



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

Sistema de registro de asistencia con códigos QR

Autores:

Lemuel David Marzola Baron

Andrés Felipe Baldovino Montiel

Alejandra Sofía Agamez Ibarra

Pedro Luis Montalvo Galindo

Leonardo Berrocal Álvarez

Alexander Toscano Ricardo





Descripción del software

En la actualidad, con el avance consecutivo del uso de las tecnologías se busca de manera eficiente poder abarcar y facilitar diversas acciones en el ámbito educativo , de esta forma el componente Sistema de Registro de Asistencia mediante Códigos QR, busca principalmente implementar una solución tecnológica e innovadora para simplificar y agilizar el proceso de registro de asistencia de eventos . Mediante el uso de códigos QR, a través de estos los usuarios podrán registrar su asistencia de manera rápida y eficiente, mientras que los responsables tendrán acceso a datos precisos y actualizados en tiempo real. Con un enfoque centrado en la usabilidad, la seguridad y la escalabilidad, este proyecto busca mejorar significativamente la gestión de la asistencia en la institución, proporcionando una solución moderna y adaptable a las necesidades cambiantes del entorno educativo.



ETAPA 1: DISEÑO DE LA APLICACIÓN Y ANÁLISIS DE REQUISITOS

1. INTRODUCCIÓN

1.1. Propósito del documento	6
1.2. Alcance del proyecto módulo de pizarra compartida	8
1.3. Definiciones y acrónimos	8

2. DESCRIPCIÓN GENERAL

2.1. Objetivos del sistema	10
2.2. Funcionalidad general	10
2.3. Usuarios del sistema	11
2.4. Restricciones	12

3. REQUISITOS FUNCIONALES

3.1. Casos de uso	13
3.2. Diagramas de flujo de casos de uso	14
3.3. Descripción detallada de cada caso de uso	14
3.4. Prioridad de requerimientos	16

4. REQUISITOS NO FUNCIONALES

4.1. Requisitos de desempeño	18
4.2. Requisitos de seguridad	19
4.3. Requisitos de usabilidad	20
4.4. Requisitos de escalabilidad	20

5. MODELADO E/R

5.1. Diagrama de entidad-relación	21
5.2. Diagrama relacional	22
5.3. Script de modelo relacional	23
5.4. Descripción de entidades y relaciones	24
5.5. Reglas de integridad referencial	25
5.6. Colecciones (NoSQL)	28

6. ANEXOS

6.1. Diagramas adicionales	29
6.2. Referencias	29



ETAPA 2: PERSISTENCIA DE DATOS CON BACKEND

7. INTRODUCCIÓN

7.1. Propósito de la etapa	30
7.2. Alcance de la etapa	30
7.3. Definiciones y acrónimos	30

8. DISEÑO DE LA ARQUITECTURA DE BACKEND

8.1. Descripción de la arquitectura propuesta	30
8.2. Componentes del backend	30
8.3. Diagramas de arquitectura	30

9. ELECCIÓN DE LA BASE DE DATOS

9.1. Evaluación de opciones (SQL o NoSQL)	31
9.2. Justificación de la elección	31
9.3. Diseño de esquema de base de datos	31

10. IMPLEMENTACIÓN DEL BACKEND

10.1. Elección del lenguaje de programación	31
10.2. Creación de la lógica de negocio	31
10.3. Desarrollo de endpoints y APIs	31
10.4. Autenticación y autorización	31

11. CONEXIÓN A LA BASE DE DATOS

11.1. Configuración de la conexión	32
11.2. Desarrollo de operaciones CRUD	32
11.3. Manejo de transacciones	32

12. PRUEBAS DEL BACKEND

12.1. Diseño de casos de prueba	32
12.2. Ejecución de pruebas unitarias y de integración	32
12.3. Manejo de errores y excepciones	32



ETAPA 3: CONSUMO DE DATOS Y DESARROLLO FRONTEND

13. INTRODUCCIÓN

13.1. Propósito de la etapa	33
13.2. Alcance de la etapa	33
13.3. Definiciones y acrónimos	33

14. CREACIÓN DE LA INTERFAZ DE USUARIO (UI)

14.1. Diseño de la interfaz de usuario (UI) con HTML y CSS	33
14.2. Consideraciones de usabilidad	33
14.3. Maquetación responsiva	33

15. PROGRAMACIÓN FRONTEND CON JAVASCRIPT (JS)

15.1. Desarrollo de la lógica del frontend	34
15.2. Manejo de eventos y comportamientos dinámicos	34
15.3. Uso de bibliotecas y frameworks (si aplica)	34

16. CONSUMO DE DATOS DESDE EL BACKEND

16.1. Configuración de conexiones al backend	34
16.2. Obtención y presentación de datos	34
16.3. Actualización en tiempo real (si aplica)	34

17. INTERACCIÓN USUARIO-INTERFAZ

17.1. Manejo de formularios y validación de datos	35
17.2. Implementación de funcionalidades interactivas	35
17.3. Mejoras en la experiencia del usuario	35

18. PRUEBAS Y DEPURACIÓN DEL FRONTEND

18.1. Diseño de casos de prueba de frontend	35
18.2. Pruebas de usabilidad	35
18.3. Depuración de errores y optimización del código	35

19. IMPLEMENTACIÓN DE LA LÓGICA DE NEGOCIO EN EL FRONTEND

19.1. Migración de la lógica de negocio desde el backend (si necesario)	36
19.2. Validación de datos y reglas de negocio en el frontend	36

20. INTEGRACIÓN CON EL BACKEND

20.1. Verificación de la comunicación efectiva con el backend	36
20.2. Pruebas de integración frontend-backend	36

ANEXOS	36
---------------------	----



Etapas 1: Diseño de la Aplicación y Análisis de Requisitos

1. Introducción

Propósito del Documento

"Sistema de registro de asistencia con código QR" es un proyecto de diseño de software educativo I, II, III, que pretende mejorar el proceso de asistencia y registro de los estudiantes en clases o eventos a través de códigos QR. La manera tradicional de tomar asistencia al comienzo de una clase le toma siempre al docente un lapso de tiempo el cual podría ser utilizado para abarcar más la temática a trabajar de igual manera en eventos a veces no se alcanza a recoger el número de datos de todos los que asisten debido a que no alcanza en material impreso o no todos tienen tiempo a llenar el material, lo cual es problemático para los docentes y organizadores del evento ya que se pierde tiempo valioso en una acción que se puede mejorar y optimizar.

Este documento, pretende documentar el proceso de diseño y análisis del sistema, desarrollo e implementación del sistema único de código QR. Este proceso se organiza en 5 fases o etapas, cada curso de diseño de software educativo desarrolla una de estas dichas fases:

Fase 1: Diseño del sistema y Análisis de Requisitos

Esta fase desarrolla el diseño y documentación necesaria para llevar a cabo la propuesta del sistema de asistencia mediante código QR. Este proceso se enmarca en la metodología Sprint de desarrollo de software, integrando las habilidades adquiridas en la Lic. informática y medios audiovisuales para desarrollar un producto innovador, que a su vez es un programa informático de carácter educativo, que busca que sus usuarios tengan experiencias amigables.

Fase 2: Persistencia de Datos con Backend – Servidor

Esta fase pone en marcha el diseño y la implementación de software desde la fase anterior, centrándose en la programación del sistema de asistencia con QR, desarrollándose la estructura, servidores, desarrollo de múltiples códigos QR. Se cubren los conceptos de sistemas de bases de datos, incluido el diseño lógico, la organización de sistemas de gestión de bases de datos, así como los lenguajes de definición y manipulación de datos SQL y NoSQL.



Fase 3: Consumo de Datos y Desarrollo Frontend – Cliente

Aquí se seleccionan las herramientas de consumo de datos y técnicas más adecuadas para lograr un producto óptimo en términos de software o hardware, de acuerdo con los requisitos funcionales y no funcionales del sistema de asistencia con Qr. En esta fase el diseño visual o gráfico del componente, es un requisito esencial en la capa de presentación del producto.

Alcance del Proyecto componente de mensajería

El sistema a desarrollar permitirá registrar de forma automática y eficiente la asistencia de los estudiantes a sus clases y eventos académicos mediante el uso de códigos QR. Cada aula, laboratorio o espacio designado para eventos contará con un código QR único que estará impreso y colocado de manera visible. Los estudiantes deberán escanear este código QR al ingresar y salir del recinto utilizando una aplicación móvil diseñada específicamente para este fin.

La interfaz será compatible con cualquier dispositivo, y permitirá a los estudiantes escanear los códigos QR de manera rápida y sencilla. Además, contará con funciones de autenticación para garantizar que solo los estudiantes registrados puedan acceder y registrar su asistencia.

Al escanear el código QR al ingresar, el sistema registrará automáticamente la hora y fecha de ingreso del estudiante a esa clase o evento. De igual manera, al escanear el código QR al salir, se registrará la hora y fecha de salida. Todos estos registros de asistencia se almacenarán en una base de datos centralizada y segura.

El sistema contará con una interfaz web accesible para profesores y administradores, donde podrán gestionar cursos, horarios y estudiantes. Además, tendrán la capacidad de generar diversos reportes de asistencia, como por estudiante, curso, fecha, etc. Estos reportes podrán ser exportados en diferentes formatos para su análisis y seguimiento.

La interfaz web también permitirá configurar reglas de asistencia, como tolerancias para retardos, justificaciones de ausencias, entre otros. Adicionalmente, el sistema enviará notificaciones automáticas a los estudiantes y padres en caso de ausencias o retardos, con el fin de mantenerlos informados y promover la asistencia regular.

En cuanto a los requisitos no funcionales, el sistema deberá cumplir con altos estándares de seguridad, con autenticación de usuarios y roles de acceso definidos. Además, deberá ser capaz de



procesar al menos 1,000 escaneos por minuto y soportar un crecimiento de hasta 20,000 estudiantes, garantizando un alto rendimiento y escalabilidad.

El sistema se integrará con el sistema de gestión académica existente de la institución educativa, lo que permitirá importar y sincronizar información de estudiantes, cursos y horarios de manera automática, evitando la necesidad de ingresar estos datos manualmente.

La implementación del proyecto se llevará a cabo en varias fases, incluyendo análisis y diseño detallado, desarrollo e integración, pruebas, y finalmente la implementación y capacitación a usuarios. Se estima un plazo de 6 meses para completar el proyecto.

Después de la implementación, se proporcionará soporte técnico inicial por 3 meses y se realizará mantenimiento y actualizaciones periódicas del sistema para garantizar su correcto funcionamiento y adaptación a nuevos requisitos que puedan surgir. Este alcance cubre los aspectos clave del proyecto y puede ser ajustado y ampliado según las necesidades específicas de la institución educativa y los comentarios de las partes interesadas involucradas.

● Definiciones y acrónimos

Este proyecto utiliza varias definiciones y abreviaturas para comprender la tecnología y el concepto de un sistema de entrada de códigos QR. Aquí están las definiciones y abreviaturas:

1. **Código QR:** Un código QR es un código de barras bidimensional que almacena información en un formato legible por máquina. Normalmente se utiliza para almacenar URL, texto, números de serie y otros tipos de información.
2. **Accesibilidad:** La capacidad de las personas con discapacidad, incluidas las personas con discapacidad visual, para utilizar eficazmente un sistema, producto o entorno.
3. **Interfaz de usuario (UI):** la parte de un sistema o aplicación que permite a los usuarios interactuar con él. Esto incluye elementos visuales como botones, formularios y menús.
4. **Experiencia de usuario (UX):** La experiencia general del usuario al interactuar con un sistema o aplicación, incluidos los aspectos emocionales y prácticos.
5. **Privacidad:** proteja la información personal y permita a los usuarios controlar cómo se usa y comparte su información.
6. **Seguridad:** Proteger la integridad y confidencialidad de los datos y sistemas para evitar el acceso no autorizado o el uso indebido.



2. Descripción General

Objetivos del Sistema

Desarrollar e implementar un sistema de registro de asistencia basado en códigos QR para mejorar la eficiencia y precisión del seguimiento de la asistencia en el entorno del aula de clases. Este sistema tiene como finalidad principal automatizar el proceso de registro de asistencia, garantizando su fiabilidad y reduciendo el tiempo dedicado a esta tarea administrativa. Al implementar este sistema, se busca proporcionar a los docentes y administradores una herramienta eficaz para gestionar la asistencia de los estudiantes en tiempo real, promoviendo así la puntualidad y la asistencia regular..

Funcionalidad General

- Desarrollar e implementar un sistema de registro de asistencia basado en códigos QR para mejorar la eficiencia y precisión del seguimiento de la asistencia en eventos. Este sistema tiene como finalidad principal automatizar el proceso de registro de asistencia, garantizando su fiabilidad y reduciendo el tiempo dedicado a esta tarea administrativa. Al implementar este sistema, se busca proporcionar a los responsables una herramienta eficaz para gestionar la asistencia de los usuarios en tiempo real, promoviendo así la puntualidad y la asistencia regular.
- Capacidad de crear diferentes tipos de eventos.
- Facilitar la creación de códigos qr únicos.
- Posibilitar el escaneo de códigos qr.
- Interactuar con el panel de control.
- Optimización del proceso de registro de asistencia e inasistencia.

Usuarios del Sistema

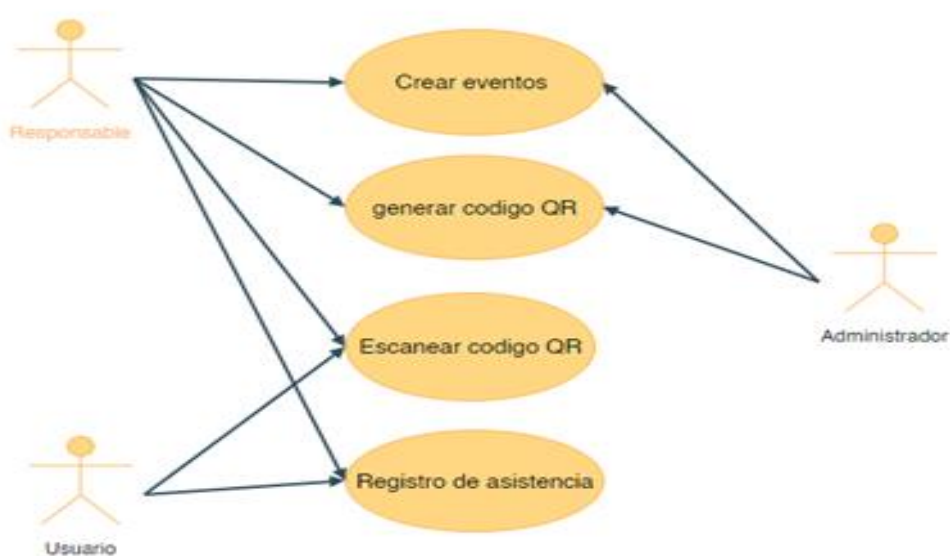
Los siguientes usuarios pueden interactuar con el inicio de sesión mediante QR, dependiendo de las funcionalidades.

Funcionalidad	Administradores	Responsable	Usuario
Crear evento	✓	✓	
Generar Código QR	✓	✓	
Escanear código QR		✓	✓
Registro de asistencia		✓	✓

3. Requisitos Funcionales

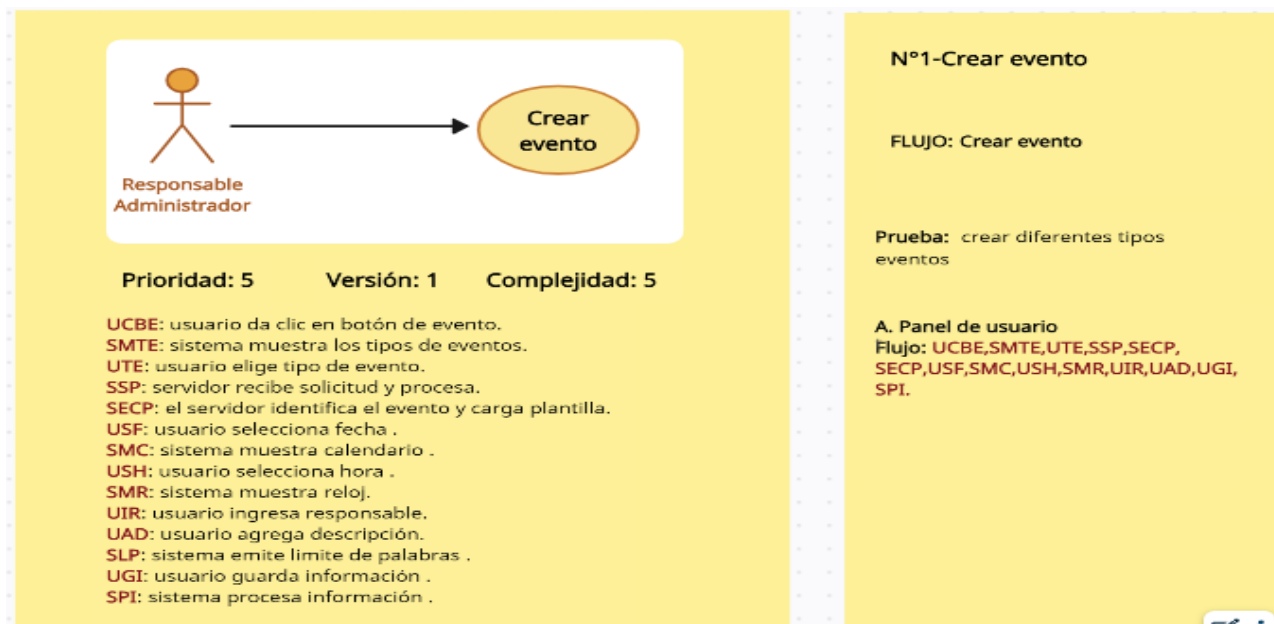
Casos de Uso

Diagrama de caso de uso

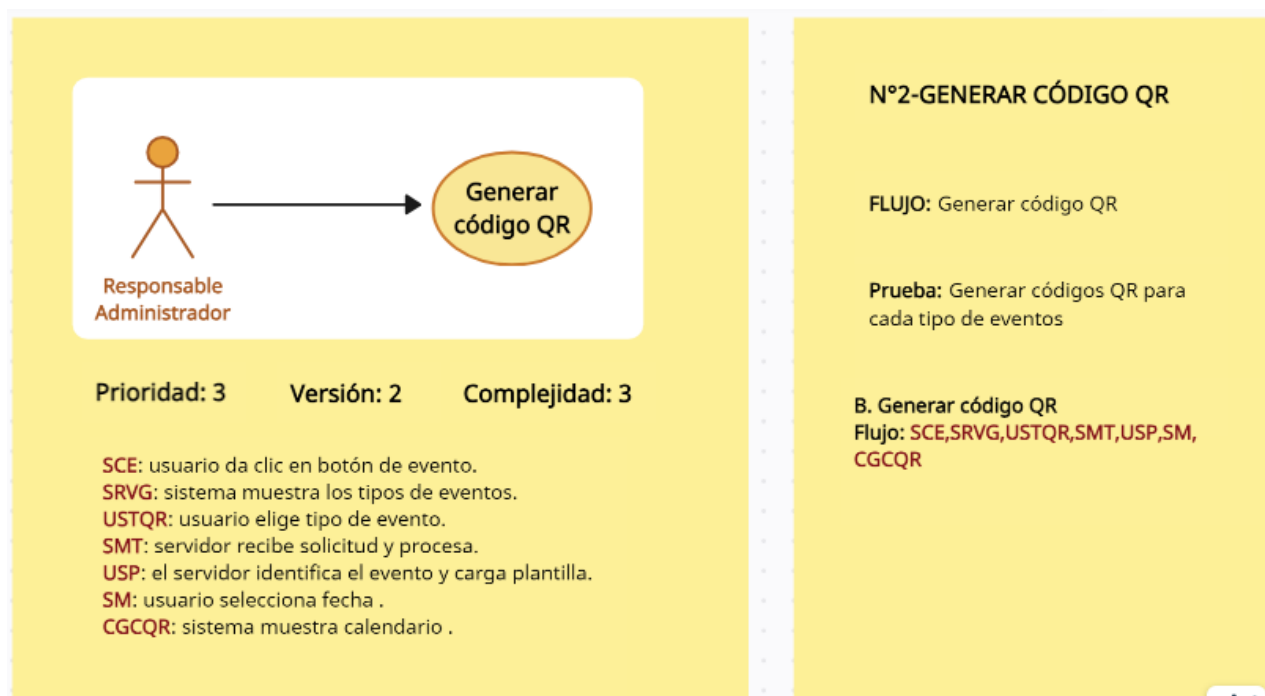


Diagramas de Flujo de Casos de Uso

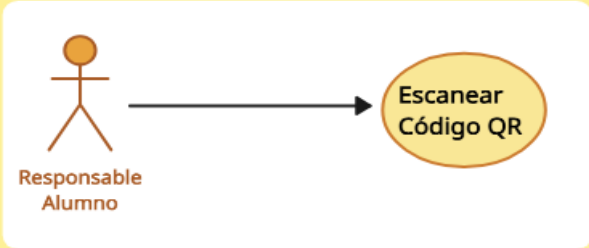
CU-1



CU-2



CU-3



Prioridad: 3 Versión: 1 Complejidad: 3

UECQR: Usuario escanea código QR .
SPC: solicitud de permiso de cámara .
PA: permiso aceptado.
LQR: lectura de QR.
SE: sistema emite enlace .
SCBD: servidor se conecta con la base de datos .
UIP: usuario ingresa a la plataforma.

N°3- ESCANEAR CÓDIGO QR

Flujo: Escanear Código QR

prueba : Escanear diferentes tipos de qr por evento

C. Escanear código QR
Flujo: **UECQR,SPC,PA,LQR,SE,SCBD,UIP**

CU-4



Prioridad: 5 versión: complejidad: 4

EUCQR: El usuario escanea el código QR.
SGI: Sistema guarda información
RSI: Registra su identificación
RA: Registrar asistencia
SCIP: Sistema carga información en pantalla
SDQR: El servidor descifra el código QR y se conecta con la base de datos
VB: Verificación en la base de datos si las credenciales son correctas
SRREI: Sistema recibe la respuesta y envía la información al responsable

N 3- REGISTRO DE ASISTENCIA

Flujo: Registro De Asistencia

PRUEBA:

D
registrar asistencia

Flujo
EUCQR, SGI, RSI, RA, SCIP, SDQR, VB, SRREI



Descripción detallada de cada caso de uso

CASO No. 1 Crear evento

ID:	CU-1	
Nombre	crear evento	
Actores	Administradores, Responsable	
Objetivo	Este caso debe permitir crear diferentes tipos de eventos para tomar su asistencia o solicitud dependiendo el evento.	
Urgencia	5	
Esfuerzo	5	
Pre-condiciones	Debe haberse registrado satisfactoriamente en el sistema	
Flujo Normal	USUARIO	SISTEMA
	El usuario ingresa a la plataforma	
		Muestra interfaz de inicio
	El usuario hace click en el botón evento	



		Despliega los tipos de eventos (cafetería, clase, conferencia.etc)
	El usuario selecciona el tipo de evento	
		El servidor recibe la solicitud y la procesa
		El servidor identifica el tipo de evento seleccionado y carga la interfaz correspondiente a este
	El usuario agrega el título del evento	
	El usuario hace click en la opción fecha	
		El sistema abre el calendario
	El usuario elige la fecha	
	El usuario hace click en la opción hora	
		Sistema abre el apartado de reloj



	Usuario selecciona la hora de inicio y la hora de fin del evento	
	Usuario ingresa los datos del responsable del evento	
	El usuario agrega descripción del evento	
		Sistema emite el límite de palabras permitidas
	Usuario hace clic en guardar	
		Sistema guarda información
		Sistema emite ventana de evento a la interfaz de inicio
Flujo alternativo 1	Si la respuesta indica que el inicio de sesión fue fallido, el navegador web muestra un mensaje de error al usuario.	
	Si el sistema no encuentra en la base de datos el usuario y contraseña proporcionados, se pedirá que ingrese nuevamente o se registre si no lo está.	



Post- condiciones	El usuario ingresa satisfactoriamente porque ya tiene una cuenta registrada	
Excepciones	El usuario no cuenta con Credenciales existentes (debe registrarse)	

CASO No. 2 Generar código QR

ID:	CU-2
Nombre	Generar QR
Actores	Administradores, Responsables
Objetivo	Este caso debe permitir Generar un código QR en un formato compatible (PDF, IMG, GIF, ETC)
Urgencia	3
Esfuerzo	3
Pre-condiciones	- Debe haberse autenticado de forma correcta en el sistema.



	- Ya debe tener la cuenta de CREA VI abierta	
Flujo Normal	USUARIO	SISTEMA
	El usuario se encuentra con su sesión activa y abierta	
	El usuario pulsa “ generar Código QR”	
		El sistema redirige las credenciales del usuario a una extensión mediante una API para generar un código QR que incluya la información de usuario y contraseña con criptografía
		El sistema convierte las credenciales el texto criptográfico
		El sistema procesa las credenciales y redirige la información al CASO No 6
		Retorna al usuario una serie de opciones de descarga para el código QR
	El usuario escoge el formato deseado	



		El sistema prepara y envía el resultado al usuario con el mensaje de “se ha realizado satisfactoriamente”
	El usuario descarga el archivo	
Flujo alternativo 1		El sistema no encuentra credenciales en el sistema por lo que no puede generar un código QR
	El usuario debe tener el rol de docente investigador o alumno para el correcto funcionamiento del código	
post-condiciones		
Excepciones	El usuario no es de los roles permitidos para generar el código QR	



CASO No. 3 Escanear código QR

ID:	CU-3	
Nombre	Escanear código QR	
Actores	asistentes al evento	
Objetivo	Este caso debe permitir la lectura del código QR y posteriormente el ingreso a la plataforma mediante el mismo.	
Urgencia	3	
Esfuerzo	3	
Pre-condiciones	<ul style="list-style-type: none"> - Debió registrarse con el método tradicional en la plataforma - Debe contar con Cámara el cliente 	
Flujo Normal	USUARIO	SISTEMA
	Ingresar el método de inicio de sesión mediante código QR	
		Solicita permiso de cámara
	Acepta el permiso y se procede con la lectura del código QR	



		Se envía la solicitud para descifrar el código QR
		El servidor descifra el código QR y se conecta con la base de datos
		Se verifica en la base de datos si las credenciales son correctas
		El navegador recibe la respuesta y envía la solicitud al navegador
	Si las credenciales son correctas, el usuario ingresa a la plataforma	
Flujo alternativo 1	Ingresa el método de inicio de sesión mediante código QR	
		Sistema no detecta cámara (da mensaje de alarma) informando que no se detecta sistema de cámara
Post-condiciones		
Excepciones	El usuario no posee una cámara. La cámara está rota o defectuosa y no puede iniciar sesión mediante código QR (Se redirige al método tradicional)	



CASO No. 4 Registro de Asistencia

ID:	CU-4	
Nombre	Registro de asistencia	
Actores	Responsable , Usuarios	
Objetivo	permite el registro de asistencia a un evento, mediante el escaneo de un código QR	
Urgencia	3	
Esfuerzo	2	
Pre-condiciones	<p>El asistente debe tener una cuenta registrada en el sistema. -</p> <p>El asistente debe tener acceso a un dispositivo móvil con cámara. -</p> <p>El código QR debe estar visible y en buen estado.</p>	
Flujo Normal	USUARIO	SISTEMA
	El usuario escanea código Qr	
		Sistema redimensiona a panel de usuario
		Sistema guarda información



	Registra su identificación	
	Registrar asistencia	
		Sistema carga información en pantalla
		El servidor descifra el código QR y se conecta con la base de datos
		Se verifica en la base de datos si las credenciales son correctas
		El sistema recibe la respuesta y envía la información al responsable
Flujo alternativo 1	Ingresa el método de inicio de sesión mediante código QR	
		Sistema no detecta cámara (da mensaje de alarma) informando que no se detecta sistema de cámara
Post-condiciones		
Excepciones	El usuario no posee una cámara. La cámara está rota o defectuosa y no puede iniciar sesión mediante	



	código QR (Se redirige al método tradicional)	
--	---	--

Prioridad de Requerimientos

A continuación, se presentan algunos requisitos

Obligatorios:

- Capacidad de crear diferentes tipos de eventos.

Importantes:

Deseables:

A partir del análisis de requerimientos, funcionalidades y el proceso de design thinking, se concreta la siguiente matriz 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					
Esfuerzo		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-4	
	3-Medio	3	6	9	12	15
			CU-5	CU-3 CU-2		
	2-Bajo	2	4	6	8	10
			CU-6			
	1-Muy bajo	1	2	3	4	5



4. Requisitos No Funcionales

Requisitos de Desempeño

- **Tiempo de Escaneo Rápido:** El sistema debe ser capaz de escanear y autenticar el código QR en un tiempo máximo de, por ejemplo, 2 segundos, para proporcionar una experiencia de usuario eficiente.
- **Compatibilidad Multiplataforma:** El sistema debe ser compatible con una variedad de dispositivos y sistemas operativos, incluyendo dispositivos móviles.
- **Tamaño de los Códigos QR:** El sistema debe ser capaz de escanear y procesar códigos QR de diferentes tamaños y resoluciones de manera eficiente.
- **Optimización de recursos:** El sistema debe ser capaz de solicitar al servidor los recursos necesarios para su ejecución, minimizando la exigencia del hardware empleado para su uso.

Requisitos de seguridad:

- **Integridad del código QR:** El componente generará código QR únicos y cifrados para prevenir el acceso no autorizado y posibles falsificaciones.
- **Autenticación segura:** garantizar la validación o autenticidad de los usuarios y prevenir accesos no autorizados, se implementará el uso de pines depurables.
- **Pruebas continuas de seguridad:** se realizarán pruebas de vulnerabilidad mediante falsificaciones de usuario (QR duplicados, similares , entre otros), para identificar posibles puntos débiles , para su corrección oportuna, garantizando la fiabilidad y seguridad del sistema de registro de asistencia con QR.



Requisitos de Usabilidad

- **Visibilidad:** El sistema debe ser totalmente visible para las personas con discapacidad visual. Proporciona soporte para lectores de pantalla para garantizar el cumplimiento de las pautas de contenido en línea.
- **Respuesta del sistema:** Si no se establece ningún proceso, el sistema debe proporcionar comentarios precisos y oportunos a los usuarios después de cada escaneo de código QR.

Requisitos de Escalabilidad

- **La carga del usuario.** El sistema debería poder manejar una cantidad significativa de usuarios simultáneos sin una degradación significativa del rendimiento.
- **Almacenamiento de datos:** el sistema debe ser escalable en términos de almacenamiento de datos. A medida que aumenta el número de usuarios y la cantidad de datos, el sistema de almacenamiento debe escalar de manera eficiente.
- **Disponibilidad y Tolerancia a Fallos:** El sistema debe estar diseñado para garantizar la alta disponibilidad y la tolerancia a fallos. Debe ser capaz de mantener el servicio incluso en caso de fallas de hardware o software.



5. Modelado E/R

A partir de la abstracción de los datos de las funcionalidades los datos preliminares recolectados son:

CU-1 Crear eventos	Fecha 8/04/2027
	Nombre del evento : “ “
	hora del evento : 9:27
	asistencia : 26

CU-2 Generar código QR	Imagen escaneada: [Nombre o ID de la imagen]
-------------------------------	--

CU- 3 Escanear código QR	id valido: “ true”
---------------------------------	--------------------

CU- 4 Registro de Asistencia	id del usuario 94873939
	hora_escáner 9:20 A.M
	Registro de errores : true



Diagrama de Entidad-Relación

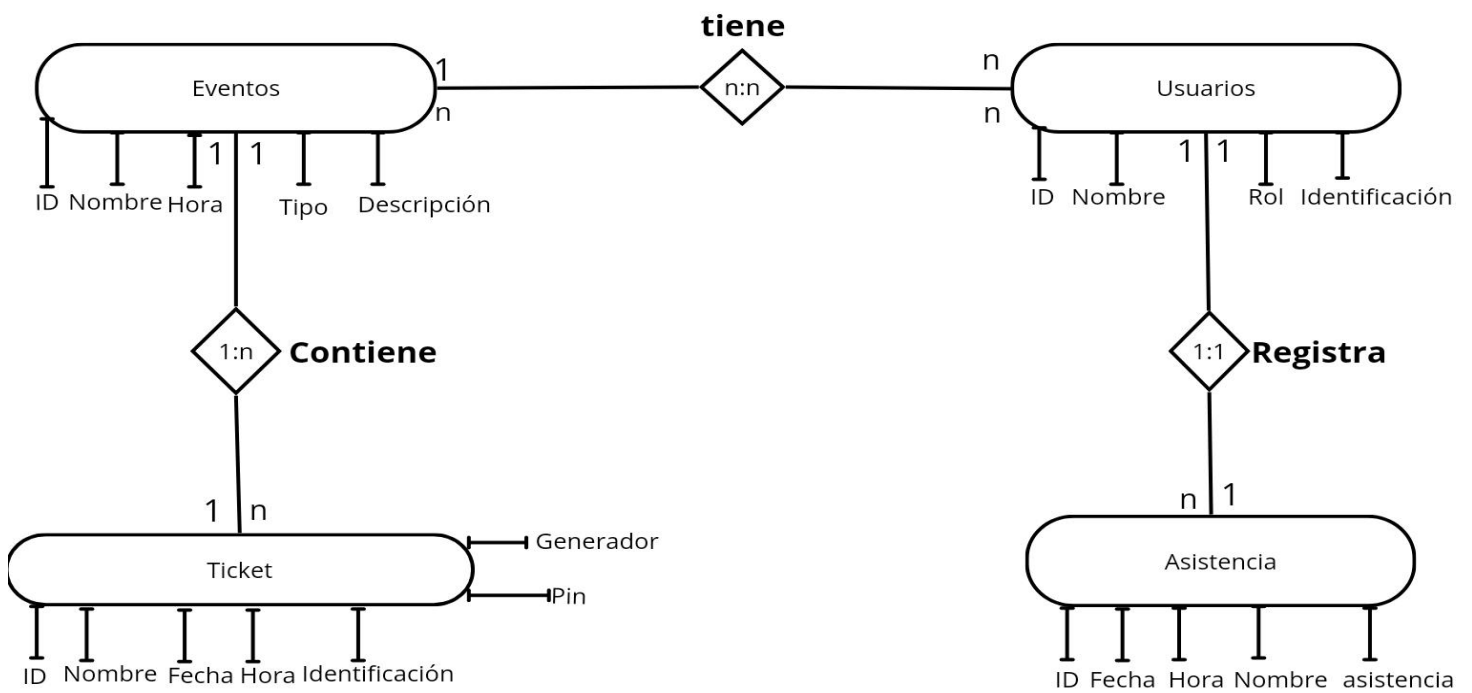
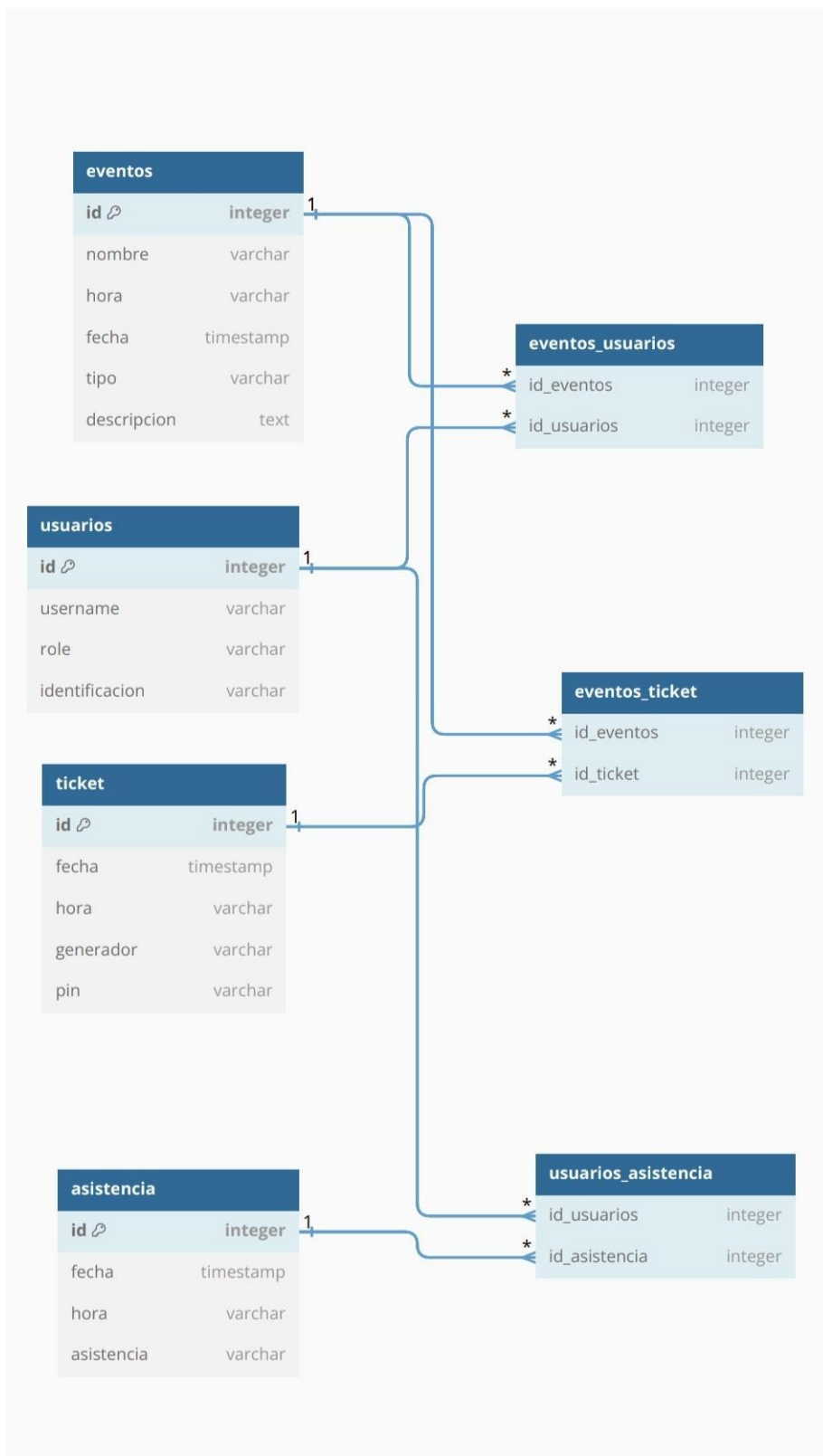


Diagrama Relacional





Script de modelo relacional

<https://dbdiagram.io/d/66196e6a03593b6b61de0b23>

Table eventos {

id integer [primary key]

nombre varchar

hora varchar

fecha timestamp

tipo varchar

descripcion text

}

Table usuarios {

id integer [primary key]

username varchar

role varchar

identificación varchar

}

Table ticket {

id integer [primary key]

fecha timestamp

hora varchar

generador varchar

pin varchar

}

Table asistencia {

id integer [primary key]

fecha timestamp

hora varchar

asistencia varchar

}

Table eventos_usuarios {

id_eventos integer [ref: > eventos.id]

id_usuarios integer [ref: > usuarios.id]

}

Table eventos_ticket {

id_eventos integer [ref: > eventos.id]

id_ticket integer [ref: > ticket.id]

}

Table usuarios_asistencia {

id_usuarios integer [ref: > usuarios.id]

id_asistencia integer [ref: > asistencia.id]

}



Descripción de Entidades y Relaciones

Entidades:

1. **User (Usuario):** Almacena información sobre los usuarios que pueden acceder a la pizarra.

Atributos: ID (identificador único), nombre de usuario, rol (como administrador, docente, estudiante), fecha de creación.

Relaciones: Cada usuario puede estar asociado con varias pizarras a través de la entidad "WhiteboardMember."

2. **Whiteboard (Pizarra):**

Representa una pizarra en la aplicación de pizarra compartida.

Atributos: ID (identificador único), título de la pizarra, descripción, fecha de creación.

Relaciones: Cada pizarra puede contener contenido a través de la relación con la entidad "Content" y puede tener plugins asociados a través de la relación con la entidad "Plugin." 3.

3. **Content (Contenido):**

Almacena contenido que se puede agregar a las pizarras, como texto, imágenes, videos, documentos, etc.

Atributos: ID (identificador único), tipo de contenido, contenido en sí, fecha de creación.

Relaciones: El contenido se asocia con un usuario a través de la relación con la entidad "User" y puede estar relacionado con comentarios.

4. **Comment (Comentario):**

Almacena comentarios realizados por los usuarios en relación con el contenido de la pizarra.

Atributos: ID (identificador único), texto del comentario, fecha de creación. **Relaciones:** Cada comentario se relaciona con el contenido específico en la entidad "Content."

5. **RevisionHistory (Historial de Revisiones):**

Registra el historial de revisiones y cambios realizados en las pizarras.

Atributos: ID (identificador único), fecha de la revisión.

Relaciones: Cada entrada de historial se relaciona con una pizarra específica en la entidad "Whiteboard."

6. **MultimediaResource (Recurso Multimedia):**



Almacena recursos multimedia, como imágenes, videos, documentos, etc. **Atributos:** ID (identificador único), tipo de recurso, ubicación o URL del recurso, título, descripción, fecha de carga.

Relaciones: Cada entrada de Recurso multimedia se relaciona con una un contenido específico en la entidad " Content."

7. Plugin:

Almacena información sobre los plugins que pueden proporcionar funcionalidades personalizadas en las pizarras.

Atributos: ID (identificador único), nombre del plugin, descripción, autor, versión, configuración.

Relaciones: Cada pizarra puede tener asociado uno o varios plugins a través de la entidad "Whiteboard."

Relaciones:

- "User" se relaciona con "WhiteboardMember" para indicar la asociación de los usuarios con las pizarras.
- "Content" se relaciona con "User" para registrar los usuarios que pueden interactuar con el contenido.
- "Content" se relaciona con "Comment" para permitir comentarios en los contenidos. - "Whiteboard" se relaciona con "Content" para indicar que una pizarra puede contener contenido.
- "Whiteboard" se relaciona con "Plugin" para permitir la asociación de plugins con las pizarras.
- "WhiteboardMember" se relaciona con "User" y "Whiteboard" para indicar la asociación de usuarios con pizarras y sus roles.
- "RevisionHistory" se relaciona con "Whiteboard" para registrar revisiones en las pizarras.
- " MultimediaResource " se relaciona con "Content" para registrar recursos asociados en el contenido.



Reglas de Integridad Referencial

1. **Integridad Referencial entre "User_Content" y "Users"**: Cada registro en la tabla "User_Content" debe estar asociado con un usuario existente en la tabla "Users" a través de la clave foránea "user_id."
2. **Integridad Referencial entre "User_Content" y "Contents"**: Cada registro en la tabla "User_Content" debe estar asociado con un contenido existente en la tabla "Contents" a través de la clave foránea "content_id."
3. **Integridad Referencial entre "Content_Comment" y "Contents"**: Cada registro en la tabla "Content_Comment" debe estar asociado con un contenido existente en la tabla "Contents" a través de la clave foránea "content_id."
4. **Integridad Referencial entre "Content_Comment" y "Comments"**: Cada registro en la tabla "Content_Comment" debe estar asociado con un comentario existente en la tabla "Comments" a través de la clave foránea "comment_id."
5. **Integridad Referencial entre "Whiteboards_WhiteboardMembers" y "Whiteboards"**: Cada registro en la tabla "Whiteboards_WhiteboardMembers" debe estar asociado con una pizarra existente en la tabla "Whiteboards" a través de la clave foránea "Whiteboards_id."
6. **Integridad Referencial entre "Whiteboards_WhiteboardMembers" y "WhiteboardMembers"**: Cada registro en la tabla "Whiteboards_WhiteboardMembers" debe estar asociado con un miembro de la pizarra existente en la tabla "WhiteboardMembers" a través de la clave foránea "whiteboardMembers_id."
7. **Integridad Referencial entre "Whiteboard_Plugin" y "Whiteboards"**: Cada registro en la tabla "Whiteboard_Plugin" debe estar asociado con una pizarra existente en la tabla "Whiteboards" a través de la clave foránea "whiteboard_id."
8. **Integridad Referencial entre "Whiteboard_Plugin" y "Plugins"**: Cada registro en la tabla "Whiteboard_Plugin" debe estar asociado con un plugin existente en la tabla "Plugins" a través de la clave foránea "plugin_id."
9. **Integridad Referencial entre "Contents_Whiteboards" y "Contents"**: Cada registro en la tabla "Contents_Whiteboards" debe estar asociado con un contenido existente en la tabla "Contents" a través de la clave foránea "Contents_id."
10. **Integridad Referencial entre "Contents_Whiteboards" y "Whiteboards"**: Cada registro en la tabla "Contents_Whiteboards" debe estar asociado con una pizarra existente en la tabla



"Whiteboards" a través de la clave foránea "whiteboard_id."

11. **Integridad Referencial entre "Content_MultimediaResource" y "Contents"**: Cada registro en la tabla "Content_MultimediaResource" debe estar asociado con un contenido existente en la tabla "Contents" a través de la clave foránea "content_id."
12. **Integridad Referencial entre "Content_MultimediaResource" y "MultimediaResources"**: Cada registro en la tabla "Content_MultimediaResource" debe estar asociado con un recurso multimedia existente en la tabla "MultimediaResources" a través de la clave foránea "MultimediaResources_id."
13. **Integridad Referencial entre "RevisionHistorys" y "Whiteboards"**: Cada registro en la tabla "RevisionHistorys" debe estar asociado con una pizarra existente en la tabla "Whiteboards" a través de la clave foránea "whiteboard_id."
14. **Integridad Referencial entre "WhiteboardMembers" y "Users"**: Cada registro en la tabla "WhiteboardMembers" debe estar asociado con un usuario existente en la tabla "Users" a través de la clave foránea "user_id."



Colecciones (NoSLQ) }

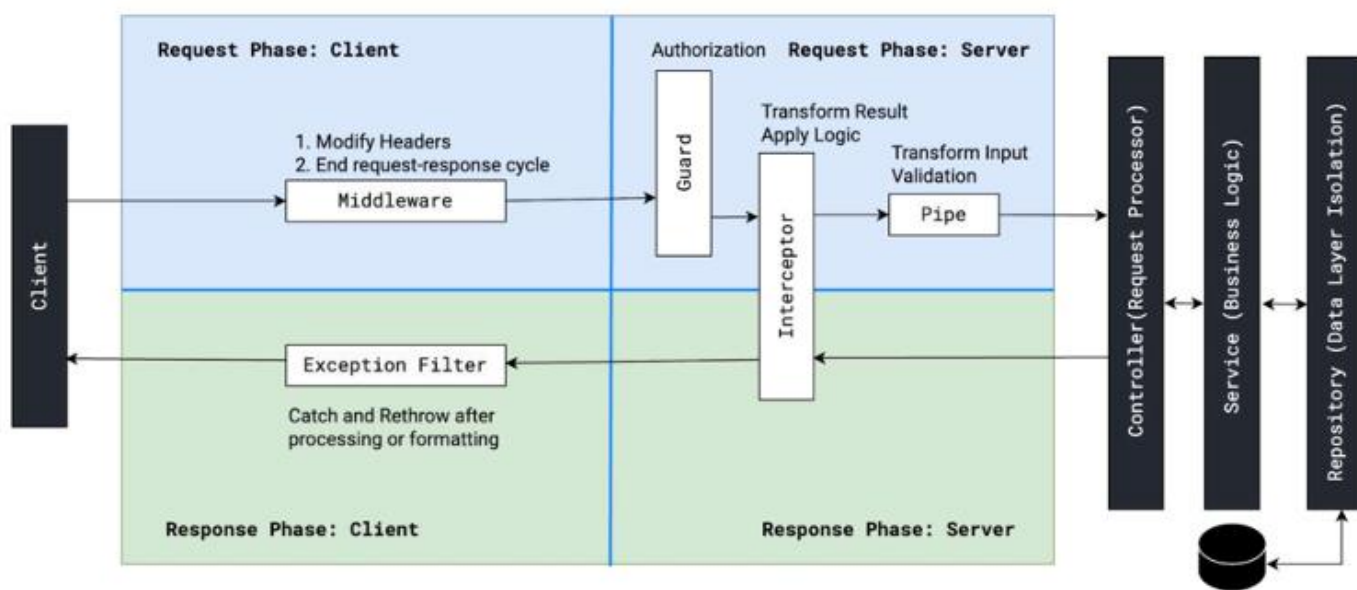
Eventos: {	Ticket{
Id: ObjectId,	Id: ObjectId,
Nombre: String,	Fecha: Date,
Hora: Date,	Hora: Date,
Fecha: Date,	Generador: String,
Tipo: String,	Pin: String,
Descripcion: String,	Id_Eventos: [ObjectId]
Id_Usuarios:[ObjectId],	}
Id_Ticket:[ObjectId]	
}	Asistencia: {
	Id: ObjectId,
Usuarios: {	Fecha: Date,
Id: ObjectId,	Hora: Date,
Username: String,	Asistencia: String,
Role: String,	Id_Usuarios: [ObjectId]
Identificacion: String,	}
Id_Asistencia: [ObjectId],	
Id_Eventos: [ObjectId],	

Etapa 2: Persistencia de Datos con Backend

7. Descripción de la Arquitectura Propuesta

Este proyecto está basado en un sistema de Nest.js que cuenta con múltiples elementos, incluyendo el cliente, el interceptor que proporciona una validación al controlador, y una interacción fluida entre el controlador, el servidor y el repositorio, así como con la base de datos. El diseño de la arquitectura permite que cada capa del sistema se comuniquen de manera eficiente, asegurando que los datos se procesen correctamente antes de ser almacenados o enviados al usuario.

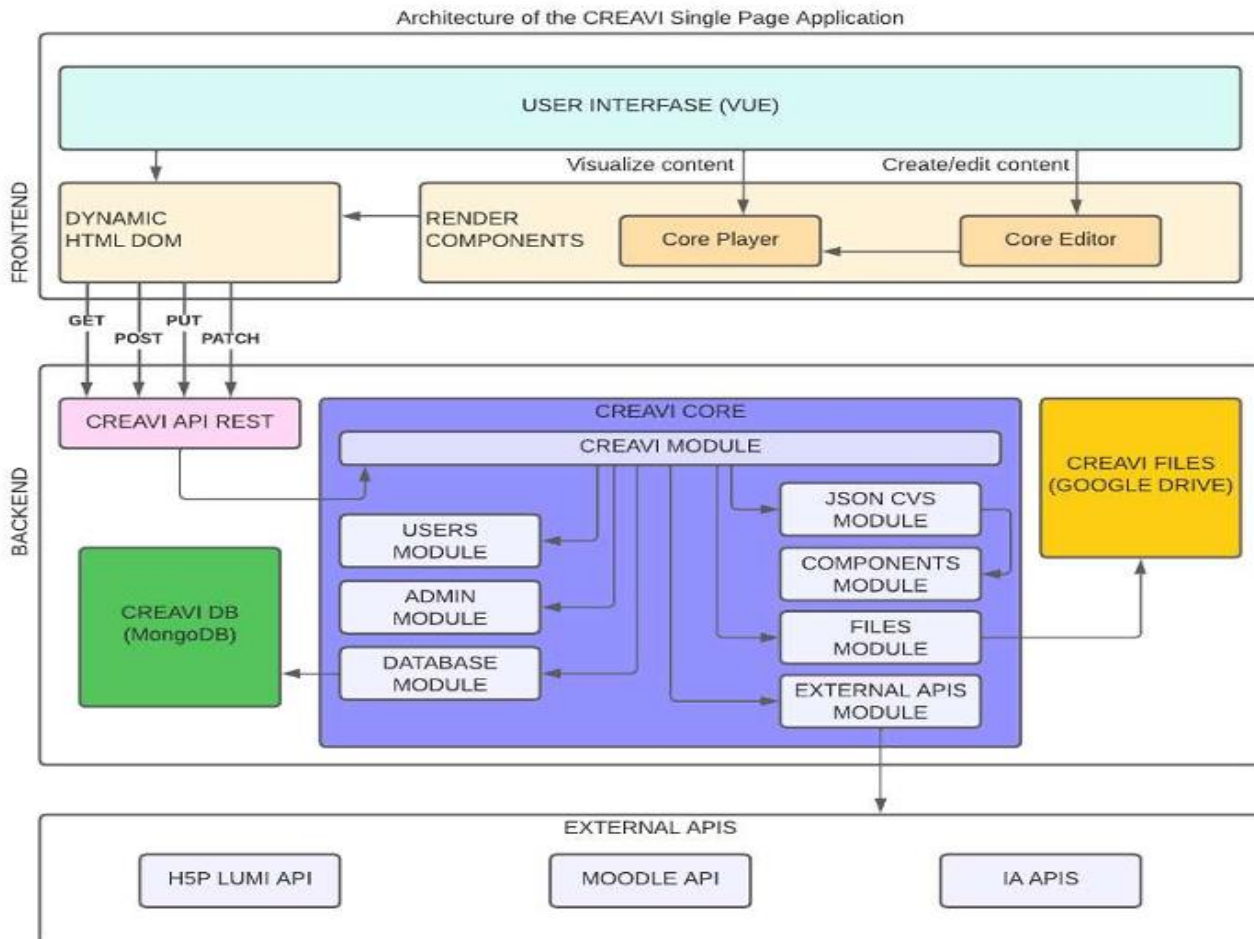
8. Diseño de la Arquitectura de Backend



Este proceso garantiza que las solicitudes del cliente se gestionen de forma organizada y segura, optimizando la comunicación entre el cliente y el servidor. Cada elemento cumple una función específica para asegurar un desempeño eficiente y confiable de la aplicación.

Componentes del Backend

Diagramas de Arquitectura

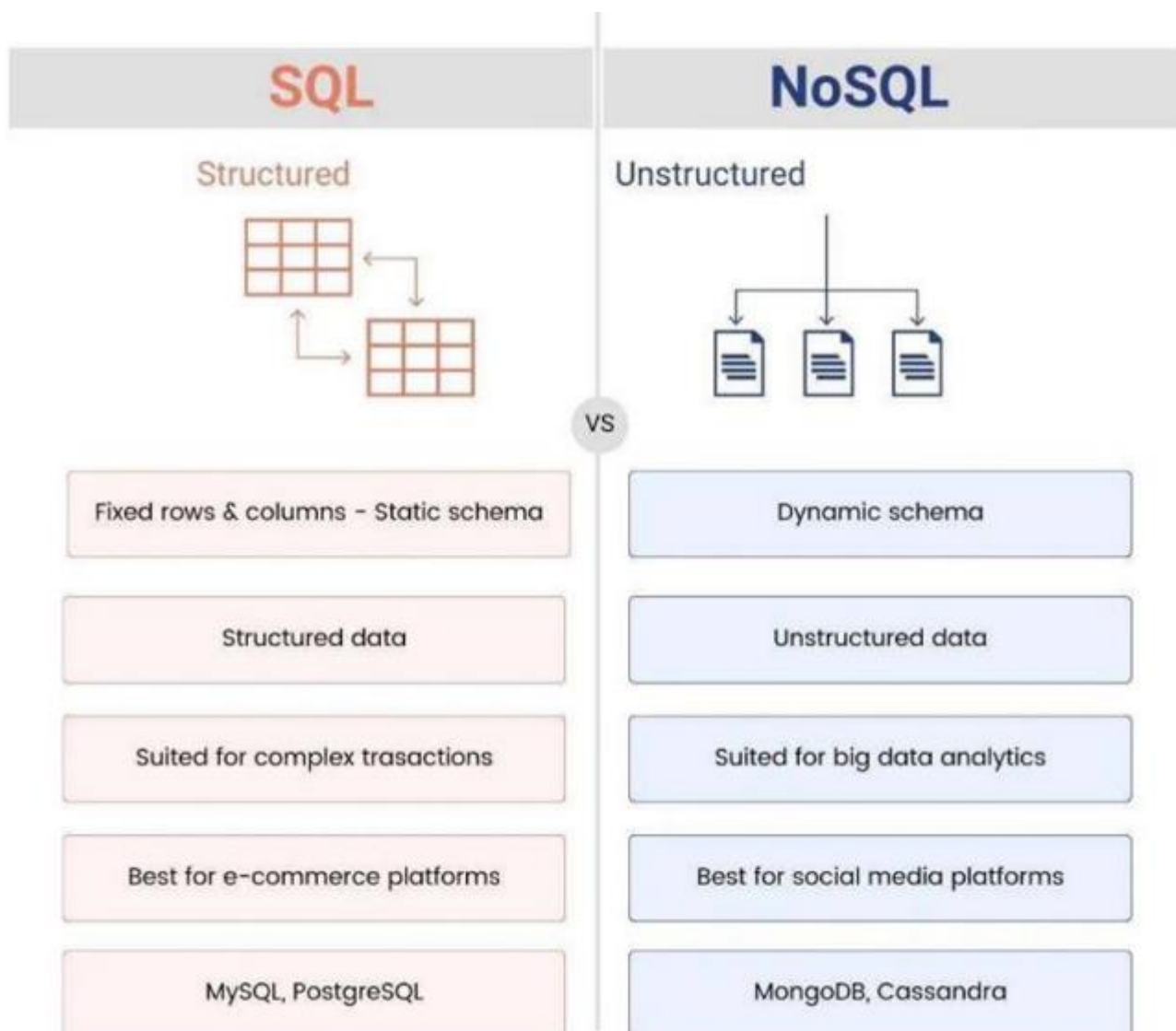


La imagen muestra la arquitectura de una aplicación de una sola página (SPA) llamada CREAVI, diseñada para ofrecer funcionalidades de edición y visualización de contenido. El **frontend**, desarrollado con Vue.js, permite a los usuarios interactuar a través de un editor y un reproductor de contenido dinámico, comunicándose con el backend mediante API REST. El **backend** incluye un núcleo modular (CREAVI Core) con componentes para la gestión de usuarios, administración, archivos y conexión con APIs externas. Los datos se almacenan en una base de datos MongoDB y los archivos se gestionan a través de Google Drive. Además, la arquitectura integra **APIs externas** como H5P Lumi, Moodle e inteligencia artificial, permitiendo funcionalidades avanzadas y adaptables para entornos educativos y colaborativos.

9. Elección de la Base de Datos

Una de las decisiones más importantes para el desarrollo del backend de un proyecto es la elección de la base de datos. Entre las opciones más populares se encuentran SQL y NoSQL, y la selección dependerá de las necesidades específicas del proyecto.

Evaluación de Opciones (SQL o NoSQL)

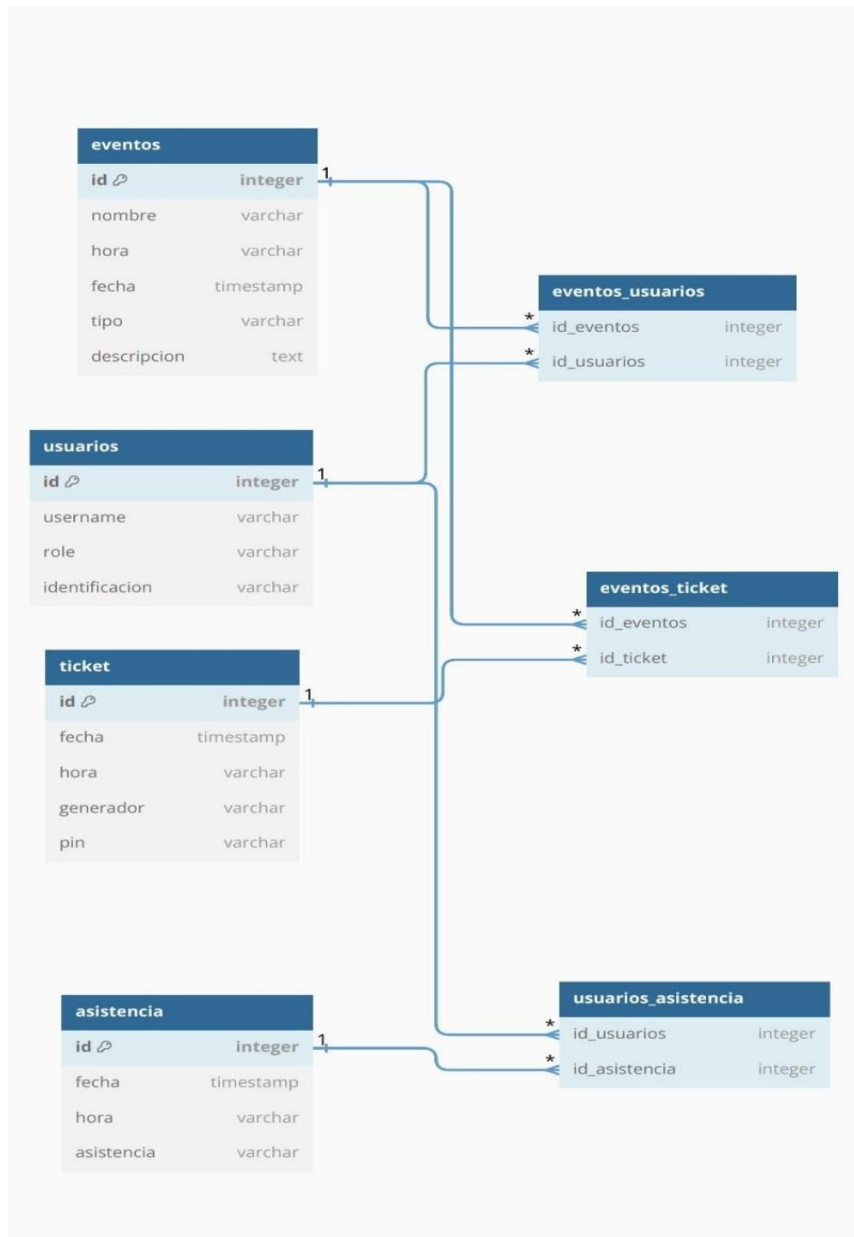




Justificación de la Elección

Se utiliza NoSQL en software de asistencia a eventos principalmente por su escalabilidad, flexibilidad y capacidad para manejar grandes volúmenes de datos en tiempo real. Las bases de datos NoSQL permiten almacenar información diversa y no estructurada sobre los asistentes, actividades y eventos, sin necesidad de un esquema rígido, lo que facilita la integración de múltiples fuentes de datos y la adaptación a cambios rápidos. Además, ofrecen alta disponibilidad y tolerancia a fallos, lo que asegura el buen funcionamiento del sistema incluso durante eventos grandes, garantizando rendimiento en tiempo real para el registro de entradas, actualizaciones de asistencia y visualización de estadísticas en vivo.

Diseño de Esquema de Base de Datos



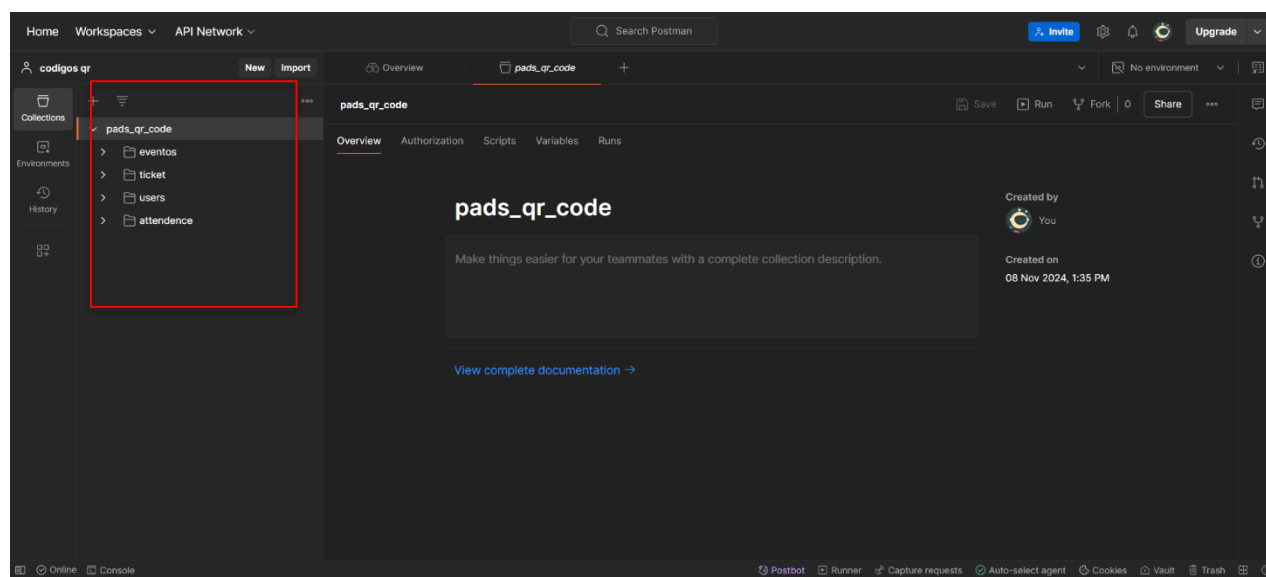
El sistema de gestión organiza toda su información en cuatro colecciones principales: Eventos, que incluye datos como el nombre, hora, fecha, tipo y descripción, permitiendo detallar las actividades planificadas; Usuarios, donde se registran el nombre, rol e identificación de cada persona involucrada, facilitando su identificación y categorización; Tickets, que almacenan información como fecha, hora, generador y pin único, utilizados para crear códigos QR personalizados para los creadores de eventos, mejorando la organización y el acceso; y Asistencias, que registra la fecha, hora y estado de asistencia, proporcionando un control claro sobre los participantes en cada evento. Estas colecciones trabajan en conjunto para garantizar un manejo eficiente de la información y una experiencia optimizada para los usuarios.

10. Implementación del Backend

Elección del Lenguaje de Programación

Para el desarrollo del backend se eligió TypeScript como lenguaje de programación, aprovechando la estructura y la robustez que ofrece en proyectos complejos. Esta elección está fundamentada en la capacidad de TypeScript para proporcionar tipado estático y un desarrollo más estructurado y escalable en aplicaciones de gran envergadura. TypeScript es compatible con Nestjs, el framework de Node.js seleccionado, que permite la construcción de aplicaciones modulares, fácilmente testeables y mantenibles. NestJS se destaca en el desarrollo de Apis rest y aplicaciones orientadas a microservicios, lo que facilita la integración y escalabilidad del sistema.

Desarrollo de Endpoints y APIs



En la anterior imagen se muestra la estructura de una colección llamada pads_qr_code en Postman. Esta colección organiza varios endpoints para manejar funciones específicas dentro de una API. Entre los elementos destacados están las carpetas eventos, ticket, users y attendance, que probablemente corresponden a diferentes recursos o módulos de la aplicación. La estructura facilita la administración y prueba de cada endpoint, lo que es esencial en el desarrollo y mantenimiento de APIs.



Home Workspaces API Network Search Postman

codigos qr New Import

Overview pads_qr_code

Save Run Fork 0 Share

Overview Authorization Scripts Variables Runs

pads_qr_code

Make things easier for your teammates with a complete collection description.

Created by You

Created on 08 Nov 2024, 1:35 PM

[View complete documentation →](#)

Postbot Runner Capture requests Auto-select agent Cookies Vault Trash

Home Workspaces API Network Search Postman

codigos qr New Import

Overview pads_qr_code

Save Run Fork 0 Share

Overview Authorization Scripts Variables Runs

pads_qr_code

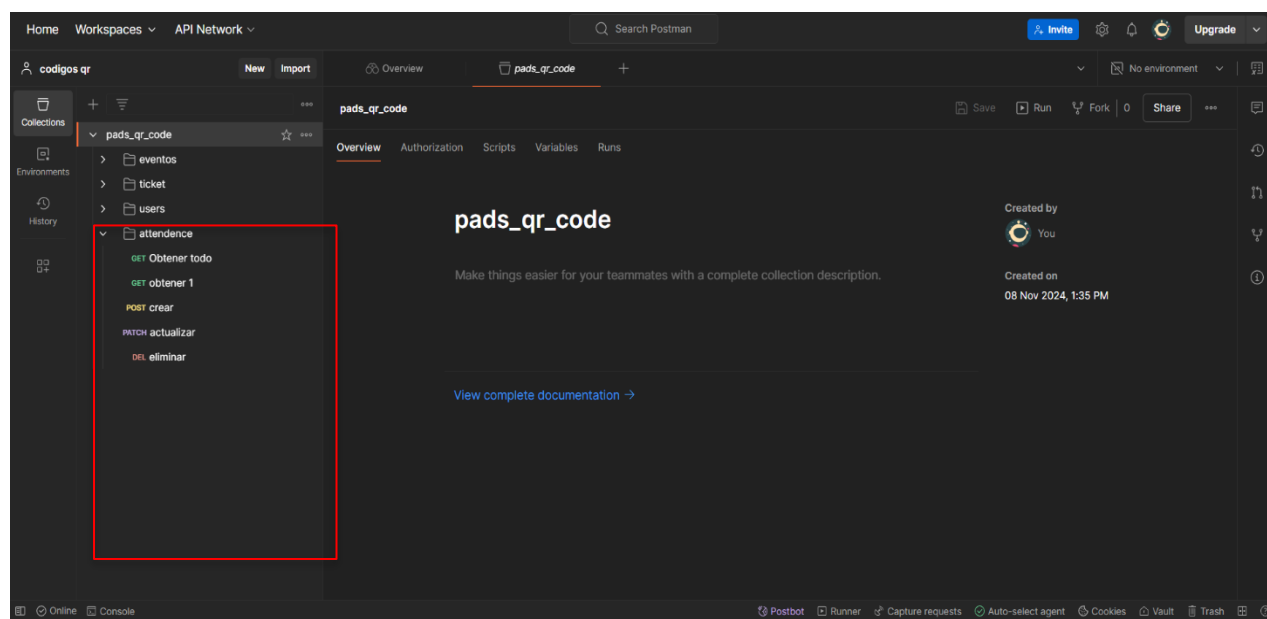
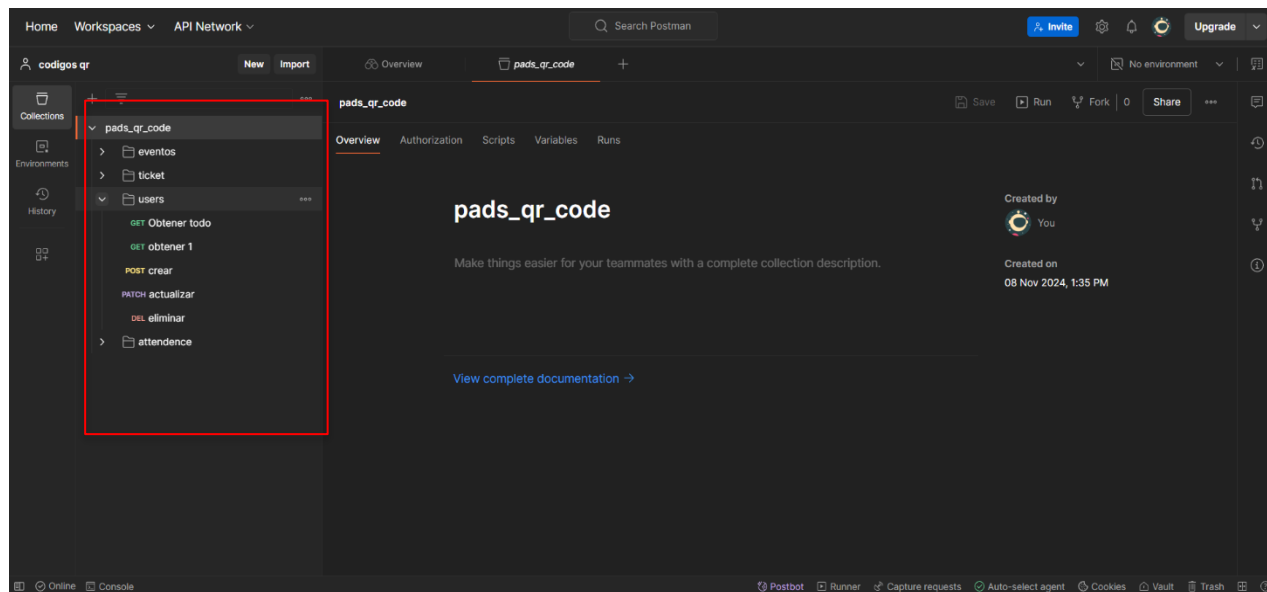
Make things easier for your teammates with a complete collection description.

Created by You

Created on 08 Nov 2024, 1:35 PM

[View complete documentation →](#)

Postbot Runner Capture requests Auto-select agent Cookies Vault Trash



En las captura, se observan con mayor detalle los métodos asociados a la colección pads_qr_code. Cada recurso incluye solicitudes específicas como GET, POST, y DELETE, con nombres como "Obtener todo", "crear", "eliminar", y "obtener 1". Las solicitudes permiten operaciones como la creación, recuperación, actualización y eliminación de registros asociados a los módulos de eventos, tickets, usuarios y asistencia.



Autenticación y Autorización

11. Conexión a la Base de Datos

Configuración de la Conexión

La conexión a la base de datos MongoDB Atlas desde el componente se realiza mediante las credenciales de acceso, la URL de conexión, y una configuración específica para asegurar el sistema. Al trabajar con MongoDB, es necesario instalar las dependencias correspondientes ejecutando: `npm install @nestjs/mongoose mongoose`.

```
src > TS app.module.ts > ...
1  import { Module } from '@nestjs/common';
2  import { AppController } from './app.controller';
3
4  import { ConfigModule } from '@nestjs/config';
5  import { ServeStaticModule } from '@nestjs/serve-static';
6  import { MongooseModule } from '@nestjs/mongoose';
7  import { join } from 'path';
8
9  import { AppService } from './app.service';
10 import { EventsModule } from './events/events.module';
11 import { UsersModule } from './users/users.module';
12 import { TicketModule } from './ticket/ticket.module';
13 // import { LogsModule } from './logs/logs.module';
14 import { AttendanceController } from './attendance/attendance.controller';
15 import { AttendanceModule } from './attendance/attendance.module';
16
```

```
17 @Module({
18   imports: [
19     ConfigModule.forRoot({ envFilePath: '.env', isGlobal: true }),
20     ServeStaticModule.forRoot({ rootPath: join(__dirname, '..', 'client') }),
21     // MongooseModule.forRoot(process.env.DB_URI),
22     MongooseModule.forRoot(
23       'mongodb+srv://aagamezibarra69:GFJuOQ7gUB6duFZI@cluster0.myaid.mongodb.net/?retryWrites=true&w=majority&appName=Cluster'
24     ),
25     EventsModule,
26     UsersModule,
27     TicketModule,
28     AttendanceModule,
29     // LogsModule
30   ],
31   controllers: [AppController],
32   providers: [AppService],
33 })
34 export class AppModule {}
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN **TERMINAL** PUERTOS COMENTARIOS

@Ldavid182 →/workspaces/pads_qr_code (main) \$



Luego, se realizó la configuración en el App Module. Aquí, el módulo MongooseModule se configuró para conectarse a la base de datos MongoDB usando la URI proporcionada, la cual incluye tanto el usuario y la contraseña como el nombre de la base de datos TaskManager_db. Esto permite el acceso seguro a la base de datos alojada en la nube.

Desarrollo de Operaciones CRUD

Las operaciones CRUD (Crear, Obtener todo, Obtener 1, Actualizar, Eliminar) son fundamentales para el funcionamiento óptimo del componente y permiten la interacción con las colecciones en MongoDB. A continuación, se detallan las implementaciones de estas operaciones para la colección de Acciones.

```
src > attendance > TS attendance.service.ts > ...
1  import { Injectable } from '@nestjs/common';
2  import { CreateAttendanceDto } from '../dto/create-attendance.dto';
3  import { UpdateAttendanceDto } from '../dto/update-attendance.dto';
4  import { InjectModel } from '@nestjs/mongoose';
5  import { Model } from 'mongoose';
6  import { Attendance } from '../schemas/attendance.schema';
7
8
9  @Injectable()
10 export class AttendanceService {
11   constructor(
12     @InjectModel(Attendance.name) private attendanceModel: Model<Attendance>,
13   ) {}
14
15   create(createAttendanceDto: CreateAttendanceDto) {
16     const createAttendance = new this.attendanceModel(createAttendanceDto);
17     return createAttendance.save();
18   }
19
20   findAll() {
21     return this.attendanceModel.find().exec();
22   }
23
24   findOne(id: string) {
25     return this.attendanceModel.findById(id).exec();
26   }
27
28   update(id: string, UpdateAttendanceDto: UpdateAttendanceDto) {
29     return this.attendanceModel
30       .findByIdAndUpdate(id, UpdateAttendanceDto, {
31         new: true,
32       })
33       .exec();
34   }
35
36   remove(id: string) {
37     return this.attendanceModel.findByIdAndDelete(id).exec();
38   }
39 }
40
41
```

Se define una clase AttendanceService con métodos para crear, obtener todos, obtener uno, actualizar y eliminar registros, cada uno implementado siguiendo las prácticas de inyección de dependencias. Los métodos utilizan DTOs (Data Transfer Objects) para validar y estructurar los



datos de entrada, garantizando que la lógica de negocio esté encapsulada y bien organizada. Esta estructura facilita la interacción eficiente con la base de datos y asegura un diseño modular y escalable del backend.

```
src > events > TS events.service.ts > ...
1  import { Injectable } from '@nestjs/common';
2  import { CreateEventDto } from '../dto/create-event.dto';
3  import { UpdateEventDto } from '../dto/update-event.dto';
4  import { InjectModel } from '@nestjs/mongoose';
5  import { Model } from 'mongoose';
6  import { Event } from '../schemas/Event.schema';
7  import { User } from 'src/users/entities/user.entity';
8
9  @Injectable()
10 export class EventsService {
11   constructor(@InjectModel(Event.name) private eventModel: Model<Event>) {}
12
13   create(CreateEventDto: CreateEventDto) {
14     const createdEvent = new this.eventModel(CreateEventDto);
15     return createdEvent.save();
16   }
17
18   findAll() {
19     return this.eventModel.find().exec();
20   }
21
22   findOne(id: string) {
23     return this.eventModel.findById(id).exec();
24   }
25
26   update(id: string, UpdateEventDto: UpdateEventDto) {
27     return this.eventModel
28       .findByIdAndUpdate(id, UpdateEventDto, {
29         new: true,
30       })
31       .exec();
32   }
33
34   remove(id: string) {
35     return this.eventModel.findByIdAndDelete(id).exec();
36   }
37 }
38
```

La clase EventsService, decorada con @Injectable(), incluye métodos para crear, obtener todos, obtener uno, actualizar y eliminar eventos. Se inyecta el modelo de eventos (eventModel) para interactuar con la base de datos, utilizando DTOs (Data Transfer Objects) que validan la estructura de los datos de entrada. Esta arquitectura modular facilita la manipulación de registros, asegura un diseño escalable y mantiene la lógica del negocio bien organizada, siguiendo las mejores prácticas en desarrollo backend.

```
src > users > TS users.service.ts > ...
1  import { Injectable } from '@nestjs/common';
2  import { CreateUserDto } from '../dto/create-user.dto';
3  import { UpdateUserDto } from '../dto/update-user.dto';
4  import { InjectModel } from '@nestjs/mongoose';
5  import { Model } from 'mongoose';
6  import { Users } from '../schemas/users.schemas'; // Importando la clase Users correctamente
7
8  @Injectable()
9  export class UsersService {
10     constructor(@InjectModel(Users.name) private usersModel: Model<Users>) {}
11
12     create(CreateUsersDto: CreateUserDto) {
13         const createdUsers = new this.usersModel(CreateUsersDto);
14         return createdUsers.save();
15     }
16
17     findAll() {
18         return this.usersModel.find().exec();
19     }
20
21     findOne(id: string) {
22         return this.usersModel.findById(id).exec();
23     }
24
25     update(id: string, UpdateUsersDto: UpdateUserDto) {
26         return this.usersModel
27             .findByIdAndUpdate(id, UpdateUsersDto, {
28                 new: true,
29             })
30             .exec();
31     }
32
33     remove(id: string) {
34         return this.usersModel.findByIdAndDelete(id).exec();
35     }
36 }
37
```

La imagen presenta el código del servicio TicketService, implementado en NestJS, que gestiona las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) relacionadas con los tickets. Utiliza el decorador Injectable para permitir la inyección de dependencias y se apoya en los Data Transfer Objects (DTO) CreateTicketDto y UpdateTicketDto para definir la estructura de los datos. A través de la inyección del modelo Ticket, el servicio puede crear nuevos tickets, recuperar todos o uno específico, actualizar información existente y eliminar tickets de la base de datos MongoDB, garantizando una gestión eficiente y estructurada de los tickets.

```
src > ticket > TS ticket.service.ts > ...
1  import { Injectable } from '@nestjs/common';
2  import { CreateTicketDto } from '../dto/create-ticket.dto';
3  import { UpdateTicketDto } from '../dto/update-ticket.dto';
4  import { InjectModel } from '@nestjs/mongoose';
5  import { Model } from 'mongoose';
6  import { Ticket } from '../schemas/ticket.schemas';
7
8  @Injectable()
9  export class TicketService {
10   constructor(@InjectModel(Ticket.name) private ticketModel: Model<Ticket>) {}
11
12   create(CreateTicketDto: CreateTicketDto) {
13     const createdTicket = new this.ticketModel(CreateTicketDto);
14     return createdTicket.save();
15   }
16
17   findAll() {
18     return this.ticketModel.find().exec();
19   }
20
21   findOne(id: string) {
22     return this.ticketModel.findById(id).exec();
23   }
24
25   update(id: string, UpdateTicketDto: UpdateTicketDto) {
26     return this.ticketModel
27       .findByIdAndUpdate(id, UpdateTicketDto, {
28         new: true,
29       })
30       .exec();
31   }
32
33   remove(id: string) {
34     return this.ticketModel.findByIdAndDelete(id).exec();
35   }
36 }
37
```

La imagen muestra el código del servicio UsersService, desarrollado con NestJS, que se encarga de gestionar las operaciones CRUD relacionadas con los usuarios. Al igual que el servicio de tickets, utiliza el decorador Injectable y los Data Transfer Objects (DTO) CreateUserDto y UpdateUserDto para manejar la creación y actualización de usuarios. La inyección del modelo Users permite al servicio crear nuevos usuarios, obtener todos o uno específico, actualizar la información existente y eliminar usuarios de la base de datos MongoDB, asegurando una administración efectiva y adecuada de los usuarios en la aplicación.



12. Pruebas del Backend

Diseño de Casos de Prueba

Ejecución de Pruebas Unitarias y de Integración Manejo de Errores y Excepciones

Etapas 3: Consumo de Datos y Desarrollo Frontend

13. Introducción

Propósito de la Etapa

Alcance de la Etapa

Definiciones y Acrónimos

14. Creación de la Interfaz de Usuario (UI) Diseño de la Interfaz de Usuario (UI) con HTML y CSS Consideraciones de Usabilidad

Maquetación Responsiva

15. Programación Frontend con JavaScript (JS) Desarrollo de la Lógica del Frontend

Manejo de Eventos y Comportamientos Dinámicos

Uso de Bibliotecas y Frameworks (si aplicable)

16. Consumo de Datos desde el Backend

Configuración de Conexiones al Backend

Obtención y Presentación de Datos

Actualización en Tiempo Real (si aplicable)

17. Interacción Usuario-Interfaz

Manejo de Formularios y Validación de Datos

Implementación de Funcionalidades Interactivas

Mejoras en la Experiencia del Usuario

18. Pruebas y Depuración del Frontend Diseño de Casos de



**UNIVERSIDAD DE
CÓRDOBA**



LICENCIATURA EN
INFORMÁTICA

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

Prueba de Frontend

Pruebas de Usabilidad

Depuración de Errores y Optimización del Código

19. 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

20. 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.