Evaluación Unidad 3

INSTITUTO PROFESIONAL SAN SEBASTIAN

“Informe Unidad 3 – Desarrollo de Software Web I”

Asignatura:
Desarrollo de Software Web I
Alumnos:
Efren Tovar Silva Rut 25698445-8
Eduardo Ahumada catalán Rut 17304258-2
Profesor:
Víctor Cofre

INDICE

INTRODUCCION

La empresa Tech Solutions, dedicada al desarrollo de soluciones informáticas, ha decidido modernizar su sistema de gestión de proyectos con el fin de optimizar sus procesos internos y asegurar una administración más eficiente de la información. Para alcanzar este objetivo se optó por la construcción de un API RESTful desarrollado en Laravel 11, uno de los frameworks más robustos y actuales del ecosistema PHP, en conjunto con un sistema de autenticación mediante JSON Web Tokens (JWT) que garantiza que todas las operaciones estén debidamente protegidas y vinculadas a usuarios registrados.

El proyecto contempla la implementación completa de operaciones CRUD (Crear, Leer, Actualizar y Eliminar), las cuales constituyen la base de cualquier sistema de gestión de datos. Cada una de estas funcionalidades fue diseñada cumpliendo con estándares de desarrollo seguro y buenas prácticas en el manejo de respuestas HTTP, asegurando la correcta validación de datos y el uso de códigos de estado apropiados para cada caso.

Objetivo de la entrega

El presente informe tiene como propósito mostrar de manera clara el funcionamiento del API desarrollado. Para ello, se explica la lógica aplicada en el código y se incluyen las pruebas realizadas en Postman que validan cada operación. Con esto se busca comprobar que la solución cumple con los objetivos de la Unidad 3, asegurando el uso correcto de Laravel, las validaciones necesarias y la protección con autenticación JWT.

ENTORNO DE DESARROLLO

SO: Windows 10/11

Stack local: Vscode, Laragon (NGINX/Apache + MySQL + PHP)

Framework: Laravel 11, PHP 8.2, Composer

Base de datos: MySQL 8 (dump incluido)

Front: Blade, Tailwind vía CDN, Font Awesome

DESARROLLO

Autenticación JWT (API)

Ruta: routes/api.php (fragmento)

```php
use App\Http\Controllers\AuthController;
use App\Http\Controllers\ProyectoController;

Route::post('/register', [AuthController::class, 'register']);
Route::post('/login',    [AuthController::class, 'login']);

Route::middleware('jwt.verify')->group(function () {
    Route::get('/proyectos',             [ProyectoController::class, 'index']);
    Route::post('/proyectos',            [ProyectoController::class, 'store']);
    Route::get('/proyectos/{id}',        [ProyectoController::class, 'show']);
    Route::match(['put','patch'], '/proyectos/{id}', [ProyectoController::class, 'update']);
    Route::delete('/proyectos/{id}',     [ProyectoController::class, 'destroy']);
});
```

Controlador: app/Http/Controllers/AuthController.php
Middleware: alias jwt.verify registrado en app/

Esperado

- POST /api/register → crea usuario (password hasheada) y devuelve token (bearer).
- POST /api/login → devuelve token (bearer).
- Cualquier ruta de /api/proyectos/ requiere Authorization: Bearer token si falta o es inválido → 401

CRUD de Proyectos (Controlador)

Ruta: app/Http/Controllers/ProyectoController.php

1) Listar proyectos (GET /api/proyectos)

```php
public function index(): JsonResponse
{
    $uid = auth('api')->id();
    $proyectos = Proyecto::with('creador:id,name,email')
        ->where('created_by', $uid)
        ->orderByDesc('id')
        ->get();

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

- Devuelve solo los proyectos del usuario autenticado
- Incluye relación creador (id, name, email).

- Si no hay registros → retorna [].

Esperado

- Éxito **200 OK**

2) Crear proyecto (POST /api/proyectos)

```
1. public function store(Request $request): JsonResponse
2. {
3.     $validador = Validator::make($request->all(), [
4.         'nombre'       => ['required','string','max:150'],
5.         'fecha_inicio' => ['required','date'],
6.         'estado'       => ['required','in:pendiente,en_progreso,finalizado,completado,pausado,cancelado,Pendiente,En
Progreso,Finalizado,Completado,Pausado,Cancelado,en progreso,En Progreso'],
7.         'responsable'  => ['required','string','max:150'],
8.         'monto'        => ['required','numeric','min:0'],
9.     ], $this->mensajes(), $this->atributos());
10.
11.     if ($validador->fails()) {
12.         return response()->json([
13.             'message' => 'Validación fallida',
14.             'errors'  => $validador->errors(),
15.         ], 422);
16.     }
17.
18.     $data = $validador->validated();
19.     $data['estado']     = $this->normalizarEstado($data['estado']); // p. ej., "En Progreso" -> "en_progreso"
20.     $data['created_by'] = auth('api')->id();
21.
22.     $proyecto = Proyecto::create($data);
23.
24.     return response()->json([
25.         'message'   => 'Proyecto creado',
26.         'proyecto' => $proyecto,
27.     ], 201);
28. }
```

- **Valida** todos los campos (obligatorios, tipos y rangos).
- **Normaliza** estado a snake_case en minúsculas.
- Asigna created_by con el usuario autenticado.

Esperado

- Éxito 201
- Error de validación 422

3) Ver proyecto por ID (GET /api/proyectos/{id})

```
1. public function show($id): JsonResponse
2. {
3.     try {
4.         $uid = auth('api')->id();
5.         $proyecto = Proyecto::with('creador:id,name,email')
6.             ->where('created_by', $uid)
```

```
            ->findOrFail($id);

        return response()->json($proyecto, 200);
    } catch (ModelNotFoundException $e) {
        return response()->json(['message' => 'Proyecto no encontrado'], 404);
    }
}
```

- Busca el proyecto **del usuario autenticado** por id.
- Si no existe (o pertenece a otro usuario) captura excepción y responde 404.

Esperado

- Encontrado **200 OK**
- No existe / no pertenece **404**

4) Actualizar proyecto (PUT/PATCH /api/proyectos/{id})

```
public function update(Request $request, $id): JsonResponse
{
    $validador = Validator::make($request->all(), [
        'nombre'       => ['required','string','max:150'],
        'fecha_inicio' => ['required','date'],
        'estado'       => ['required','in:'.implode(',', $this->estadosPermitidos)],
        'responsable'  => ['required','string','max:150'],
        'monto'        => ['required','numeric','min:0'],
    ], $this->mensajes(), $this->atributos());

    if ($validador->fails()) {
        return response()->json([
            'message' => 'Validación fallida',
            'errors'  => $validador->errors(),
        ], 422);
    }

    try {
        $uid = auth('api')->id();
        $proyecto = Proyecto::where('created_by', $uid)->findOrFail($id);

        $data = $validador->validated();
        $data['estado'] = $this->normalizarEstado($data['estado']);

        $proyecto->update($data);

        return response()->json([
            'message'  => 'Proyecto actualizado',
            'proyecto' => $proyecto->fresh('creador:id,name,email'),
        ], 200);
    } catch (ModelNotFoundException $e) {
        return response()->json(['message' => 'Proyecto no encontrado'], 404);
    }
}
```

- Valida igual que store.
- Normaliza estado y actualiza solo si el proyecto es del usuario.

Esperado

- Éxito **200 OK**
- No existe / no es del usuario 404.
- Datos inválidos 422.

5) Eliminar proyecto (DELETE /api/proyectos/{id})

```
1. public function destroy($id)
2. {
3.     try {
4.         $uid = auth('api')->id();
5.         $proyecto = Proyecto::where('created_by', $uid)->findOrFail($id);
6.         $proyecto->delete();
7.
8.         return response()->noContent(); // 204
9.     } catch (ModelNotFoundException $e) {
10.         return response()->json(['message' => 'Proyecto no encontrado'], 404);
11.     }
12. }
```

- Elimina el proyecto si es del usuario.
- Usa respuesta vacía en éxito.

Esperado

- Éxito **204 (sin body).**
- No existe / no es del usuario 404

Validación y normalización

Mensajes y alias

```
1. private function mensajes(): array { /* required, string, date, in, numeric, min */ }
2. private function atributos(): array {
3.   return ['nombre'=>'nombre','fecha_inicio'=>'fecha de inicio','estado'=>'estado','responsable'=>'responsable','monto'=>'monto'];
4. }
```

Normalización de estado:

```
1. private function normalizarEstado(string $estado): string {
2.     $estado = trim($estado);
3.     $estado = str_replace(' ', '_', $estado);
4.     return mb_strtolower($estado);
5. }
```

Esperado

- Entradas como "En Progreso" o "en progreso" se guardan como "en_progreso".
- Errores de validación siempre devuelven **422** con message + errors (clave por campo).

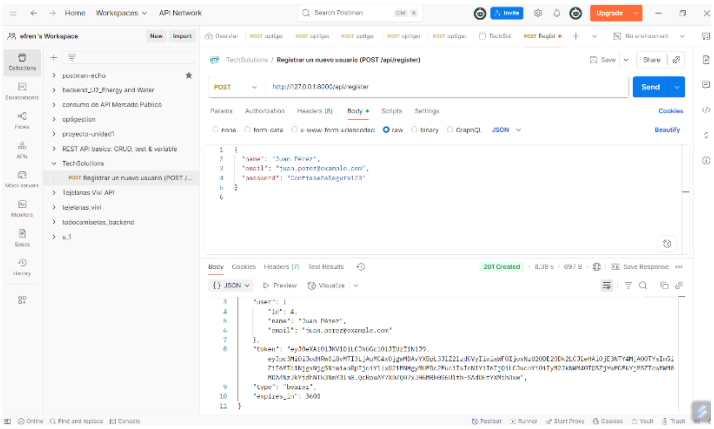
CAPTURAS DE POSTMAN Y EVIDENCIAS

Registro de nuevos usuarios

Se implementó el método register en el controlador AuthController. Esta función permite crear un nuevo usuario en la base de datos validando los campos de entrada (nombre, correo electrónico y contraseña). Además, la contraseña se guarda encriptada y el sistema genera un token JWT para el inicio de sesión inmediato.

Captura de Postman:

- POST /api/register exitoso devuelve 201 Created con los datos básicos del usuario y el token de autenticación.

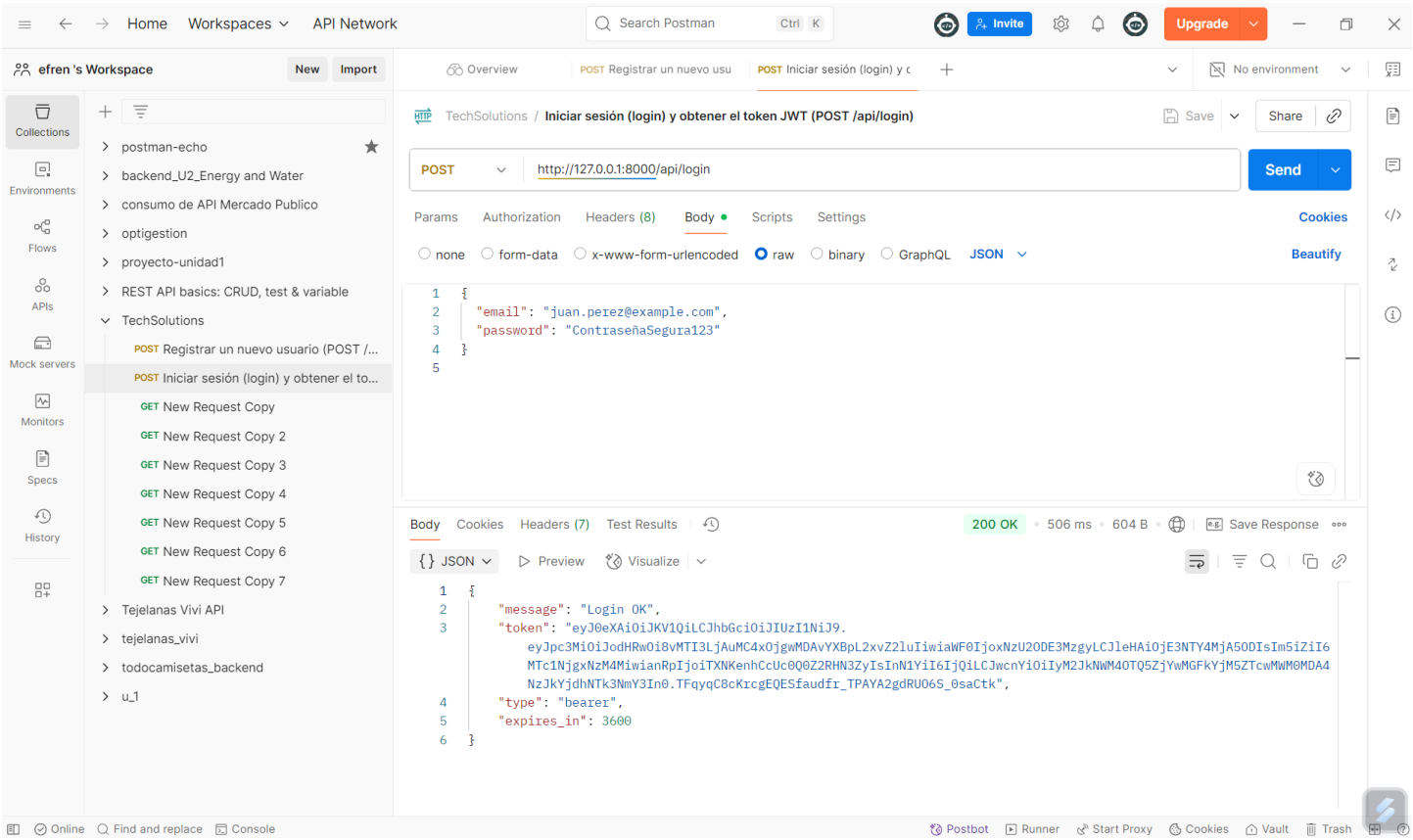


Inicio de sesión y obtención de token JWT

Se implementó el método login en el controlador AuthController. Este proceso valida las credenciales del usuario (correo y contraseña) y, si son correctas, devuelve un token JWT que permite acceder a todas las rutas protegidas del API. Dicho token debe incluirse en la cabecera Authorization con el formato *Bearer Token* para realizar operaciones posteriores.

Captura de Postman:

- POST /api/login exitoso devuelve 200 OK con el token JWT, el tipo (*bearer*) y el tiempo de expiración.

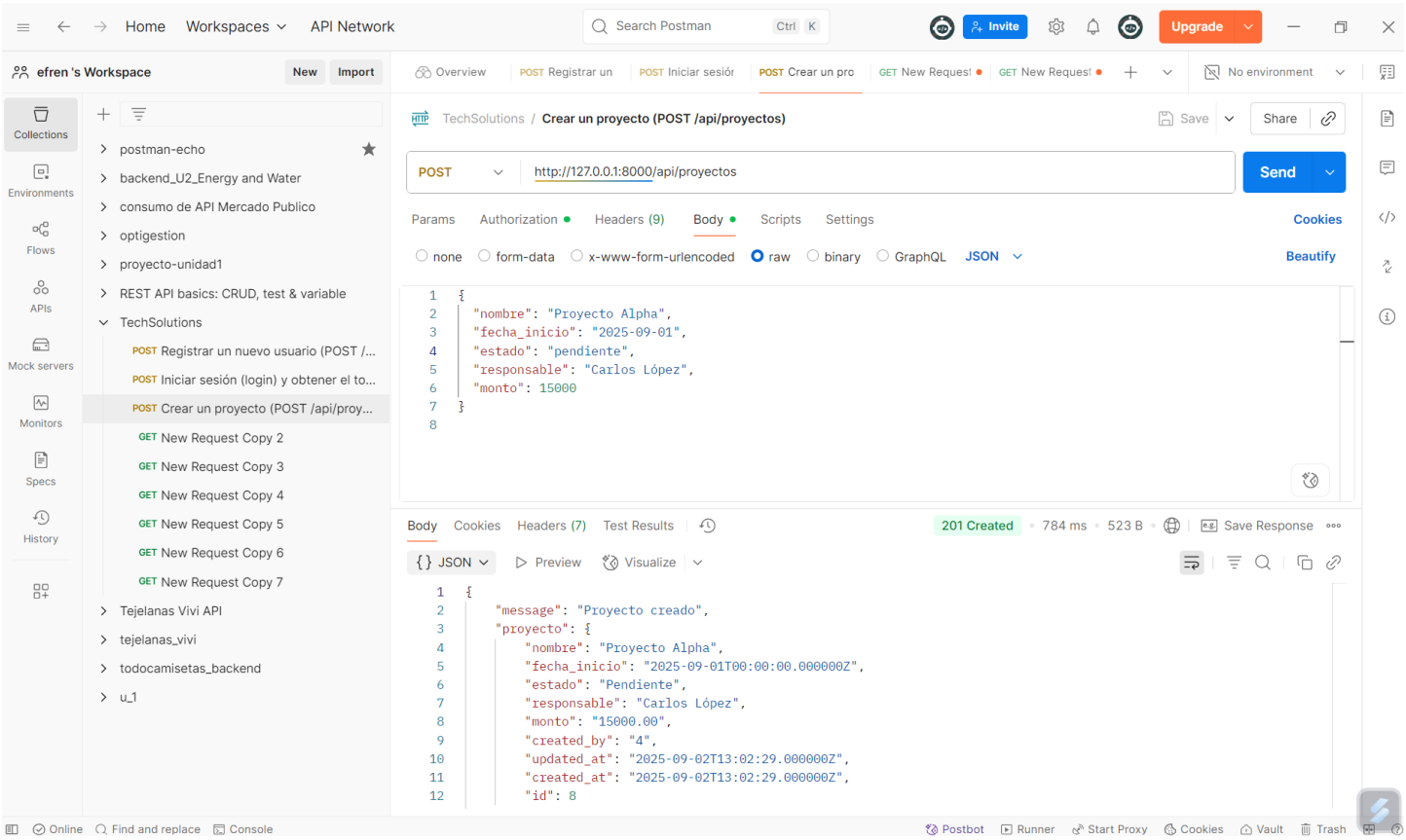


Insertar nuevos registros

Se implementó el método store en el controlador ProyectoController. Este método valida los campos obligatorios (nombre, fecha de inicio, estado, responsable y monto) y, si los datos son correctos, inserta un nuevo proyecto en la base de datos asociado al usuario autenticado. Además, devuelve la información completa del proyecto creado.

Captura de Postman:

- POST /api/proyecto exitoso devuelve 201 Created con el mensaje de confirmación y todos los datos del proyecto registrado.

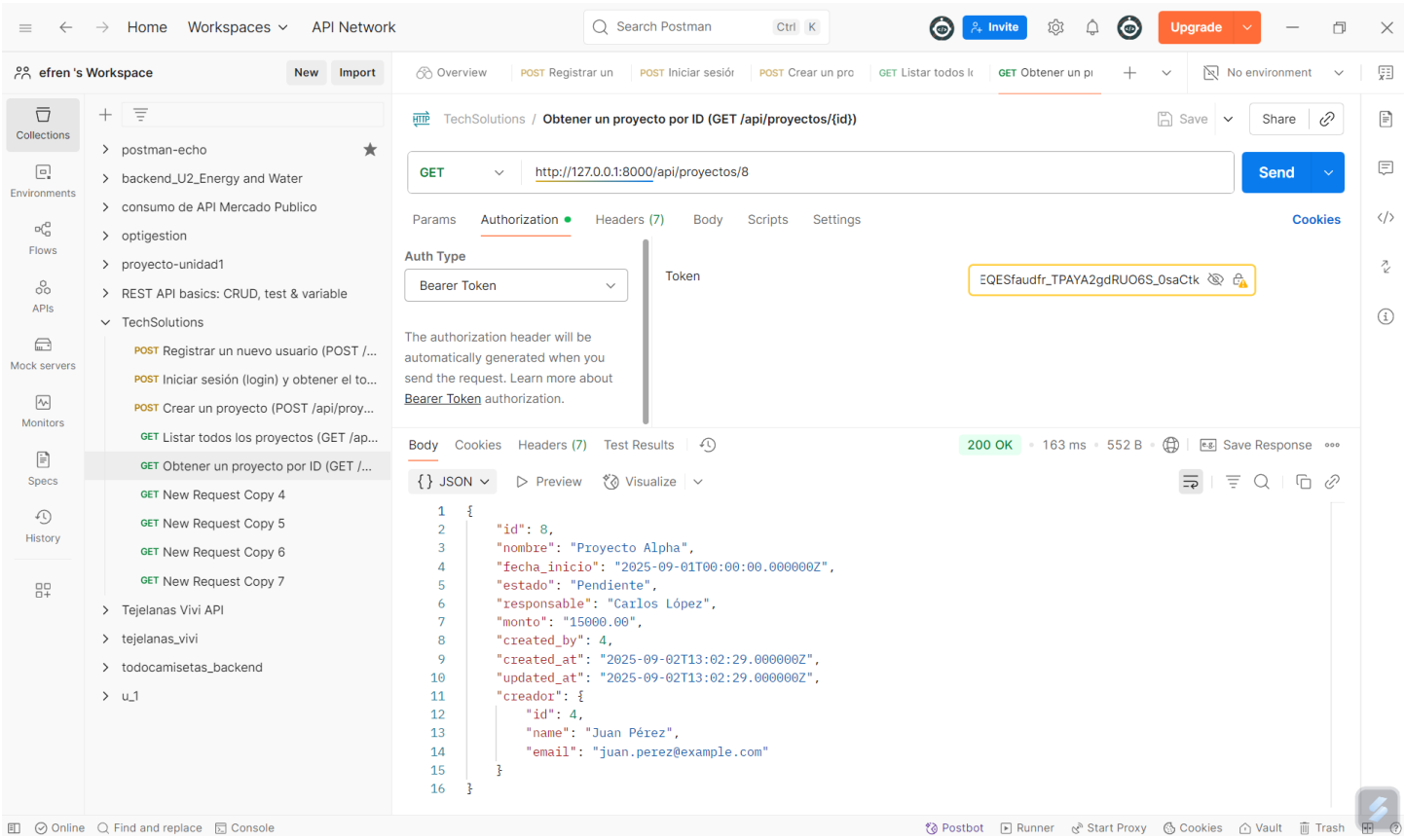


Consultar proyecto por ID

Se implementó el método show en el controlador ProyectoController. Esta función permite recuperar un proyecto específico utilizando su identificador, siempre que pertenezca al usuario autenticado. Si el proyecto existe, devuelve todos los campos asociados junto con la información del creador.

Captura de Postman:

- GET /api/proyectos/{id} exitoso devuelve 200 OK con el objeto completo del proyecto solicitado.

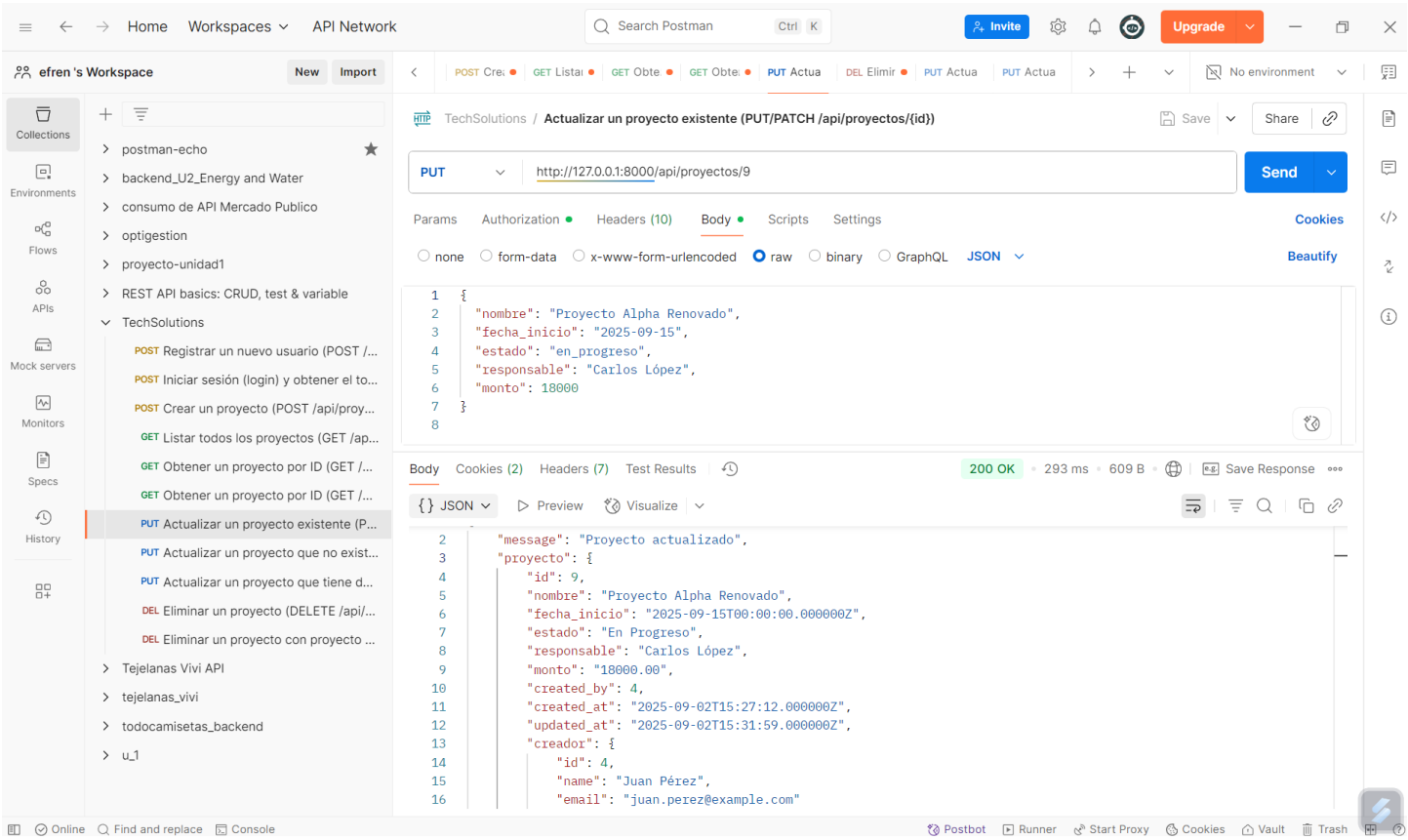


Actualizar un proyecto existente

Se implementó el método update en el controlador ProyectoController. Este método valida nuevamente todos los campos obligatorios y permite modificar los datos de un proyecto ya creado. Si la información es correcta, se actualiza el registro en la base de datos y se devuelve el objeto con los nuevos valores.

Captura de Postman:

- PUT /api/proyectos/{id} exitoso devuelve 200 OK con el mensaje de confirmación y todos los campos actualizados del proyecto.

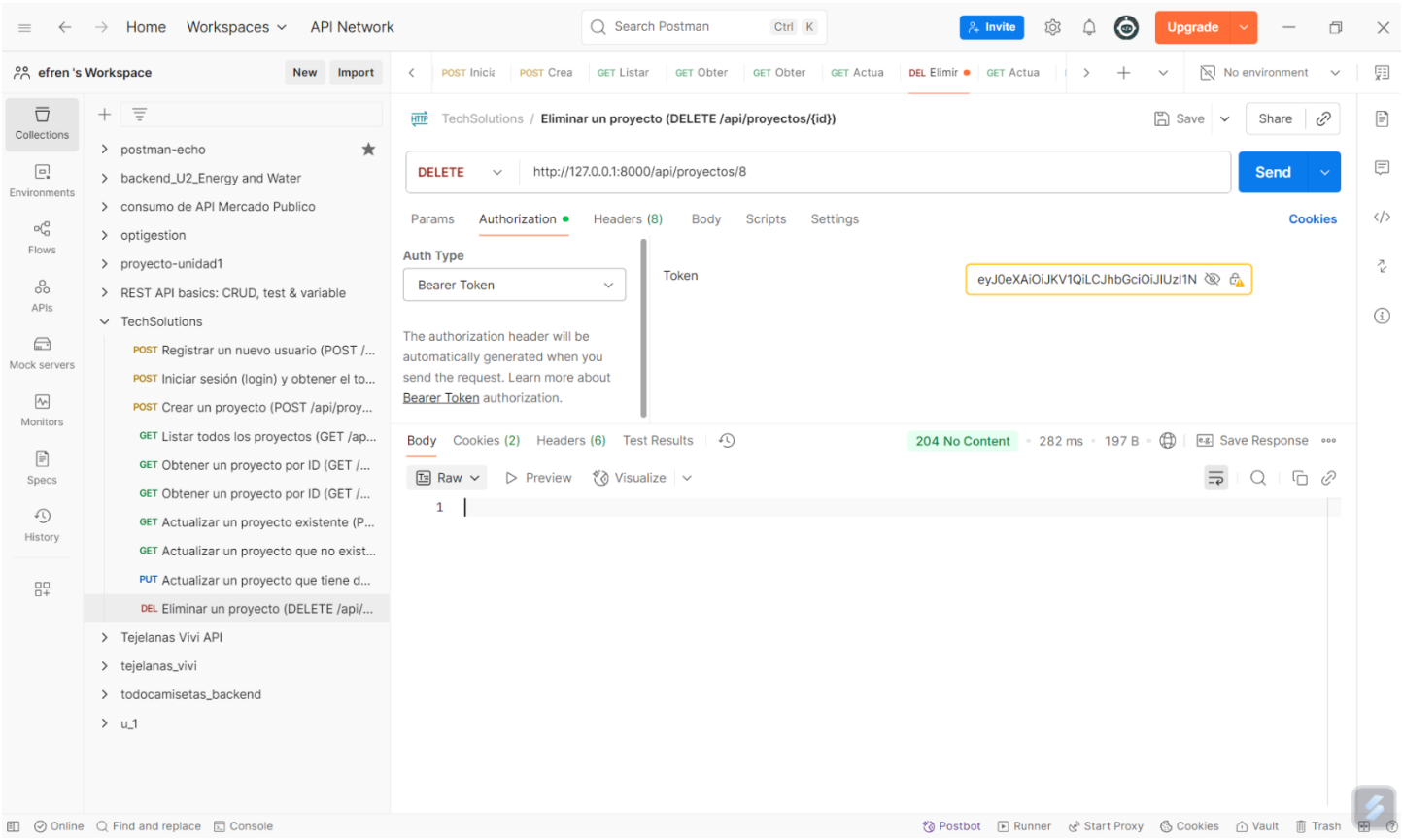


Eliminar un proyecto

Se implementó el método destroy en el controlador ProyectoController. Esta función permite borrar un proyecto específico del usuario autenticado a partir de su identificador. Si la operación es exitosa, la respuesta se envía sin contenido, siguiendo las buenas prácticas de los servicios REST.

Captura de Postman:

- DELETE /api/proyectos/{id} exitoso devuelve 204 No Content indicando que el proyecto fue eliminado correctamente.



En la carpeta del proyecto, se encuentra toda la evidencia de las pruebas realizadas. Encontrarás más fotografías de respaldo, una carpeta con capturas de pantalla de Postman

(C:\laragon\www\Tech-Solutions\pruebas_postman\captura de pantalla)

y la colección de Postman utilizada

(C:\laragon\www\Tech-Solutions\pruebas_postman\TechSolutions.postman_collection.json).

CONCLUCIONES

La elaboración de este informe permitió documentar de manera práctica el desarrollo y la validación de un API REST construido con Laravel 11 y protegido mediante JWT. A lo largo de las pruebas realizadas en Postman se evidenció el funcionamiento de cada una de las operaciones CRUD, mostrando tanto los casos exitosos como las situaciones de error (validaciones, recursos no encontrados o intentos de acceso sin autorización).

Estas evidencias no solo confirman que el sistema responde con los códigos HTTP adecuados en cada escenario, sino que también reflejan la importancia de incorporar mecanismos de seguridad, validación de datos y manejo de errores consistentes en el desarrollo de software

También Se incluye el archivo desarrollo_software_1.sql en la ruta database/backup/ como respaldo completo de la base de datos utilizada, con los registros generados

Ruta: "C:\laragon\www\Tech-Solutions\database\backup\desarrollo_software_1.sql"

También este es el link para github https://github.com/ejts29/tech-solutions-eva2

Además, también incluí Las pruebas Que se realizaron. Encontrarás más fotografías de respaldo, una carpeta con capturas de pantalla de Postman

(C:\laragon\www\Tech-Solutions\pruebas_postman\captura de pantalla)

y la colección de Postman utilizada

(C:\laragon\www\Tech-Solutions\pruebas_postman\TechSolutions.postman_collection.json).

REFERENCIAS

A3rxander. (s. f.). *How to implement JWT authentication in Laravel 11 (jwt-auth v2.x)*. *Medium*. Recuperado de https://medium.com/@a3rxander/how-to-implement-jwt-authentication-in-laravel-11-26e6d7be5a41

Instituto Profesional Santo Tomás. (s. f.). *Eva del IPSS*. Recuperado de https://eva.ipss.cl/course/view.php?id=979

Instituto Profesional Santo Tomás. (2025). *Apunte 3* [Recurso en plataforma EVA]. Recuperado de https://eva.ipss.cl/pluginfile.php/117301/mod_resource/content/6/Apunte%203.pdf

JWT.io. (s. f.). *Introduction to JSON Web Tokens*. Recuperado de https://jwt.io/introduction

Laravel. (2024). *Laravel Documentation (v11)*. Recuperado de https://laravel.com/docs/11.x

Laravel Documentation. (s. f.). *HTTP Controllers – Laravel 11.x*. Recuperado de https://documentacionlaravel.com/docs/11.x/controllers

Laravel Documentation. (s. f.). *Validation – Laravel 11.x*. Recuperado de https://documentacionlaravel.com/docs/11.x/validation

Mindicador. (s. f.). *API de indicadores económicos de Chile*. Recuperado de https://mindicador.cl/