

API de Gestión y Autenticación de Clientes – TongueTrek

Autor:

Mauricio Villegas

CC:1022144753

Profesor:

Guillermo de la peña

Fecha:

30 de junio de 2025



1. Introducción

El desarrollo de aplicaciones web y móviles modernas exige un manejo de usuarios que no solo sea funcional, sino fundamentalmente seguro y confiable. El presente proyecto aborda la creación de un servicio de backend (API RESTful) para la aplicación **TongueTrek**, diseñado para gestionar el ciclo de vida completo de sus clientes. Este sistema va más allá de un simple registro, implementando una arquitectura robusta que prioriza la protección de los datos y la experiencia del usuario.

La relevancia de este proyecto radica en la implementación de mecanismos de seguridad estándar en la industria. En un entorno digital donde las brechas de seguridad son cada vez más comunes, es crucial garantizar la integridad de las credenciales de los usuarios. Por ello, este sistema integra encriptación de contraseñas y datos sensibles, protección contra ataques de fuerza bruta mediante el bloqueo de cuentas, y un flujo de autenticación seguro basado en tokens (JWT). Adicionalmente, se ha puesto especial énfasis en la accesibilidad y la usabilidad, desarrollando un sistema de recuperación de cuenta interactivo y un canal de comunicación constante a través de notificaciones por correo electrónico, asegurando que el usuario siempre tenga el control y la visibilidad sobre su cuenta.

2. Objetivo General y Específicos

Objetivo General

Desarrollar una API RESTful **segura, funcional y escalable** para la gestión integral de clientes de la aplicación TongueTrek, garantizando la confidencialidad de los datos, la integridad de la autenticación y una experiencia de usuario confiable a través de todo su ciclo de vida en la plataforma.

Objetivos Específicos

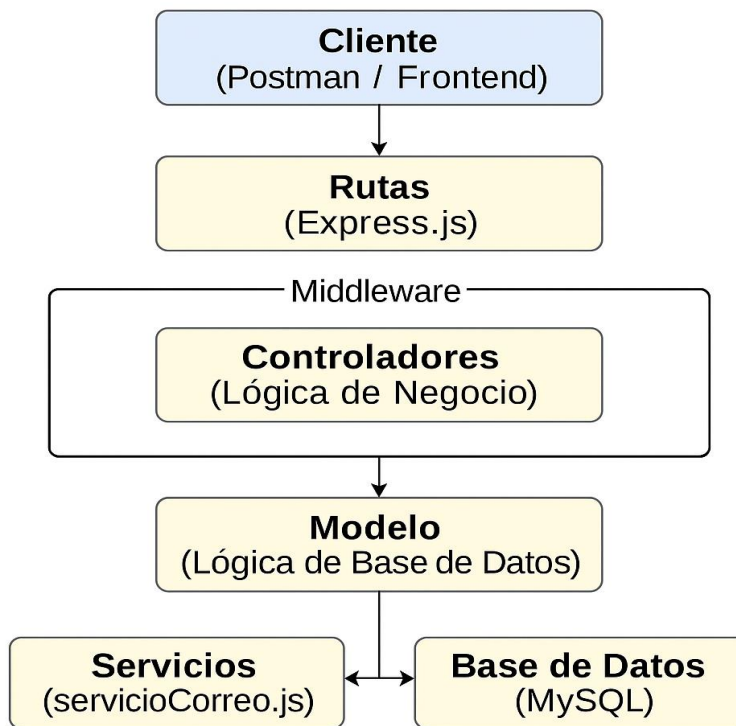
- **Implementar un sistema de registro y login seguro:**
 - Validar los datos de entrada del usuario.
 - Encriptar todas las contraseñas y respuestas de seguridad utilizando el algoritmo bcrypt antes de su almacenamiento.
- **Desarrollar un mecanismo de autenticación robusto:**
 - Generar tokens de acceso (JWT) tras un inicio de sesión exitoso.

- Implementar un sistema de control de sesiones "stateful" mediante el registro y la validación de tokens en la base de datos.
- Asegurar que las rutas sensibles de la aplicación estén protegidas y solo sean accesibles con un token válido y activo.
- **Proteger la aplicación contra intentos maliciosos:**
 - Diseñar e implementar una lógica para contar los intentos fallidos tanto en el login como en la recuperación de cuenta.
 - Bloquear automáticamente las cuentas de los usuarios que excedan el número máximo de intentos permitidos para prevenir ataques de fuerza bruta.
- **Construir un flujo de recuperación de cuenta seguro e interactivo:**
 - Permitir al usuario recuperar el acceso a su cuenta mediante la validación de preguntas de seguridad personalizadas.
 - Generar un token de reseteo de corta duración y de un solo propósito para autorizar el cambio de contraseña de forma segura.
- **Establecer un canal de comunicación con el usuario:**
 - Integrar un servicio de envío de correos electrónicos (Nodemailer) para notificar al usuario sobre eventos importantes en su cuenta, como el registro, la modificación de datos y los cambios de contraseña.

3. Arquitectura del Sistema

La API está diseñada siguiendo una arquitectura en capas, un patrón común en el desarrollo de software que promueve la separación de responsabilidades, la modularidad y la facilidad de mantenimiento. Cada capa tiene un propósito específico y se comunica con las demás de una manera ordenada.

Diagrama de Arquitectura



Descripción de los Módulos (Capas)

- **Cliente (Postman / Frontend):** Es el punto de entrada. Representa cualquier aplicación (en nuestro caso, Postman) que consume la API. El cliente envía peticiones HTTP (GET, POST, PUT, etc.) a las rutas definidas en el servidor.
- **Capa de Rutas (vista/Cliente/ClienteRuta.js):** Actúa como el "recepcionista" de la aplicación. Utilizando **Express.js**, esta capa define los *endpoints* (las URLs) que la API expone al mundo exterior. Su única responsabilidad es recibir una petición y dirigirla al controlador adecuado, pasando primero por los middlewares que sean necesarios.
- **Capa de Middleware (middleware/authMiddleware.js):** Son los "guardias de seguridad" que se interponen entre la ruta y el controlador. Nuestro authMiddleware intercepta cada petición a una ruta protegida para:
 1. Verificar la existencia y validez de un token JWT.

2. Consultar la base de datos para asegurar que el token no ha sido revocado. Si la validación es exitosa, permite que la petición continúe hacia el controlador; de lo contrario, la rechaza con un error de autorización.
- **Capa de Controladores (controlador/Cientes/ClienteControlador.js):** Es el "cerebro" de la aplicación. Aquí reside la **lógica de negocio**. Un controlador recibe la petición (ya validada por el middleware), procesa los datos de entrada, toma decisiones (ej: ¿la contraseña es correcta?, ¿la cuenta está bloqueada?) y orquesta las acciones a seguir, llamando a la capa de modelo para interactuar con la base de datos.
 - **Capa de Modelo (modelo/ClienteModelo.js):** Es la única capa que tiene permiso para "hablar" directamente con la base de datos. Su responsabilidad es abstraer las consultas SQL. El controlador le pide "guarda este usuario" o "busca este token", y el modelo se encarga de escribir y ejecutar la consulta INSERT, SELECT o UPDATE correspondiente. Esto mantiene el código SQL aislado y organizado.
 - **Base de Datos (MySQL):** Es la capa final, donde los datos persisten. Almacena toda la información de manera estructurada en las tablas clientes, perfil y tokens.
 - **Servicios (servicios/servicioCorreo.js):** Es un módulo transversal que puede ser llamado desde cualquier controlador. En nuestro caso, el servicioCorreo encapsula toda la lógica para conectarse con un proveedor externo (Gmail) y enviar correos, manteniendo esa funcionalidad reutilizable y separada de la lógica de negocio principal.
 - **4. Tecnologías Utilizadas**

Para el desarrollo de esta API, se ha seleccionado un conjunto de tecnologías modernas, populares y robustas basadas en el ecosistema de JavaScript y Node.js. La elección de estas herramientas se basa en su rendimiento, su amplia comunidad de soporte y la facilidad para construir aplicaciones escalables.

A continuación, se presenta un resumen de las tecnologías y librerías clave empleadas en el proyecto:

Categoría	Tecnología/Librería	Propósito Principal en el Proyecto
Entorno de Ejecución	Node.js	Permite ejecutar JavaScript del lado del servidor, siendo la base sobre la que corre toda la aplicación.
Framework de Servidor	Express.js	Facilita la creación de la API RESTful, gestionando las rutas, las peticiones HTTP y los middlewares de una manera simple y organizada.
Base de Datos	MySQL	Sistema de gestión de bases de datos relacional elegido para almacenar de forma persistente y estructurada todos los datos de la aplicación.
Conexión a BD	mysql2	Driver de Node.js optimizado para MySQL. Ofrece un rendimiento superior y soporte para funcionalidades asíncronas (async/await).
Seguridad (Encriptación)	bcrypt	Librería estándar para el hashing de contraseñas. Transforma las contraseñas en un formato ilegible y seguro antes de guardarlas.
Seguridad (Autenticación)	jsonwebtoken (JWT)	Implementa la generación y verificación de JSON Web Tokens, el estándar utilizado para crear las "credenciales de acceso" de los usuarios.
Notificaciones	nodemailer	Módulo para el envío de correos electrónicos desde Node.js, utilizado para todas las notificaciones transaccionales de la aplicación.

Gestión de Entorno	dotenv	Permite cargar variables de entorno desde un archivo .env, manteniendo las credenciales y secretos fuera del código fuente.
Middleware	cors	Habilita y configura las políticas de Cross-Origin Resource Sharing, un mecanismo de seguridad esencial para controlar qué dominios pueden acceder a la API.

5. Modelo de Base de Datos

En esta sección se detalla la estructura de la base de datos MySQL que soporta toda la lógica de la aplicación. El modelo fue diseñado de forma normalizada para garantizar la integridad de los datos, evitar la redundancia y facilitar la escalabilidad futura.

Diagrama Entidad-Relación (Descripción Textual)

El modelo se compone de tres tablas principales: clientes, perfil y tokens.

- La tabla **clientes** es la entidad central del sistema.
- Existe una relación de **uno a uno (1:1)** entre clientes y perfil. Cada cliente puede tener un único perfil con su información extendida, y cada perfil pertenece a un único cliente. Esta relación se establece a través de la clave foránea cliente_id.
- Existe una relación de **uno a muchos (1:N)** entre clientes y tokens. Cada cliente puede tener muchos tokens generados a lo largo del tiempo (uno por cada inicio de sesión), pero cada token pertenece a un único cliente.

Descripción de las Tablas

Tabla: clientes

Esta tabla es el núcleo del sistema. Almacena la información esencial de identidad y las credenciales de seguridad de cada usuario.

Columna	Tipo (Simplificado)	Descripción
id	INT (PK)	Identificador único y autoincremental para cada cliente.
documento	VARCHAR	Documento de identidad único del cliente, usado para la recuperación.
nombre	VARCHAR	Nombre completo del cliente.
correo	VARCHAR	Correo electrónico único, usado para el login y las notificaciones.
telefono	VARCHAR	Número de contacto del cliente.
contrasena	VARCHAR(255)	Almacena la contraseña del usuario encriptada con bcrypt.
estado	ENUM	'activo' o 'bloqueado'. Controla si el usuario puede iniciar sesión.
intentos_fallidos	INT	Contador de intentos de login fallidos consecutivos.
intentos_preguntas_fallidos	INT	Contador de intentos fallidos al responder las preguntas de seguridad.

Tabla: perfil

Esta tabla almacena información extendida y personal del usuario, incluyendo los datos necesarios para el flujo de recuperación de cuenta.

Columna	Tipo (Simplificado)	Descripción
idPerfil	INT (PK)	Identificador único y autoincremental para cada fila de perfil.

Columna	Tipo (Simplificado)	Descripción
cliente_id	INT (FK)	Clave foránea que conecta esta fila con el id de la tabla clientes.
direccion	VARCHAR	Dirección de residencia del usuario.
fechaexpedicion	DATE	Fecha de expedición del documento de identidad.
pregunta1, 2, 3	VARCHAR	Almacena el texto de las preguntas de seguridad elegidas por el usuario.
respuesta1, 2...	VARCHAR(255)	Almacena el hash bcrypt de las respuestas a las preguntas de seguridad.

Tabla: tokens

Actúa como un registro de sesiones, permitiendo un control granular sobre cada token emitido y habilitando un sistema de cierre de sesión seguro.

Columna	Tipo (Simplificado)	Descripción
id	INT (PK)	Identificador único y autoincremental para cada token.
documento	VARCHAR	Documento del usuario al que pertenece el token, para fácil referencia.
token	TEXT	La cadena de texto completa del JWT generado.
fecha_expiracion	TIMESTAMP	La fecha y hora exactas en que el token dejará de ser válido.
estado	ENUM	'activo' o 'revocado'. Permite invalidar un token antes de que expire.
tipo	ENUM	'acceso' o 'reseteo'. Diferencia los tokens de login de los de recuperación.

6. Flujos de Usuario

Esta sección describe los procesos clave que un usuario puede realizar en la API, detallando la secuencia de interacciones desde la solicitud inicial del cliente hasta la respuesta final del servidor, incluyendo las interacciones con la base de datos y el envío de notificaciones.

Flujo de Registro de Nuevo Usuario

Este es el punto de partida para cualquier cliente nuevo en la plataforma.

1. **Solicitud del Cliente:** El usuario envía sus datos de registro (documento, nombre, correo, teléfono y contraseña) a través de una petición POST al endpoint `/api/cliente`.
2. **Validación en el Controlador:** El `ClienteControlador` recibe la petición. Primero, verifica que todos los campos obligatorios hayan sido enviados.
3. **Encriptación de la Contraseña:** Si los datos son válidos, el sistema utiliza `bcrypt` para generar un "hash" seguro de la contraseña. Este proceso es irreversible y garantiza que la contraseña nunca se almacene en texto plano.
4. **Persistencia en Base de Datos:** El controlador llama al `ClienteModelo`, que ejecuta una consulta INSERT para crear una nueva fila en la tabla `clientes`. El nuevo usuario se crea con un estado de 'activo' y los contadores de intentos en 0.
5. **Notificación de Bienvenida:** Inmediatamente después de confirmar que el usuario fue creado, el sistema utiliza el servicio de correo (`Nodemailer`) para enviar un email de bienvenida a la dirección de correo proporcionada.
6. **Respuesta Final:** La API responde al cliente con un estado 201 Created y un mensaje de éxito, confirmando que el registro se ha completado.

Flujo de Inicio de Sesión (Login)

Este proceso valida la identidad de un usuario y le otorga una credencial de acceso temporal y segura.

1. **Solicitud del Cliente:** El usuario envía su correo y contraseña a través de una petición POST al endpoint `/api/login`.
2. **Verificación de Usuario:** El controlador busca en la tabla `clientes` un usuario que coincida con el correo proporcionado.

3. Lógica de Seguridad:

- **Estado de la Cuenta:** El sistema verifica si la columna estado del usuario es 'bloqueado'. Si lo es, la petición es rechazada inmediatamente.
- **Verificación de Contraseña:** Se utiliza bcrypt.compare para comparar la contraseña enviada con el hash almacenado en la base de datos.
- **Manejo de Intentos Fallidos:** Si la contraseña es incorrecta, se incrementa el contador intentos_fallidos en la base de datos. Si este contador llega a 3, el estado de la cuenta se cambia a 'bloqueado'.

4. Generación de Sesión Exitosa:

- **Si la contraseña es correcta**, el contador intentos_fallidos se reinicia a 0.
- Se genera un **Token de Acceso (JWT)**, que contiene el id, correo y documento del usuario, con una validez de 1 hora.
- Este nuevo token se guarda en la tabla tokens con un tipo de 'acceso' y un estado de 'activo'.

5. **Respuesta Final:** La API responde con un estado 200 OK, devolviendo información básica del usuario y el token de acceso.

Flujo de Cierre de Sesión (Logout)

Este flujo permite invalidar de forma segura una sesión activa antes de que el token expire.

1. **Solicitud del Cliente:** Un usuario autenticado envía una petición POST a /api/cerrar-sesion, incluyendo su token de acceso en la cabecera Authorization.
2. **Validación del Middleware:** El authMiddleware intercepta la petición, verifica la validez del token y consulta la tabla tokens para asegurar que el token esté 'activo'.
3. **Revocación en Base de Datos:** Si el token es válido, el ClienteControlador llama al modelo para ejecutar una consulta UPDATE sobre la tabla tokens, cambiando el estado de ese token específico a 'revocado'.

4. **Respuesta Final:** La API responde con un 200 OK y un mensaje de éxito. A partir de este momento, ese token ya no podrá ser utilizado para acceder a rutas protegidas.

Flujo de Recuperación de Cuenta

Este es el proceso más complejo, diseñado para que un usuario que ha olvidado su contraseña pueda recuperarla de forma segura.

1. Paso A: Obtener Preguntas de Seguridad:

- El usuario proporciona su número de documento en una petición GET a `/api/recuperar/preguntas/:documento`.
- El sistema busca al cliente y luego su perfil asociado para devolverle las tres preguntas de seguridad que él mismo configuró, sin revelar las respuestas.

2. Paso B: Validar Respuestas:

- El usuario envía su documento y las tres respuestas a POST `/api/recuperar/validar`.
- El sistema compara cada respuesta enviada con los hashes correspondientes almacenados en la tabla perfil usando `bcrypt.compare`.
- **Si alguna respuesta es incorrecta**, se incrementa el contador `intentos_preguntas_fallidos`. Si llega a 3, la cuenta se bloquea.
- **Si todas las respuestas son correctas**, el contador de intentos se reinicia. Se genera un **Token de Reseteo** especial de corta duración (5 minutos) y se guarda en la tabla tokens con el tipo 'reseteo'. Se envía un correo de notificación al usuario y se le devuelve el `resetToken`.

3. Paso C: Restablecer la Contraseña:

- El usuario envía el `resetToken` y su nueva contraseña a POST `/api/recuperar/restablecer`.
- El sistema verifica que el `resetToken` sea válido, no haya expirado y sea del tipo correcto.
- Se encripta la nueva contraseña con `bcrypt` y se ejecuta un UPDATE en la tabla clientes para guardarla.

- Se envía un correo final al usuario notificándole que su contraseña ha sido cambiada con éxito.
- La API responde con un 200 OK.

7. Guía de Pruebas en Postman

Esta sección detalla los flujos de prueba para demostrar todas las funcionalidades de la API. Se asume que el usuario de prueba, **Mauricio Villegas**, no existe previamente en la base de datos.

7.1 Flujo Principal de Usuario (Camino Feliz)

Paso A: Registro de Usuario

Campo Valor

Método POST

URL http://localhost:4545/api/cliente

Body raw > JSON

```
{  
  "t1": "1035123456",  
  "t2": "Mauricio Villegas",  
  "t3": "mauricaldeville2016@gmail.com",  
  "t4": "3011234567",  
  "t5": "Pru3baSegura!"  
}
```

Resultado en Caso de Éxito:

- **En Postman:** Recibirás un estado 201 Created y el siguiente mensaje, confirmando la creación:

```
{  
  "mensaje": "Cliente creado con éxito.",  
}
```

```
"id": 1
}
```

- **En tu Correo:** Recibirás un email de bienvenida de "TongueTrek" en tu bandeja de entrada.
- **En la Base de Datos:** Se creará una nueva fila en la tabla clientes con los datos proporcionados. La contraseña estará encriptada, el estado será 'activo' y los contadores de intentos estarán en 0.

Resultado en Caso de Error:

- **En Postman:** Si algún dato no cumple las validaciones (ej: la contraseña no tiene mayúscula o el documento ya existe), recibirás un error 400 Bad Request o 409 Conflict con un mensaje específico:

JSON

// Ejemplo si falta la mayúscula en la contraseña

```
{
  "errores": [
    "La contraseña (t5) debe contener al menos una letra mayúscula y un número."
  ]
}
```

Paso B: Inicio de Sesión

Campo Valor

Método POST

URL http://localhost:4545/api/login

Body raw > JSON

JSON

```
{
  "t1": "mauricaldeville2016@gmail.com",
  "t2": "Pru3baSegura!"
}
```

}

|  **Acción:** | Copia el valor del token de la respuesta para los siguientes pasos. |

✓ **Resultado en Caso de Éxito:**

- **En Postman:** Recibirás un estado 200 OK con tus datos de usuario y el token de acceso.

 $\{$

```
"mensaje": "Inicio de sesión exitoso",
```

```
"usuario": { ... },
```

```
"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXLTJ5IiwiaWF0IjoiMTY1MjY0MjY0In0="
```

}

- **En la Base de Datos:** Se creará una nueva fila en la tabla tokens con el nuevo token, un estado de 'activo' y un tipo de 'acceso'.

✖ Resultado en Caso de Error:

- **En Postman:** Si la contraseña es incorrecta, recibirás un error 401 Unauthorized con el conteo de intentos restantes. Si la cuenta está bloqueada, recibirás un 403 Forbidden.
- **En la Base de Datos:** Se incrementará el contador intentos_fallidos en la tabla clientes por cada fallo.

Paso C: Completar Perfil de Seguridad

Campo	Valor
-------	-------

Método POST

URL `http://localhost:4545/api/perfil`

Authorization Bearer Token (Pega el token del paso anterior).

Body raw > JSON

JSON

 $\{$

"direccion": "Calle Falsa 123, Itagüí",

```
"fechaexpedicion": "2015-01-01",  
"pregunta1": "Nombre de mi primera mascota",  
"respuesta1": "Rocky",  
"pregunta2": "Ciudad favorita",  
"respuesta2": "Tokio",  
"pregunta3": "Videojuego favorito",  
"respuesta3": "The Witcher 3"  
}
```

✓ Resultado en Caso de Éxito:

- **En Postman:** Recibirás un 201 Created con el mensaje {"mensaje": "Perfil completado y guardado exitosamente."}.
- **En la Base de Datos:** Se creará una nueva fila en la tabla perfil conectada a tu cliente_id. Las columnas respuesta1, respuesta2 y respuesta3 contendrán textos largos y encriptados.

7.2 Flujo de Recuperación de Cuenta

Paso A: Obtener Preguntas

Campo Valor

Método GET

URL http://localhost:4545/api/recuperar/preguntas/1035123456

✓ Resultado en Caso de Éxito:

- **En Postman:** Recibirás un 200 OK con un JSON que contiene únicamente las tres preguntas que configuraste, sin las respuestas.

Paso B: Validar Respuestas

Campo Valor


Método POST

URL http://localhost:4545/api/recuperar/validar

Campo Valor

Body raw > JSON

```
{  
  "documento": "1035123456",  
  "respuesta1": "Rocky",  
  "respuesta2": "Tokio",  
  "respuesta3": "The Witcher 3"  
}
```

|  **Acción:** | Copia el resetToken de la respuesta. |

✓ **Resultado en Caso de Éxito:**

- **En Postman:** Recibirás un 200 OK con un mensaje de éxito, tu correo y el resetToken.
- **En tu Correo:** Recibirás una notificación de que se ha iniciado un proceso de recuperación.
- **En la Base de Datos:** Se creará una nueva fila en la tabla tokens con el resetToken y el tipo 'reseteo'.

✗ **Resultado en Caso de Error:**

- **En Postman:** Si una respuesta es incorrecta, recibirás un 401 Unauthorized con el conteo de intentos restantes. Si fallas 3 veces, la cuenta se bloqueará y recibirás un 403 Forbidden.
- **En la Base de Datos:** Se incrementará el contador intentos_preguntas_fallidos en la tabla clientes.

Paso C: Restablecer Contraseña

Campo Valor

Método POST

URL http://localhost:4545/api/recuperar/restablecer

Campo Valor

Body raw > JSON

```
{  
  "resetToken": "pega_el_resetToken_aqui",  
  "nuevaContrasena": "MiContraseñaFinal987!"  
}
```

✓ Resultado en Caso de Éxito:

- **En Postman:** Recibirás un 200 OK con {"mensaje": "Contraseña actualizada exitosamente..."}
- **En tu Correo:** Recibirás una notificación confirmando el cambio de contraseña.
- **En la Base de Datos:** En la tabla clientes, la columna contrasena se actualizará con el nuevo hash, y los contadores de intentos se reiniciarán a 0.

8. Caso de Uso y Pruebas de Demostración

A continuación, se presenta un caso de uso detallado que demuestra el flujo completo de la API, desde la creación de un nuevo usuario hasta la recuperación de su cuenta. Este escenario sirve como una guía de pruebas integral para validar todas las funcionalidades implementadas.

Usuario de Prueba:

- **Correo:** mauriciovill018@gmail.com
- **Documento:** 1030405060
- **Contraseña:** DemoTest123!

Paso 1: Registrar Usuario

- **POST** /api/cliente
- **Body:**

JSON

```
{  
  "t1": "1030405060",  
  "t2": "Mauricio Villegas (Demo)",  
  "t3": "mauriciovill018@gmail.com",
```

```
"t4": "3123456789",  
"t5": "DemoTest123!"  
}
```

Paso 2: Iniciar Sesión

- **POST** /api/login

- **Body:**

JSON

```
{  
  "t1": "mauriciovill018@gmail.com",  
  "t2": "DemoTest123!"  
}
```

- **Acción:** Copiar el token de la respuesta.

Paso 3: Completar Perfil

- **POST** /api/perfil

- **Authorization:** Bearer Token (Pega el token anterior).

- **Body:**

JSON

```
{  
  "direccion": "Universidad de Antioquia",  
  "fechaexpedicion": "2020-10-10",  
  "pregunta1": "Color favorito",  
  "respuesta1": "Verde",  
  "pregunta2": "Segundo nombre",  
  "respuesta2": "Andres",  
  "pregunta3": "Equipo de futbol",  
  "respuesta3": "Medellin"  
}
```

Paso 4: Recuperar Cuenta

A. Obtener Preguntas

- **GET** /api/recuperar/preguntas/1030405060

B. Validar Respuestas

- **POST** /api/recuperar/validar

- **Body:**

JSON

```
{  
  "documento": "1030405060",  
  "respuesta1": "Verde",  
  "respuesta2": "Andres",  
  "respuesta3": "Medellin"  
}
```

- **Acción:** Copiar el resetToken de la respuesta.

C. Restablecer Contraseña

- **POST** /api/recuperar/restablecer

- **Body:**

JSON

```
{  
  "resetToken": "pega_el_resetToken_aqui",  
  "nuevaContrasena": "NuevaClaveFinal123!"  
}
```