CHECK REVISION BACKEND

1. Estructura del proyecto

- ¿Tiene una **estructura organizada** y escalable?
  - Separación por capas: routes, controllers, models, middlewares, services, etc.
- ¿Usa patrón MVC u otro patrón relevante (ej. repository/service layer)?
- ¿Tiene un archivo principal (server.js o app.js) claro y legible?

---

2. Rutas y controladores

- ¿Las rutas (routes/) están claramente definidas y separadas por recurso?
- ¿Los controladores (controllers/) manejan la lógica de negocio y no están en las rutas?
- ¿Se documentaron las rutas con Swagger u otra herramienta?

---

3. Conexión y manejo de base de datos

- ¿Usa un ORM como Sequelize, Prisma o una librería como Knex, Mongoose (si fuera MongoDB)?
- ¿Las consultas están optimizadas y seguras?
- ¿Se implementó correctamente la conexión a la base de datos?

---

4. Middleware y validaciones

- ¿Se usan middlewares para tareas comunes (autenticación, logging, manejo de errores)?
- ¿Hay validación de entrada con bibliotecas como express-validator o Joi?
- ¿Se evita la exposición de datos sensibles?

---

5. Autenticación y autorización

- ¿El backend tiene autenticación (login)?
  - ¿Con tokens JWT, sesiones, OAuth u otro mecanismo?
- ¿Hay autorización según roles de usuario?
- ¿Las rutas protegidas verifican permisos correctamente?

Instructor: Jesús Ropero Barbosa
jesus.ropero.b@gmail.com

6. Pruebas y errores

- ¿El backend maneja adecuadamente errores (try/catch, middlewares de error)?
- ¿Hay pruebas unitarias o de integración (con Jest, Mocha, etc.)?
- ¿Hay consistencia en los mensajes de error y respuestas al cliente?

7. Seguridad

- ¿Usa variables de entorno (.env) para claves y configuración sensible?
- ¿Evita vulnerabilidades comunes (XSS, CSRF, SQL Injection)?
- ¿Las contraseñas están cifradas con bcrypt o similar?
- ¿Tiene CORS correctamente configurado?

8. Consistencia y estilo del código

- ¿Se sigue una convención de nombres coherente (camelCase, PascalCase)?
- ¿El código es limpio, legible y comentado adecuadamente?
- ¿Usa herramientas de estilo como Prettier, ESLint?

9. Manejo de dependencias

- ¿El archivo package.json está bien organizado?
- ¿Solo se incluyen dependencias necesarias?
- ¿Se usan scripts útiles para desarrollo (npm run dev, npm test, etc.)?

10. Documentación técnica

- ¿Tiene un README.md explicativo con:
  - Cómo ejecutar el proyecto
  - Rutas principales del API
  - Dependencias importantes
  - Ejemplo de .env?
- ¿Se incluye documentación del API (Swagger, Postman)?

Ejemplo de herramientas y buenas prácticas

Tema	Herramientas/Buenas prácticas
Autenticación	JWT + bcrypt
Validación	express-validator, Joi
Estructura del código	Separación de rutas, controladores y modelos
Base de datos	Sequelize / Knex / Raw SQL
Pruebas	Jest / Mocha + Supertest
Seguridad	Helmet, dotenv, sanitización de entrada
Documentación de rutas	Swagger, Postman