19-9-2024

Thiago Lionel Vergara

7MO 4ta

**Documentación de la API del Sistema “Asistencia QR”**

Automatización del registro de asistencia de profesores mediante códigos QR

**Índice**

1. **Introducción**
   * Propósito del documento
   * Descripción general de la API
   * Descripción del sistema "Asistencia QR"
2. **Instalación y Configuración**
   * Requisitos previos
   * Instalación de dependencias
   * Configuración de variables de entorno
3. **Estructura del Proyecto**
4. **Modelos de Base de Datos**
   * Profesor
   * Asistencia
   * Curso
   * Materia
   * Horario
5. **Rutas de la API**
   * Profesores
   * Asistencias
   * Login
   * Cursos
   * Horarios
   * Materias
6. **Endpoints Detallados**
   * Endpoints de Profesores
   * Endpoints de Asistencias
   * Endpoints de Cursos
   * Endpoints de Horarios
   * Endpoints de Materias
   * Endpoint de Login
7. **Conexión a la Base de Datos**
8. **Dependencias**
9. **Consideraciones Finales**
   * Manejo de errores
   * Seguridad
   * Futuras mejoras

**1. Introducción**

* **Propósito del documento**:  
  Esta documentación describe la implementación y funcionamiento de una API para el sistema **"Asistencia QR"**, diseñado para mejorar el registro de asistencia de los profesores mediante el uso de códigos QR.
* **Descripción general de la API**:  
  La API de **Asistencia QR** facilita la gestión de los profesores, sus asistencias, cursos, horarios y materias. Incluye autenticación basada en JWT y una arquitectura backend diseñada con Node.js y PostgreSQL.
* **Descripción del sistema "Asistencia QR"**:  
  El sistema está enfocado en registrar la llegada de los profesores a clases mediante la lectura de un código QR, generado al iniciar sesión en la plataforma. La administración del sistema incluye funciones CRUD para profesores, cursos, materias y horarios.

**2. Instalación y Configuración**

* **Requisitos previos**:
  + Node.js v20.x o superior.
  + PostgreSQL.
  + Un entorno donde ejecutar la API, como una máquina local o un servidor.
* **Instalación de dependencias**: Ejecutar el siguiente comando para instalar las dependencias necesarias:

*npm install*

**Configuración de variables de entorno**: Crear un archivo .env en la raíz del proyecto con las siguientes variables:

*POSTGRES\_DATABASE="verceldb"*

*POSTGRES\_USER="default"*

*POSTGRES\_PASSWORD="jXIrS4D8ycBY"*

*SECRET\_KEY=ffa3cfc0381f105bb68697a0724114e940b2f28d2bd206fd2a8b62b33e87ae8c*

**3. Estructura del Proyecto**

* **controllers/**: Controladores que manejan las solicitudes HTTP.
* **models/**: Modelos Sequelize que representan las tablas en la base de datos.
* **database/**: Archivo de configuración y conexión a la base de datos.
* **routes/**: Define las rutas de la API.
* **index.js**: Punto de entrada principal del servidor Express.

Memoria actualizada

Perfecto, tomando en cuenta que el proyecto se llama **"Asistencia QR"** y que estamos entregando la documentación de la API de este sistema, aquí te dejo cómo quedaría la estructura final con este nombre y contexto:

**Documentación de la API del Sistema "Asistencia QR"**

**Carátula**

**Título**:  
**Documentación de la API del Sistema "Asistencia QR"**

**Subtítulo**:  
Automatización del registro de asistencia de profesores mediante códigos QR

**Autor**:  
[Tu nombre o el nombre del equipo]

**Fecha**:  
[Fecha actual]

**Índice**

1. **Introducción**
   * Propósito del documento
   * Descripción general de la API
   * Descripción del sistema "Asistencia QR"
2. **Instalación y Configuración**
   * Requisitos previos
   * Instalación de dependencias
   * Configuración de variables de entorno
3. **Estructura del Proyecto**
4. **Modelos de Base de Datos**
   * Profesor
   * Asistencia
   * Curso
   * Materia
   * Horario
5. **Rutas de la API**
   * Profesores
   * Asistencias
   * Cursos
   * Horarios
   * Materias
   * Login
6. **Endpoints Detallados**
   * Endpoints de Profesores
   * Endpoints de Asistencias
   * Endpoints de Cursos
   * Endpoints de Horarios
   * Endpoints de Materias
   * Endpoint de Login
7. **Conexión a la Base de Datos**
8. **Dependencias**
9. **Consideraciones Finales**
   * Manejo de errores
   * Seguridad
   * Futuras mejoras

**1. Introducción**

* **Propósito del documento**:  
  Esta documentación describe la implementación y funcionamiento de una API REST para el sistema **"Asistencia QR"**, diseñado para automatizar el registro de asistencia de los profesores mediante el uso de códigos QR.
* **Descripción general de la API**:  
  La API de **Asistencia QR** facilita la gestión de los profesores, sus asistencias, cursos, horarios y materias. Incluye autenticación basada en JWT y una arquitectura backend diseñada con Node.js y PostgreSQL.
* **Descripción del sistema "Asistencia QR"**:  
  El sistema está enfocado en registrar la llegada de los profesores a clases mediante la lectura de un código QR, generado al iniciar sesión en la plataforma. La administración del sistema incluye funciones CRUD para profesores, cursos, materias y horarios.

**2. Instalación y Configuración**

* **Requisitos previos**:
  + Node.js v20.x o superior.
  + PostgreSQL.
  + Un entorno donde ejecutar la API, como una máquina local o un servidor.
* **Instalación de dependencias**: Ejecutar el siguiente comando para instalar las dependencias necesarias:

bash

Copiar código

npm install

* **Configuración de variables de entorno**: Crear un archivo .env en la raíz del proyecto con las siguientes variables:

env

Copiar código

POSTGRES\_DATABASE=nombre\_base\_datos

POSTGRES\_USER=usuario

POSTGRES\_PASSWORD=contraseña

SECRET\_KEY=clave\_secreta\_jwt

**3. Estructura del Proyecto**

* **controllers/**: Controladores que manejan las solicitudes HTTP.
* **models/**: Modelos Sequelize que representan las tablas en la base de datos.
* **database/**: Archivo de configuración y conexión a la base de datos.
* **routes/**: Define las rutas de la API.
* **index.js**: Punto de entrada principal del servidor Express.

**4. Modelos de Base de Datos**

**Profesor (ModelProfesor.js)**

* **Descripción**: Almacena la información de los profesores.
* **Columnas**: id, nombre, apellido, dni, telefono, correo, password.

**Asistencia (ModelAsistencia.js)**

* **Descripción**: Registra las asistencias de los profesores.
* **Columnas**: AsistenciaID, Fecha, HoraLlegada, ProfesorID.

**Curso (ModelCursos.js)**

* **Descripción**: Representa los cursos.
* **Columnas**: CursoID, Anio, Division.

**Materia (ModelMateria.js)**

* **Descripción**: Almacena las materias disponibles.
* **Columnas**: MateriaID, NombreMateria.

**Horario (ModelHorario.js)**

* **Descripción**: Gestiona los horarios de los profesores.
* **Columnas**: HorarioID, Dia, CursoID, ProfesorID, MateriaID, fechaInicio, fechaFin.

**Rutas / Endpoints de la API**

Estas rutas permiten la interacción con los controladores de tu API. Todas las rutas están prefijadas con /api, por lo que los endpoints completos se verán así:

<https://backend-asistencia-qr.vercel.app/api>

**A. Rutas para Profesores**

Estas rutas manejan las operaciones CRUD sobre los profesores.

(Las Response que se verán a continuación son solo ejemplos)

**GET /api/profesores**  
Obtiene todos los profesores registrados en la base de datos.

* + **Response**:
  + [ { "id": 1, "nombre": "Juan", "apellido": "Pérez", "dni": "12345678", "telefono": "1122334455", "correo": "juan.perez@escuela.com" } ]

**GET /api/profesores/:id**  
Obtiene los detalles de un profesor específico mediante su ID.

* **Response**:
* { "id": 1, "nombre": "Juan", "apellido": "Pérez", "dni": "12345678", "telefono": "1122334455", "correo": "juan.perez@escuela.com" }

**POST /api/profesores**  
Crea un nuevo profesor en la base de datos.

* **Body**:
* { "nombre": "Juan", "apellido": "Pérez", "dni": "12345678", "telefono": "1122334455", "correo": "juan.perez@escuela.com", "password": "1234" }
* **Response**:
* { "message": "Registro Creado correctamente" }

**PUT /api/profesores/:id**  
Actualiza los datos de un profesor por su ID.

* **Body**: { "nombre": "Juan", "apellido": "Pérez", "dni": "87654321", "telefono": "5544332211" }
* **Response**:
* { "message": "Registro actualizado correctamente" }

**DELETE /api/profesores/:id**  
Elimina un profesor y todas sus relaciones (horarios y asistencias) por su ID.

* **Response**: { "message": "Profesor y todas sus relaciones eliminadas correctamente" }

**B. Rutas para Asistencia**

Estas rutas manejan las operaciones relacionadas con la asistencia de los profesores.

**POST /api/login/asistencia**  
Registra la asistencia de un profesor mediante un token QR.

* + **Body**: { "qrToken": "( *token generado para el qr*)” }
  + Response:
  + {"message": "Asistencia registrada con éxito"}

**GET /api/login/contarAsistencias/:profesorId**  
Cuenta el número total de asistencias e inasistencias de un profesor específico.

* **Response**:
* { "message": "Totales de asistencias e inasistencias del profesor con ID X", "asistencias": 5, "inasistencias": 1 }

**C. Ruta de Login**

Esta ruta maneja el inicio de sesión de los profesores.

* **POST /api/login**  
  Autentica al profesor usando su correo y contraseña, y devuelve un token QR.
  + **Body**: "correo": "profesor@escuela.com", "password": " (*contraseña del profesor*) "
  + Response:
  + { "message": "Hola Juan Pérez, bienvenido!", "qrToken": "jwt\_token" }

**D. Rutas para Cursos**

* **GET /api/listado/cursos**  
  Devuelve un listado de todos los cursos registrados en la base de datos.
  + **Response**: [ { "CursoID": 1, "Anio": 3, "Division": "A" } ]

**E. Rutas para Horarios**

* **POST /api/insertar/horario**  
  Inserta un nuevo horario para un profesor en un curso específico.
  + **Body**: { "nombreProfesor": "Juan Pérez", "nombreCurso": "1ro 1ra", "nombreMateria": "Matemática", "dia": "Lunes", "fechaInicio": "08:00", "fechaFin": "10:00" }
  + **Response**:
  + { "message": "Horario insertado exitosamente", "nuevoHorario": { "ProfesorID": 1, "CursoID": 1, "MateriaID": 3, "Dia": "Lunes", "fechaInicio": "08:00:00", "fechaFin": "10:00:00" } }

**F. Rutas para Materias**

* **POST /api/insertar/materias**  
  Inserta una nueva materia en la base de datos.
  + **Body**: { "NombreMateria": "Matemática" }
  + **Response**:
  + { "message": "Materia insertada exitosamente", "nuevaMateria": { "MateriaID": 1, "NombreMateria": "Matemática" } }
* **GET /api/listado/materias**

Devuelve un listado de todas las materias registradas.

* **Response:**
* [ { "MateriaID": 1, "NombreMateria": "Matemática" } ]

**7. Conexión a la Base de Datos**

* La conexión a la base de datos PostgreSQL se maneja a través de **Sequelize**.
* **Configuración SSL**: Se permite la conexión SSL con PostgreSQL, ideal para entornos de producción.

**8. Dependencias Utilizadas**

Lista de las dependencias utilizadas en el proyecto:

1. **bcryptjs** (^2.4.3):
   * **Descripción**: Biblioteca para hashear y comparar contraseñas utilizando el algoritmo bcrypt.
   * **Uso**: Se utiliza en el controlador ProfesorController.js para hashear contraseñas antes de almacenarlas en la base de datos, y en LoginController.js para comparar contraseñas durante el inicio de sesión.
2. **cors** (^2.8.5):
   * **Descripción**: Middleware para habilitar **CORS** (Cross-Origin Resource Sharing) en la API, permitiendo que pueda ser accedida desde otros dominios.
   * **Uso**: Se usa en index.js para configurar CORS en todas las solicitudes a la API.
3. **dotenv** (^16.4.5):
   * **Descripción**: Carga variables de entorno desde un archivo .env al entorno de Node.js.
   * **Uso**: En db.js para cargar las credenciales de la base de datos y en LoginController.js para cargar la clave secreta utilizada por JWT.
4. **express** (^4.19.2):
   * **Descripción**: Framework web minimalista para Node.js.
   * **Uso**: Es la base de toda la API, proporcionando las funcionalidades para crear servidores y manejar rutas y middleware.
5. **jsonwebtoken** (^9.0.2):
   * **Descripción**: Implementación de **JWT (JSON Web Tokens)** para crear y verificar tokens de autenticación.
   * **Uso**: Se utiliza en LoginController.js para crear tokens JWT que autentican a los profesores, y en AsistenciaController.js para verificar los tokens al registrar la asistencia.
6. **pg** (^8.12.0):
   * **Descripción**: Cliente para interactuar con bases de datos PostgreSQL desde Node.js.
   * **Uso**: En db.js como el módulo de conexión a la base de datos PostgreSQL.
7. **sequelize** (^6.37.1):
   * **Descripción**: **ORM (Object Relational Mapper)** para Node.js, que facilita la interacción con bases de datos SQL.
   * **Uso**: En los modelos de la API (ModelProfesor.js, ModelAsistencia.js, ModelCursos.js, ModelMateria.js, ModelHorario.js) para definir las tablas y manejar la persistencia de los datos en PostgreSQL.
8. **nodemon** (^3.1.0):
   * **Descripción**: Herramienta que reinicia automáticamente el servidor Node.js cuando detecta cambios en los archivos.
   * **Uso**: Aunque no es parte del código de la API, se utiliza en desarrollo para reiniciar automáticamente el servidor mientras se trabaja.

**9. Consideraciones Finales**

**Manejo de errores**

* **Errores comunes**: 404 Not Found, 500 Internal Server Error.
* **Validaciones**: Verificación de datos entrantes y respuestas claras en caso de errores.

**Seguridad**

* **JWT**: Los tokens JWT protegen las rutas y autentican a los profesores.
* **SSL**: La conexión a la base de datos está asegurada con SSL.

**Futuras mejoras**

* **Implementar roles de usuario**:  
  Definir roles como administrador, profesor y director, cada uno con diferentes permisos y acceso a ciertas funcionalidades dentro del sistema. Esto permitiría una mayor personalización y control de acceso, aumentando la seguridad y la claridad en el uso de la plataforma.
* **Posible integración con otros servicios educativos**:  
  Integrar el sistema con otras plataformas de gestión educativa, como sistemas de calificaciones, gestión de alumnos o implementación con plataformas de enseñanza online. Esto mejoraría la eficiencia en el manejo de datos del colegio, permitiendo una mejor gestión y centralizada de las actividades escolares.

**Repositorio de la API:**

[**https://github.com/0lio21/Backend-AsistenciaQR**](https://github.com/0lio21/Backend-AsistenciaQR)