

# Unitat 7. Serveis web

2n DAW - IES María Enríquez

# Resultats d'aprenentatge i criteris d'avaluació

**RA7. Desenvolupa serveis web reutilitzables i accessibles mitjançant protocols web, verificant el seu funcionament.**

Criteris d'avaluació:

- a) S'han reconegut les **característiques pròpies i l'àmbit d'aplicació dels serveis web**.
- b) S'han reconegut els **avantatges d'utilitzar serveis web** per a proporcionar accés a funcionalitats incorporades a la lògica de negoci d'una aplicació.
- c) S'han identificat les tecnologies i els protocols implicats en el **consum de serveis web**.
- d) S'han utilitzat els **estàndards i architectures més difosos** i implicats en el desenvolupament de serveis web.
- e) S'ha programat un **servei web**.
- f) S'ha **verificat el funcionament del servei web**.
- g) S'ha **consumit el servei web**.
- h) S'ha **documentat un servei web**.

# Resultats d'aprenentatge i criteris d'avaluació

**RA8. Genera pàgines web dinàmiques analitzant i utilitzant tecnologies i frameworks del servidor web que afigen codi al llenguatge de marques.**

Criteris d'avaluació:

- a) S'han identificat les diferències entre l'**execució de codi en el servidor i en el client web**.
- b) S'han reconegut els avantatges d'**unir totes dues tecnologies** en el procés de desenvolupament de programes.
- c) S'han identificat les **tecnologies i frameworks relacionades amb la generació per part del servidor** de pàgines web amb guions embeguts.
- d) S'han utilitzat aquestes **tecnologies i frameworks per a generar pàgines web** que incloguen **interacció amb l'usuari**.
- e) S'han utilitzat aquestes tecnologies i frameworks, per a generar pàgines web que incloguen **verificació** de formularis.
- f) S'han utilitzat aquestes tecnologies i frameworks per a generar pàgines web que incloguen **modificació dinàmica del seu contingut** i la seua estructura.
- g) S'han aplicat aquestes tecnologies i frameworks en la **programació d'aplicacions web**.

# Índex de la unitat

---

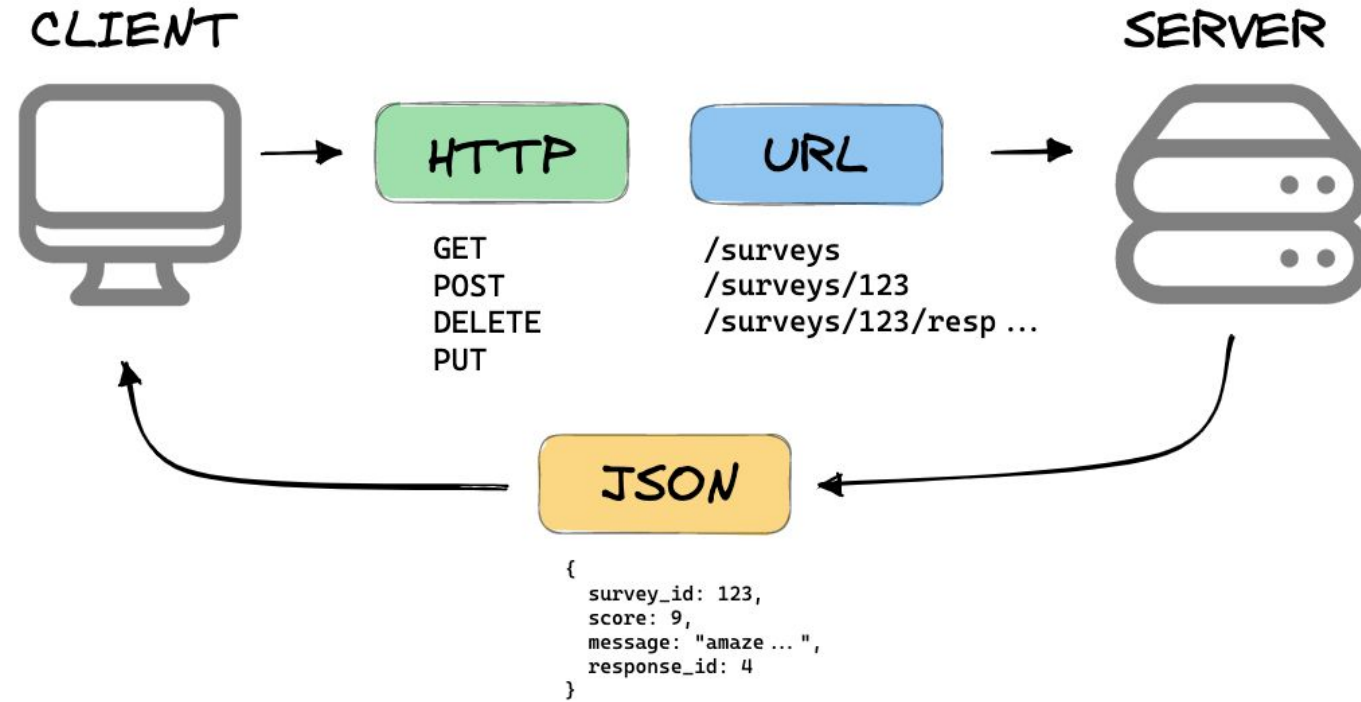
1. Concepte de servei REST
2. Creació de serveis REST
3. Prova de serveis REST
4. Autenticació basada en tokens



# 1. Concepte de servei REST

---

WHAT IS A REST API?



# 1. Concepte de servei REST | **codis d'estat**

---

**1XX** petició incompleta

**2XX** petició vàlida.

**200** tot ok

**201** insert o nou recurs ok

**204** tot ok, però no es retorna res.

**3XX** redirecció:

**301** recurs s'ha canviat permanentment a altra URL

**304** recurs no ha canviat des de l'última vegada. Ús caché.

# 1. Concepte de servei REST

**4xx** error:

**404** no existeix el recurs sol·licitat

**403** no autoritzat

**400** dades de petició no vàlides (formulari)

**5xx** error en el servidor:

**500** error intern

**504** timeout

## HTTP STATUS CODES

### 2xx Success

**200**

Success / OK

### 3xx Redirection

**301**

Permanent Redirect

**302**

Temporary Redirect

**304**

Not Modified

### 4xx Client Error

**401**

Unauthorized Error

**403**

Forbidden

**404**

Not Found

**405**

Method Not Allowed

### 5xx Server Error

**501**

Not Implemented

**502**

Bad Gateway

**503**

Service Unavailable

**504**

Gateway Timeout

# 1. Concepte de servei REST | **servei REST**

---

Estil d'**arquitectura REST** (representational state transfer) es **caracteritza** per:

- **No cal** mantenir **l'estat** entre el client i el servidor
- Els recursos s'identifiquen amb **URIs**
- Operacions: **GET** | **POST**(insert) | **PUT**(update) | **DELETE**
- Resposta en **JSON o XML**

**Avantatges:**

Interoperabilitat | Simplicitat | Escalabilitat



# 1. Concepte de servei REST | **format JSON**

---

Javascript Object Notation: objecte de javascript en cadenes de text:

```
let persona = {  
  nombre: "Nacho",  
  edad: 39  
};
```

```
{"nombre": "Nacho", "edad": 39}
```

```
let personas = [  
  { nombre: "Nacho", edad: 39},  
  { nombre: "Mario", edad: 4},  
  { nombre: "Laura", edad: 2},  
  { nombre: "Nora", edad: 10}  
];
```

```
[{"nombre": "Nacho", "edad": 39},  
 {"nombre": "Mario", "edad": 4},  
  {"nombre": "Laura", "edad": 2},  
 {"nombre": "Nora", "edad": 10}]
```

## 2. Creació serveis REST | controladors d'API

---

Creem el controlador que el deixarà en la ruta App\Http\Controller\Api

```
php artisan make:controller Api/LibroController --api --model=Libro
```

Ens crearà un controlador amb totes les **funcions** que necessitem:

index → GET

store → POST

show → GET

update → PUT

destroy → DELETE

## 2. Creació serveis REST | rutes d'API

---

A partir de la versió 10 de Laravel, cal instal·lar el mòdul API si es preten utilitzar el servidor com a servidor d'API REST:

```
php artisan install:api
```

A més en `bootstrap/app.php` afegim el nou arxiu de configuració de routes per a l'API:

```
return Application::configure(basePath: dirname(__DIR__))
    ->withRouting(
        web: __DIR__ . '/../routes/web.php',
        api: __DIR__ . '/../routes/api.php',
        commands: __DIR__ . '/../routes/console.php',
        health: '/up',
```

## 2. Creació serveis REST | rutes d'API

---

En el arxiu `routes/api.php` crearem una ruta associada al controlador de recursos:

```
use App\Http\Controllers\Api\LibroController;  
...  
Route::apiResource('libros', LibroController::class);
```

D'aquesta manera s'ha definit la ruta `api/libros`

```
php artisan route:list
```

## 2. Creació serveis REST | **serveis GET**

---

En el mètode `index` del controlador:

```
public function index()
{
    $libros = Libro::get();
    return $libros;
}
```

Si accedim a `api/libros` obtindrem el llistat de llibres en format JSON (JSON formatter)

## 2. Creació serveis REST | **serveis GET**

---

En el mètode `show` del controlador:

```
public function show(Libro $libro)
{
    return $libro;
}
```

Si accedim a `api/libros/1` obtindrem les dades d'eixe llibre

## 2. Creació serveis REST | personalització serveis GET

Afegir clàusules `hidden` per evitar enviar algun camp en el model:

```
protected $hidden = ['password'];
```

Definir les dades a enviar amb un array:

```
public function show(Libro $libro)
{
    return [
        'titulo' => $libro->titulo,
        'editorial' => $libro->editorial
    ];
}
```

## 2. Creació serveis REST | personalització serveis GET

```
public function index()  
{  
    $libros = Libro::with('autor')->get();  
    return $libros->map(function ($libro) {  
        return [  
            'titulo' => $libro->titulo,  
            'editorial' => $libro->editorial,  
            'autor' => $libro->autor->nombre  
        ];  
    });  
}
```



## 2. Creació serveis REST | personalització serveis GET

Personalitzar el codi de resposta amb `response() -> json(...)`

```
public function index()  
{  
    $libros = Libro::get();  
    return response()->json($libros, 200);  
}  
...  
public function show(Libro $libro)  
{  
    return response()->json($libro, 200);  
}
```

## 2. Creació serveis REST | resta de serveis

Mètode `store` → POST

```
public function store(Request $request)
{
    $libro = new Libro();
    $libro->titulo = $request->titulo;
    $libro->editorial = $request->editorial;
    $libro->precio = $request->precio;
    $libro->autor()->associate(Autor::findOrFail($request->autor_id));
    $libro->save();

    return response()->json($libro, 201);
}
```

## 2. Creació serveis REST | resta de serveis

---

Mètode `update` → PUT

```
public function update(Request $request, Libro $libro)
{
    $libro->titulo = $request->titulo;
    $libro->editorial = $request->editorial;
    $libro->precio = $request->precio;
    $libro->autor()->associate(Autor::findOrFail($request->autor_id));
    $libro->save();

    return response()->json($libro, 200);
}
```

## 2. Creació serveis REST | resta de serveis

---

Mètode `destroy` → DELETE

```
public function destroy(Libro $libro)
{
    $libro->delete();
    return response()->json(null, 204);
}
```

```
public function destroy(Libro $libro)
{
    $libro->delete();
    return response()->json($libro);
}
```

## 2. Creació serveis REST | validació de dades

---

Podem validar mitjançant els corresponents Requests que vam crear utilitzant la classe `App\Http\Requests\LibroRequest`:

```
public function store(LibroRequest $request)
{
    ...
}
...
public function update(LibroRequest $request, Libro $libro)
{
    ...
}
```

## 2. Creació serveis REST | respostes d'error

---

Per capturar i personalitzar les respostes d'error haurem d'editar l'arxiu `bootstrap/app.php` de la següent forma:

```
use Illuminate\Foundation\Application;
use Illuminate\Foundation\Configuration\Exceptions;
use Illuminate\Foundation\Configuration\Middleware;
use Symfony\Component\HttpKernel\Exception\NotFoundHttpException;
use Illuminate\Validation\ValidationException;
use Illuminate\Http\Request;
use Illuminate\Database\Eloquent\ModelNotFoundException;
...
(continua)
```

```
->withExceptions(function (Exceptions $exceptions) {
    $exceptions->render(function (ModelNotFoundException $e, Request $request) {
        if ($request->is('api*')) {
            return response()->json(['error' => 'Recurso no encontrado'],404);
        }
    });
    $exceptions->render(function (NotFoundException $e, Request $request) {
        if ($request->is('api*')) {
            return response()->json(['error' => 'Recurso no encontrado'],404);
        }
    });
    $exceptions->render(function (ValidationException $e, Request $request) {
        if ($request->is('api*')) {
            return response()->json(['error' => 'Datos no válidos'],400);
        }
    });
    $exceptions->render(function (\Throwable $e, Request $request) {
        if ($request->is('api*')) {
            return response()->json(['error' => 'Error: ' . $e->getMessage()],500);
        }
    });
})->create();
```



# 3. Prova de serveis REST

---

Extensió **ThunderClient** per a Visual Studio Code

# Exercici 1

---

Crea un servidor API Rest que permeti gestionar l'inventari informàtic d'un centre. Tindrà els models ubicació (id, nom) i dispositiu (id, nom, descripció, estat i ubicació).

La descripció pot tindre valor nul.

L'estat pot ser: operatiu, reparació o baixa.

# 4. Autenticació basada en Tokens

---

Casos en que **no** podem fer **ús de sessions**:

- Aplicació d'escriptori
- Aplicació mòbil

Alternativa, ús de **tokens**.

# 4. Autenticació basada en Tokens

---

## Com funciona?

1. **L'usuari** envia les seues **credencials** ( login i password ) normalment en format **JSON**.
2. El **servidor valida** aquestes credencials i, si són correctes, **genera** una cadena de text anomenada **token** que servirà per identificar unívocament l'usuari a partir d'aquell moment. Aquest token ha de ser **enviat** al **client** que es va validar
3. A partir d'aquest punt, el **client** ha **d'adjuntar** el **token** com a part de la informació a cada petició que fa a una zona d'accés restringit, de manera que el servidor pugui consultar el token i comprovar si correspon amb algun usuari autoritzat. Aquest token normalment s'envia a una **capçalera** de la petició anomenada **Authorization**

# 4. Autenticació basada en Tokens

---

## Implementació en Laravel

1. **Opció 1:** Laravel nadiu.
  - a. Afegir camp a taula usuaris.
  - b. Manipulació manual senzilla
  
2. **Opció 2:** Laravel Sanctum.
  - a. Afegir taula per a la gestió de tokens.
  - b. APIs i SPAs

# 4. Autenticació basada en Tokens

---

## Projecte d'exemple

1. Crear projecte testToken
2. Configurar accés base de dades en .env
3. Configurar migració taula users per a que utilitze usuarios. Només deixarem la id, login, password i timestamps.
4. Modificar factory, seeder i model per a fer ús d'Usuario.
5. Configurem middleware
6. Creeem controlador API
7. Omplim el codi del controlador
8. Configurem les rutes

# 4. Autenticació basada en Tokens

---

## 1. **Crear projecte** projecteTestToken

```
laravel new projecteTestToken
```

# 4. Autenticació basada en Tokens

---

## 2. Configurar accés **base de dades** en `.env`

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=testToken
DB_USERNAME=testToken
DB_PASSWORD=testToken
```



## 4. Autenticació basada en Tokens

---

3. Configurar migració taula users per a que utilitze usuarios. Només deixarem la id, login, password i timestamps.

```
public function up()
{
    Schema::create('usuarios', function (Blueprint $table) {
        $table->id();
        $table->string('login')->unique;
        $table->string('password');
        $table->timestamps();
    });
}
```

# 4. Autenticació basada en Tokens

---

4. Modificar **factory**, seeder i model per a fer ús d'Usuario.

```
class UsuarioFactory extends Factory
{
    public function definition(): array
    {
        return [
            'login' => $this->faker->word,
            'password' => bcrypt('1234')
        ];
    }
}
```

# 4. Autenticació basada en Tokens

---

4. Modificar factory, seeder i **model** per a fer ús d'**Usuario**.

```
class Usuario extends Authenticatable
{
    use HasFactory, Notifiable;
    ...
}
```

## 5. Configurem middleware en **bootstrap/app.php**

```
->withExceptions( function (Exceptions $exceptions) {  
    $exceptions->render( function (NotFoundException $e, Request $request) {  
        if ($request->is('api/*')) {  
            return response()->json([ 'error' => 'Recurso no encontrado' ], 404);  
        }  
    });  
    $exceptions->render( function (ValidationException $e, Request $request) {  
        if ($request->is('api/*')) {  
            return response()->json([ 'error' => 'Datos no válidos' ], 400);  
        }  
    });  
    $exceptions->render( function (AuthenticationException $e, Request $request) {  
        if ($request->is('api/*')) {  
            return response()->json([ 'error' => 'Usuario no autenticado o token  
inválido' ], 401);  
        }  
    });  
    $exceptions->render( function (\Throwable $e, Request $request) {  
        if ($request->is('api/*')) {  
            return response()->json([ 'error' => 'Error: ' . $e->getMessage()], 500);  
        }  
    });  
})->create();
```

# 4. Autenticació basada en Tokens

---

## 6. Creem controlador API

```
php artisan make:controller Api/PruebaController --api
```

## 7. Omplim el codi del controlador

```
class PruebaController extends Controller
{
    public function index()
    {
        return response()->json(['mensaje' => 'Accediendo a index']);
    }

    public function store(Request $request)
    {
        return response()->json(['mensaje' => 'Insertando'], 201);
    }

    public function show($id)
    {
        return response()->json(['mensaje' => 'Ficha de ' . $id]);
    }

    public function update(Request $request, $id)
    {
        return response()->json(['mensaje' => 'Actualizando elemento']);
    }

    public function destroy($id)
    {
        return response()->json(['mensaje' => 'Borrando elemento']);
    }
}
```

# 4. Autenticació basada en Tokens

---

8. Configurem les rutes en routes/api.php (cal crear l'arxiu)

```
Route::apiResource('prueba', PruebaController::class);
```

Afegim la ruta en bootstrap/app.php

```
->withRouting(  
    web: __DIR__.'../../routes/web.php',  
    commands: __DIR__.'../../routes/console.php',  
    api: __DIR__.'../../routes/api.php',  
    health: '/up',  
);
```

# 4. Autenticació basada en Tokens

---

## Mode Nadiu

Modifiquem la migració per afegir el camp **api\_token**

```
public function up()
{
    Schema::create('usuarios', function (Blueprint $table) {
        $table->id();
        $table->string('login')->unique;
        $table->string('password');
        $table->string('api_token', 60)->unique()->nullable();
        $table->timestamps();
    });
}
```

```
php artisan migrate:fresh --seed
```



# 4. Autenticació basada en Tokens

---

## Mode Nadiu

En `config/auth.php` modifiquem per utilitzar el model `Usuario` i apis

```
'providers' => [  
    'users' => [  
        'driver' => 'eloquent',  
        'model' => App\Models\ Usuario::class,  
    ],  
    ...  
'defaults' => [  
    'guard' => 'api',  
    ...  
],  
...  
'guards' => [  
    'web' => [  
        ...  
    ],  
    'api' => [  
        'driver' => 'token',  
        'provider' => 'users'  
    ]  
]
```

# 4. Autenticació basada en Tokens

---

## Mode Nadiu

Per protegir les rutes, creem el controlador LoginController

```
php artisan make:controller Api/LoginController
```

```
class LoginController extends Controller
{
  public function login(Request $request)
  {
    $usuario = Usuario::where('login', $request->login)->first();

    if (!$usuario ||
        !Hash::check($request->password, $usuario->password))
    {
      return response()->json(
        ['error' => 'Credenciales no válidas'], 401);
    }
    else
    {
      $usuario->api_token = Str::random(60);
      $usuario->save();
      return response()->json(['token' => $usuario->api_token]);
    }
  }
}
```

# 4. Autenticació basada en Tokens

---

## Mode Nadiu

Afegim la ruta de login a `routes/api.php`

```
Route::post('login', [LoginController::class, 'login']);
```

# 4. Autenticació basada en Tokens

---

## Mode Nadiu

Afegim el middleware en el constructor de PruebaController

```
class PruebaController extends Controller
{
    public function __construct()
    {
        $this->middleware('auth:api',
            ['except' => ['index', 'show']]);
    }
}
```

# 4. Autenticació basada en Tokens

---

## Mode Sanctum

Instal·lem Sanctum:

```
php artisan install:api
```

Podem configurar el temps d'expiració en minuts en config/sanctum.php

```
'expiration' => 5,
```

# 4. Autenticació basada en Tokens

---

## Mode Sanctum

Afegim HasApiTokens al nostre model Usuario

```
class Usuario extends Authenticatable
{
    use HasApiTokens, HasFactory, Notifiable;
    ...
}
```

# 4. Autenticació basada en Tokens

---

## Mode Sanctum

Generem el controlador de login

```
php artisan make:controller Api/LoginController
```



```
use App\Http\Controllers\Controller;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Hash;
use App\Models\Usuario;

class LoginController extends Controller
{
    public function login(Request $request)
    {
        $usuario = Usuario::where('login', $request->login)->first();

        if (!$usuario ||
            !Hash::check($request->password, $usuario->password))
        {
            return response()->json(
                ['error' => 'Credenciales no válidas'], 401);
        }
        else
        {
            return response()->json(['token' =>
                $usuario->createToken($usuario->login)->plainTextToken]);
        }
    }
}
```

# 4. Autenticació basada en Tokens

---

## Mode Sanctum

En routes/api.php afegim la ruta a LoginController

```
Route::post('login', [LoginController::class, 'login']);
```

# 4. Autenticació basada en Tokens

---

## Mode Sanctum

Modifiquem el constructor de PruebaController

```
namespace App\Http\Controllers\Api;

use Illuminate\Http\Request;
use Illuminate\Routing\Controller;

class PruebaController extends Controller
{
    public function __construct()
    {
        $this->middleware('auth:sanctum',
            ['except' => ['index', 'show']]);
    }
}
```

# 4. Autenticació basada en Tokens

## Proves

GET

http://127.0.0.1:8001/api/prueba/

Send

Query

Headers 2

Auth

Body

Tests

Pre Run

Query Parameters

☐

parameter

value

GET

http://127.0.0.1:8001/api/prueba/2

Send

Query

Headers 2

Auth

Body

Tests

Pre Run

Query Parameters

☐

parameter

value

Status: 200 OK Size: 32 Bytes Time: 212 ms

Response Headers 7 Cookies Results Docs

```
1 {  
2   "mensaje": "Accediendo a index"  
3 }
```

Copy

Status: 200 OK Size: 24 Bytes Time: 17 ms

Response Headers 7 Cookies Results Docs

```
1 {  
2   "mensaje": "Ficha de 2"  
3 }
```

# 4. Autenticació basada en Tokens

## Proves

POST

Query Headers 2 Auth Body Tests Pre Run

HTTP Headers

☐ Raw

<input checked="" type="checkbox"/>	Accept	application/json
<input checked="" type="checkbox"/>	User-Agent	Thunder Client (https://www.thunderclient.cc)
<input type="checkbox"/>	header	value

PUT

Query Headers 2 Auth Body Tests Pre Run

HTTP Headers

☐ Raw

<input checked="" type="checkbox"/>	Accept	application/json
<input checked="" type="checkbox"/>	User-Agent	Thunder Client (https://www.thunderclient.cc)
<input type="checkbox"/>	header	value

Status: 401 Unauthorized Size: 56 Bytes Time: 18 ms

Response Headers 7 Cookies Results Docs {} ≡

```
1 {  
2   "error": "Usuario no autenticado o token inválido"  
3 }
```

Status: 401 Unauthorized Size: 56 Bytes Time: 32 ms

Response Headers 7 Cookies Results Docs {} ≡

```
1 {  
2   "error": "Usuario no autenticado o token inválido"  
3 }
```

# 4. Autenticació basada en Tokens

## Proves

DELETE ⌵ http://127.0.0.1:8001/api/prueba/2 Send

Query

Headers 2

Auth

Body

Tests

Pre Run

HTTP Headers ⌵ Raw

☒ Accept

application/json

☒ User-Agent

Thunder Client (https://www.thunderclient.cc

☐ header

value

POST ⌵ http://127.0.0.1:8001/api/login/ Send

Query

Headers 2

Auth

Body 1

Tests

Pre Run

JSON

XML

Text

Form

Form-encode

GraphQL

Binary

JSON Content Format

```
1 {
2   "login": "dolor",
3   "password": "1234"
4 }
```

Status: **401 Unauthorized** Size: **56 Bytes** Time: **23 ms**

Response

Headers 7

Cookies

Results

Docs

```
1 {
2   "error": "Usuario no autenticado o token inválido"
3 }
```

Status: **200 OK** Size: **72 Bytes** Time: **358 ms**

Response

Headers 7

Cookies


Results

Docs

```
1 {
2   "token": "TCorvFicaLMHSxp64Dwog4d0G6wgP1Ut0PxJDNJb5IGanf53TcqalzhU
3   eoJA"
```

# 4. Autenticació basada en Tokens

## Proves

POST 

http://127.0.0.1:8001/api/prueba/

Send

Query

Headers <sup>2</sup>

Auth <sup>1</sup>

Body

Tests

Pre Run

None

Basic

Bearer

OAuth 2

NTLM

AWS

Bearer Token

TCorvFicaLMHSxp64Dwog4dOG6wgP1Ut0PxJDNJb5IGanf53TcqalzhUeoJA

Status: 201 Created   Size: 24 Bytes   Time: 30 ms



Response

Headers <sup>7</sup>

Cookies

Results


Docs

1 {

2    "mensaje": "Insertando"

3 }

PUT 

http://127.0.0.1:8001/api/prueba/2

Send

Query

Headers <sup>2</sup>

Auth <sup>1</sup>

Body

Tests

Pre Run

None

Basic

Bearer

OAuth 2

NTLM

AWS

Bearer Token

TCorvFicaLMHSxp64Dwog4dOG6wgP1Ut0PxJDNJb5IGanf53TcqalzhUeoJA

Status: 200 OK   Size: 35 Bytes   Time: 15 ms

Response

Headers <sup>7</sup>

Cookies

Results

Docs

1 {

2    "mensaje": "Actualizando elemento"

3 }

# 4. Autenticació basada en Tokens

## Proves

DELETE

▼

http://127.0.0.1:8001/api/prueba/2

Send

Query

Headers 2

Auth 1

Body

Tests

Pre Run

None

Basic

Bearer

OAuth 2

NTLM

AWS

Bearer Token

TCorvFicaLMHSxp64Dwog4dOG6wgP1Ut0PxJDNJb5IGanf53TcqalzhUeoJA

GET

▼

http://127.0.0.1:8001/api/pruebaasdadasd

Send

Query

Headers 2

Auth 1

Body

Tests

Pre Run

None

Basic

Bearer

OAuth 2

NTLM

AWS

Bearer Token

TCorvFicaLMHSxp64Dwog4dOG6wgP1Ut0PxJDNJb5IGanf53TcqalzhUeoJA

Status: 200 OK   Size: 31 Bytes   Time: 17 ms

Response

Headers 7

Cookies

Results

Docs

1

{

2

"mensaje": "Borrando elemento"

3

}

Status: 404 Not Found   Size: 33 Bytes   Time: 29 ms

Response

Headers 7

Cookies

Results

Docs

1

{

2

"error": "Recurso no encontrado"

3

}