Evidencia GA6-220501123-AA2-EV01

Codificar la API de Acuerdo a Los Requerimientos del Software

Aprendiz: Juan Carlos Lopez Moreno

Instructor: Alvaro Esteban Betancourt Matoma

Área Técnica

Servicio Nacional de Aprendizaje – SENA

Técnico en Programación de Aplicaciones y Servicios Para la Nube

Código: 3186263

Diciembre 2025

Introducción

El desarrollo de interfaces de programación de aplicaciones (APIs) se ha consolidado como un pilar fundamental en la arquitectura de software moderna, permitiendo la comunicación eficiente entre sistemas heterogéneos mediante estándares establecidos y protocolos bien definidos. En este contexto, las APIs RESTful representan el paradigma predominante para el diseño de servicios web, aprovechando los principios arquitectónicos de REST (Representational State Transfer) para ofrecer soluciones escalables, mantenibles y coherentes con los estándares HTTP.

Este informe documenta el proceso de diseño, implementación y validación de una API RESTful que facilita el intercambio de información estructurada en formato JSON. El desarrollo se fundamenta en una comprensión profunda de los conceptos arquitectónicos REST, la aplicación rigurosa de buenas prácticas de ingeniería de software y el aprovechamiento óptimo de las capacidades que proporciona un Entorno de Desarrollo Integrado (IDE) moderno.

La implementación aborda los requerimientos específicos establecidos en el componente formativo de Construcción de API (GA6-220501123-AA2-EV01), considerando aspectos críticos como la correcta definición de endpoints, el manejo apropiado de métodos HTTP, la estructuración coherente de respuestas JSON, la gestión adecuada de códigos de estado HTTP y la implementación de mecanismos que garanticen la robustez y confiabilidad del servicio.

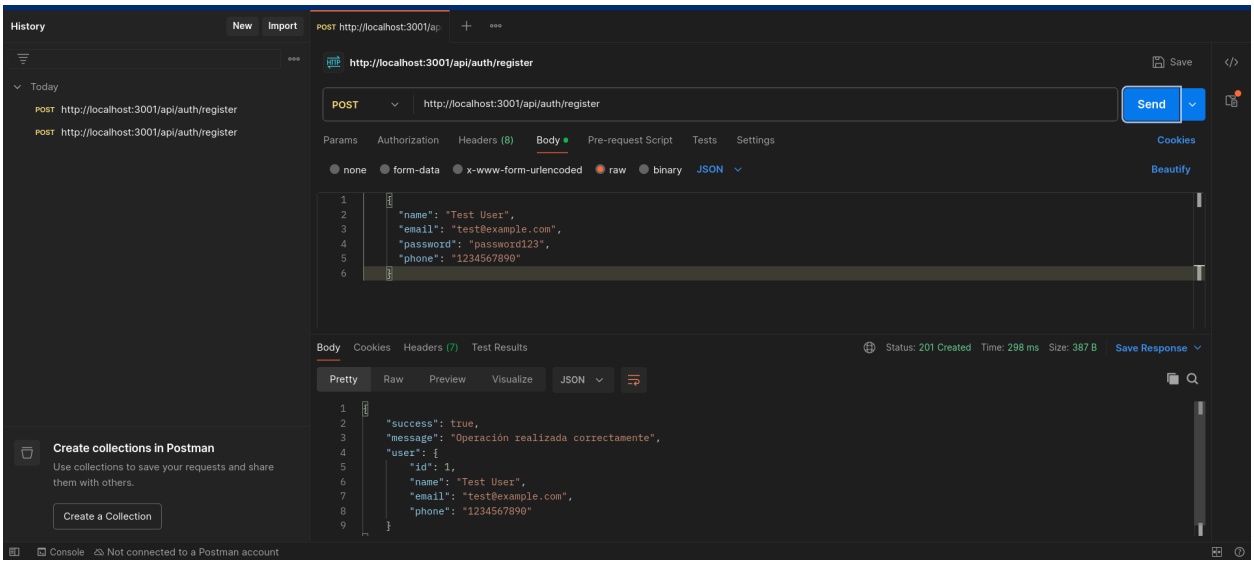
Para asegurar la calidad y funcionalidad de la solución desarrollada, se ha integrado Postman como herramienta de testing, permitiendo la validación exhaustiva de cada endpoint implementado, la verificación del comportamiento esperado ante diferentes escenarios de uso y la documentación práctica de las capacidades de la API.

A través de este documento, se presenta el código fuente desarrollado, acompañado de la justificación técnica de las decisiones de diseño adoptadas, la descripción detallada de la

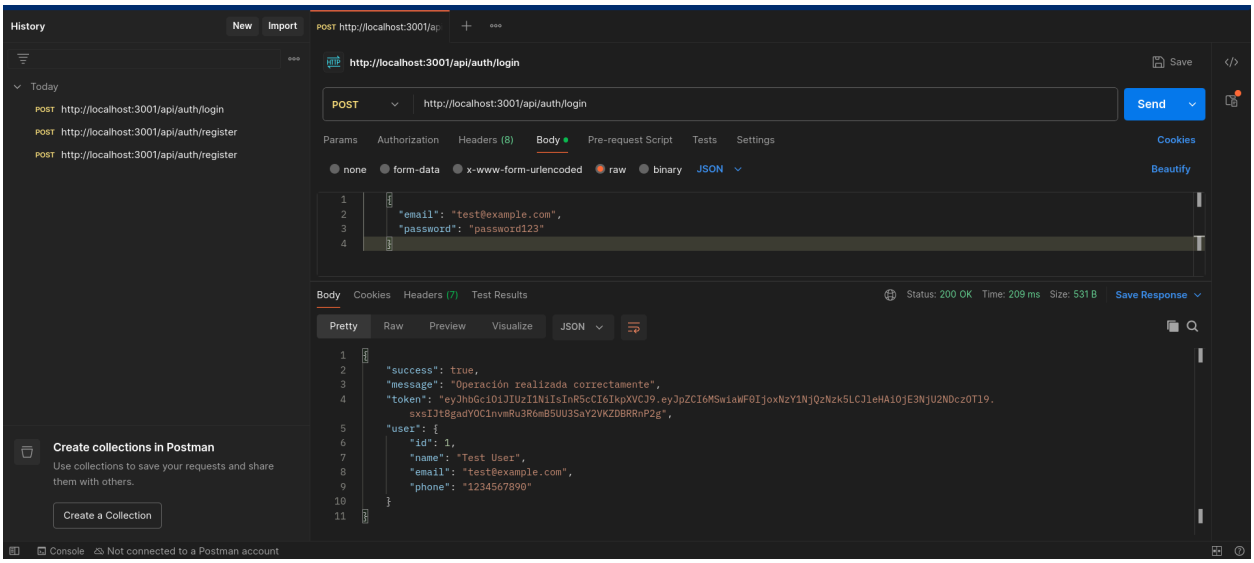
estructura del proyecto y las evidencias de las pruebas realizadas, demostrando así el cumplimiento integral de los objetivos planteados en la actividad de aprendizaje.

Test de la API

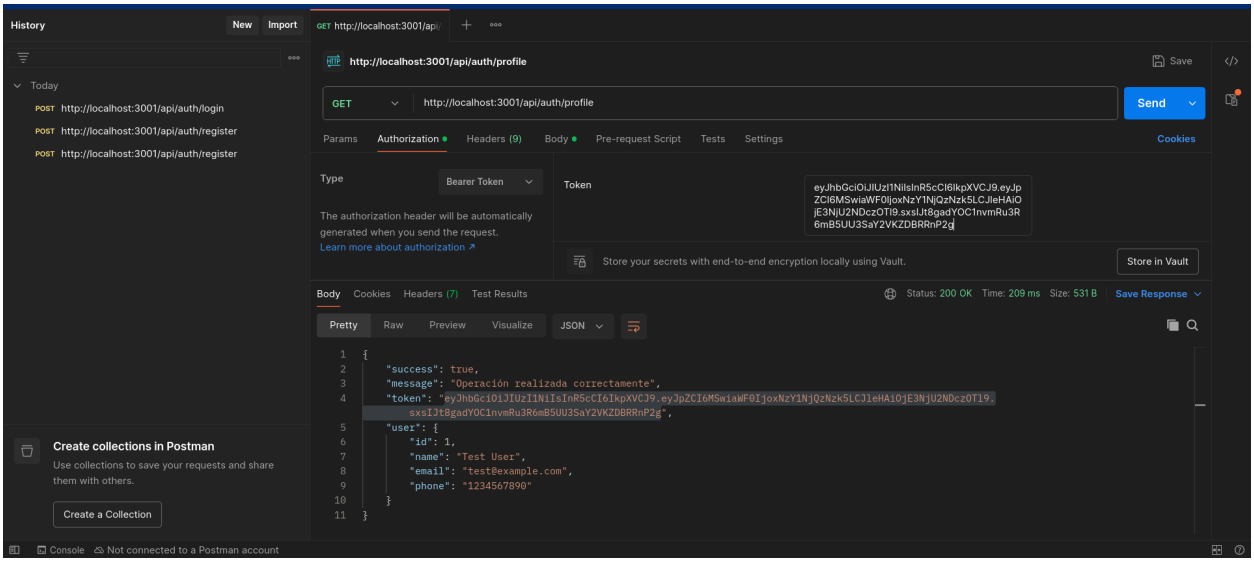
Registro de Usuario (POST /api/auth/register): El proceso de testing inicia con la validación del endpoint de registro de usuarios, donde se envía una petición POST al recurso `http://localhost:3001/api/auth/register` con un payload JSON conteniendo los campos name, email, password y phone. La API procesa la solicitud en 298 ms, retornando un código de estado HTTP 201 Created, lo cual es semánticamente correcto para la creación de recursos nuevos. La respuesta JSON estructurada incluye los campos success (true), message ("Operación realizada correctamente") y un objeto user con los datos del usuario recién creado, excluyendo información sensible como la contraseña. Este endpoint implementa correctamente el proceso de hash de contraseñas mediante bcrypt antes de persistir los datos en la base de datos SQLite, como se evidencia en las capturas posteriores de la estructura de datos.



Autenticación de Usuario (POST /api/auth/login): Posteriormente se valida el endpoint de autenticación, enviando una petición POST a `http://localhost:3001/api/auth/login` con las credenciales del usuario previamente registrado (email y password en formato JSON). El servidor procesa la solicitud en 209 ms, retornando un código HTTP 200 OK junto con una respuesta estructurada que incluye un token JWT (JSON Web Token) y los datos del usuario autenticado. El token generado sigue el estándar JWT con su estructura característica de tres segmentos separados por puntos (header.payload.signature), y será utilizado en las peticiones subsecuentes que requieran autenticación. Este endpoint valida correctamente las credenciales comparando la contraseña en texto plano proporcionada con el hash almacenado en la base de datos, y solo genera el token si la autenticación es exitosa.

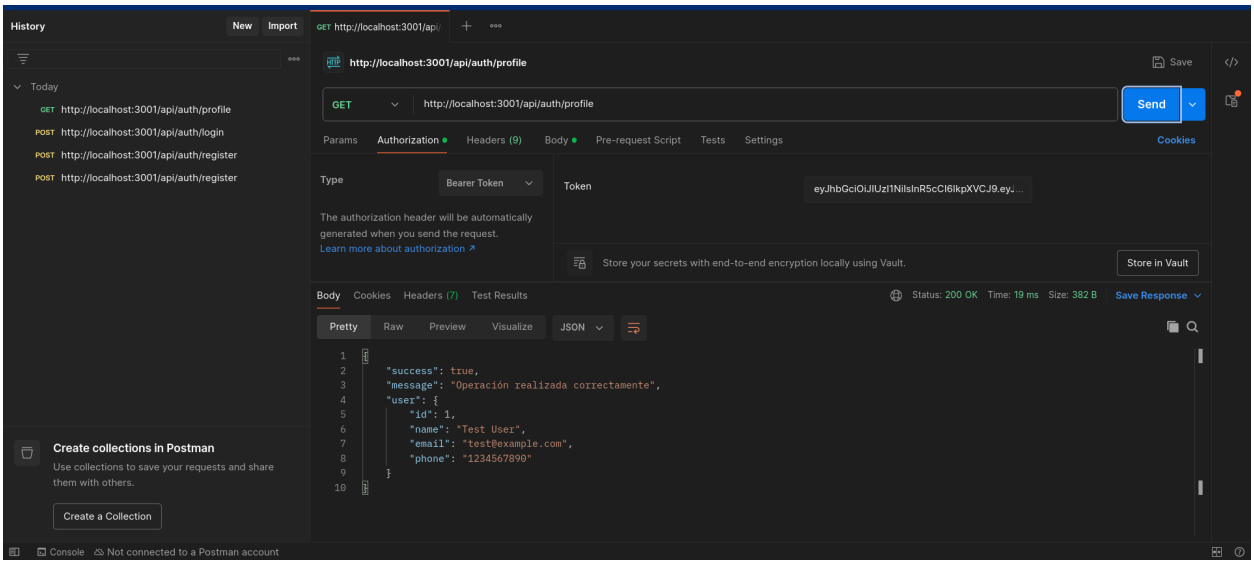


Consulta de Perfil con Token Extendido (GET /api/auth/profile): Se realiza una tercera prueba del endpoint de perfil utilizando un token JWT diferente, esta vez con un tiempo de respuesta de 209 ms y un tamaño de respuesta de 531 bytes. La particularidad de esta prueba radica en que la respuesta incluye no solo los datos del usuario autenticado, sino también el campo token en el payload JSON, sugiriendo una funcionalidad de renovación o refresh del JWT. El endpoint mantiene la coherencia en la estructura de respuesta con los campos success, message, token y user, demostrando que el middleware de autenticación valida correctamente diferentes tokens JWT y permite el acceso a recursos protegidos. Esta implementación evidencia buenas prácticas en la gestión de sesiones mediante tokens stateless.



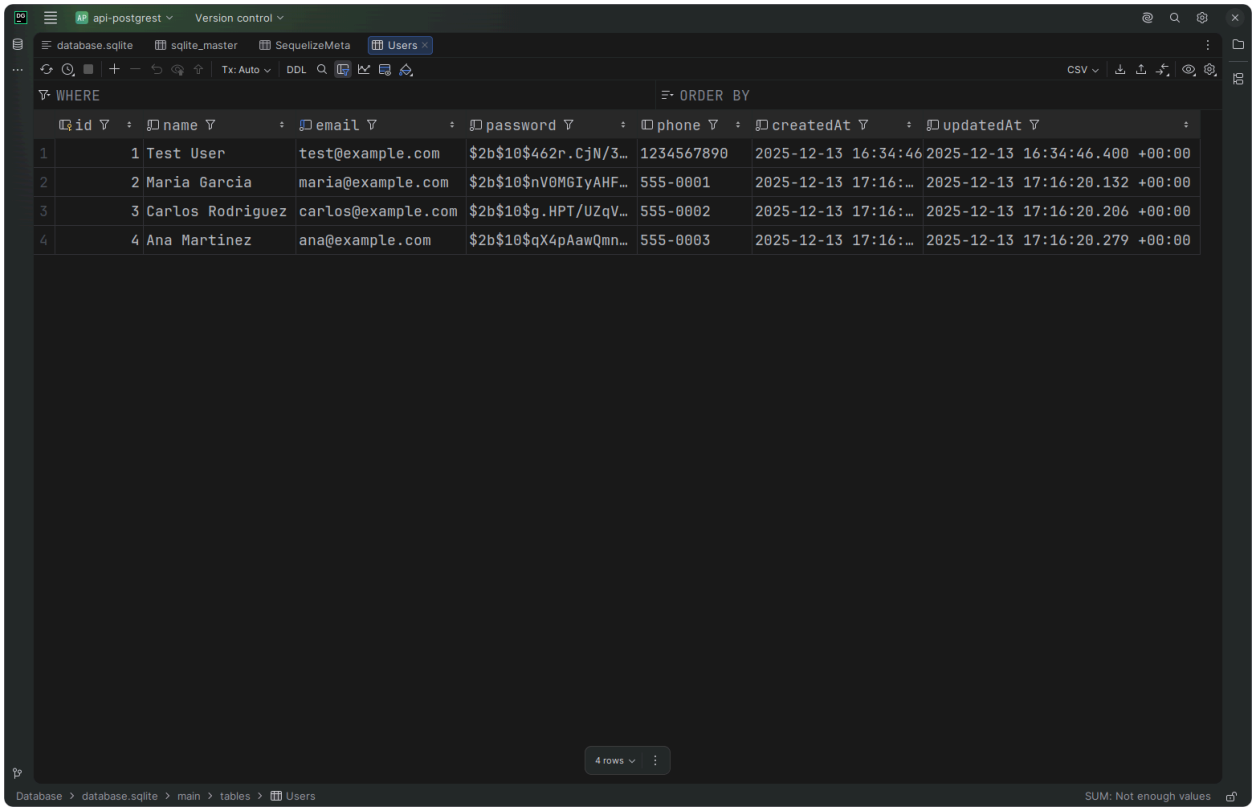
Consulta de Perfil Autenticado (GET /api/auth/profile): Se ejecuta una petición GET al endpoint protegido `http://localhost:3001/api/auth/profile`, utilizando el token JWT obtenido en el proceso de login mediante el esquema de autenticación Bearer Token. El servidor procesa la solicitud en apenas 19 ms, retornando un código HTTP 200 OK con los datos del usuario autenticado en formato JSON. La respuesta excluye información sensible como la contraseña hasheada, retornando únicamente los campos id, name, email y phone del usuario.

Este endpoint demuestra la correcta implementación de un middleware de autenticación que valida el token JWT, extrae la información del usuario del payload del token, consulta los datos actualizados en la base de datos y retorna la información de manera segura. El excelente tiempo de respuesta evidencia la eficiencia del sistema de autenticación basado en tokens.



Persistencia en Base de Datos SQLite: Finalmente, se presenta la visualización de la tabla Users en la base de datos SQLite, donde se confirma la correcta persistencia de los datos registrados a través de la API. La estructura de la tabla incluye los campos id (integer, primary key, autoincrement), name (text), email (text, unique), password (text), phone (text), createdAt (datetime) y updatedAt (datetime), implementando así un esquema completo de auditoría temporal. Se observan cuatro registros de usuarios con sus contraseñas correctamente hasheadas mediante bcrypt, evidenciado por el prefijo característico `$2b$10$` seguido de la sal y el hash resultante. Los timestamps de creación y actualización demuestran que el ORM o la lógica de persistencia gestiona automáticamente estos campos, siguiendo las convenciones de bases de datos relacionales. Esta captura valida que la API no solo funciona correctamente en términos de

respuestas HTTP, sino que también persiste adecuadamente los datos con las medidas de seguridad implementadas, específicamente el hashing de contraseñas antes del almacenamiento.



Para poder visualizar el código se puede hacer uso del siguiente link en el cual está el repositorio del código realizado para el desarrollo de la API: [Respositorio GitHub]