



SEGURIDAD EN APLICACIONES WEB

TABLA DE CONTENIDO

- 01 Autenticación y Autorización**
- 02 CORS**
- 03 Protección Contra Ataques Comunes**
- 04 Ejemplo API**

AUTENTICACIÓN

La autenticación es el proceso de verificar la identidad de un usuario. En las aplicaciones web, esto se suele lograr mediante credenciales como un nombre de usuario y una contraseña.

Otros métodos más avanzados incluyen autenticación de dos factores (2FA), donde el usuario debe proporcionar una segunda prueba, como un código enviado a su teléfono. Este paso asegura que quien accede es quien dice ser.

AUTORIZACIÓN

Una vez autenticado, el sistema determina qué acciones o recursos puede acceder el usuario, en función de su rol o permisos. Este proceso se llama autorización.

Roles y permisos

- Roles: Permiten agrupar usuarios bajo ciertas capacidades o privilegios comunes, por ejemplo, "Administrador", "Usuario estándar", "Invitado".
- Permisos: Especifican las acciones que un usuario o rol puede realizar, como "leer", "escribir", "eliminar", etc.

JWT (JSON WEB TOKENS)

JWT es un estándar abierto basado en JSON que permite el intercambio seguro de información entre las partes como un objeto JSON.

Se utiliza principalmente en sistemas de autenticación basada en tokens, donde, en lugar de mantener una sesión en el servidor, se utiliza un token que se firma con una clave secreta.



JWT (JSON WEB TOKENS)

Componentes del JWT:

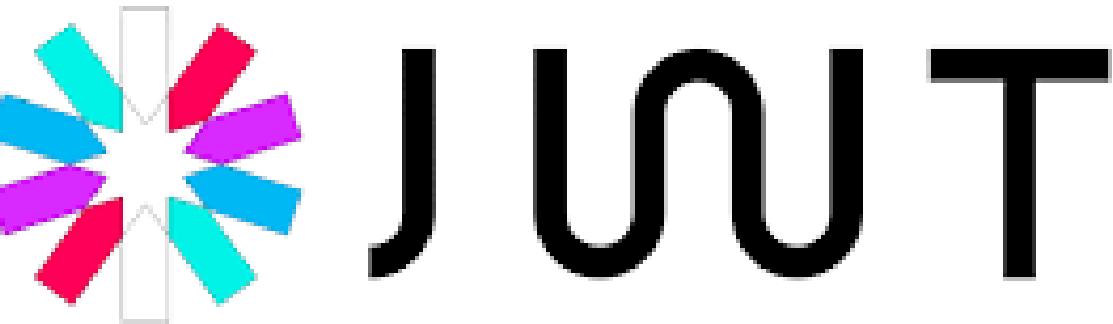
- Header: Indica el tipo de token y el algoritmo de encriptación.
- Payload: Contiene la información que se quiere transmitir (como el ID de usuario).
- Signature: Asegura que el token no ha sido alterado.



JWT (JSON WEB TOKENS)

Flujo de uso de JWT:

- El servidor genera un JWT tras la autenticación exitosa.
- El cliente almacena este JWT, generalmente en el almacenamiento local o cookies.
- En cada solicitud posterior, el cliente envía el JWT para que el servidor valide la identidad del usuario.



CORS

CORS (Cross-Origin Resource Sharing) es un mecanismo de seguridad que controla el acceso de recursos entre diferentes orígenes. Este permite a los servidores especificar cuáles dominios pueden acceder a los recursos del servidor.

CORS permite solicitudes entre dominios mediante el uso de políticas de acceso controladas por el servidor.

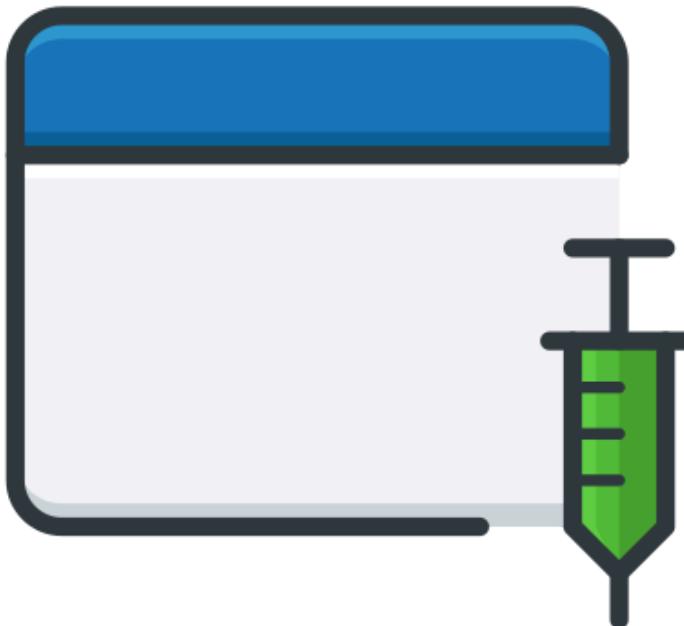


PROTECCIÓN CONTRA ATAQUES COMUNES

xss (Cross-Site Scripting)

Inyección de scripts maliciosos en páginas web que pueden robar información del usuario o ejecutar acciones en su nombre.

Este se previene mediante la validación y sanitización de entradas para evitar la ejecución de scripts maliciosos.

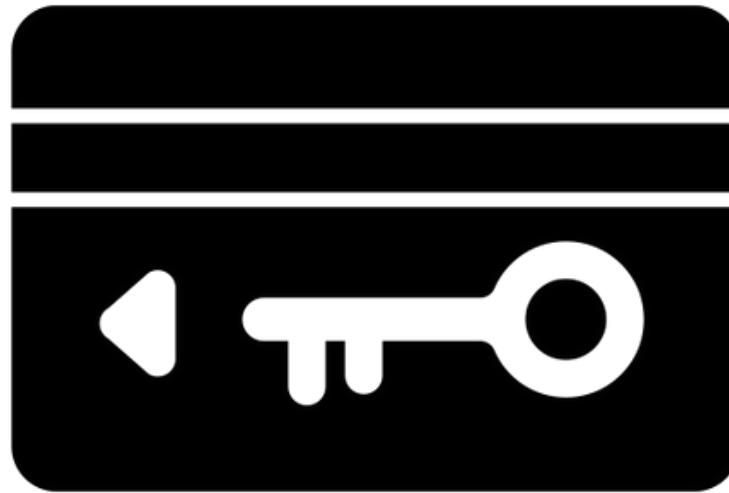


PROTECCIÓN CONTRA ATAQUES COMUNES

CSRF (Cross-Site Request Forgery)

Ataques que engañan a un usuario autenticado para realizar acciones no deseadas en una aplicación web.

Este se previene mediante el uso de tokens CSRF para cada solicitud para verificar la autenticidad de la misma.



PROTECCIÓN CONTRA ATAQUES COMUNES

SQL Injection

Inyección de consultas SQL maliciosas para manipular la base de datos de formas no autorizadas.

Este se previene mediante el uso de ORM y consultas preparadas.



IMPORTACIÓN DE LIBRERÍAS



```
// Importar Express, Mongoose, y otras  
dependencias  
const express = require("express");  
const mongoose = require("mongoose");  
const jwt = require("jsonwebtoken");  
const bcrypt = require("bcrypt");  
const app = express();  
  
// Middleware para manejar datos en formato JSON  
app.use(express.json());  
  
// Conectar a MongoDB  
mongoose.connect(MONGODB_URI, {});
```

DEFINICIÓN DE ESQUEMAS DE DATOS



```
// Definir el esquema y el modelo para Items
const itemSchema = new mongoose.Schema({
  name: String,
});
const Item = mongoose.model("Item", itemSchema);

// Definir el esquema y el modelo para Usuarios
const userSchema = new mongoose.Schema({
  username: { type: String, unique: true },
  password: String,
});
const User = mongoose.model("User", userSchema);
```

JSON Y AUTENTICACIÓN CON JWT

```
// Middleware para autenticar token JWT
const authenticateToken = (req, res, next) => {
  const authHeader = req.headers["authorization"];
  const token = authHeader && authHeader.split(" ")[1];
  if (token == null) return res.sendStatus(401);

  jwt.verify(token, "your_jwt_secret", (err, user) => {
    if (err) return res.sendStatus(403);
    req.user = user;
    next();
  });
};
```

REGISTRO DE USUARIOS

```
● ● ●

// Ruta para registrar nuevos usuarios
app.post("/register", async (req, res) => {
  try {
    const hashedPassword = await
bcrypt.hash(req.body.password, 10);
    const newUser = new User({
      username: req.body.username,
      password: hashedPassword,
    });
    await newUser.save();
    res.status(201).send("Usuario registrado");
  } catch (err) {
    res.status(400).send(err.message);
  }
});
```

AUTENTICACIÓN Y GENERACIÓN DE TOKEN

```
// Ruta para autenticar usuarios y obtener un token JWT
app.post("/login", async (req, res) => {
  const user = await User.findOne({ username:
    req.body.username });
  if (user == null) return res.status(400).send("Usuario
no encontrado");

  try {
    if (await bcrypt.compare(req.body.password,
      user.password)) {
      const token = jwt.sign({ username: user.username },
        "your_jwt_secret");
      res.json({ token });
    } else {
      res.status(403).send("Contraseña incorrecta");
    }
  } catch {
    res.status(500).send("Error en el servidor");
  }
});
```

CREAR UN ÍTEM (POST /ITEMS)

```
● ● ●  
  
// Crear (Create) un nuevo ítem  
app.post("/items", authenticateToken, async (req, res) =>  
{  
  try {  
    const newItem = new Item({  
      name: req.body.name,  
    });  
    await newItem.save();  
    res.status(201).json(newItem);  
  } catch (err) {  
    res.status(400).send(err.message);  
  }  
});
```

LEER TODOS LOS ÍTEMES (GET /ITEMS)

```
// Leer (Read) todos los ítems
app.get("/items", authenticateToken, async (req, res) => {
  try {
    const items = await Item.find();
    res.json(items);
  } catch (err) {
    res.status(500).send(err.message);
  }
});
```

LEER UN ÍTEM POR ID (GET /ITEMS/ID)



```
// Leer (Read) un ítem por ID
app.get("/items/:id", authenticateToken, async (req, res)
=> {
  try {
    const item = await Item.findById(req.params.id);
    if (!item) return res.status(404).send("El ítem no fue
encontrado.");
    res.json(item);
  } catch (err) {
    res.status(500).send(err.message);
  }
});
```

ACTUALIZAR UN ÍTEM POR ID (PUT /ITEMS/:ID)

```
// Actualizar (Update) un ítem por ID
app.put("/items/:id", authenticateToken, async (req, res)
=> {
  try {
    const item = await Item.findById(req.params.id);
    if (!item) return res.status(404).send("El ítem no fue
encontrado.");

    item.name = req.body.name;
    await item.save();
    res.json(item);
  } catch (err) {
    res.status(400).send(err.message);
  }
});
```

ELIMINAR UN ÍTEM POR ID (DELETE /ITEMS/:ID)



```
// Eliminar (Delete) un ítem por ID
app.delete("/items/:id", authenticateToken, async (req,
res) => {
  try {
    const item = await
Item.findByIdAndDelete(req.params.id);
    if (!item) return res.status(404).send("El ítem no fue
encontrado.");
    res.json(item);
  } catch (err) {
    res.status(500).send(err.message);
  }
});
```

INICIAR EL SERVIDOR



```
// Iniciar el servidor
app.listen(3000, () => {
  console.log("Servidor corriendo en el puerto 3000");
});
```

GRACIAS

```
2 const fetch = require('node-fetch');
3 const log = require('winston');
4 let embed;
5
6
7 function transform(data) {
8     // Promise.resolve is required here
9     return transformPromise(data);
10 }
11
12 function removeHeader(data) {
13     return prev.then(() => {
14         $(':header').remove();
15         const children = $(data).children();
16         if ($(':header').length > 0) {
17             $(header).append(children);
18             $(children).remove();
19         }
20         return header;
21     });
22 }
23
24 return Promise.resolve(data);
```