Módulo de Autenticación con Nicola Framework

1. Flujo de Datos para el Proceso de Login

El proceso sigue una secuencia estricta para garantizar la seguridad y la correcta manipulación de la información:

1. **Cliente (Frontend):** Envía una solicitud `POST` al *endpoint* de login. El cuerpo de la solicitud debe ser un objeto JSON con las credenciales: `email` y `password`. El `Content-Type` debe ser `application/json`.
2. **Nicola (Router/Middleware - `Insulator`):** El *router* intercepta la solicitud. Si se ha configurado el *middleware* `Insulator` (como se hace en las rutas), este valida inmediatamente el *schema* de los datos recibidos (comprobando que existan `email` y `password` y que sean de tipo `string`).
    - **Control de Errores:** Si la validación de `Insulator` falla (datos incompletos o tipo incorrecto), el *framework* detiene el flujo y responde automáticamente con un **Error 400 (Bad Request)**, protegiendo al *controller* de datos inválidos.
3. **Nicola (Controller):** Si los datos son válidos, el *controller* toma el control. Su misión principal es interactuar con el sistema de base de datos (en este caso, se asume **Supabase**) para verificar la identidad. Pregunta a la DB si la combinación `user/pass` es legítima utilizando el método de autenticación correspondiente (e.g., `supabase.auth.signInWithPassword`).
4. **Nicola (Coherer - Generación del Token):** Una vez verificada la identidad, el *controller* utiliza la utilidad `Coherer` del *framework*. `Coherer` es responsable de **GENERAR SU PROPIO JWT** (el "Nicola Token"). Este token es fundamental porque está **firmado con la clave secreta del servidor (`NICOLA_SECRET`)**, lo que asegura que solo este *backend* pueda verificar su autenticidad en futuras solicitudes.
5. **Respuesta (Backend a Cliente):** La respuesta final al cliente contiene el "Nicola Token" generado. Este token debe ser almacenado por el cliente y adjuntado en los *headers* de todas las solicitudes futuras a rutas privadas.

2. Estructura del Módulo de Autenticación

El módulo se compone de tres partes esenciales: Rutas, Controlador y Middleware.2.1. Rutas (`src/app/routes/AuthRoutes.js`)

Misión: Definir los *endpoints* públicos (`/login` y `/register`) y aplicar la validación inicial de *schema* (`Insulator`).

// Nota Importante: Si se decide no usar Insulator, la validación de la

// existencia y tipo de los campos DEBE ser realizada manualmente al

// inicio de cada método del controlador (AuthController).

2.2. Controlador (`src/app/controllers/AuthController.js`)

Misión: Contener la lógica de negocio. Esto incluye la verificación de credenciales con el proveedor de identidad (Supabase) y la emisión del JWT propio.

Requisitos Críticos:

1. **Validación Manual (Si no se usa Insulator):** Se debe chequear la existencia de `email` y `password` al inicio.
2. **Verificación de Identidad:** Utilizar `supabase.auth.signInWithPassword` (o método equivalente) para confirmar que las credenciales son válidas.
3. **Generación del Token Propio (JWT):** Este es el paso clave. Se utiliza `Coherer.sign(payload, options)`:
    - **`payload`:** Objeto con datos esenciales (mínimo `userId`).
    - **`options`:** Incluye el tiempo de expiración (ej. `{ expiresIn: "1h" }`).

Ejemplo de Firma del Token (Detalle):

// Asumiendo que `userId` se obtiene tras la verificación exitosa de Supabase

const payload = { userId: '123e4567-e89b-12d3-a456-426614174000', rol: 'admin' };

const token = Coherer.sign(payload, { expiresIn: "1h" });

// El token resultante se enviará al cliente.

Nota sobre la Seguridad (`NICOLA_SECRET`):

- `Coherer` utiliza la variable de entorno `process.env.NICOLA_SECRET` para firmar el token.
- Si esta variable no está configurada, el *framework* fallará intencionalmente, mostrando un mensaje de error claro: `"Please configure, NICOLA_SECRET in the .env file"`. **Es vital configurar esta variable con una cadena larga y compleja.**

2.3. Middleware (`src/app/middlewares/AuthMiddleware.js`)

Misión: Proteger todas las rutas que requieren que el usuario esté autenticado.

Requisitos del Middleware:

1. **Lectura del Header:** Debe buscar el token en el *header* `Authorization` (formato estándar: `"Bearer <token>"`).
2. **Validación del Token:** Usar **`Coherer.verify(token)`**. Este método realiza dos comprobaciones:
    - Verifica que el token sea auténtico (que haya sido firmado con `NICOLA_SECRET`).
    - Verifica que el token no haya expirado.
3. **Resultado de la Verificación:**
    - **Éxito:** Si el token es válido, `Coherer.verify()` decodifica el *payload*. Este *payload* se adjunta a `req.user` (ej. `req.user = { userId: ... }`) y se llama a `next()` para continuar con la ejecución del *controller*.
    - **Fallo:** Si la verificación falla (token inválido, expirado, o no presente), se establece `res.statusCode = 401` y se termina la respuesta con un mensaje de **"Unauthorized"**.

El Servicio de Conexión (`src/services/SupabaseClient.js`)

- **Agrega esto al ticket:**
    - **Archivo:** `src/services/SupabaseClient.js`
    - **Misión:** Exportar la instancia configurada de Supabase.

El "Cableado" en el Servidor (`app.js`)

- **Agrega esto al ticket:**
    - **Archivo:** `app.js` (Modificación)
    - **Misión:** Importar y activar las rutas de Auth.
    -

Configuración de Entorno (`.env`)
Fragmento de código
# ... otras variables ...
SUPABASE_URL=https://tuproyecto.supabase.co
SUPABASE_KEY=tu-anon-key

CRÍTICO: La llave maestra para firmar tokens

NICOLA_SECRET=UnaCadenaSuperLargaYCompleja_QueNadieAdivine_2026