓	✓		
Recibir Notas	✓	✓	✓	✓

Restricciones

Solo el rol docente es quien pude crear los grupos de trabajos y también Limitaciones en la participación activa, como requisitos mínimos para contribuir o cumplir con ciertos criterios para unirse a proyectos específicos.

3. Requisitos Funcionales

Comunicación en Tiempo Real:

- Herramientas de chat o mensajería instantánea para facilitar la comunicación entre los usuarios mientras trabajan en el mismo proyecto.

Creación y Edición Colaborativa de Contenido:

- Capacidades para crear y editar documentos, presentaciones, hojas de cálculo, etc., de forma colaborativa en tiempo real.

Requisito de Colaboración:

- El software debe permitir la colaboración en tiempo real entre los miembros del equipo, incluyendo chat, videoconferencias y compartición de documentos.
- Debe contar con un sistema de seguimiento de tareas y proyectos para asignar y supervisar las actividades de los miembros del equipo.

Foros y Comunidades:

- Espacios donde los usuarios puedan participar en discusiones, hacer preguntas y compartir recursos relacionados con el contenido educativo.

Seguridad y Privacidad:

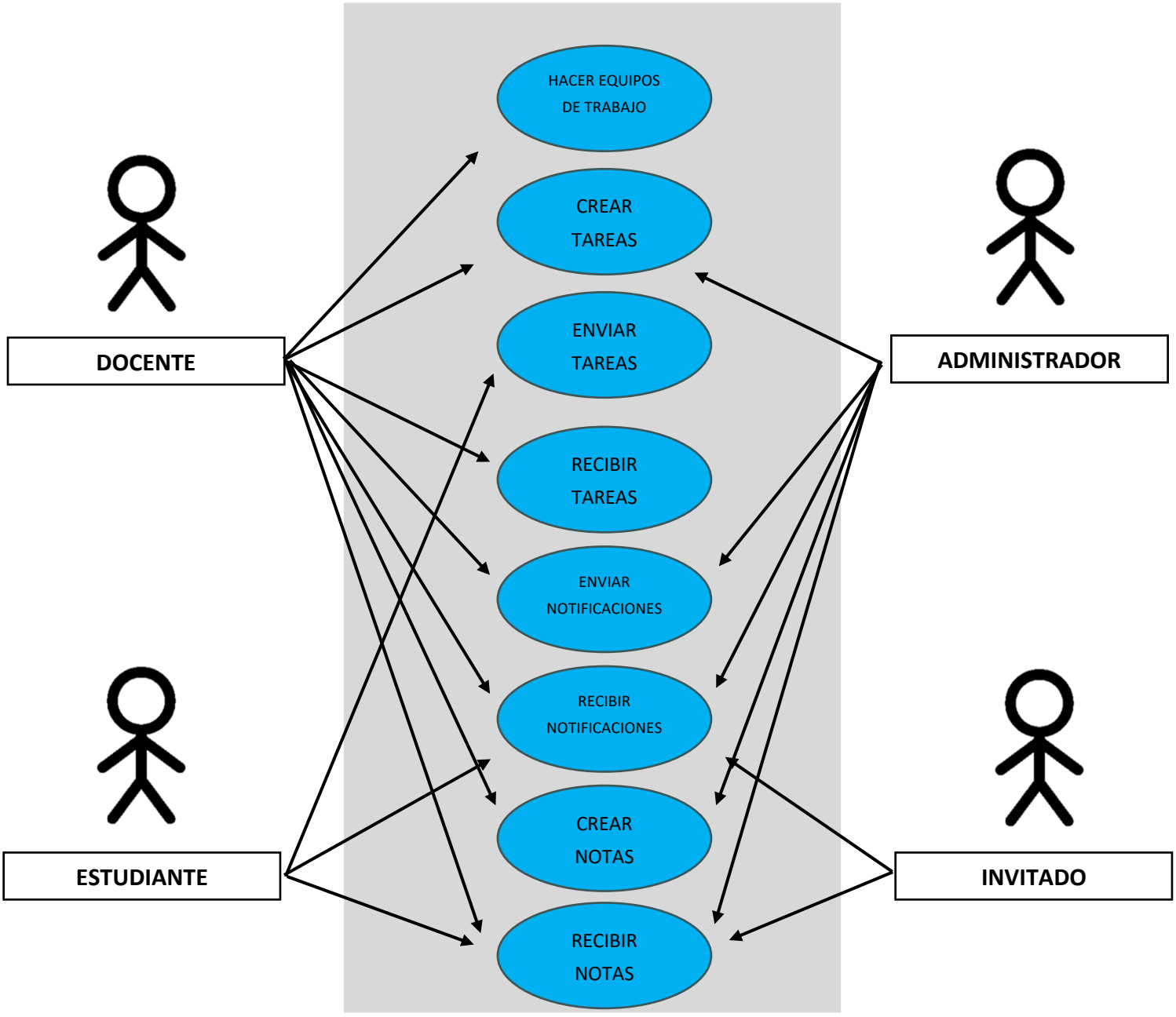
- Garantizar la seguridad de los datos y la privacidad de los usuarios, especialmente si se trata de una plataforma educativa que involucra a estudiantes.

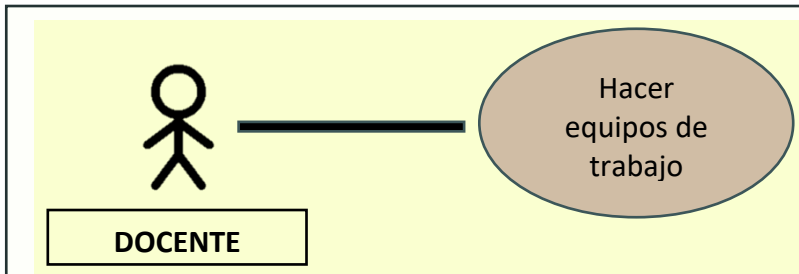
Requisito de Evaluación Continua:

- El software debe permitir la recopilación y análisis de datos para evaluar la eficacia y la usabilidad de las soluciones desarrolladas.
- Debe permitir la recopilación de retroalimentación de usuarios y administradores.

Casos de Uso

Diagrama de caso de uso





Prioridad: requerido
Versión: 1
Complejidad: ALTA

Hacer equipos de trabajo

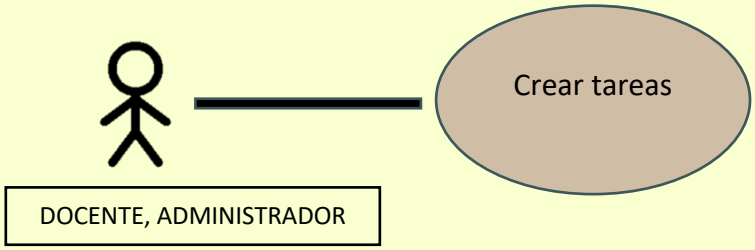
Flujo: Hacer equipos de trabajo
Prueba: Seleccionar crear grupo de trabajo, crear grupo de trabajo

Descripción detallada de cada caso de uso

CASO No. 1 Hacer equipos de trabajo

ID:	CU-1	
Nombre	Hacer equipos de trabajo	
Actores	Docente investigador, Docente invitado	
Objetivo	Este caso debe permitir hacer equipos de trabajo	
Urgencia	5	
Esfuerzo	5	
Pre-condiciones	- Debe haberse autenticado de forma correcta en el sistema. - Los grupos de trabajo deben estar seleccionados	
Flujo Normal	Docente	Docente
	Selecciona hacer grupos de trabajo	
		Despliega una pantalla de creación de grupos
	Gestiona la pantalla de grupo	
	Guarda el grupo	
		Guarda en base de datos
		Notifica a los estudiantes
Flujo alternativo 1		
Flujo alternativo 2		
Post-condiciones		

Exepciones		



Prioridad: requerido
Versión: 1
Complejidad: media

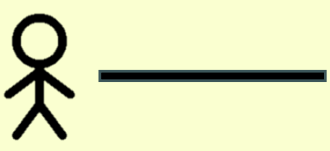
Crear tareas

Flujo: Seleccionamos nueva área y se la asignamos a un grupo de trabajo

CASO No.2 Crear tareas

ID:	CU-2	
Nombre	Crear tareas	
Actores	Docente, Administrador	
Objetivo	Este caso debe permitir Crear tareas	
Urgencia	5	
Esfuerzo	4	
Pre-condiciones	- Debe haberse autenticado de forma correcta en el sistema. - Tener asignados los grupos de trabajo para crear la tarea por selección	
Flujo Normal	Docente	Sistema
	Seleccionar nueva tarea	
		Despliega lista de grupos de trabajo
	Seleccionar Grupo de trabajo asignado	
		Muestra integrantes del grupo

		Espacio debajo para crear tarea
	Genera la tarea por grupos	
		Guarda la tarea
Flujo alternativo 1		
Flujo alternativo 2		
Post-condiciones		
Excepciones		



Enviar tareas

ESTUDIANTES

Prioridad: requerido

Versión: 1

Enviar tarea

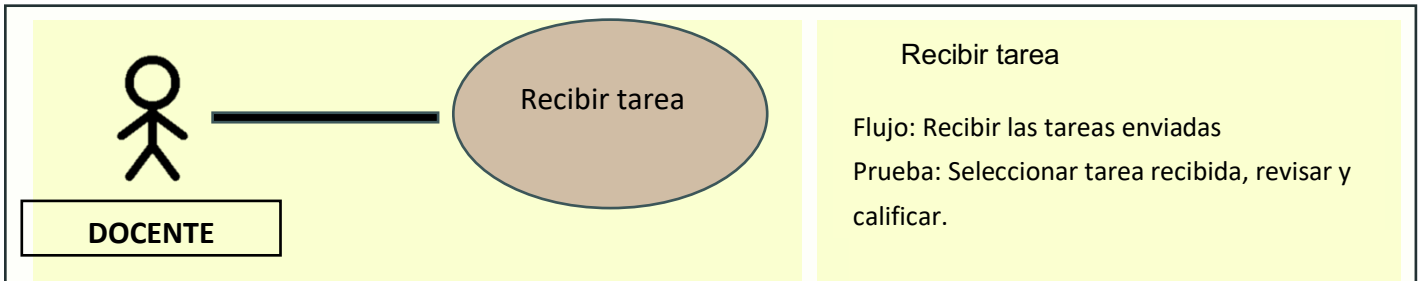
Flujo: Enviar tarea asignada

Prueba: Ejecutar tarea y enviar

CASO No. 3 Enviar Tarea

ID:	CU- 3	
Nombre	Enviar Tarea	
Actores	Grupos de trabajo (Estudiantes)	
Objetivo	Este caso debe permitir enviar tarea	
Urgencia	4	
Esfuerzo	4	
Pre-condiciones	- Debe haberse autenticado de forma correcta en el sistema. - Los grupos de trabajo deben estar seleccionados - Tarea terminada y lista para enviar	
Flujo Normal	Estudiantes	Sistema
	Selecciona tarea que se les asigno al grupo	
		Muestras las tareas que están pendientes para el grupo
	El estudiante ejecuta la tarea	
		El sistema guarda la tarea

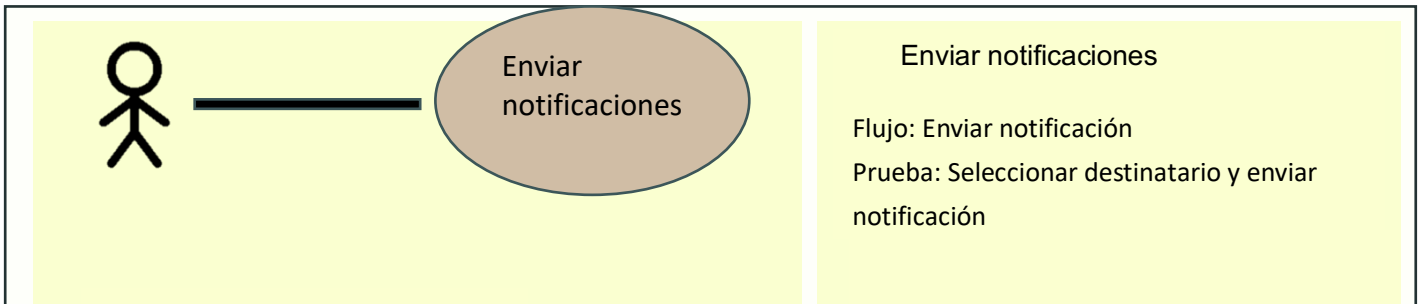
Flujo alternativo 1		
Flujo alternativo 2		
Post-condiciones		
Excepciones		



CASO No. 4 Recibir tarea

ID:	CU- 4	
Nombre	Recibir tarea	
Actores	Docente	
Objetivo	Este caso debe permitir recibir las tareas que asignaron a cada grupo de trabajo.	
Urgencia	4	
Esfuerzo	4	
Pre-condiciones	- Debe haberse autenticado de forma correcta en el sistema. - Los grupos de trabajo deben estar seleccionados - Tarea enviada por estudiantes	
Flujo Normal	Docente	Sistema
	Selecciona Tarea enviadas	
		Muestras las tareas que fueron enviadas
	Seleccionar la tarea a revisar	
	Valora la tarea	
		Guarda información

Flujo alternativo 1		
Flujo alternativo 2		
Post-condiciones		
Excepciones		

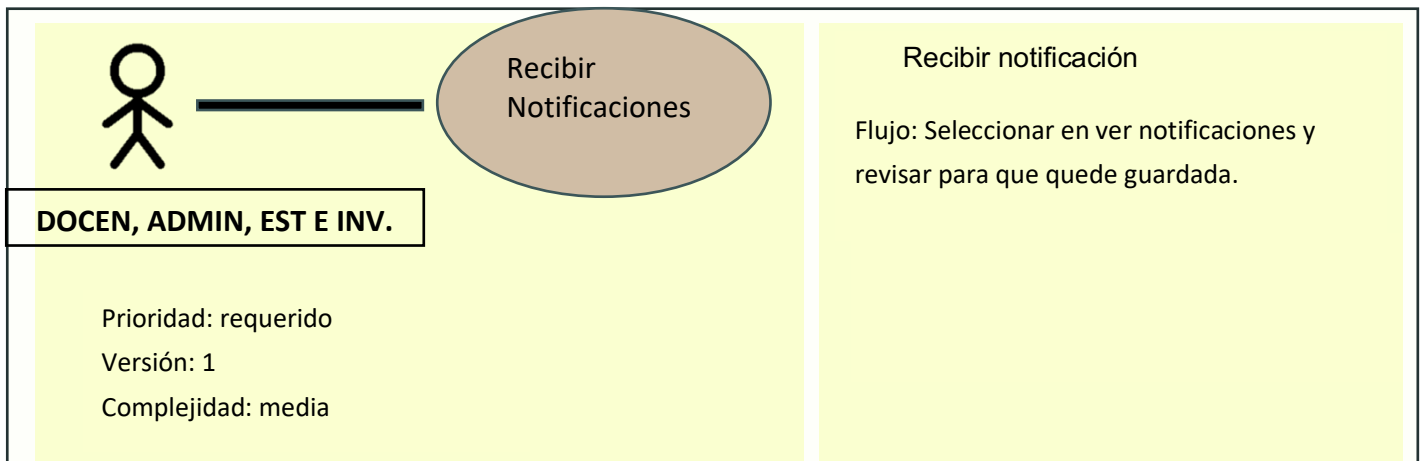


DOCENTE Y ADMINITRADOR

CASO No. 5 Enviar Notificaciones

ID:	CU-5	
Nombre	Enviar Notificaciones	
Actores	Docente y Administrado	
Objetivo	Este caso debe permitir hacer equipos de trabajo	
Urgencia	5	
Esfuerzo	4	
Pre-condiciones	- Debe haberse autenticado de forma correcta en el sistema.	
Flujo Normal	Docente	Sistema
		Interfaz de la pantalla de inicio donde muestra una campanita de notificaciones
	Selecciona campanita de notificaciones	
		Muestra 2 opciones de enviar notificaciones o notificaciones recibidas
	Selecciona Enviar notificaciones	
		Interfaz donde permite redactar la notificación y seleccionar destinatarios

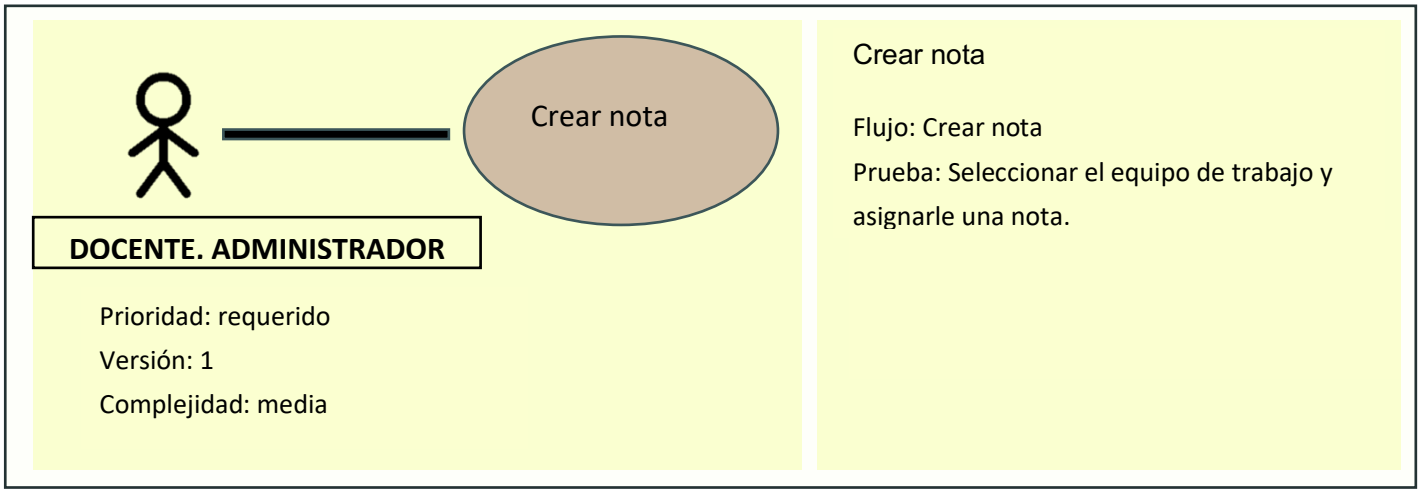
	Redacta la notificación y selecciona los destinatarios	
		Envía la notificación
		Guarda la notificación
Flujo alternativo 1		
Flujo alternativo 2		
Post-condiciones		
Excepciones		



CASO No. 6 Recibir Notificaciones

ID:	CU-6	
Nombre	Recibir Notificaciones	
Actores	Docente, estudiante administrador e invitado	
Objetivo	Este caso debe permitir recibir notificaciones	
Urgencia	4	
Esfuerzo	4	
Pre-condiciones	- Debe haberse autenticado de forma correcta en el sistema. - Los grupos de trabajo deben estar seleccionados	
Flujo Normal	Docente, estudiante, administrador e invitado	Sistema
		Interfaz de la pantalla de inicio donde muestra una campanita de notificaciones con numero de notificaciones si abrir
	Selecciona campanita de notificaciones	
		Muestra la opción de notificaciones recibidas
	Selecciona notificaciones recibidas	
		Interfax donde permite ver la notificación y quien envió la notificación
	Ve la notificación	

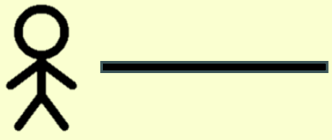
		Notifica que fue guardada
Flujo alternativo 1		
Flujo alternativo 2		
Post-condiciones		
Excepciones		



CASO No. 7 Crear Nota

ID:	CU-7	
Nombre	Crear Nota	
Actores	Docentes y administrador	
Objetivo	Este caso debe permitir crear notas a estudiantes	
Urgencia	5	
Esfuerzo	4	
Pre-condiciones	- Debe haberse autenticado de forma correcta en el sistema. - Los grupos de trabajo deben estar seleccionados	
Flujo Normal	Docente	Sistema
	Selecciona ver grupos de trabajo	
		Despliega lista de grupos
	Seleccionar la opción de asignar nota por grupo	
		Muestra grupo de estudiantes
	Asignar nota por grupo	
		Guardar información
Flujo alternativo 1		
Flujo alternativo 2		

Post-condiciones		
Excepciones		



Recibir nota

DOCENTE, ESTUDIANTE

Prioridad: requerido

Versión: 1

Complejidad: media

Recibir nota

Flujo: Recibir nota

Prueba: Seleccionar notificaciones y ver la notificación recibida

CASO No. 8 Recibir Nota

ID:	CU-8	
Nombre	Recibir Nota	
Actores	Docentes, estudiantes	
Objetivo	Este caso debe permitir recibir notas	
Urgencia	4	
Esfuerzo	4	
Pre-condiciones	- Debe haberse autenticado de forma correcta en el sistema. - Los grupos de trabajo deben estar seleccionados	
Flujo Normal	Docente	Sistema
		Interfaz de la pantalla de inicio donde muestra una campanita de notificaciones con numero para ver cuantas notas hay disponibles
	Selecciona campanita de notificaciones	
		Muestra la opción de notificaciones recibidas
	Selecciona notificaciones recibidas	
		Interfaz donde permite ver la nota recibida y quien la envió
	Ve la nota y selecciona guardar	
Flujo alternativo 1		

Flujo alternativo 2		
Post-condiciones		
Excepciones		

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

https://asana.com/es/resources/priority-matrix

4. Requisitos No Funcionales

Rendimiento:

- Tiempo de respuesta: El sistema debe proporcionar tiempos de respuesta rápidos para garantizar una experiencia de usuario ágil.
- Capacidad de carga: El software debe ser capaz de manejar una cantidad específica de usuarios concurrentes sin degradación significativa del rendimiento.

Disponibilidad:

- El sistema debe estar disponible las 24 horas del día, los 7 días de la semana, o de acuerdo con un acuerdo de nivel de servicio (SLA) establecido.

Seguridad:

- Autenticación y autorización: Debe haber un sólido sistema de autenticación de usuarios y control de acceso para garantizar la privacidad y seguridad de los datos.
- Protección contra amenazas: El sistema debe estar protegido contra ataques cibernéticos, como inyección de SQL, ataques de denegación de servicio (DDoS) y otros.

Escalabilidad:

- El sistema debe ser escalable para adaptarse al crecimiento de usuarios y la expansión de la base de datos.

Usabilidad:

- La interfaz de usuario debe ser intuitiva y cumplir con los principios de diseño centrado en el usuario para garantizar una experiencia de usuario positiva.

Mantenimiento:

- Debe haber facilidad en el mantenimiento del sistema, incluyendo actualizaciones de software, corrección de errores y administración de bases de datos.

Compatibilidad:

- El software debe ser compatible con múltiples navegadores web, sistemas operativos y dispositivos, para llegar a un público amplio.

Rendimiento de la Base de Datos:

- Las consultas a la base de datos deben ser eficientes y optimizadas para mantener un rendimiento rápido.

Documentación:

- Debe existir una documentación completa y actualizada para usuarios, administradores y desarrolladores.

Costo:

- Eficiencia en el Uso de Recursos: Optimizar el uso de recursos, como servidores, para minimizar costos operativos.

Requisitos de Desempeño

Tiempo de Carga:

- El software debe cargar la interfaz de usuario principal en menos de 2 segundos en condiciones normales de red y servidor.

Respuesta a Eventos:

- Las acciones del usuario, como hacer clic en botones o enviar formularios, deben generar una respuesta del sistema en menos de 1 segundo.

Procesamiento de Datos:

- El sistema debe ser capaz de procesar y almacenar grandes volúmenes de datos de usuario de manera eficiente, sin demoras significativas.

Rendimiento de Consultas de Bases de Datos:

- Las consultas a la base de datos deben ejecutarse en menos de 200 milisegundos para garantizar una experiencia de usuario ágil.

Escalabilidad:

- El sistema debe ser capaz de escalar para manejar un aumento del 50% en la carga de trabajo sin degradación del rendimiento.

Requisitos de Seguridad

Tiempo de Carga:

- El software debe cargar la interfaz de usuario principal en menos de 2 segundos en condiciones normales de red y servidor.

Respuesta a Eventos:

- Las acciones del usuario, como hacer clic en botones o enviar formularios, deben generar una respuesta del sistema en menos de 1 segundo.

Procesamiento de Datos:

- El sistema debe ser capaz de procesar y almacenar grandes volúmenes de datos de usuario de manera eficiente, sin demoras significativas.

Rendimiento de Consultas de Bases de Datos:

- Las consultas a la base de datos deben ejecutarse en menos de 200 milisegundos para garantizar una experiencia de usuario ágil.

Escalabilidad:

- El sistema debe ser capaz de escalar para manejar un aumento del 50% en la carga de trabajo sin degradación del rendimiento.

Tiempo de Respuesta en Colaboración en Tiempo Real:

- En el caso de aplicaciones de colaboración en tiempo real, como videoconferencias, el tiempo de respuesta para la transmisión y recepción de datos debe ser inferior a 200 milisegundos para garantizar una comunicación fluida.

Uso de Recursos:

- El software debe utilizar los recursos del sistema de manera eficiente, evitando consumir en exceso CPU, memoria y ancho de banda.

Requisitos de Usabilidad

Diseño Intuitivo:

- El software debe contar con una interfaz de usuario intuitiva que permita a los usuarios navegar y realizar tareas de manera sencilla y sin esfuerzo.

Consistencia de la Interfaz:

- Todas las partes del software deben seguir un diseño coherente, incluyendo la disposición de elementos, el uso de colores y la ubicación de botones y menús.

Retroalimentación Visual:

- El software debe proporcionar retroalimentación visual inmediata cuando los usuarios interactúan con él, como cambios de color en botones al pasar el cursor sobre ellos.

Facilidad de Aprendizaje:

- El software debe ser fácil de aprender para los nuevos usuarios, con tutoriales, guías de inicio y asistencia contextual.

Eficiencia de Uso:

- El software debe permitir a los usuarios realizar tareas de manera eficiente, minimizando los pasos necesarios para completar acciones comunes.

Requisitos de Escalabilidad

Escalabilidad Horizontal:

- El software debe ser capaz de escalar horizontalmente, es decir, agregar más servidores o recursos de hardware para manejar un aumento en la carga de trabajo sin afectar el rendimiento.

Escalabilidad Vertical:

- El software debe ser capaz de escalar verticalmente, lo que implica la capacidad de aumentar los recursos de hardware en un servidor individual para manejar cargas de trabajo más pesadas.

Balanceo de Carga Automático:

- El sistema debe contar con un mecanismo de balanceo de carga automático que distribuya las solicitudes entre los servidores disponibles de manera equitativa.

Administración de Recursos Eficiente:

- Debe utilizar eficientemente los recursos disponibles, como CPU, memoria y almacenamiento, para optimizar el rendimiento y minimizar los costos.

Elasticidad:

- El software debe ser elástico y capaz de adaptarse dinámicamente a cambios en la carga de trabajo, escalando hacia arriba o hacia abajo según sea necesario.

Alta Disponibilidad:

- Debe garantizar una alta disponibilidad, con mecanismos de redundancia y conmutación por error para minimizar el tiempo de inactividad.

Gestión de Base de Datos Escalable:

- Si el software utiliza bases de datos, debe ser capaz de escalar la capacidad de almacenamiento y la capacidad de procesamiento de la base de datos de manera eficiente.

Monitorización y Ajuste Automático:

- El software debe contar con herramientas de monitorización y ajuste automático que permitan detectar problemas de rendimiento y aplicar correcciones automáticamente.

Despliegue en la Nube:

- Debe ser capaz de desplegarse en entornos de nube pública o privada para aprovechar los recursos de la nube y escalar según sea necesario.

Plan de Escalabilidad:

- Debe contar con un plan claro y documentado para la escalabilidad, que incluya umbrales de carga, métricas de rendimiento y procedimientos para escalado.

5. Modelado E/R

Diagrama de Entidad-Relación

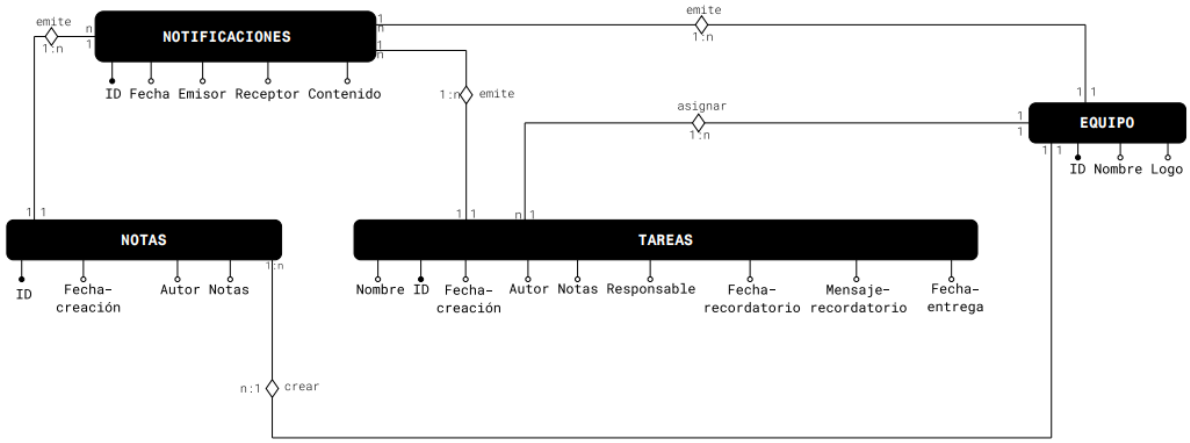
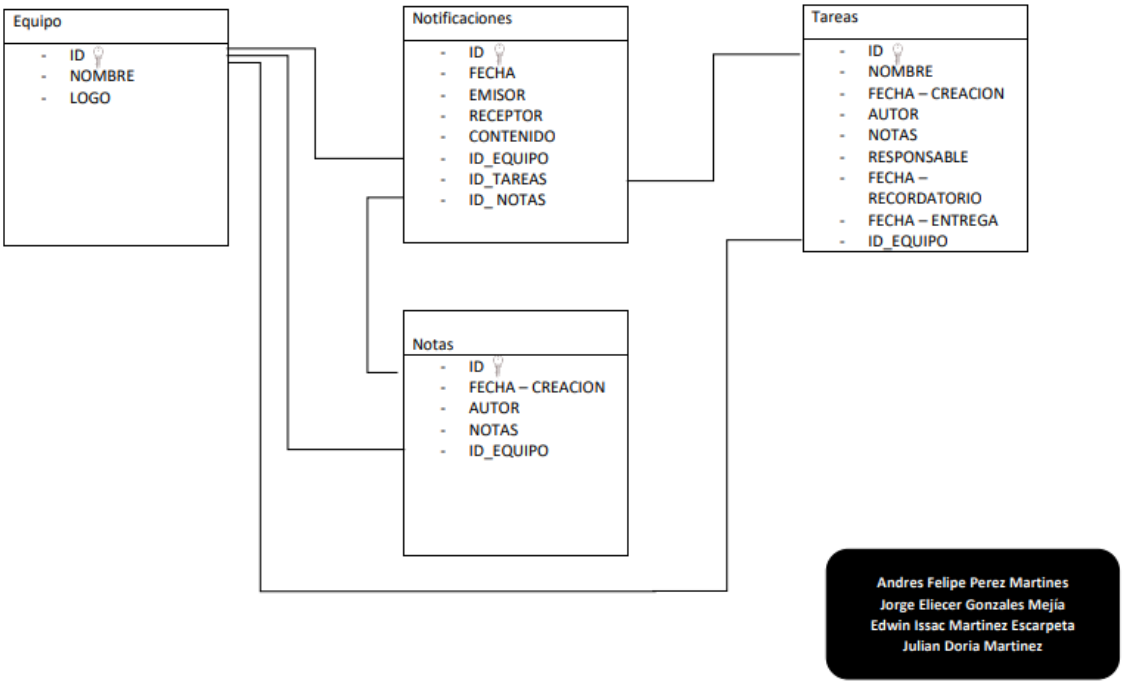


Diagrama Relacional



Andres Felipe Perez Martines
Jorge Eliecer Gonzales Mejía
Edwin Issac Martinez Escarpeta
Julian Doria Martinez

Descripción de Entidades y Relaciones

Entidades:

Notifications (Notificaciones):

Envía notificaciones en tiempo real para recordar o informar sobre alguna actividad pendiente por terminar o mensaje escrito por los compañeros de trabajo

Atributos: Id (Identificador único), fecha, emisor, receptor, contenido.

Relaciones: El equipo pude recibir notificaciones y recorderis de las tareas así mismo con las notas.

Grades (Notas):

Prosador de texto utilizado para tomar notas, escribir, crear lista y realizar anotaciones.

Atributos: Id (Identificador único),fecha-creacion, autor y notas.

Relaciones: Cada nota se relaciona con las entidades notificaciones y equipo

Tasks (Tareas):

Nos ayudaran a ejercitar, integrar conocimiento y repasar lo que quedo pendiente.

Atributos: Id (Identificador único), nombre, fecha-creación, autor, notas, responsable, fecha-recordatorio, mensaje-recordatorio, fecha de entrega.

Relaciones: Cada tarea está relacionada los equipos de trabajo y notificaciones enviadas como recordatorio.

Equipment (Equipo):

Un grupo de integrantes con habilidades y destrezas donde se organizan para llegar a u objetivo predeterminado.

Atributos: Id (Identificador único), nombre, y logo

Relaciones: Cada equipo de trabajo está relacionado a una serie de notificaciones de notas donde se envían las tareas y actividades a realizar.

Relaciones:

-“equipo” se relaciona con “tareas” para asignar las tareas al equipo.

-“equipo” se relaciona con “notas” para tomar apuntes de algún tema en específico.

-“equipo” se relaciona con “notificaciones” para recibir notificaciones de algunas notas o tareas asignadas.

-“notas” se relaciona con “notificaciones” para notificar las notas al equipo de trabajo

-“tareas” se relaciona con “notificaciones” para notificar las tareas a equipo de trabajo.

Colecciones (NoSLQ)

Equipo: {

_id: Objectld,

nombre: String,

logo: [Objectld],

}

Notificaciones: {

_id: Objectld,

fecha: Date,

emisor: Sring,

receptor: String,

contenido: String,

}

Notas: {

_id: Objectld,

fecha_creacion: Date,

autor: string,

notas: String,

}

Tareas: {

_id: Objectld,

nombre: String,

fecha_creacion: Date,

autor: String,

notas: Sring,

responsable: String,

fecha_recordatorio: Date,

mensaje_recordatorio: Date,

fecha_entrega: Date,

}

- Etapa 2: Persistencia de Datos con Backend

6. Introducción

En el contexto de un backend en una plataforma educativa, la persistencia de datos juega un papel fundamental en el almacenamiento y gestión eficiente de la información asociada con usuarios, contenido educativo, actividades colaborativas y más. Este proceso asegura la coherencia y la integridad de los datos a lo largo del tiempo, permitiendo que la plataforma funcione de manera consistente.

Propósito de la Etapa

El propósito de la etapa implica seleccionar la tecnología de base de datos adecuada, definir esquemas de almacenamiento eficientes y considerar estrategias de acceso y recuperación de datos. Además, se explorarán aspectos como la escalabilidad, la seguridad y la integridad de los datos para garantizar un sistema robusto y confiable.

Alcance de la Etapa

Tiene como objetivo principal identificar y diseñar la solución de persistencia de datos para el backend de la plataforma educativa, abarcando desde la evaluación de tecnologías de base de datos hasta la definición de esquemas y estrategias de acceso eficientes. Se busca seleccionar la tecnología más apropiada, considerando la escalabilidad y la seguridad, y diseñar esquemas que reflejen la estructura lógica de la información a gestionar. Además, se integran estrategias para facilitar el trabajo colaborativo, asegurando una gestión eficiente de datos compartidos entre usuarios y estableciendo conexiones efectivas entre la base de datos y la lógica de negocio de la plataforma, proporcionando así una base sólida para el desarrollo colaborativo futuro.

-

7. Diseño de la Arquitectura de Backend

Descripción de la Arquitectura Propuesta

En nuestro proyecto, la arquitectura del backend basada en Nest.js se caracteriza por su enfoque modular, escalabilidad y facilidad de mantenimiento. Nest.js utiliza un diseño modular y basado en controladores que se asemeja a la arquitectura de Angular, lo que proporciona una estructura organizada y fácil de entender.

La arquitectura consta de varios elementos clave:

Módulos:

La aplicación está estructurada en módulos independientes, cada uno con su propio conjunto de controladores, servicios y modelos de datos. Esto facilita la modularidad y permite una mejor organización del código.

Controladores:

Los controladores gestionan las solicitudes HTTP y se encargan de dirigirlas a los servicios correspondientes. Siguen el patrón de diseño de inyección de dependencias para acceder a los servicios necesarios.

Servicios:

Los servicios contienen la lógica de negocio de la aplicación. Están diseñados para ser reutilizables y pueden ser inyectados en diferentes partes del código, como controladores o otros servicios.

Modelos de Datos:

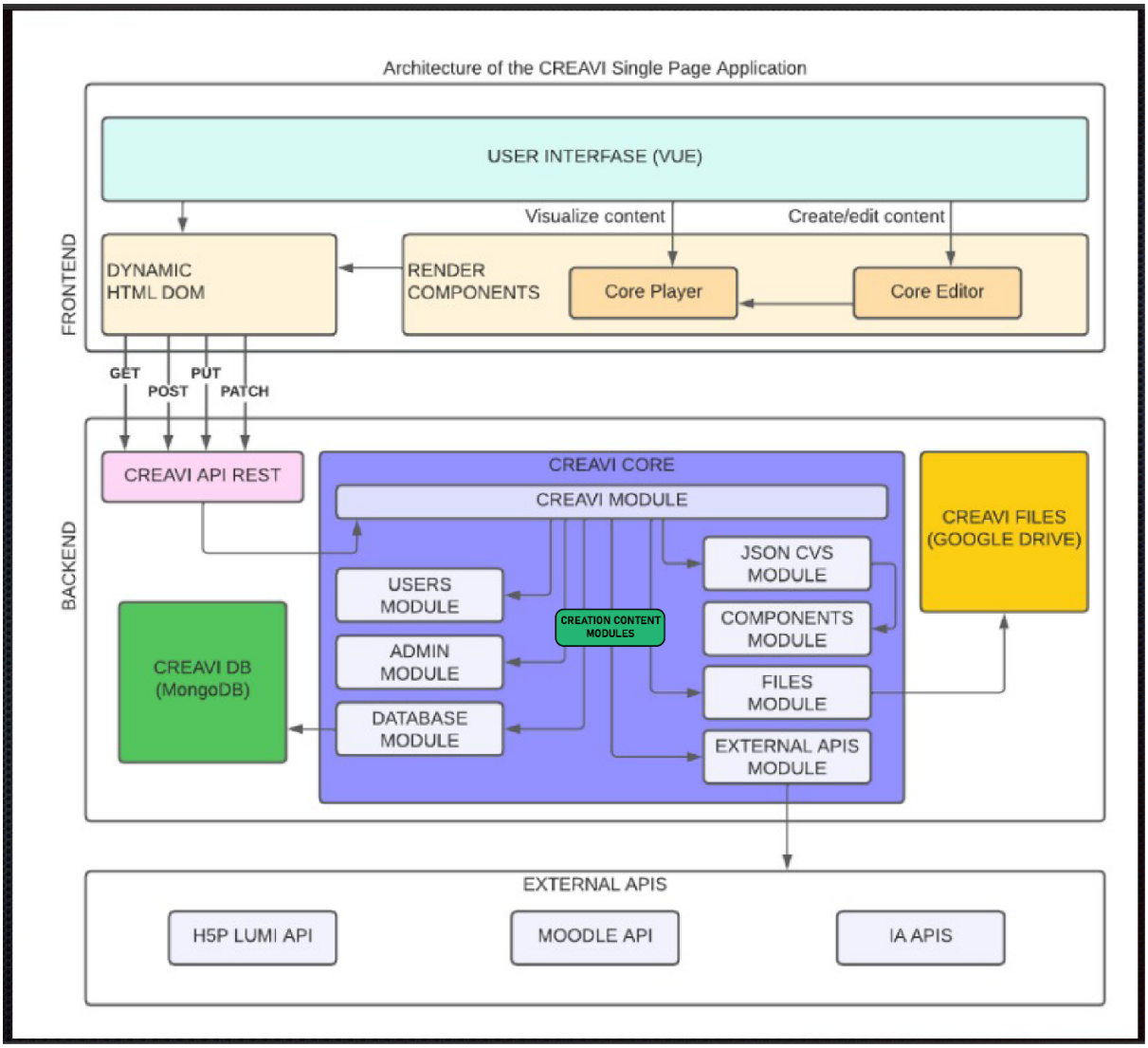
Definimos modelos de datos para representar las entidades del dominio de nuestra aplicación. Estos modelos pueden incluir entidades de base de datos, DTOs (Data Transfer Objects) para la comunicación y otros objetos relacionados con la lógica de negocio.

Esta arquitectura basada en Nest.js proporciona un marco sólido para el desarrollo del backend, promoviendo la organización, la reutilización y la mantenibilidad del código, aspectos cruciales para el éxito a largo plazo de nuestro proyecto.

Componentes del Backend

En nuestro backend construido con Nest.js, los componentes clave incluyen módulos, controladores que manejan solicitudes HTTP, servicios que contienen la lógica de negocio, modelos de datos para representar entidades, middleware para procesar solicitudes, una capa de acceso a la base de datos (por ejemplo, con TypeORM), interceptores y guards para el control de flujo, configuración de la aplicación, logging para seguimiento de eventos, pruebas unitarias e integración, inyección de dependencias para la gestión de dependencias, y una API RESTful que sigue mejores prácticas de diseño. Estos componentes trabajan en conjunto para crear un backend organizado, modular y eficiente.

Diagramas de Arquitectura



-

-

8. Elección de la Base de Datos

Evaluación de Opciones (SQL o NoSQL)

SQL (Structured Query Language) y NoSQL (Not Only SQL) representan dos enfoques diferentes para el diseño de bases de datos. SQL se basa en un modelo relacional, empleando tablas estructuradas y un esquema predefinido. Ofrece transacciones ACID para garantizar la consistencia de los datos. NoSQL, por otro lado, proporciona flexibilidad en la estructura de datos, permitiendo adaptarse a cambios sin esquemas predefinidos. Es altamente escalable horizontalmente y puede ofrecer un rendimiento más rápido en operaciones específicas. La elección entre SQL y NoSQL depende de los requisitos específicos de cada proyecto en cuanto a estructura de datos, consistencia y escalabilidad. En muchos casos, se elige una combinación de ambos para aprovechar sus fortalezas en diferentes situaciones.

Justificación de la Elección

La elección entre SQL y NoSQL depende en gran medida de las necesidades específicas de un proyecto y de las características particulares de los datos y operaciones que se deben manejar. Aquí hay algunas justificaciones comunes para elegir uno u otro:

Justificación para SQL:

Estructura Relacional:

Si los datos se pueden modelar eficientemente en tablas relacionales con relaciones claras entre ellas, SQL es una opción sólida.

Integridad y Consistencia:

Cuando es fundamental garantizar la integridad y consistencia de los datos, SQL, con su soporte para transacciones ACID, proporciona un nivel de garantía elevado.

Esquema Fijo:

Si los requisitos del proyecto son estables y el esquema de datos no cambia con frecuencia, un modelo con un esquema fijo puede ser beneficioso.

Justificación para NoSQL:

Estructura Flexible:

Si los datos son no estructurados o cambian con frecuencia, NoSQL ofrece flexibilidad para adaptarse a estos cambios sin necesidad de un esquema predefinido.

Escalabilidad Horizontal:

Cuando se anticipa un alto volumen de datos y tráfico, NoSQL, con su capacidad para escalar horizontalmente en clústeres de servidores, puede ser más adecuado.

Modelos de Datos Específicos:

Si el modelo de datos de documentos, grafos, clave-valor o columnares se adapta mejor a la naturaleza de los datos y las consultas, NoSQL ofrece opciones especializadas.

Rendimiento Específico:

Para aplicaciones que priorizan el rendimiento en operaciones específicas y consultas, NoSQL puede ser más eficiente gracias a su diseño simplificado y enfoque en escalabilidad.

9. Implementación del Backend

Elección del Lenguaje de Programación

JavaScript es un lenguaje de programación diseñado en un principio para añadir interactividad a las páginas webs y crear aplicaciones web. A pesar de la similitud en el nombre, no está relacionado con Java. Se emplea en el desarrollo de páginas web para tareas como cambiar automáticamente la fecha de una página, hacer que una página aparezca en una ventana emergente al hacer clic en un enlace o que un texto o imagen cambien al pasar el ratón por encima. También suele emplearse para hacer encuestas y formularios. Se ejecuta en el ordenador del visitante a la web, por lo que no requiere descargas constantes desde el sitio web.

Creación de la Lógica de Negocio

Notificaciones:

Las notificaciones son cruciales para mantener a los usuarios informados sobre cambios relevantes en el sistema, actualizaciones en tareas, nuevos comentarios, invitaciones a equipos, entre otros. Proporcionan una vía efectiva de comunicación en tiempo real, lo que es esencial para la colaboración.

Equipos de Trabajo:

Los equipos de trabajo son fundamentales para organizar a los usuarios en grupos que colaboran en proyectos específicos. Facilitan la asignación de tareas, la comunicación interna y la gestión eficiente de la información relacionada con un conjunto particular de actividades colaborativas.

Notas:

Las notas pueden ser valiosas para la toma de apuntes, la documentación de ideas, la colaboración en documentos compartidos y la creación de contenido colaborativo. Son herramientas esenciales para el intercambio de información y la construcción de conocimiento compartido.

Tareas:

Las tareas representan acciones específicas que deben realizarse en el contexto del trabajo colaborativo. La gestión efectiva de tareas incluye la asignación, el seguimiento del progreso y la comunicación sobre cualquier cambio o actualización en relación con estas. Son centrales para la coordinación y ejecución eficiente de proyectos colaborativos.

Desarrollo de Endpoints y APIs

(GET, POST, PUT, PATCH) se refieren a métodos HTTP, que son comandos que indican la acción que se debe realizar en un recurso específico. Aquí hay una breve descripción de cada uno:

GET:

Se utiliza para solicitar datos de un recurso específico. Es un método "seguro" en el sentido de que no debería cambiar el estado del servidor.

POST:

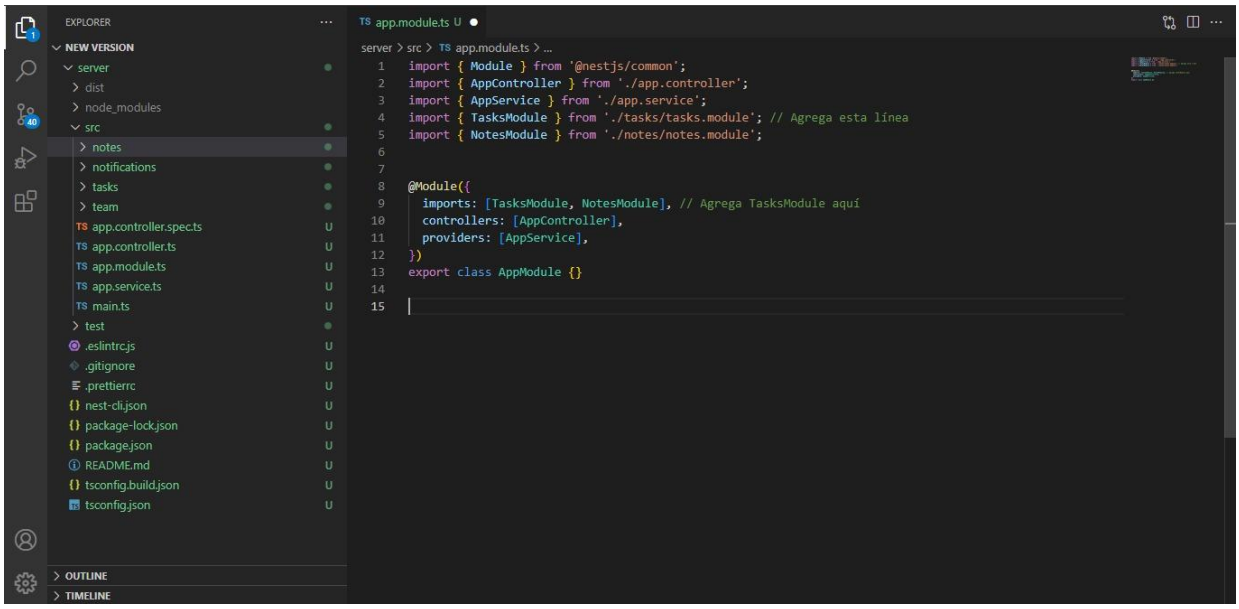
Se utiliza para enviar datos al servidor para crear un recurso nuevo. Es comúnmente utilizado para enviar formularios y cargar archivos.

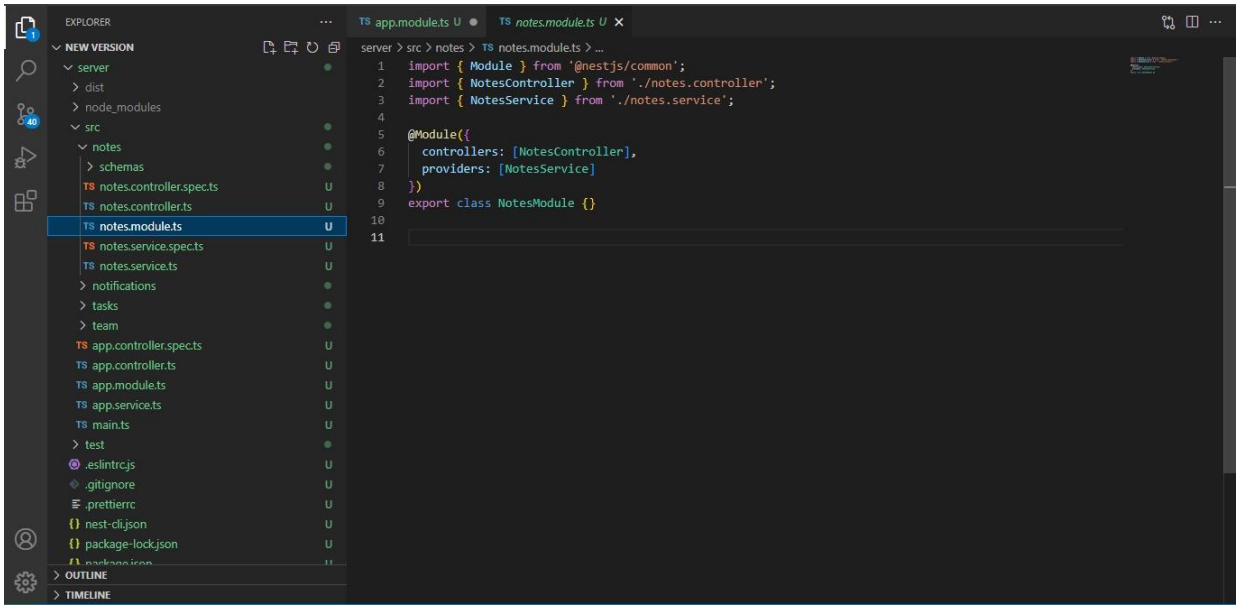
PUT:

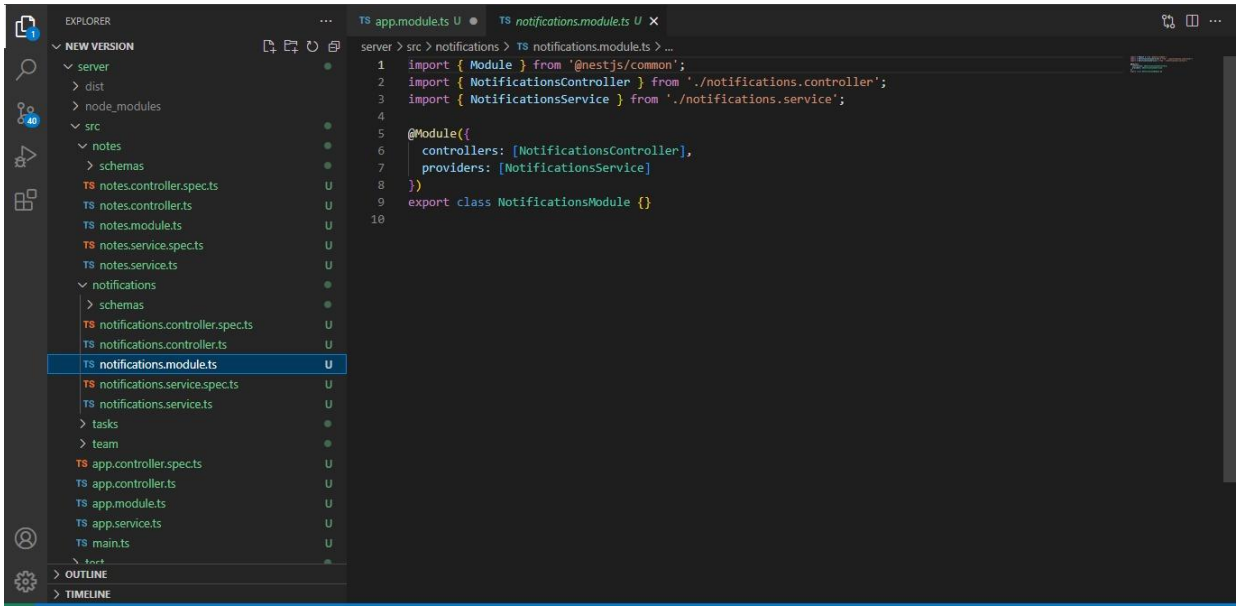
Se utiliza para actualizar un recurso existente en el servidor. La solicitud PUT generalmente incluye toda la representación del recurso, incluso si solo se están actualizando algunos campos.

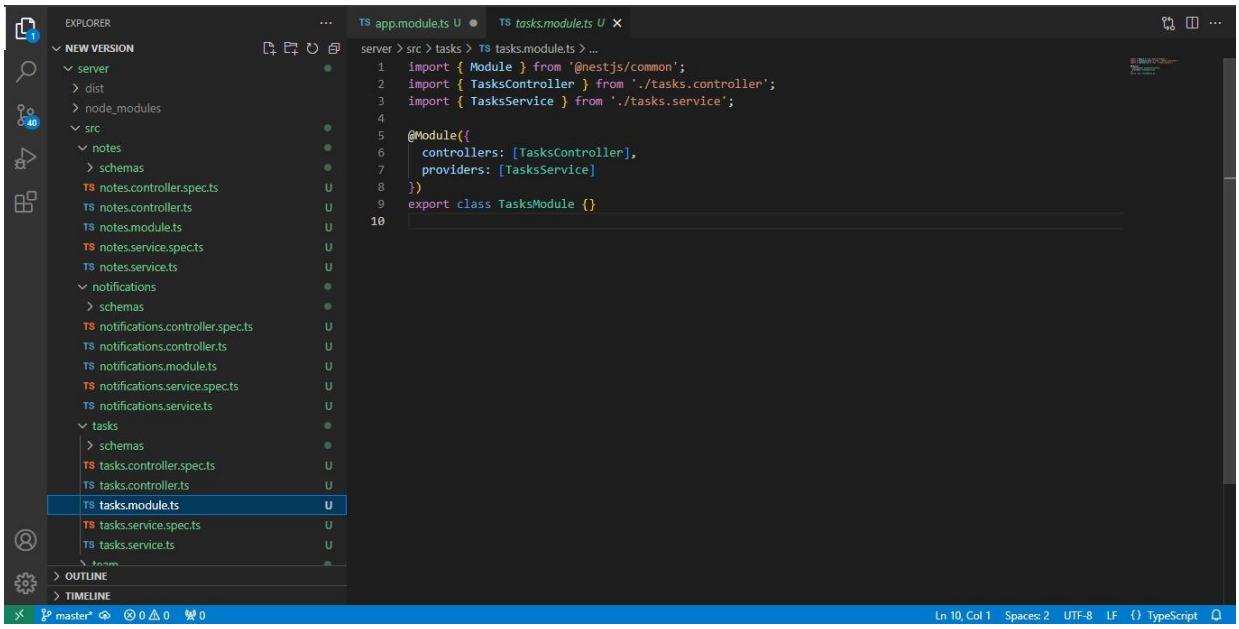
PATCH:

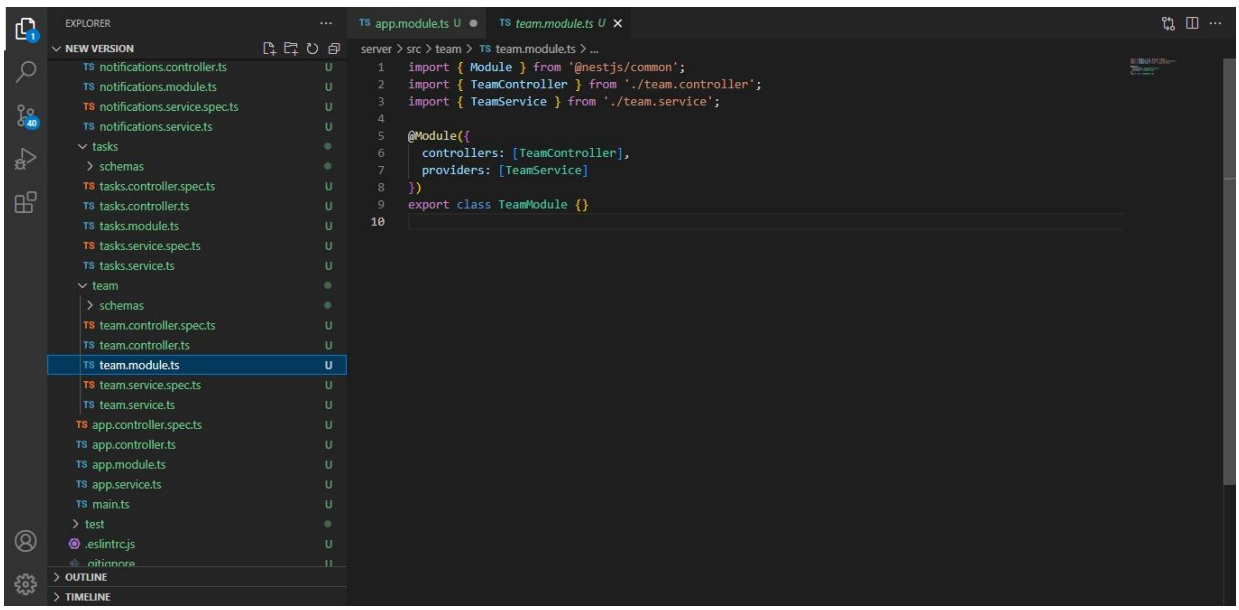
Similar a PUT, pero se utiliza para actualizar parcialmente un recurso en lugar de reemplazarlo por completo. La solicitud PATCH contiene solo los cambios que se deben aplicar.

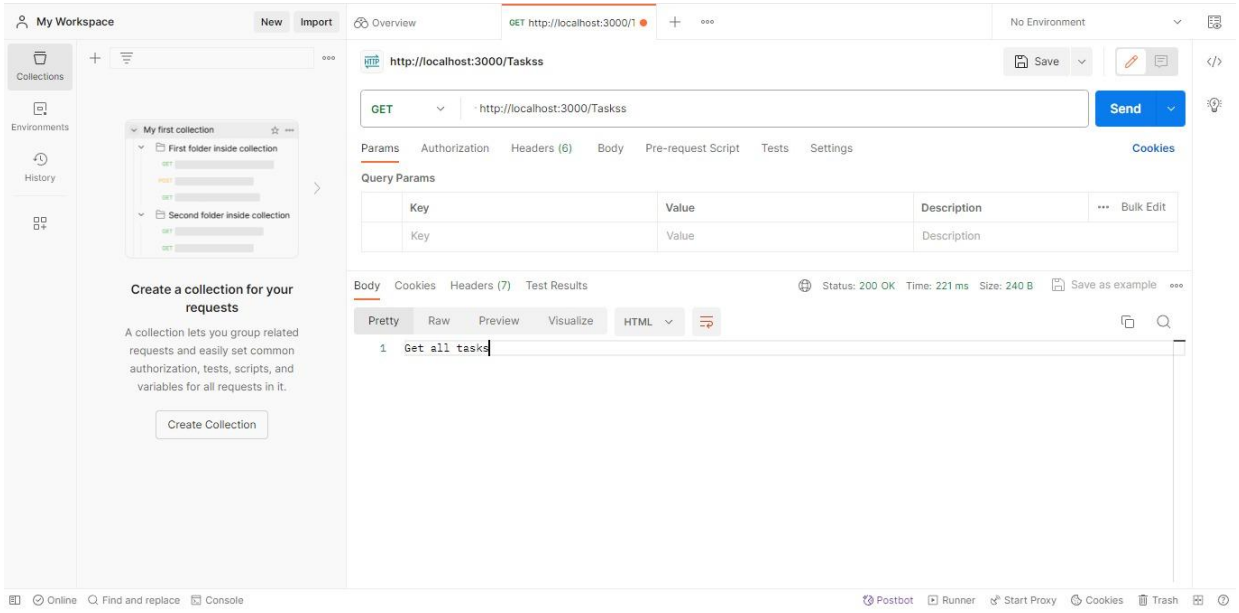

EXPLORER
NEW VERSION
server
dist
node_modules
src
notes
notifications
tasks
team
app.controller.spec.ts
app.controller.ts
app.module.ts
app.service.ts
main.ts
test
.eslintrc.js
.gitignore
.prettierrc
nest-cli.json
package-lock.json
package.json
README.md
tsconfig.build.json
tsconfig.json
OUTLINE
TIMELINE
app.module.ts U
server > src > app.module.ts > ...
import { Module } from '@nestjs/common';
import { AppController } from './app.controller';
import { AppService } from './app.service';
import { TasksModule } from './tasks/tasks.module'; // Agrega esta línea
import { NotesModule } from './notes/notes.module';
@Module({
imports: [TasksModule, NotesModule], // Agrega TasksModule aquí
controllers: [AppController],
providers: [AppService],
})
export class AppModule {}

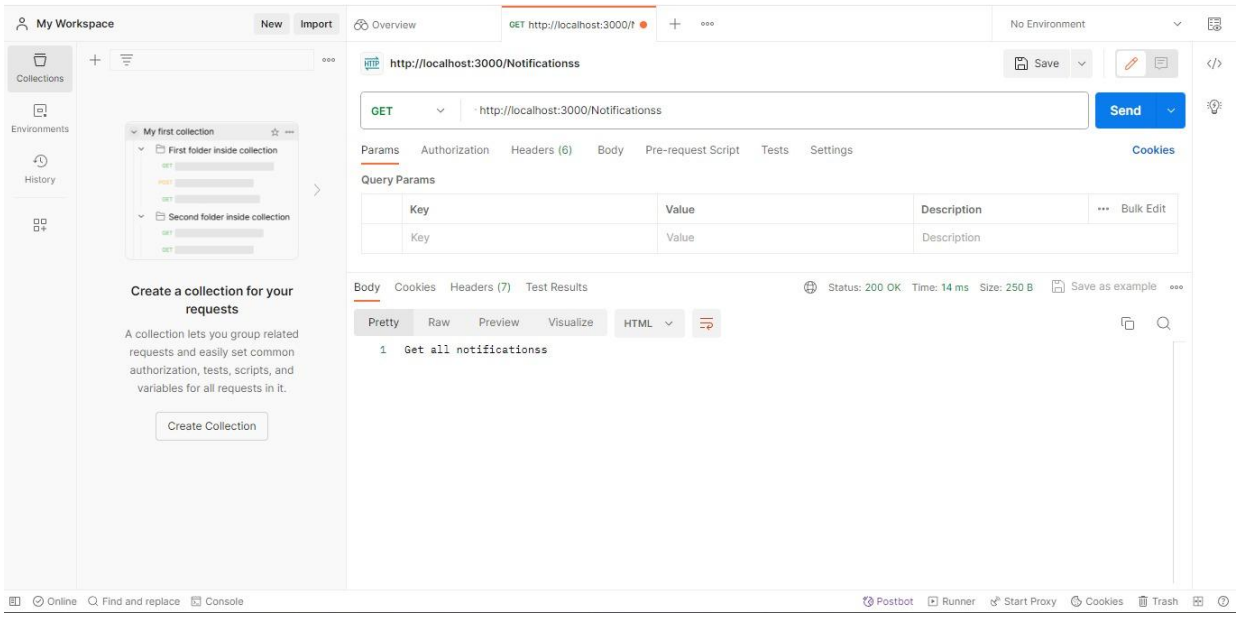

EXPLORER
NEW VERSION
server
dist
node_modules
src
notes
schemas
notes.controller.spec.ts
notes.controller.ts
notes.module.ts
notes.service.spec.ts
notes.service.ts
notifications
tasks
team
app.controller.spec.ts
app.controller.ts
app.module.ts
app.service.ts
main.ts
test
.eslintrc.js
.gitignore
.prettierrc
nest-cli.json
package-lock.json
OUTLINE
TIMELINE
app.module.ts U
notes.module.ts U
server > src > notes > notes.module.ts > ...
import { Module } from '@nestjs/common';
import { NotesController } from './notes.controller';
import { NotesService } from './notes.service';
@Module({
controllers: [NotesController],
providers: [NotesService]
})
export class NotesModule {}

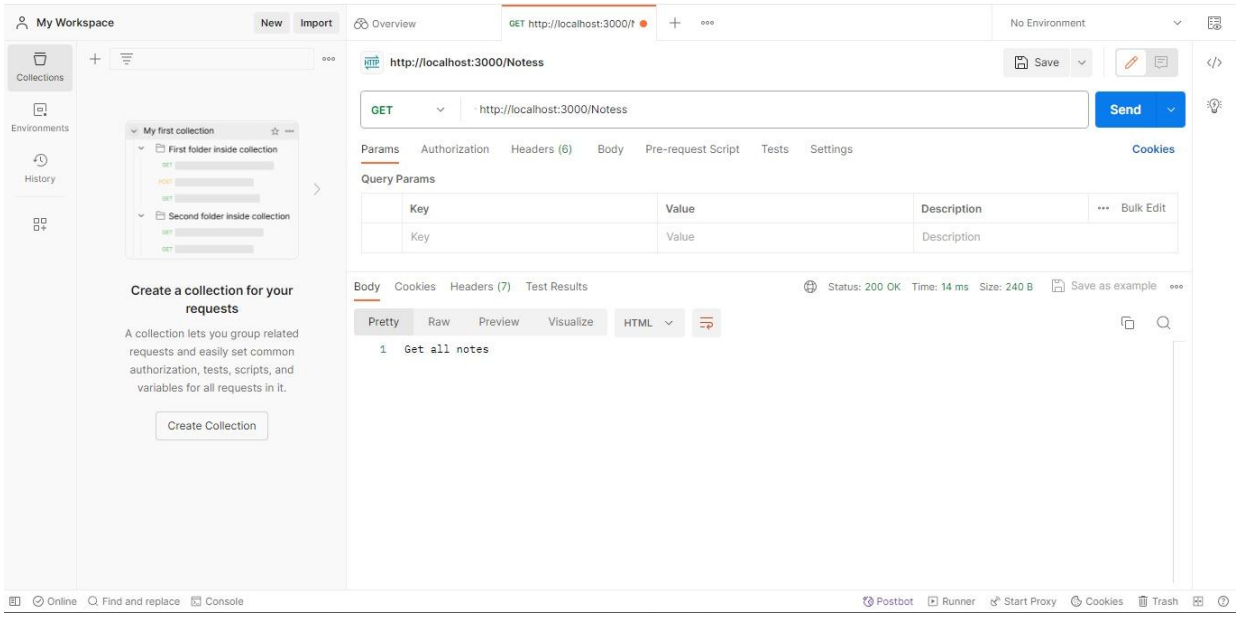

EXPLORER
NEW VERSION
server
dist
node_modules
src
notes
schemas
notes.controller.spec.ts
notes.controller.ts
notes.module.ts
notes.service.spec.ts
notes.service.ts
notifications
schemas
notifications.controller.spec.ts
notifications.controller.ts
notifications.module.ts
notifications.service.spec.ts
notifications.service.ts
tasks
team
app.controller.spec.ts
app.controller.ts
app.module.ts
app.service.ts
main.ts
OUTLINE
TIMELINE
app.module.ts U
notifications.module.ts U
server > src > notifications > notifications.module.ts > ...
import { Module } from '@nestjs/common';
import { NotificationsController } from './notifications.controller';
import { NotificationsService } from './notifications.service';
@Module({
controllers: [NotificationsController],
providers: [NotificationsService]
})
export class NotificationsModule {}

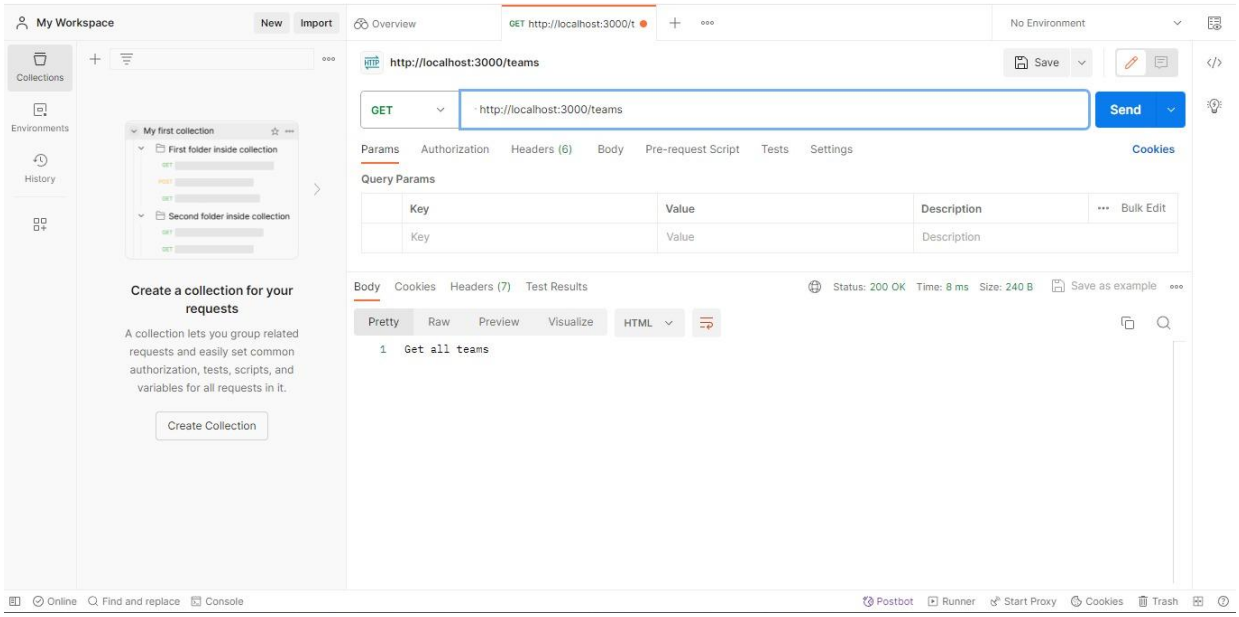

EXPLORER
NEW VERSION
server
dist
node_modules
src
notes
schemas
notes.controller.spec.ts
notes.controller.ts
notes.module.ts
notes.service.spec.ts
notes.service.ts
notifications
schemas
notifications.controller.spec.ts
notifications.controller.ts
notifications.module.ts
notifications.service.spec.ts
notifications.service.ts
tasks
schemas
tasks.controller.spec.ts
tasks.controller.ts
tasks.module.ts
tasks.service.spec.ts
tasks.service.ts
OUTLINE
TIMELINE
app.module.ts U
tasks.module.ts U
server > src > tasks > tasks.module.ts > ...
import { Module } from '@nestjs/common';
import { TasksController } from './tasks.controller';
import { TasksService } from './tasks.service';
@Module({
controllers: [TasksController],
providers: [TasksService]
})
export class TasksModule {}
master*
Ln 10, Col 1
Spaces: 2
UTF-8
LF
TypeScript


EXPLORER
NEW VERSION
notifications.controller.ts
notifications.module.ts
notifications.service.spec.ts
notifications.service.ts
tasks
schemas
tasks.controller.spec.ts
tasks.controller.ts
tasks.module.ts
tasks.service.spec.ts
tasks.service.ts
team
schemas
team.controller.spec.ts
team.controller.ts
team.module.ts
team.service.spec.ts
team.service.ts
app.controller.spec.ts
app.controller.ts
app.module.ts
app.service.ts
main.ts
test
.eslintrc.js
OUTLINE
TIMELINE
app.module.ts U
team.module.ts U
server > src > team > team.module.ts > ...
import { Module } from '@nestjs/common';
import { TeamController } from './team.controller';
import { TeamService } from './team.service';
@Module({
controllers: [TeamController],
providers: [TeamService]
})
export class TeamModule {}


My Workspace
New
Import
Overview
GET http://localhost:3000/1
No Environment
Collections
Environments
History
http://localhost:3000/Taskss
Save
GET
http://localhost:3000/Taskss
Send
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Body
Cookies
Headers (7)
Test Results
Status: 200 OK
Time: 221 ms
Size: 240 B
Save as example
Pretty
Raw
Preview
Visualize
HTML
1 Get all tasks
My first collection
First folder inside collection
Second folder inside collection
Create a collection for your requests
A collection lets you group related requests and easily set common authorization, tests, scripts, and variables for all requests in it.
Create Collection
Online
Find and replace
Console
Postbot
Runner
Start Proxy
Cookies
Trash


My Workspace
New
Import
Overview
GET http://localhost:3000/t
No Environment
Collections
Environments
History
http://localhost:3000/Notificationss
Save
GET
http://localhost:3000/Notificationss
Send
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Body
Cookies
Headers (7)
Test Results
Status: 200 OK
Time: 14 ms
Size: 250 B
Save as example
Pretty
Raw
Preview
Visualize
HTML
1 Get all notificationss
My first collection
First folder inside collection
Second folder inside collection
Create a collection for your requests
A collection lets you group related requests and easily set common authorization, tests, scripts, and variables for all requests in it.
Create Collection
Online
Find and replace
Console
Postbot
Runner
Start Proxy
Cookies
Trash





Autenticación y Autorización

10. Conexión a la Base de Datos

Configuración de la Conexión

Instalación de Node.js:

- Descargar e Instalar Node.js:

Descargo e instalo Node.js desde nodejs.org.

- Verificar la Instalación:

Abro la terminal y verifico que Node.js y npm estén instalados con node -v y npm -v.

Instalación de Nest.js:

- Instalar Nest.js Globalmente:

Ejecuto en la terminal: npm install -g @nestjs/cli

- Crear y Ejecutar un Proyecto:

Creo un nuevo proyecto Nest.js con: nest new nombre-del-proyecto

- Luego, voy al directorio del proyecto y ejecuto:
  cd nombre-del-proyecto
  npm run start

Conexión con MongoDB Atlas:

Crear Cuenta en MongoDB Atlas:

- Creo una cuenta en MongoDB Atlas.

Configurar Clúster:

- Creo un clúster en MongoDB Atlas y configuro la lista blanca de IP.

Crear Usuario y Obtener Cadena de Conexión:

- Creo un usuario con permisos y obtengo la cadena de conexión.

Conectar la Aplicación a MongoDB Atlas:

- Uso mongoose o un paquete similar en mi proyecto Nest.js y configuro la conexión con la cadena obtenida.

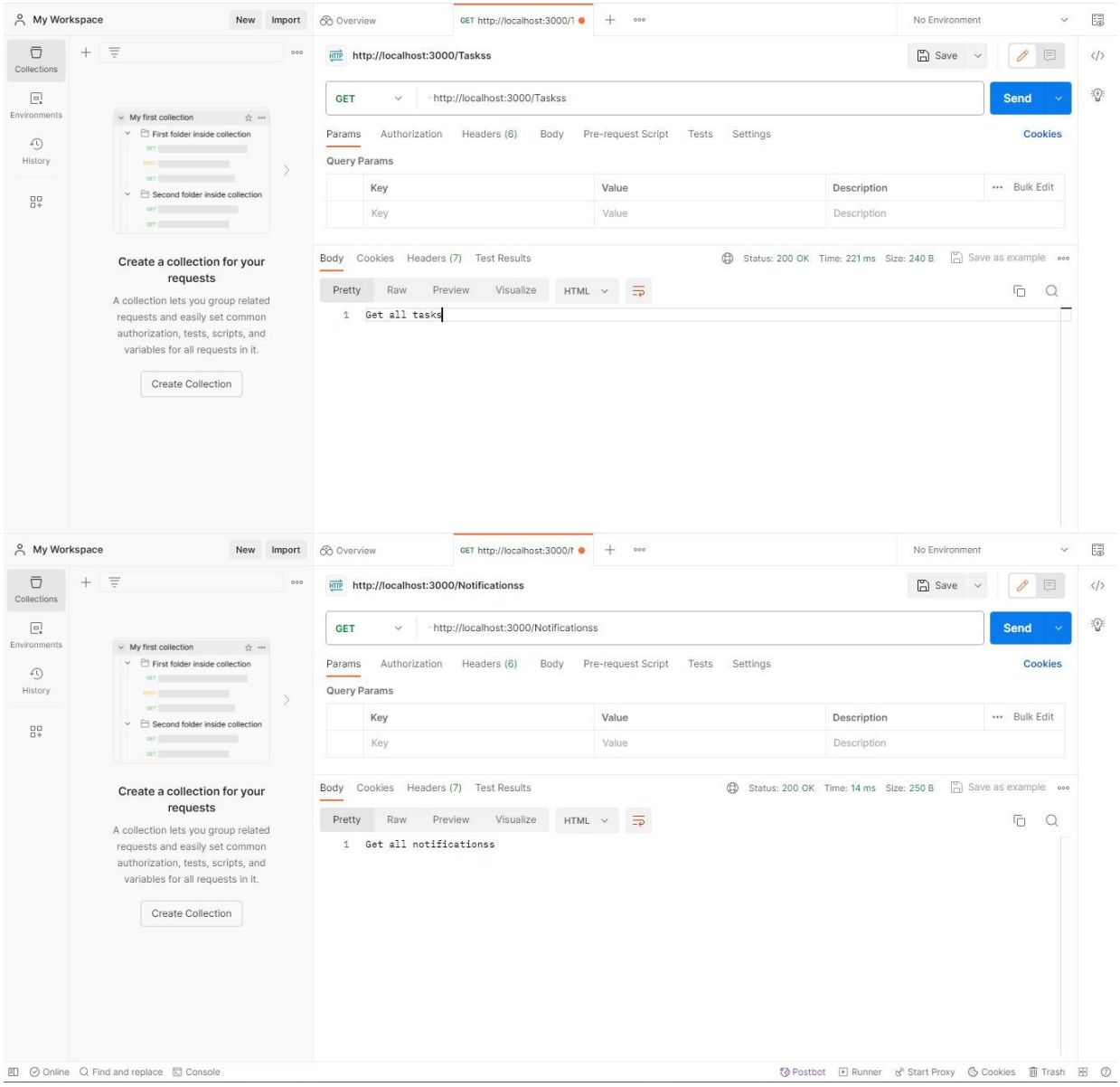
Desarrollo de Operaciones CRUD

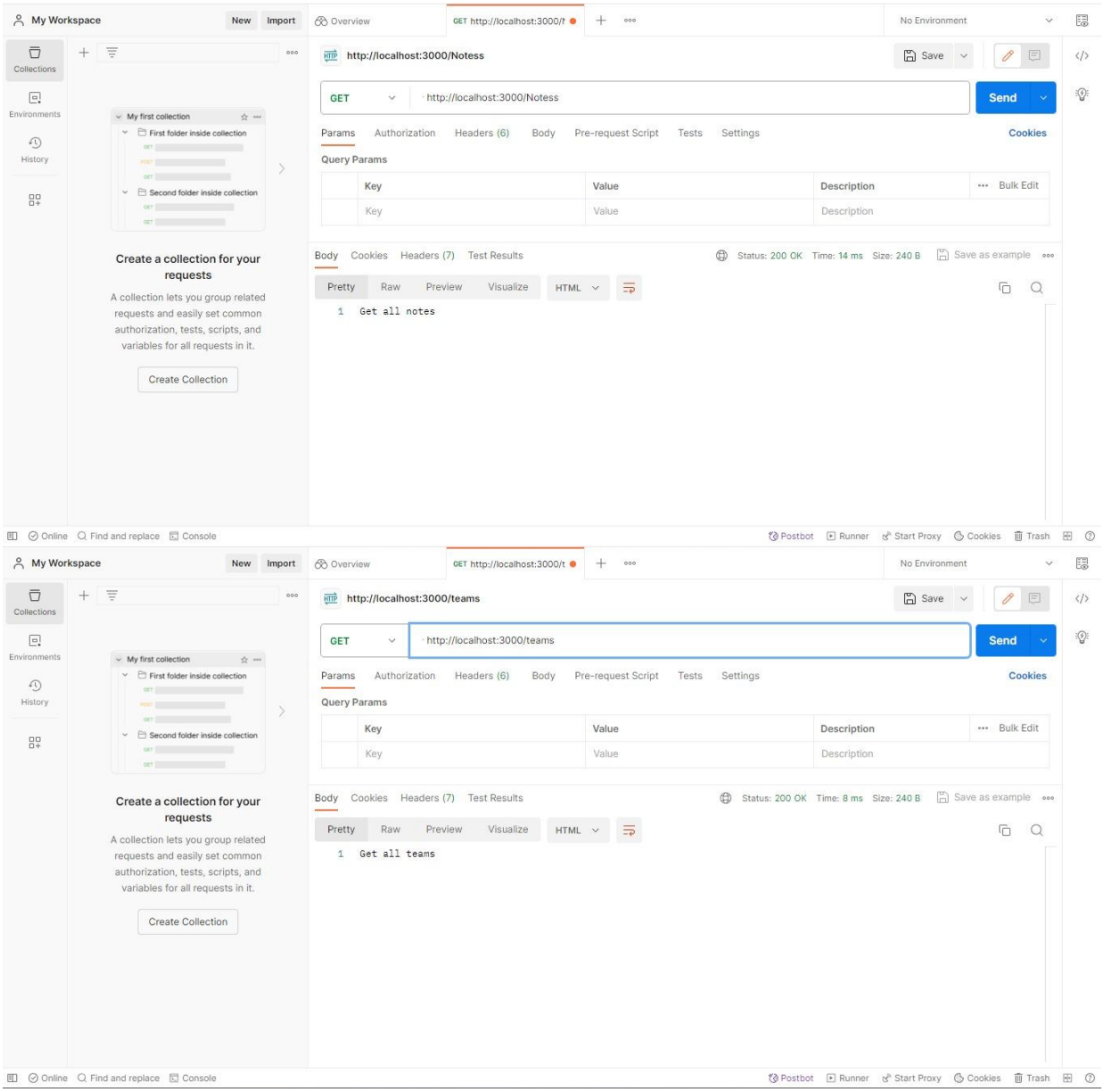
Manejo de Transacciones

-

11. Pruebas del Backend

Diseño de Casos de Prueba


My Workspace
New
Import
Overview
GET http://localhost:3000/1
No Environment
Collections
Environments
History
http://localhost:3000/Taskss
Save
GET
http://localhost:3000/Taskss
Send
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Body
Cookies
Headers (7)
Test Results
Status: 200 OK
Time: 221 ms
Size: 240 B
Save as example
Pretty
Raw
Preview
Visualize
HTML
1 Get all tasks
My first collection
First folder inside collection
Second folder inside collection
Create a collection for your requests
A collection lets you group related requests and easily set common authorization, tests, scripts, and variables for all requests in it.
Create Collection
My Workspace
New
Import
Overview
GET http://localhost:3000/t
No Environment
Collections
Environments
History
http://localhost:3000/Notificationss
Save
GET
http://localhost:3000/Notificationss
Send
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Body
Cookies
Headers (7)
Test Results
Status: 200 OK
Time: 14 ms
Size: 250 B
Save as example
Pretty
Raw
Preview
Visualize
HTML
1 Get all notificationss
My first collection
First folder inside collection
Second folder inside collection
Create a collection for your requests
A collection lets you group related requests and easily set common authorization, tests, scripts, and variables for all requests in it.
Create Collection
Online
Find and replace
Console
Postbot
Runner
Start Proxy
Cookies
Trash



Ejecución de Pruebas Unitarias y de Integración

Manejo de Errores y Excepciones

- Etapa 3: Consumo de Datos y Desarrollo Frontend

12. Introducción

Propósito de la Etapa

Alcance de la Etapa

Definiciones y Acrónimos

-

13. Creación de la Interfaz de Usuario (UI)

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

Consideraciones de Usabilidad

Maquetación Responsiva

-

14. Programación Frontend con JavaScript (JS)

Desarrollo de la Lógica del Frontend

Manejo de Eventos y Comportamientos Dinámicos

Uso de Bibliotecas y Frameworks (si aplicable)

-

15. Consumo de Datos desde el Backend

Configuración de Conexiones al Backend

Obtención y Presentación de Datos

Actualización en Tiempo Real (si aplicable)

-

16. Interacción Usuario-Interfaz

Manejo de Formularios y Validación de Datos

Implementación de Funcionalidades Interactivas

Mejoras en la Experiencia del Usuario

-

17. Pruebas y Depuración del Frontend

Diseño de Casos de Prueba de Frontend

Pruebas de Usabilidad

Depuración de Errores y Optimización del Código

-

18. 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

-

19. 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 la pizarra 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 de la pizarra 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.