# INTRODUCCIÓN AL DESARROLLO BACKEND

## CON NODEJS

2023

ANDÉN Centro de Innovación y Emprendimientos Tecnológicos

SECRETARÍA DE EXTENSIÓN UNIVERSITARIA UTN - FRC SEU

UTN Facultad Regional Córdoba

Agencia CÓRDOBA JOVEN

CÓRDOBA entre todos

# Autenticación y autorización en una API

- Seguridad en las Aplicaciones
- Amenazas comunes de seguridad
- Autenticación y autorización en una API
- ¿Qué es la autenticación?
  - Métodos de autenticación
  - Implementación de la autenticación
- ¿Qué es la autorización?
  - Métodos de autorización
  - Implementación de la autorización
- Autenticación y autorización con OAuth 2.0 + JWT.

# Seguridad en Aplicaciones

**Por qué es importante?**

- Proteger contra amenazas y ataques cibernéticos.
- Preservar la privacidad de los datos y la confianza del usuario.
- Evitar daños a la reputación y pérdidas financieras.

**Principios**

- Diseño seguro desde el inicio.
- Implementación de buenas prácticas de codificación.
- Mantenimiento regular y actualización de sistemas.
- Implementación de protección de capas de seguridad.
- Validación y filtrado de datos de entrada.

**Componentes**

- Autenticación y autorización.
- Encriptación de datos.
- Protección contra vulnerabilidades y ataques conocidos.

# Amenazas comunes de seguridad

- **Cross-Site Scripting (XSS):** Un ataque XSS ocurre cuando los atacantes inyectan código malicioso en las respuestas de la API. Puede permitir que los atacantes roben información confidencial o realicen acciones en nombre del usuario legítimo.
- **Inyección de código (Code Injection):** La inyección de código se produce cuando se inserta código malicioso en las entradas de la API. Puede permitir a los atacantes ejecutar comandos no autorizados en el servidor o acceder a datos confidenciales.
- **DDoS (Distributed Denial of Service):** Un ataque DDoS tiene como objetivo abrumar la API con una gran cantidad de solicitudes simultáneas. Puede causar la interrupción del servicio, dejando la API inaccesible para los usuarios legítimos.
- **Fuga de información (Information Leakage):** La fuga de información ocurre cuando la API revela detalles confidenciales o sensibles en sus respuestas.
- **Autenticación y autorización inadecuadas:** El uso de métodos de autenticación y autorización débiles o mal configurados puede permitir el acceso no autorizado a recursos protegidos. Los atacantes pueden aprovechar esto para obtener privilegios no autorizados en la API.

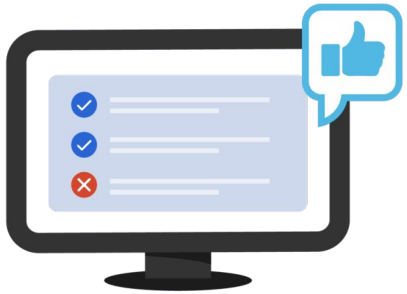
# ¿Cómo identificar un problema de seguridad?

- **Realizar pruebas de penetración (penetration testing):** Esta técnica implica simular ataques reales a la API para descubrir posibles vulnerabilidades y puntos débiles.
- **Revisar el código fuente y la configuración:** Realizar una revisión minuciosa del código fuente de la API y su configuración puede ayudarte a identificar posibles problemas de seguridad.
- **Monitorear y analizar los registros de actividad:** Los registros de actividad de la API pueden proporcionar pistas sobre intentos de ataques o comportamientos sospechosos.
- **Realizar análisis estático y dinámico del código:** Utiliza herramientas de análisis estático y dinámico para identificar posibles vulnerabilidades en el código de la API.
- **Seguir las mejores prácticas de seguridad:** Asegúrate de seguir las mejores prácticas de seguridad en el desarrollo y la implementación de la API. Esto incluye el uso de métodos de autenticación y autorización.
- **Participar en programas de recompensas por errores (bug bounty programs):** Algunas organizaciones ofrecen programas de recompensas por encontrar y reportar problemas de seguridad en sus APIs.

# ¿Qué es la autenticación / autorización?

**¿Qué es la autenticación?**

La autenticación es el proceso de **verificar la identidad de un usuario** para garantizar que tiene los permisos adecuados para acceder a un sistema o recurso.

**¿Qué es la autorización?**

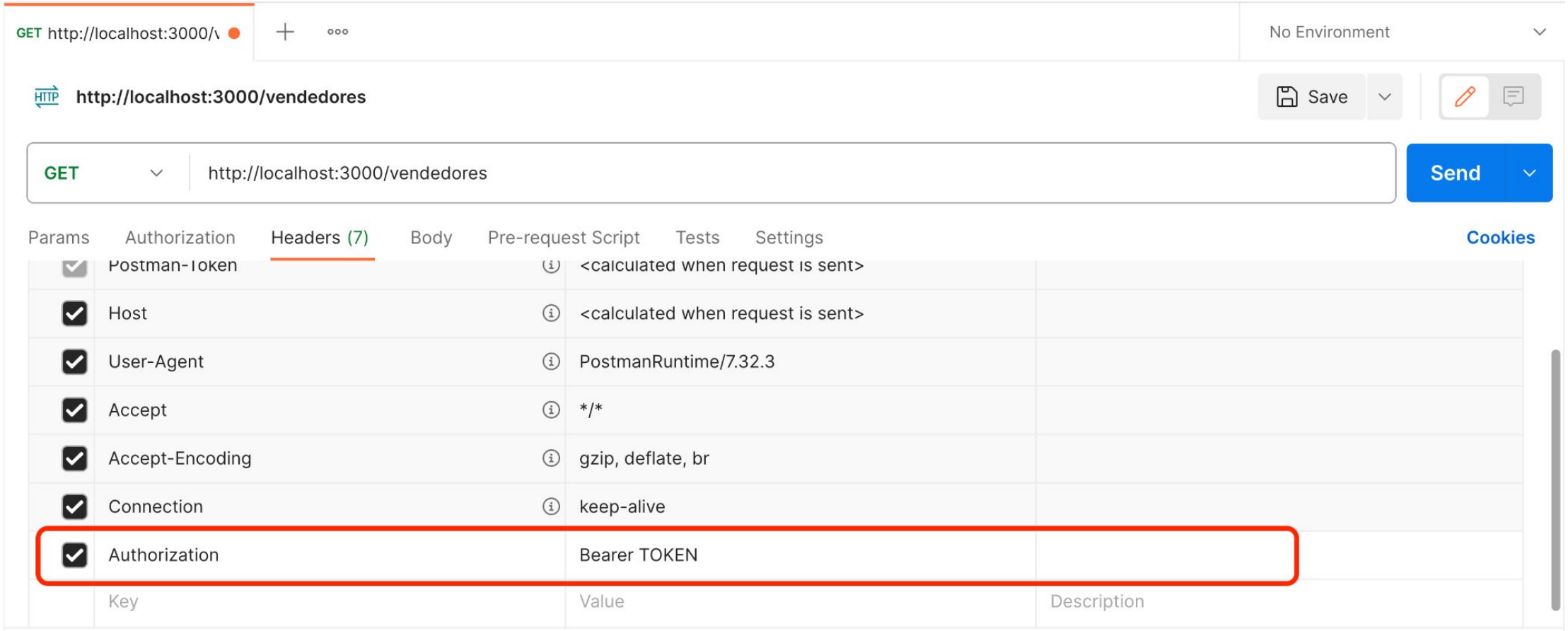
La autorización es el proceso de determinar qué recursos o acciones están permitidos para un usuario autenticado en función de sus **permisos y roles**.

# Autenticación y Autorización: Tokens

- En el contexto de la autenticación y autorización, un token es un objeto que representa la identidad y los permisos de un usuario o una aplicación.
- Los tokens se utilizan para verificar y autorizar el acceso a recursos protegidos, como APIs o servicios.
- En una API se envía en cada petición en un Header HTTP llamado **Authorization** de la siguiente forma

| | | |
|---|---|---|
| ☑ | Authorization | Bearer TOKEN |

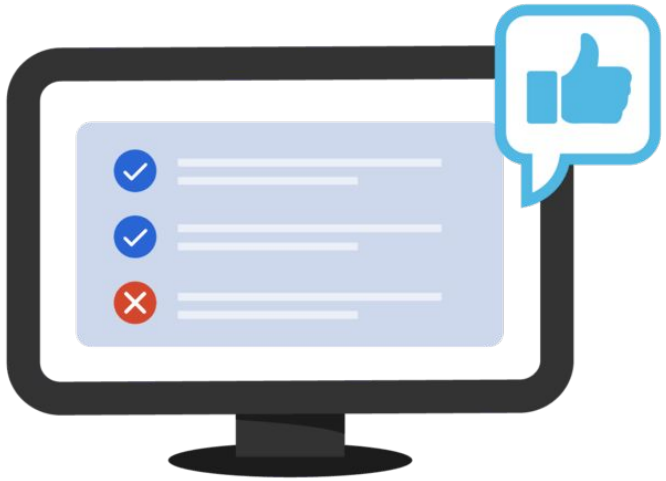
# Token en Postman



# Autenticación de API

- **Autenticación: Verificando la Identidad**
  - La autenticación en una API implica verificar la identidad de los usuarios o aplicaciones que acceden a ella.
  - El objetivo es asegurarse de que solo los usuarios autorizados puedan interactuar con los recursos de la API.
- **Métodos de Autenticación Comunes:**
  - **Tokens de Acceso (Access Tokens):** Los usuarios obtienen un token de acceso después de autenticarse, el cual se utiliza para realizar solicitudes a la API. Ejemplo: OAuth 2.0.
  - **Claves API (API Keys):** Se asigna una clave única a cada usuario o aplicación para autenticar las solicitudes. Es importante mantener esta clave segura y privada.

# Autorización de API

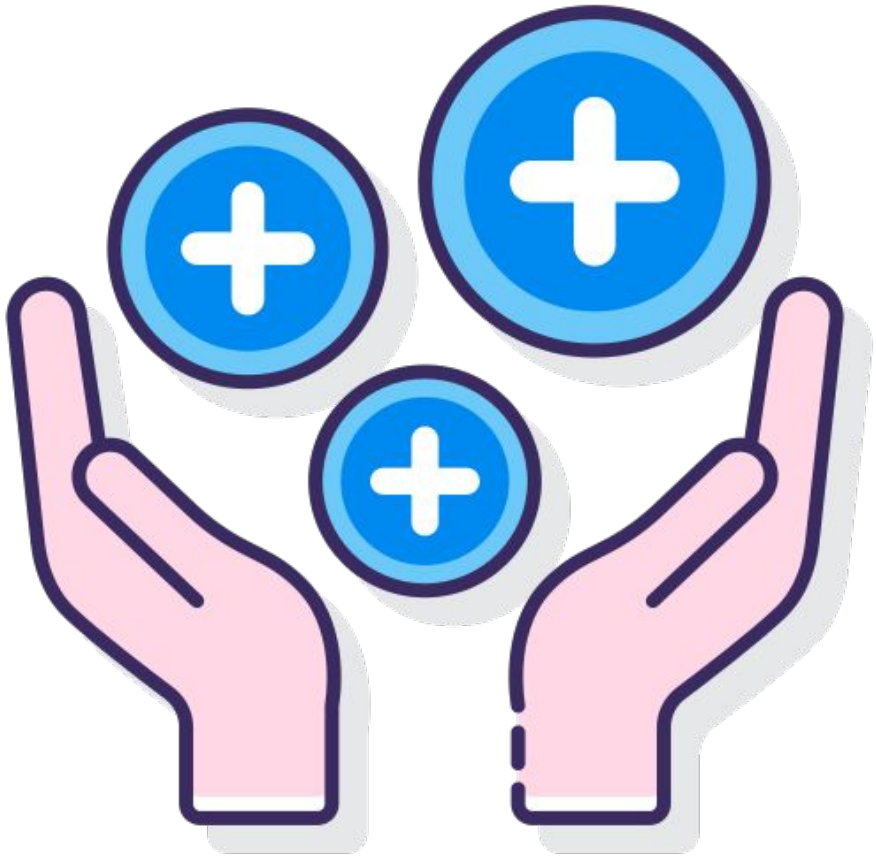


- **Autorización: Controlando el Acceso a Recursos**
  - La autorización se refiere a establecer qué acciones y recursos pueden acceder los usuarios autenticados en la API.
  - Es importante definir y aplicar políticas de autorización para garantizar la seguridad y protección de los datos.

- **Estrategias de Autorización:**
  - **Basada en Roles (Role-Based):** Los usuarios se asignan a roles específicos con permisos predefinidos. Ejemplo: administrador, usuario normal, invitado.
  - **Basada en Acciones (Action-Based):** Se definen permisos granulares para acciones individuales. Ejemplo: leer, escribir, actualizar, eliminar.
  - **Control de Acceso Basado en Atributos (Attribute-Based Access Control, ABAC):** Los permisos se basan en atributos y características del usuario, recurso y contexto.

# Beneficios de la Autenticación y Autorización de API

- Mejora la seguridad y protección de los datos sensibles.
- Permite controlar el acceso a los recursos y acciones específicas.
- Ayuda a cumplir con regulaciones y estándares de seguridad.
- Establece confianza entre la API y los usuarios.



# Autenticación con API Key


- Un API Key es una clave de autenticación utilizada para identificar y autorizar el acceso a una API.
- Se trata de un valor único que se proporciona al cliente de la API y se incluye en las solicitudes para verificar la autorización.

**Funcionalidad de API Key:**

- **Identificación:** El API Key permite identificar de manera única a un cliente o aplicación que realiza la solicitud a la API.
- **Autorización:** El API Key se utiliza para verificar si el cliente tiene permisos para acceder a los recursos o servicios de la API.

**¿Quién las genera?:**

- Las API Keys pueden ser generadas por el proveedor de la API y proporcionadas a los clientes, o pueden ser generadas automáticamente por el sistema.
- Se recomienda utilizar métodos seguros para la generación de API Keys, como la generación de claves criptográficamente seguras.

# Autenticación con API Key

**¿Cómo se usa?:**

- El cliente incluye el API Key en cada solicitud a la API, ya sea a través de la cabecera de autorización, un parámetro de consulta o un campo específico en el cuerpo de la solicitud.
- La API verifica la validez y autorización del API Key antes de procesar la solicitud.


**Beneficios**

- Fácil implementación.
- Control de acceso.
- Seguridad.
- Escalabilidad.

**Consideraciones de seguridad:**

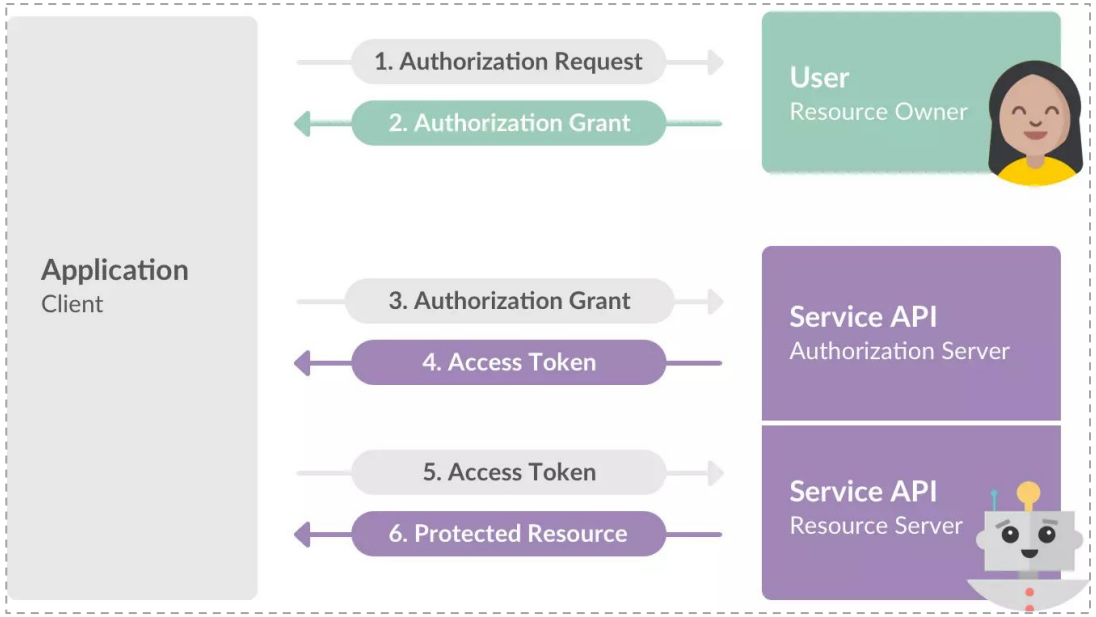
- Las API Keys deben ser tratadas como secretos y almacenadas de manera segura por los clientes.
- Se pueden implementar medidas adicionales, como límites de uso, renovación periódica de claves y restricciones de IP, para aumentar la seguridad de la API Key.

# Autenticación con OAuth 2.0


- OAuth 2.0 es un protocolo de autorización estándar utilizado para permitir a los usuarios otorgar acceso a sus datos a aplicaciones de terceros sin compartir sus credenciales.
- Proporciona un flujo seguro y confiable para autenticar y autorizar solicitudes de acceso a recursos protegidos en nombre de un usuario.
- Consiste en términos generales en generar un ACCESS TOKEN que nos da acceso a las APIs. Este ACCESS TOKEN no es fijo, sino que tiene un tiempo de vida y luego de eso es necesario generar uno nuevo.

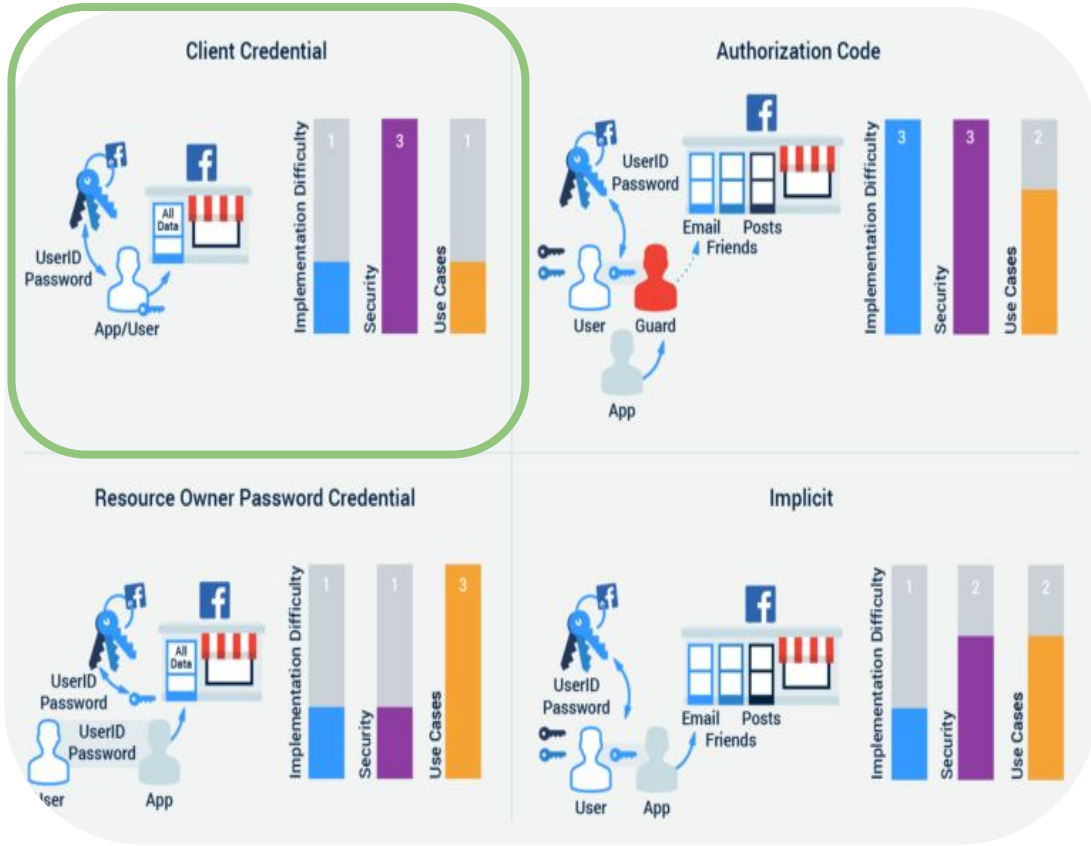
# Partes Involucradas en OAuth 2.0

- **Propietario del Recurso (Resource Owner):** El usuario final que posee los datos o recursos protegidos y otorga acceso a ellos.
- **Cliente (Client):** La aplicación de terceros que solicita acceso a los recursos en nombre del propietario del recurso.
- **Servidor de Autorización (Authorization Server):** El servidor que autentica al propietario del recurso y emite tokens de acceso al cliente.
- **Servidor de Recursos (Resource Server):** El servidor que almacena y protege los recursos a los que se accede.



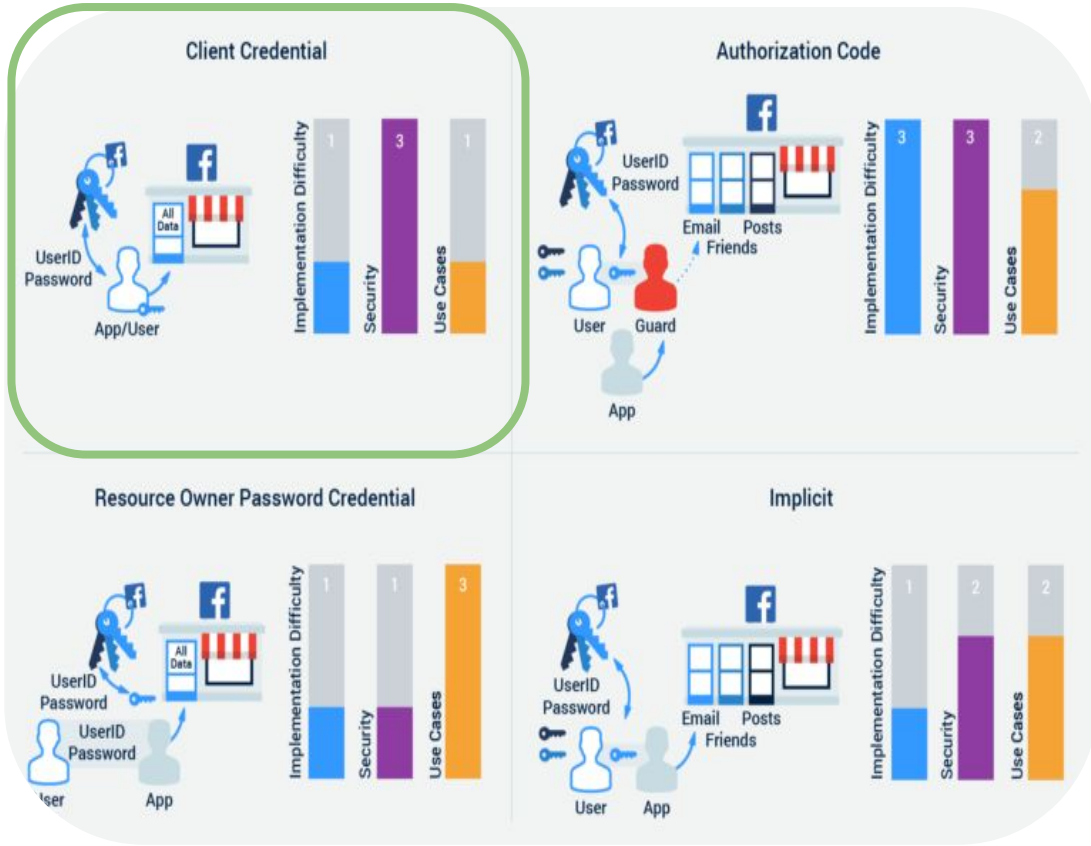
# OAuth 2.0: Flujos para Generar Tokens

- **Authorization Code Grant:**
  - Flujo más común y recomendado para aplicaciones web y móviles.
  - Requiere una interacción directa con el servidor de autorización para obtener un código de autorización y luego canjearlo por un token de acceso.
- **Implicit Grant:**
  - Flujo simplificado para aplicaciones JavaScript o móviles nativas.
  - El token de acceso se obtiene directamente en el proceso de redireccionamiento sin un código de autorización intermedio.
- **Resource Owner Password Credentials Grant:**
  - Flujo utilizado cuando el cliente obtiene directamente las credenciales del usuario.
  - Menos recomendado debido a posibles riesgos de seguridad, pero puede ser útil en ciertos escenarios.
- **Client Credentials Grant:**
  - Flujo utilizado por aplicaciones cliente confiables (sin participación del usuario).
  - El cliente autentica directamente con el servidor de autorización y obtiene un token de acceso.



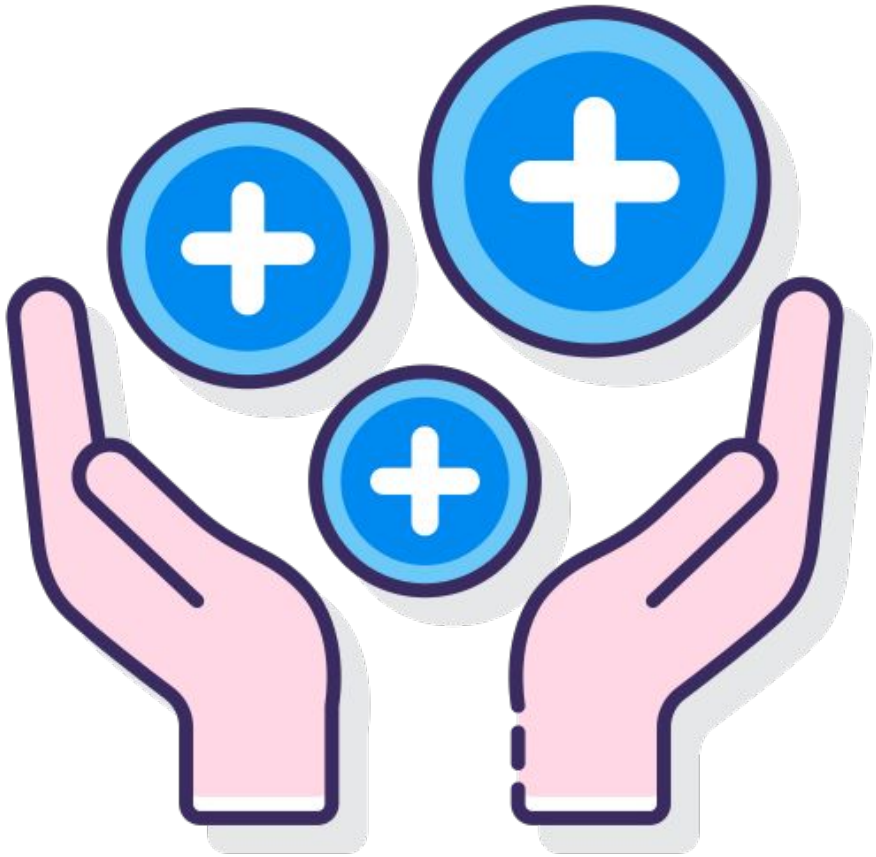
# OAuth 2.0: Flujos para Generar Tokens

- **Authorization Code Grant:**
  - Flujo más común y recomendado para aplicaciones web y móviles.
  - Requiere una interacción directa con el servidor de autorización para obtener un código de autorización y luego canjearlo por un token de acceso.
- **Implicit Grant:**
  - Flujo simplificado para aplicaciones JavaScript o móviles nativas.
  - El token de acceso se obtiene directamente en el proceso de redireccionamiento sin un código de autorización intermedio.
- **Resource Owner Password Credentials Grant:**
  - Flujo utilizado cuando el cliente obtiene directamente las credenciales del usuario.
  - Menos recomendado debido a posibles riesgos de seguridad, pero puede ser útil en ciertos escenarios.
- **Client Credentials Grant:**
  - Flujo utilizado por aplicaciones cliente confiables (sin participación del usuario).
  - El cliente autentica directamente con el servidor de autorización y obtiene un token de acceso.



# Beneficios de OAuth 2.0

- Permite a los usuarios controlar y revocar el acceso a sus datos por parte de aplicaciones de terceros.
- Proporciona una forma segura de compartir recursos protegidos sin compartir las credenciales del usuario.
- Estándar ampliamente adoptado y compatible con diferentes tipos de aplicaciones y plataformas.



# Autorización con OAuth 2.0: Scopes

- Scope es un mecanismo en OAuth 2.0 para **limitar el acceso** de una aplicación a la cuenta de un usuario. Una aplicación puede solicitar uno o más alcances, esta información luego se presenta al usuario en la pantalla de consentimiento y el token de acceso emitido a la aplicación se limitará a los alcances otorgados.
- La especificación de OAuth permite que el servidor de autorizaciones o el usuario modifique los alcances otorgados a la aplicación en comparación con lo que se solicita, aunque no hay muchos ejemplos de servicios que hagan esto en la práctica.
- Esta confirmación de Scopes se debe hacer en el **Servidor de Autorización** por cada API.
- Ejemplo:
    - **read:productos**: Permite al cliente acceder a los datos de productos.
    - **write:productos:** Permite al cliente cargar o modificar productos.
    - **read:vendedores:** Permite al cliente acceder a los datos de vendedores.
    - **write:vendedores:** Permite al cliente cargar o modificar vendedores.

# Servidor de Autorización

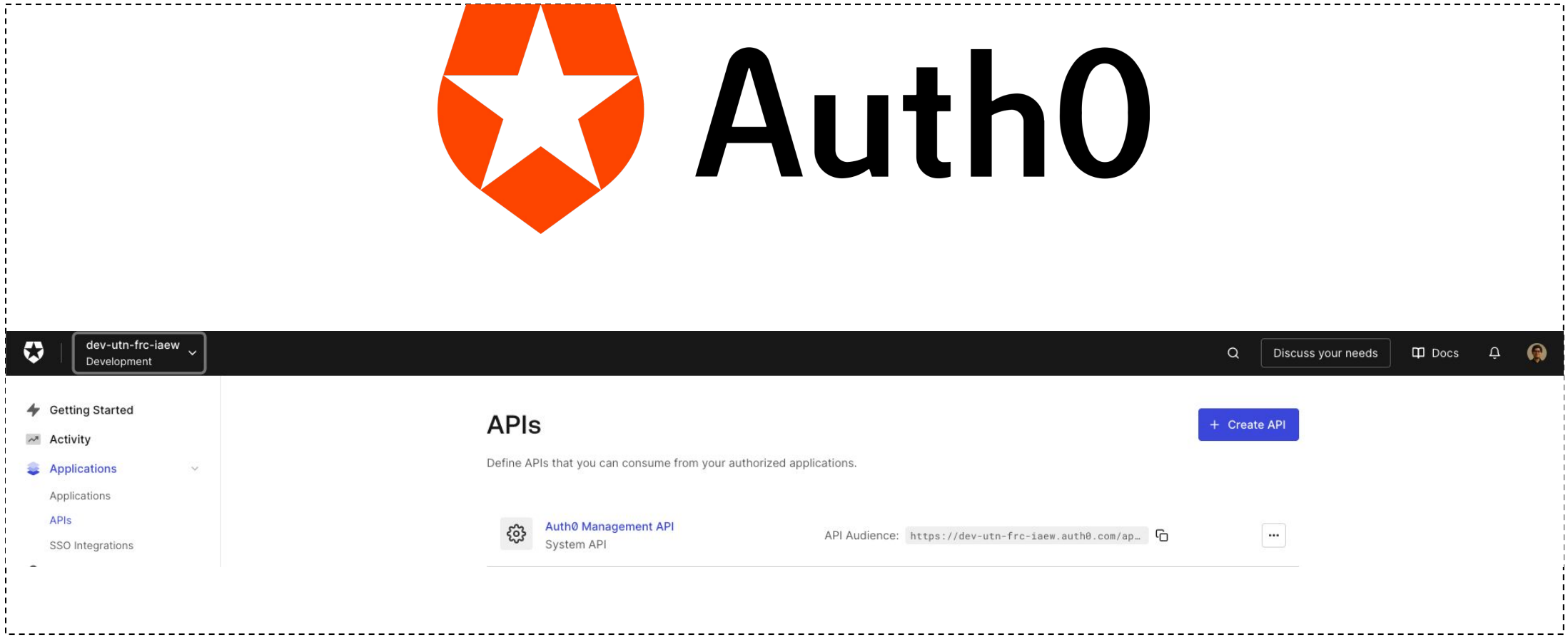
**¿Qué es un servidor de autorización?**

- Es un componente que gestiona y controla los permisos y privilegios de los usuarios en una aplicación o sistema.
- Se encarga de autenticar a los usuarios y proporcionarles tokens de acceso para acceder a recursos protegidos.
- Podemos crear nuestro propio servidor de Autorización o utilizar alguno disponible de la nube.

**Ejemplos de servidores de autorización populares:**

- **Auth0**
- **Okta**
- **Keycloak**
- **AWS Cognito**
- **Azure Active Directory**

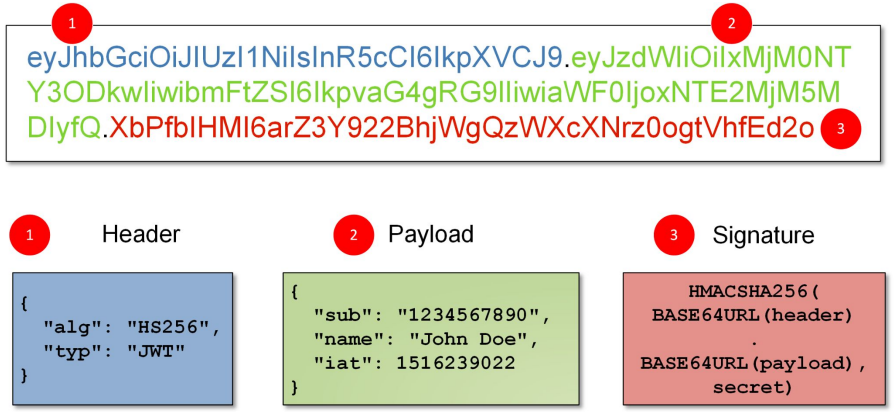
# Ejemplo Servidor de Autorización: Auth0



# JWT (JSON Web Tokens)

- JWT es un estándar abierto (RFC 7519) para transmitir información de forma segura como objetos JSON.
- Consiste en tres partes: encabezado (header), carga útil (payload) y firma (signature).
- https://jwt.io/
- **Importante!!!** Un JWT lo podemos usar como access token de OAuth 2.0.

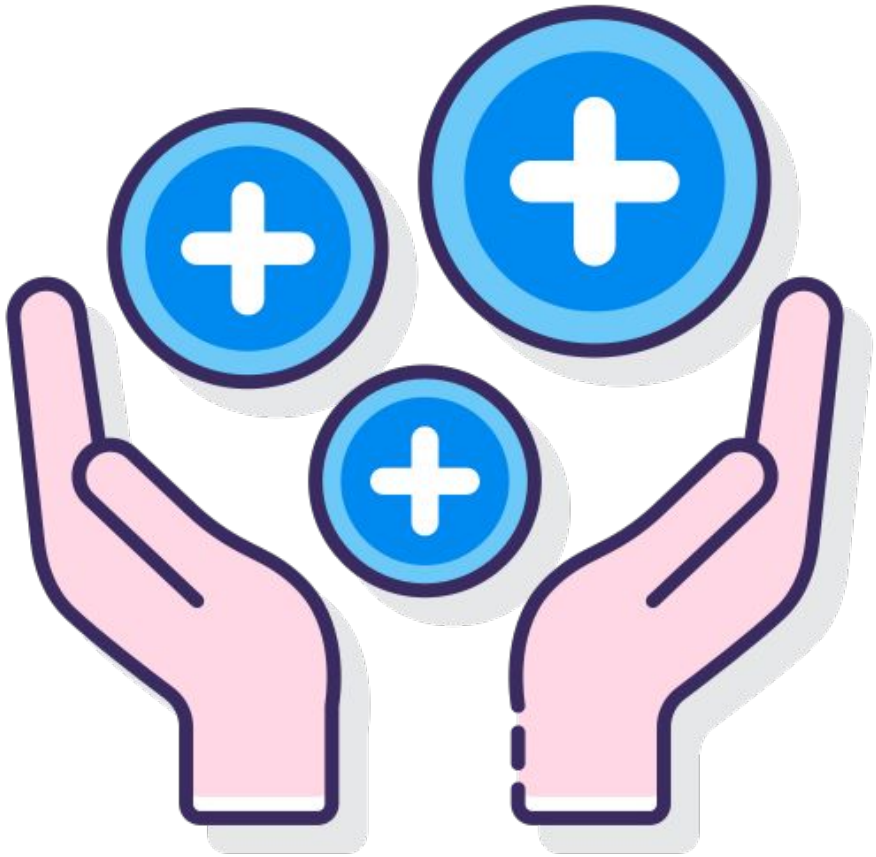
# Estructura de JWT




- **Encabezado (Header):**
  - Tipo de token: JWT
  - Algoritmo de firma utilizado (por ejemplo, HMAC SHA256, RSA, etc.)
- **Carga útil (Payload):**
  - Contiene la información que se desea transmitir.
  - Puede incluir datos como el identificador del usuario, roles, permisos, etc.
- **Firma (Signature):**
  - Verifica la integridad del token.
  - Se genera utilizando la clave secreta o la clave pública/privada según el algoritmo de firma.

# Beneficios de JWT

- **Seguridad:** Los JWT se pueden firmar digitalmente, lo que garantiza su integridad y autenticidad.
- **Portabilidad:** Los JWT son autónomos y autocontenido, lo que los hace ideales para comunicaciones entre servicios.
- **Escalabilidad:** Los JWT son fáciles de implementar y no requieren almacenamiento en el servidor.



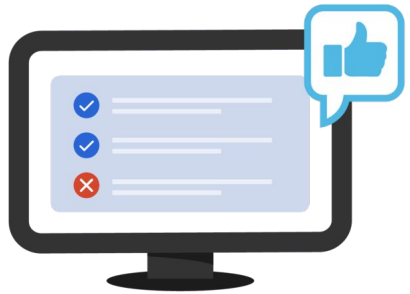
# Uso de JWT

**Uso de JWT en autenticación:**

- El **Servidor de Autorización** emite un JWT al autenticar al usuario.
- El cliente almacena el token y lo envía en cada solicitud subsiguiente.
- El **Servidor de Recursos** verifica la firma y la validez del token para autorizar la solicitud.


**Uso de JWT en autorización:**

- El **Servidor de Recursos** valida el JWT para extraer información sobre el usuario y sus permisos.
- Basado en los datos del token, el **Servidor de Recursos** permite o deniega el acceso a los recursos protegidos.



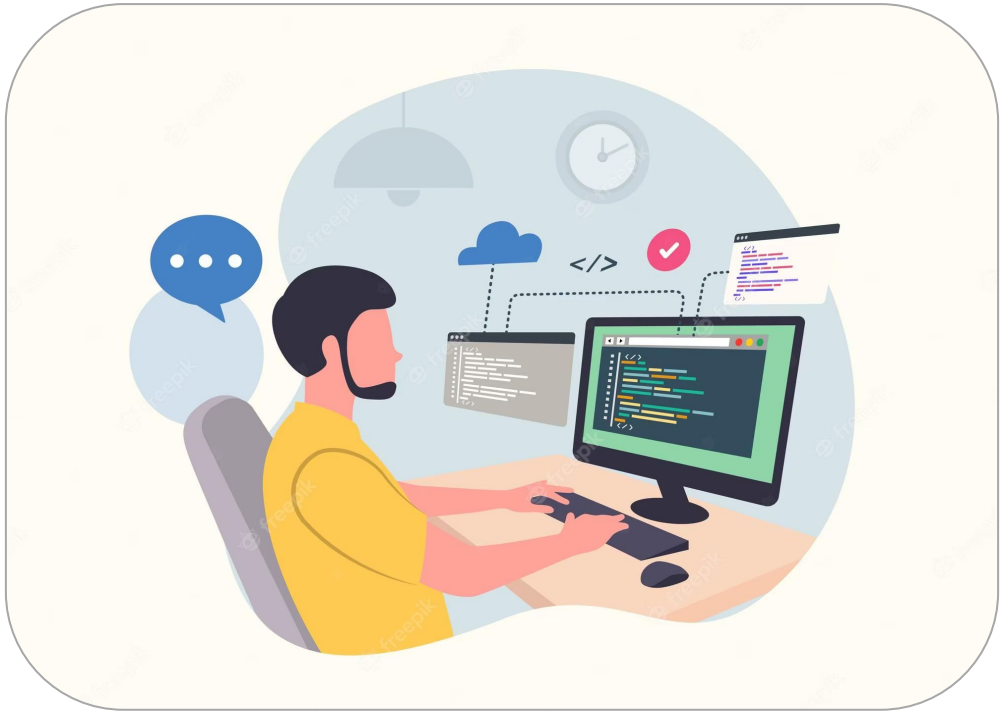
# Bueno, Vamo a Codea!!!

Vamos a utilizar la librería para resolver la autenticación y la autorización:
**express-oauth2-jwt-bearer**

1. Ejemplo Autenticación con OAuth 2.0 + JWT
2. Ejemplo Autorización OAuth 2.0.

# Actividad 6: Paso a Paso Seguridad API

- Seguir las instrucciones de la actividad publicada en la UVE.



# MUCHAS GRACIAS

ANDÉN Centro de Innovación y Emprendimientos Tecnológicos

SECRETARÍA DE EXTENSIÓN UNIVERSITARIA UTN - FRC SEU

UTN Facultad Regional Córdoba

Agencia CÓRDOBA JOVEN

CÓRDOBA entre todos