# Requerimientos

* [Postman](https://www.postman.com/downloads/)
* Visual Studio Code
* NodeJs

# Cookies

La idea de una cookie es simple: el servidor envía un poco de información y el navegador lo almacena durante un período de tiempo configurable.

### Las cookies no son secretas para el usuario

Todas las cookies que el servidor envía al cliente están disponibles para que el cliente las vea. No hay razón por la que no puedas enviar algo encriptado para proteger su contenido, pero rara vez hay necesidad de esto (¡al menos si no estás haciendo nada malo!).

Las cookies firmadas, de las que hablaremos más adelante, pueden ofuscar el contenido de la cookie, pero esto no es criptográficamente seguro de miradas indiscretas.

### El usuario puede eliminar o deshabilitar las cookies

Los usuarios tienen control total sobre las cookies y los navegadores permiten eliminar las cookies una por una o individualmente. A menos que no esté tramando nada bueno, no hay una razón real para que los usuarios hacer esto, pero es útil durante las pruebas. Los usuarios también pueden rechazar las cookies, lo cual es más problemático: solo las aplicaciones web más simples pueden prescindir de las cookies

### Las cookies regulares pueden ser manipuladas

Cada vez que un navegador realiza una solicitud a su servidor que tiene una cookie asociada, y confía ciegamente en el contenido de esa cookie, se está abriendo para ataque. El colmo sería ejecutar código contenido en una cookie. Para asegurarse de que las cookies no se manipulen, use cookies firmadas.

### Las cookies se pueden utilizar para ataques

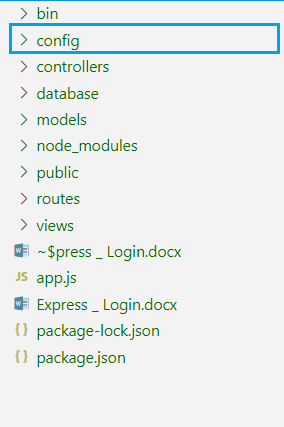
Recientemente ha surgido una categoría de ataques denominados ataques de secuencias de comandos en sitios cruzados (XSS). Una técnica de ataques XSS consiste en JavaScript malicioso que modifica el contenido de las cookies. Esta es una razón adicional para no confiar en el contenido de las cookies que regresan a su servidor. El uso de cookies firmadas ayuda (la manipulación será evidente en una cookie firmada, ya sea que el usuario o JavaScript malicioso la hayan modificado), hay también una configuración que especifica que las cookies deben ser modificadas solo por el servidor. Las cookies pueden tener una utilidad limitada, pero ciertamente son más seguras.

Las cookies no son mágicas: cuando el servidor desea que el cliente almacene una cookie, envía un encabezado llamado Set-Cookie que contiene nombre/valor , y cuando un cliente envía una solicitud a un servidor para el que tiene cookies, envía múltiples encabezados de solicitud de cookies que contienen el valor de las galletas

Instalar el módulo Cookie-Parser

**npm install --save cookie-parser**

Crear una carpeta de configuración llamado **conf** en la raíz del proyecto



Agregar el siguiente contenido al archivo

|  |
| --- |
| module.exports = {      cookieSecret: 'UtNg#2020',  }; |

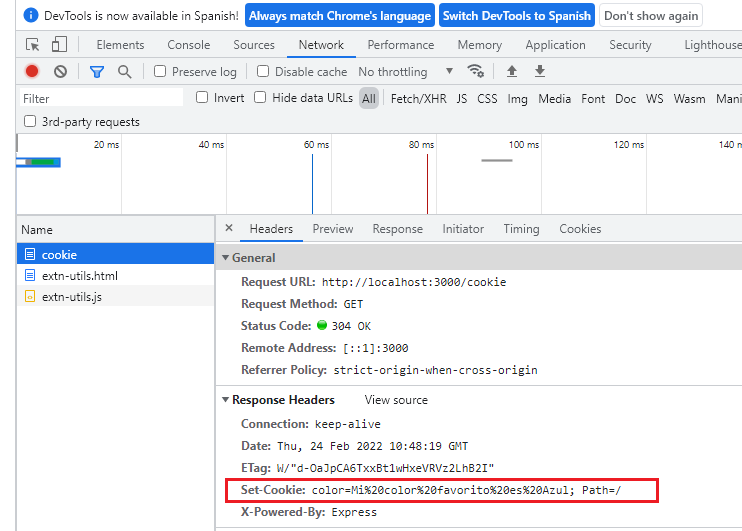
Agregar la configuración en archivo **app.js**

|  |
| --- |
| var cookieParser = require('cookie-parser');  var credentials = require('./config/credentials');  app.use(cookieParser(credentialCookie.cookieSecret)); |

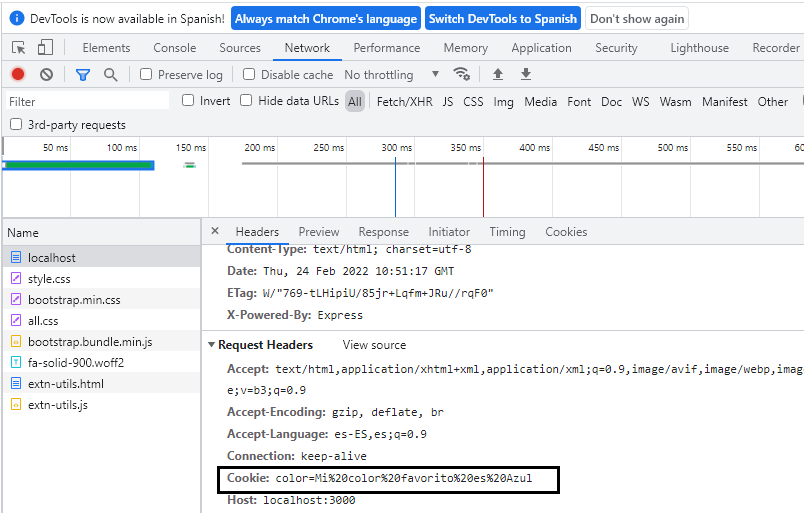
Ubicar el archivo de ruta **index.js** y agregar el siguiente path

|  |
| --- |
| router.get('/cookie',(req, res) => {      res.cookie('color' , 'Mi color favorito es Azul');      res.send('Cookie is set');  }); |

Ingresa la URL <http://localhost:3000/cookie> e inspecciona el header

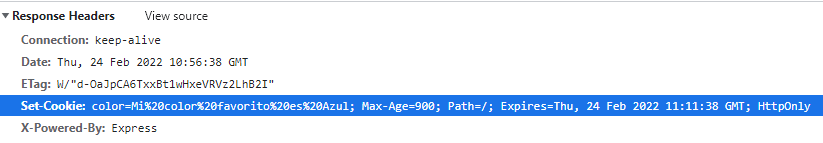


El navegador envía de vuelta la información de las cookies para comprobar ingrese ahora la URL <http://localhost:3000/> , inspeccionar



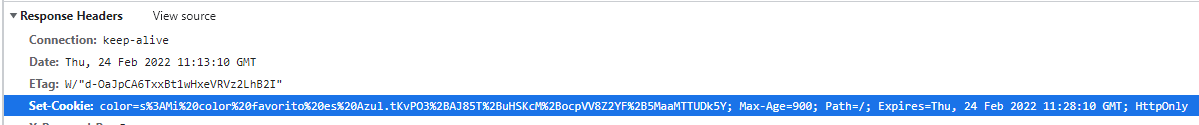
Podemos modificar el tiempo de expiración

|  |
| --- |
| res.cookie('color' , 'Mi color favorito es Azul', { maxAge: 900000, httpOnly: true}); |

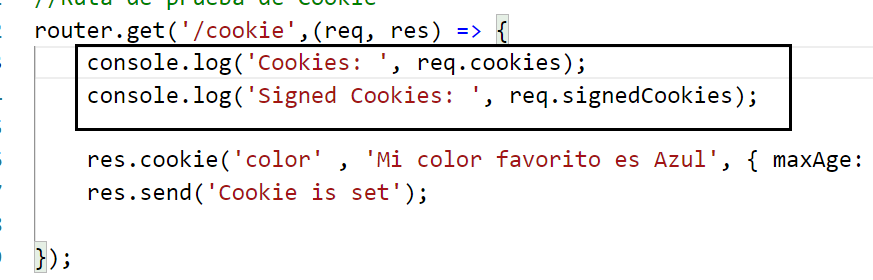


Ahora vamos a establecer una Cookie firmada

|  |
| --- |
| res.cookie('color' , 'Mi color favorito es Azul', { maxAge: 900000, httpOnly: true, signed:true}); |



Código para verificar Cookies





Desafío:

* Establece al menos dos cookies.
* Inspeccionar si se guardaron las cookies
* Firmar las cookies
* Imprimir las cookies a consola

Archivo .env

* Instalar el módulo dotenv

**Npm install dotenv --save**

Crear archivo con extensión .env en la carpeta raíz del proyecto agregar el siguiente contenido

**DB\_URL** = mongodb+srv://gds0351:gds0351@cluster0.jnmbt.mongodb.net/login?retryWrites=true&w=majority

**KEY\_SECRET** = UtNg#2022

Configurar DotEnv en archivo app.js, agregar la instrucción en la parte superior del archivo

require ('dotenv').config();

Modificar el archivo de conexión a BD para hacer referencia a la URL

**require('dotenv').config();**

function connMongo() {

    var mongoose = require('mongoose');

**var mongoDB = process.env.DB\_URL;**

    console.log(process.env.DB\_URL);

    mongoose.connect(mongoDB, { useNewUrlParser: true, useUnifiedTopology: true });

    var db = mongoose.connection;

    db.on('error', console.error.bind(console, 'Error de Conexion a MongoDB'));

}

module.exports = connMongo

Sesiones

* Instalar el modulo [express-session](https://www.npmjs.com/package/express-session)

**npm install express-session –save**

Configurar el express-session en app.js

**const session = require('express-session');**

//Manejo session

app.set('trust proxy', 1);

app.use(session({

    secret: process.env.KEY\_SECRET,

    resave: false,

    saveUninitialized: true,

    cookie: { secure: true }

}));

# Encriptamiento de la información

Instalar el módulo bcrypt

**npm install --save bcrypt**

Modificar el modelo user.js para encriptar el campo Password

const mongoose = require('mongoose');

const Schema = mongoose.Schema;

**const bcrypt = require('bcrypt');**

const UserSchema = new Schema({

username: { type: String, minlength: 8, maxlength: 15, required: true },

password: { type: String, required: true },

email: { type: String }

});

**UserSchema.pre('save', function(next) {**

**const user = this;**

**bcrypt.hash(user.password, 10, (error, hash) => {**

**user.password = hash**

**next()**

**});**

**});**

module.exports = mongoose.model('User', UserSchema);

# Modificar el método de controlador que verifica Ingreso

Agregar la referencia de bcrypt

**const bcrypt = require('bcrypt')**

Modficar la función de verificación de Login

exports.user\_login\_verify = function(req, res) {

let usuario = req.body.username;

let pass = req.body.password;

console.log('Usuario: ' + usuario + " Pass: " + pass);

if (usuario && pass) {

**User.findOne({ username: usuario }, (error, user) => {**

**if (user) {**

**bcrypt.compare(pass, user.password, (error, same) => {**

**if (same) { //Coinciden**

**//Almacena datos a sesión**

req.session.usuario = usuario;

**res.render('index', { title: 'Bienvenido' });**

**} else {**

**let data = {**

**title: 'Ingresar al Sistema',**

**message: 'Usuario o contraseña incorrecto',**

**layout: false**

**}**

**res.render('login', data);**

**}**

**})**

} else {

**let data = {**

**title: 'Ingresar al Sistema',**

**message: 'Usuario No Existe',**

**layout: false**

**}**

**res.render('login', data);**

}

})

} else {

**let data = {**

**title: 'Ingresar al Sistema',**

**message: 'Usuario o Contraseña Incorrecto',**

**layout: false**

**}**

**res.render('/', data);**

}

};

Protegiendo páginas con Middleware

Si ingresa la url <http://localhost:3000/home> permite ingresar a la página sin pasar por el Login.

Despliega pantalla de bienvenida

# Implementando verificación de páginas

* Crear carpeta middleware dentro del proyecto a la misma altura de controllers
* Crear archivo **authentication.js** dentro de carpeta middleware.

const User = require('../models/user')

module.exports = (req, res, next) => {

    console.log('Verificando Inicio de Sesión');

    User.findById(req.session.usuario, (error, user) => {

        if (error || !user)

            return res.redirect('/')

        next()

    })

};

* Modificamos las rutas que sean necesarias para que primero verifique si usuario está LogIn

/\* No es necesario verificar \*/

router.get('/', controller.user\_login);

router.get('/formRegister', controller.register\_show);

router.post('/addUser', controller.user\_register);

router.post('/verify', controller.user\_login\_verify);

//Protege páginas

router.get('/home', authMiddleware, controller.user\_home);

router.get('/logout', authMiddleware, controller.user\_logout);

Prueba la funcionalidad agregando un re