Guia del Proyecto - API Node.js con SQLite

Resumen rapido

Proyecto: API en Node.js usando programacion orientada a objetos. Base de datos: SQLite.

Esta guia explica paso a paso como configurar, probar y usar el sistema de inscripciones usando la consola de Windows y curl. Se usa solo texto; no hay capturas.

Requisitos previos

- Tener instalado Node LTS
- Tener Visual Studio Code o cualquier editor para ver el proyecto
- Abrir la consola de Windows (CMD o PowerShell) y posicionarse en la carpeta del proyecto donde esta package.json

  IMPORTANTE: El backend valida el DNI como string. Siempre enviar el DNI entre comillas en el JSON: "dni":"41342897". Si se envia sin comillas, la API puede responder: {"error":"DNI invalido. Debe ser numerico y hasta 8 digitos."}

Como iniciar el sistema

1. Abrir la consola de Windows en la carpeta del proyecto
2. Ejecutar:

```
npm install
```

1. Iniciar el servidor:

```
npm start
```

Si todo salio bien, la consola mostrara que el servidor esta corriendo en http://localhost:3000

Datos de ejemplo que se usan en esta guia

- DNI (ejemplo): "41342897"
- id de alumno (ejemplo): 1
- id de carrera (Desarrollo de Software): 2
- ids de materias disponibles en este ejemplo: 44 y 45
- id de periodo de ejemplo: 1

Usar estos valores en los comandos de prueba para replicar exactamente lo que se muestra.

Probar las funciones del sistema (curl desde consola)

1) Ver carreras disponibles

```
curl http://localhost:3000/api/carreras
```

2) Buscar un alumno por DNI

Enviar el DNI como string (con comillas):

```
curl -X POST http://localhost:3000/api/alumnos/buscar-dni -H "Content-Type: application/json" -d "{\"dni\":\"41342897\"}"
```

Si existe, devuelve los datos del alumno, sus carreras y su historial de materias. Si no existe, devolvera 404 o un error segun configuracion.

3) Verificar si hay un periodo de inscripcion activo para la carrera

Nota: ahora el periodo depende de la carrera. Se debe enviar el id de la carrera en la query.

```
GET /api/periodos/activo?carreraId=2
```

Ejemplo con curl:

```
curl "http://localhost:3000/api/periodos/activo?carreraId=2"
```

Si no hay periodo activo para esa carrera, bloquear avance.

4) Ver las materias de la carrera

```
curl http://localhost:3000/api/carreras/2/materias
```

Devuelve la lista completa de materias y sus correlativas.

5) Ver materias habilitadas para inscribirse (segun correlativas y estados)

Se requiere el id del alumno y la carrera:

```
curl "http://localhost:3000/api/alumnos/1/materias-posibles?carreraId=2"
```

El backend devolvera solo las materias en las que el alumno puede inscribirse. El frontend debe usar esta respuesta para habilitar o deshabilitar controles.

6) Crear la inscripcion

Usar el id del alumno, el id de la carrera y las materias seleccionadas (ejemplo: 44 y 45):

```
curl -X POST http://localhost:3000/api/inscripciones -H "Content-Type:
application/json" -d "{\"alumnoId\":1,\"carreraId\":2,\"materiasIds\":
[44,45],\"periodoId\":1}"
```

El backend volvera a validar todo. Si esta correcto, creara la inscripcion y devolvera el registro creado.

Configurar datos iniciales (ejemplo rapido)

A continuacion se muestran comandos para crear los datos minimos en caso de una BD vacia. Usar los ids de respuesta para reemplazar cuando el sistema devuelva otro id.

Crear la carrera Desarrollo de Software

```
curl -X POST http://localhost:3000/api/admin/carreras -H "Content-Type:
application/json" -d "{\"nombre\":\"Desarrollo de Software\"}"
```

Crear materias (ejemplo)

```
curl -X POST http://localhost:3000/api/admin/materias -H "Content-Type:
application/json" -d "{\"nombre\":\"Matematica I\",\"anio\":1}"

curl -X POST http://localhost:3000/api/admin/materias -H "Content-Type:
application/json" -d "{\"nombre\":\"Programacion I\",\"anio\":1}"

curl -X POST http://localhost:3000/api/admin/materias -H "Content-Type:
application/json" -d "{\"nombre\":\"Base de Datos I\",\"anio\":1}"

curl -X POST http://localhost:3000/api/admin/materias -H "Content-Type:
application/json" -d "{\"nombre\":\"Programacion II\",\"anio\":2}"
```

Asignar materias a la carrera (ejemplo usando carreraId 2)

```
curl -X POST http://localhost:3000/api/admin/carreras/2/materias -H "Content-
Type: application/json" -d "{\"materiasIds\":[44,45,46,47]}"
```

Crear correlativas (ejemplo: Programacion II requiere Programacion I)

```
curl -X POST http://localhost:3000/api/admin/correlativas -H "Content-Type:
application/json" -d "{\"materiaId\":45,\"correlativaId\":44}"
```

Crear un alumno de ejemplo

```
curl -X POST http://localhost:3000/api/admin/alumnos -H "Content-Type:
application/json" -d "{\"dni\":\"41342897\",\"nombre\":\"Juan\",\"apellido\":
\"Perez\",\"email\":\"juan@mail.com\",\"fechaNacimiento\":\"2000-05-15\",
\"telefono\":\"1234567890\",\"direccion\":\"Calle Falsa 123\",\"localidad\":
\"Buenos Aires\",\"activo\":1}"
```

Crear un periodo activo para la carrera (ejemplo usando carreraId 2)

```
curl -X POST http://localhost:3000/api/admin/periodos -H "Content-Type:
application/json" -d "{\"fechaInicio\":\"2025-11-01\",\"fechaFin\":
\"2025-11-30\",\"activo\":1,\"carreraId\":2,\"cupoLimite\":100}"
```

Endpoints principales para usar desde el frontend

Resumen del flujo que debe implementar el frontend:

1. Buscar persona por DNI
2. POST /api/alumnos/buscar-dni (body: { "dni": "41342897" })
3. Si existe, permitirle continuar
4. Mostrar carreras asociadas
5. Seleccionar una carrera
6. Verificar periodo activo para la carrera: GET /api/periodos/activo?carreraId={id}
7. Si hay periodo activo, continuar
8. Consultar materias de la carrera
9. GET /api/carreras/{idCarrera}/materias
10. Consultar materias en las que el alumno se puede inscribir
11. GET /api/alumnos/{idAlumno}/materias-posibles?carreraId={idCarrera}
12. El frontend habilita/deshabilita controles segun esta lista
13. Confirmar y crear la inscripcion
14. POST /api/inscripciones (body: { alumnoId, carreraId, materiasIds, periodoId })

Reglas de negocio

- El periodo activo depende de la carrera (periodo tiene carreraId)
- Un alumno solo puede inscribirse si hay un periodo activo para la carrera seleccionada
- Un alumno solo puede inscribirse a materias habilitadas segun correlativas y su estado
- No se puede inscribir dos veces a la misma materia
- La carrera debe estar asociada al alumno
- El backend vuelve a validar todo al crear la inscripcion (periodo activo, materias posibles, cupos)

Errores comunes y soluciones

- **DNI invalido al buscar**: asegurar enviar JSON con el DNI como string: "dni":"41342897"
- **Servidor no inicia**: ejecutar `npm install` y revisar logs
- **No hay periodo activo para la carrera**: verificar que al crear el periodo se envio `carreraId` correcto
- **Alumno no encuentra materias habilitadas**: revisar correlativas y estados en `materias_aprobadas`
- **Curl con error de comillas**: en PowerShell usar comillas simples para -d o escapar comillas internas
- **Error 404**: comprobar ruta y metodo HTTP
- **ECONNREFUSED**: servidor no esta corriendo o puerto distinto

---

Estructura del proyecto y objetos principales

El proyecto esta organizado por capas para mantener el codigo claro. A continuacion una descripcion de las clases/objetos principales y sus relaciones:

Modelos / Objetos

- Alumno
- campos: id, dni (string), nombre, apellido, email, fechaNacimiento, telefono, direccion, localidad, activo
- relaciones: carreras (lista), inscripciones (lista), materiasAprobadas (lista)
- Carrera
- campos: id, nombre
- relaciones: materias (lista)
- Materia
- campos: id, nombre, anio
- relaciones: correlativas (lista de ids de Materia)
- MateriaAprobada
- campos: id, alumnoId, materiaId, estado (APROBADA, INSCRIPTO, REGULAR, CURSANDO, LIBRE), nota, fechaUltimoEstado, profesorId, periodoId
- Inscripcion
- campos: id, fechaInscripcion, alumnoId, periodoId
- relaciones: materiasInscriptas (lista de materiaIds)

- PeriodoInscripcion

- campos: id, fechaInicio, fechaFin, activo, cupoLimite, carreraId

- Profesor

- campos: id, nombre, apellido

Capas y donde encontrar las clases

- src/modelos/ : definicion de objetos y clases que representan las tablas
- src/repositorios/ : clases que ejecutan consultas a SQLite (CRUD)
- src/servicios/ : logica de negocio (validaciones, calculo de materias habilitadas, transacciones)
- src/controllers/ : handlers HTTP que exponen los endpoints
- src/routes/ : definicion de rutas y middlewares
- src/db/ : archivo sqlite y scripts de inicializacion

Checklist rapido para probar el flujo (usar los datos de ejemplo)

1. Iniciar servidor: `npm start`
2. Crear carrera (si no existe): `curl -X POST ... /api/admin/carreras` (nombre: Desarrollo de Software)
3. Crear materias si es necesario
4. Crear alumno con dni "41342897" (si no existe)
5. Crear periodo para carreraId 2
6. Verificar: `curl "http://localhost:3000/api/periodos/activo?carreraId=2"`
7. Buscar alumno por DNI: `curl -X POST /api/alumnos/buscar-dni -d "{\"dni\": \"41342897\"}"`
8. Consultar materias posibles: `curl "http://localhost:3000/api/alumnos/1/materias-posibles?carreraId=2"`
9. Crear inscripcion con materias [44,45]

Fin de la guia.