Ciencia Latina Revista Científica Multidisciplinar, Ciudad de México, México.
ISSN 2707-2207 / ISSN 2707-2215 (en línea), mayo-junio 2025,
Volumen 9, Número 3.

https://doi.org/10.37811/cl_rcm.v9i1

# APIS RESTFUL PARA INTEROPERABILIDAD ENTRE APLICACIÓN MÓVIL Y APLICACIÓN WEB

## RESTFUL APIS FOR INTEROPERABILITY BETWEEN MOBILE AND WEB APPLICATIONS

**Jorge Cein Villanueva Guzmán**
Tecnológico Nacional de México – Instituto Tecnológico de Villahermosa

**Ezequiel Gómez Domínguez**
Tecnológico Nacional de México – Instituto Tecnológico de la Zona Olmeca

**Pedro Zamora Reséndiz**
Tecnológico Nacional de México – Instituto Tecnológico de Querétaro

**Antonio Priego Clemente**
Tecnológico Nacional de México – Instituto Tecnológico de Villahermosa

**Edgar Landero García**
Tecnológico Nacional de México – Instituto Tecnológico de Villahermosa

**DOI:** https://doi.org/10.37811/cl_rcm.v9i3.18325

# APIs RESTful para interoperabilidad entre aplicación Móvil y aplicación Web

**Jorge Cein Villanueva Guzmán**[1]
jorge.vg@villahermosa.tecnm.mx
https://orcid.org/0000-0003-1307-0801
Tecnológico Nacional de México – Instituto Tecnológico de Villahermosa
México

**Ezequiel Gómez Domínguez**
ezequiel.gd@zolmeca.tecnm.mx
https://orcid.org/0009-0008-3996-951X
Tecnológico Nacional de México – Instituto Tecnológico de la Zona Olmeca
México

**Pedro Zamora Reséndiz**
pedrozamora.resendiz@gmail.com
https://orcid.org/0009-0009-3190-5631
Tecnológico Nacional de México – Instituto Tecnológico de Querétaro
México

**Antonio Priego Clemente**
antonio.pc@villahermosa.tecnm.mx
https://orcid.org/0009-0002-7159-9498
Tecnológico Nacional de México – Instituto Tecnológico de Villahermosa
México

**Edgar Landero García**
l20301099@villahermosa.tecnm.mx
https://orcid.org/0009-0000-7196-5887
Tecnológico Nacional de México – Instituto Tecnológico de Villahermosa
México

**RESUMEN**

Este proyecto aborda la necesidad de integrar eficazmente plataformas digitales para fortalecer la promoción de productos artesanales en Tabasco. El objetivo principal fue diseñar una solución tecnológica que conectara de forma segura y eficiente la aplicación móvil y la plataforma web existente del IFAT, mediante APIs RESTful que permiten el intercambio de información en tiempo real. La metodología contempló el uso de arquitecturas escalables, microservicios y protocolos de autenticación para garantizar la seguridad y flexibilidad del sistema. Como resultado, se logró una infraestructura que permite la comunicación eficiente entre la aplicación móvil y la aplicación web. Se concluye que la interoperabilidad tecnológica puede ser una aliada clave para el desarrollo cultural y comercial en entornos digitales.

***Palabras clave:*** APIs RESTful, aplicación web, aplicación móvil, arquitectura de microservicios

[1] Autor principal.
Correspondencia: jorge.vg@villahermosa.tecnm.mx

# RESTful APIs for interoperability between mobile and web applications

**ABSTRACT**

This project addresses the need to effectively integrate digital platforms to strengthen the promotion of handicraft products in Tabasco. The main objective was to design a technological solution that would securely and efficiently connect the mobile application and the existing IFAT web platform, through RESTful APIs that allow the exchange of information in real time. The methodology contemplated the use of scalable architectures, microservices and authentication protocols to ensure the security and flexibility of the system. As a result, an infrastructure that allows efficient communication between the mobile application and the web application was achieved. It is concluded that technological interoperability can be a key ally for cultural and commercial development in digital environments.

***Keywords*:** RESTful APIs, web application, mobile application, microservices architecture

*Artículo recibido 07 mayo 2025*

*Aceptado para publicación: 13 junio 2025*

## INTRODUCCIÓN

En un contexto cada vez más digitalizado, la integración tecnológica entre plataformas es clave para garantizar servicios eficientes, seguros y accesibles. La interoperabilidad —entendida como la capacidad de diferentes sistemas para compartir y utilizar información de forma coherente— se ha convertido en un aspecto crucial para el éxito de las organizaciones que buscan integrar tecnologías y sistemas diversos. La capacidad de conectar y hacer funcionar juntos diferentes sistemas, aplicaciones y tecnologías es fundamental para garantizar una operación eficiente y segura(SOAINT, 2024). En este marco, el Instituto para el Fomento de las Artesanías de Tabasco (IFAT) ha identificado una oportunidad estratégica para fortalecer la presencia digital de los artesanos locales mediante la interconexión de su aplicación móvil con su plataforma web. Esta iniciativa no solo busca mejorar la experiencia del usuario, sino también empoderar a las comunidades artesanales mediante el acceso directo a nuevos mercados.

Diversos estudios destacan el impacto positivo de las tecnologías web, como las APIs RESTful y la arquitectura de microservicios, en la mejora de procesos de negocio y en la transformación digital de sectores tradicionales(Newman, 2021). Estas herramientas permiten construir soluciones escalables, seguras y flexibles, capaces de evolucionar con las necesidades del entorno. No obstante, su aplicación en proyectos culturales y sociales —como el comercio artesanal— aún enfrenta desafíos en términos de diseño técnico y adaptabilidad comunitaria.

El problema central de esta investigación radica en la falta de integración entre los canales digitales del IFAT, lo que limita el flujo de información, la actualización oportuna de contenidos y la experiencia de los usuarios, tanto artesanos como consumidores. Esta desconexión obstaculiza el propósito de visibilizar y comercializar eficientemente los productos artesanales tabasqueños, afectando directamente el alcance y el impacto de las estrategias digitales implementadas.

A partir de este diagnóstico, los objetivos de este estudio son: diseñar e implementar una solución de integración entre la aplicación móvil y la plataforma web del IFAT mediante APIs RESTful; garantizar la interoperabilidad, seguridad y escalabilidad del sistema; y evaluar su impacto en la experiencia del usuario y la autonomía comercial de los artesanos. En consecuencia, las preguntas de investigación que guían este trabajo son: ¿Cómo puede una arquitectura de microservicios con APIs RESTful mejorar la

integración tecnológica entre plataformas del IFAT? ¿Qué beneficios aporta esta integración a la gestión de contenidos y al empoderamiento digital de los artesanos?

**MARCO TEÓRICO**

- ***APIs***

De acuerdo con el autor (Tarkar & Parker, 2018) Restful APIs y APIs se han convertido en una herramienta esencial de desarrollo backend. Sin APIs, habrá un montón de problemas de seguridad en el sistema. El mundo moderno de desarrollo móvil y web dependen esencialmente de Restful APIs y APIs para la autenticación, Fetching Restful APIs y APIs se han convertido en una herramienta esencial de desarrollo backend.

Con una API, el desarrollador sólo tiene que centrarse en diseñar el front-end de la aplicación móvil. El backend de la aplicación será el mismo que el de la aplicación web. El desarrollador tiene que hacer llamadas a la API para realizar las principales operaciones tales como: mostrar los datos, enviarlos y eliminarlos.

Una API (interfaz de programación de aplicaciones) es un conjunto de herramientas, definiciones y protocolos que se utiliza para integrar los servicios y el software de aplicaciones. Es lo que permite que sus productos y servicios se comuniquen con otros, sin tener que diseñar permanentemente una infraestructura de conectividad nueva (Red Hat, 2018).

Las API pueden ser de tres tipos: privadas (para uso interno exclusivamente), compartidas (con socios específicos para brindar intecambio de información) o públicas (cualquier persona con acceso puede desarrollar aplicaciones que interactúen con las API para fomentar la innovación).

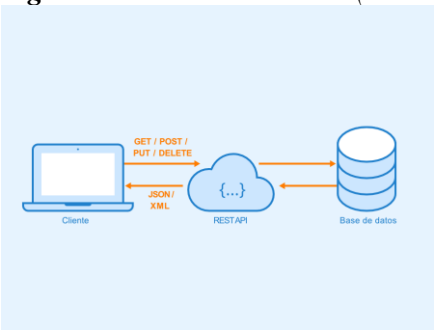
- ***RESTFUL***

REST significa Transferencia de Estado Representacional (Representational State Transfer), refiriéndose a un estilo de arquitectura de software que cuenta con 6 principios enfocados principalmente en cómo se definen y abordan los recursos en red en la web (Seobility Wiki, 2024). La arquitectura REST es una arquitectura cliente-servidor diseñada para usar HTTP, que se clasifica como un protocolo de comunicación sin estado(Christensen, 2009).

Durante los últimos años, el estilo arquitectónico REST se ha utilizado para guiar el diseño y desarrollo de la arquitectura de la Web moderna (Fielding, 2000). En la arquitectura REST, los clientes y servidores

intercambian representaciones de recursos utilizando una interfaz y un protocolo estandarizados. Estas características ayudan a proporcionar aplicaciones REST simples y ligeras. Por lo tanto, con respecto al alcance de la confiabilidad, los servicios RESTFUL superan las limitaciones de los servicios Simple Object Access Protocol (SOAP) y logran mejores resultados, especialmente en las comunicaciones móviles (Chen et al., 2005; McFaddin et al., 2008).

***Figura 1***. *API REST. Fuente: (Seobility Wiki, 2024)*




- ***Interoperabilidad***

La interoperabilidad es la capacidad de dos o más sistemas para intercambiar información y utilizarla de forma coherente (Amazon Web Services, 2024). Las arquitecturas orientadas a servicios, y en particular los microservicios, promueven el modularidad y el desacoplamiento entre componentes, lo que facilita la escalabilidad y el mantenimiento de soluciones complejas. Al dividir las funcionalidades en servicios pequeños y autónomos, es posible desplegar y actualizar cada módulo de manera independiente sin afectar el funcionamiento global.

- ***WebSockets***

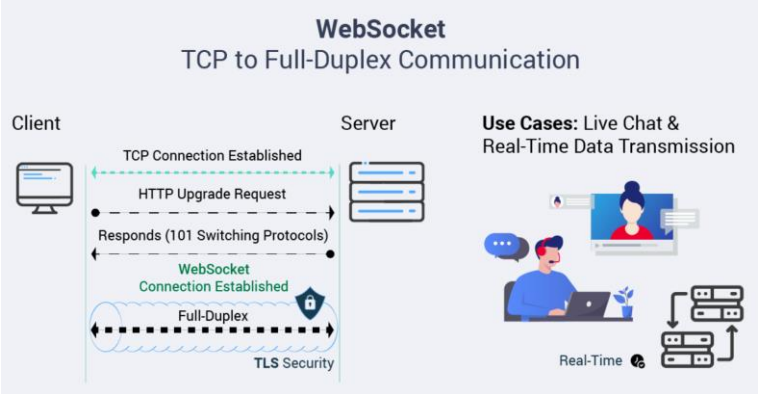
Los websockets ofrecen una solución para establecer una conexión bidireccional entre en un servidor y un cliente sobre internet, especialmente bien soportada por los navegadores web(Soligo et al., 2023).

El protocolo WebSocket permite la comunicación bidireccional entre un cliente ejecutar código no confiable en un entorno controlado en un host remoto que ha optado por recibir comunicaciones de ese código. El modelo utilizado para esto es el modelo de seguridad basado en el origen que se usa comúnmente por navegadores web. El protocolo consiste en un protocolo de apertura seguido de la estructura básica del mensaje, superpuesta sobre TCP. El objetivo de esta tecnología es para proporcionar un mecanismo basado en navegador aplicaciones que necesitan comunicación bidireccional con

servidores que no dependa de abrir múltiples conexiones HTTP (por ejemplo, usando XMLHttpRequest o <iframe>s y sondeo largo) (Fette & Melnikov, 2011).

El protocolo WebSocket intenta abordar los objetivos de las tecnologías HTTP bidireccionales existentes en el contexto de la infraestructura HTTP existente; como tal, está diseñado para funcionar sobre los puertos HTTP 80 y 443, así como para soportar proxies e intermediarios HTTP, aunque esto implique cierta complejidad específica del entorno actual (Fette & Melnikov, 2011).

***Figura 2.*** *Funcionamiento webSocket. Fuente: (Wheaton, 2024)*




## METODOLOGÍA

Para llevar a cabo esta investigación, se adoptó un enfoque metodológico (Coelho, 2020) aplicado(Vizcaíno Zúñiga et al., 2023) y tecnológico, orientado al diseño e implementación de soluciones informáticas centradas en la mejora de procesos digitales. El proyecto se desarrolló en cuatro fases interrelacionadas, utilizando principios del desarrollo ágil para garantizar flexibilidad y adaptabilidad durante su ejecución.

La **primera fase** se diseñó la arquitectura del sistema basada en microservicios, definiendo los módulos clave y la estructura de las APIs RESTful necesarias para establecer la interoperabilidad.

Actualmente la aplicación web se encuentra funcionando y cuenta con los siguientes módulos: productos, puntos ventas, inventario, administración de usuarios. A partir de los módulos anteriores, se elaboraron las siguientes APIs:

- /punto-ventas/all - Despliega todos los puntos de venta
- /punto-venta/get-punto-venta?id=1 - Despliega un punto de venta (correspondiente a su id)
- /products/ - Despliega todos los productos

- /products/1 - Actualiza los datos de los productos en referencia de su id (Funcional con método PUT o PATCH)
- /supplier/ - Despliega todos los proveedores
- /supplier/update/id - Actualiza información de los proveedores en referencia a su id (funcional con método PUT o PATCH)
- /supplier/delete/id – Elimina proveedores en referencia a su id (funcional con el metodo DELETE)
- /image/upload/id - Carga imagen a productos existentes en referencia a su id (funcional con el método POST)
- /auth/register - Realiza el registro de un usuario
- /auth/login – Realiza el logueo de un usuario

Estas APIs funcionarán como punto de partida a lo que se mostrará en la aplicación móvil, por lo que cada API cumple con una función en especifica ya bien sea que muestren, editen, o eliminen cierta información que requiera ser mostrada en la aplicación móvil.

En la **segunda fase** se comprobó el funcionamiento de todas las APIs listadas en la primera fase. Se utilizó la herramienta Postman para la validación inicial de las rutas de servicio. Se ejecutaron todas las APIs en Postman para comprobar su funcionamiento. Cada archivo debe devolver la información en formato JSON.

La **tercera fase** implicó la configuración de las APIs desde el frontend de la aplicación web. Para lo anterior, fue necesario configurar las rutas de las APIs en el framework Yii2, el cual es en el que reside la aplicación web actual.

Se aplicaron las siguientes reglas:

***Tabla 1.*** *Reglas de configuración de las APIs en el Framework Yii2.*

| Tipo de regla | Aportaciones |
|---|---|
| **Verbo + Ruta** ('GET,HEAD products') | Separa claramente operaciones de lectura y escritura, evitando ambigüedades REST. |
| **RegEx tokens** (<id:\d+>) | Valida y captura parámetros antes de llegar al controlador, reduciendo verificaciones manuales. |
| yii\\rest\\UrlRule | Genera *en bloque* todas las rutas CRUD estándar para supplier (GET /supplier, PUT /supplier/3, etc.). También agrega soporte automático para opciones como OPTIONS (CORS pre-flight) y HEAD. |
| **Rutas personalizadas** (punto-venta/all) | Permiten exponer acciones fuera del CRUD (reportes, filtros complejos) sin romper la convención REST. |
| **Fallback genéricos** | Brindan flexibilidad para cualquier controlador/acción nueva sin tocar la configuración, pero se ubican al final para no "robar" coincidencias a reglas más específicas. |

Estas fases permitieron validar la viabilidad técnica del sistema, medir su impacto en términos de eficiencia operativa y satisfacción de los usuarios y sentar las bases para futuras mejoras y ampliaciones del ecosistema digital del IFAT.

- ***Herramientas utilizadas***

• **Flutter**: Kit de desarrollo de interfaz de usuario multiplataforma empleado para construir toda la capa visual de la aplicación móvil (Flutter, 2025). Permitió compilar un único código fuente para Android e iOS, asegurando consistencia entre plataformas y reduciendo los tiempos de desarrollo. Se aprovecharon widgets nativos y patrones de gestión de estado (Provider) para lograr transiciones fluidas y rendimiento nativo.

• **Dart**: Lenguaje de programación accesible y productivo para desarrollar aplicaciones para cualquier plataforma (Dart, 2025). Su tipado estático y sintaxis moderna facilitaron el mantenimiento

del código, mientras que la compilación JIT (durante el desarrollo) y AOT (en producción) aceleró el ciclo de pruebas y optimizó el rendimiento final.

- **Android Studio:** Entorno de desarrollo integrado (IDE) usado para emular dispositivos, perfilar la aplicación y depurar errores (Google Inc, 2025). Los emuladores AVD y los plugins de Flutter agilizaron las pruebas en diferentes resoluciones y versiones de Android, garantizando compatibilidad y estabilidad.
- **PHP**: Lenguaje del lado del servidor generalista y Open Source, especialmente concebido para el desarrollo de aplicaciones web (The PHP Group, 2025) con el que se implementó la lógica de negocio de las APIs. Su madurez, amplia comunidad y extensiones nativas de seguridad facilitaron el manejo eficiente de peticiones HTTP y la serialización de datos en JSON.
- **Yii2 Framework:** Framework MVC sobre PHP (Yii, 2025) que estructuró la aplicación web y los servicios REST. Sus generadores de código (Gii), ActiveRecord ORM y filtros de acceso agilizaron la creación de endpoints seguros, escalables y fáciles de mantener.
- **MySQL**: Sistema gestor de bases de datos relacional (Manual, 2023) donde se almacenan usuarios, inventario, pedidos y catálogos de artesanías. Se definieron índices, claves foráneas y procedimientos almacenados para asegurar integridad referencial y tiempos de respuesta óptimos.
- **Docker**: Plataforma de contenedores utilizada para empaquetar los servicios (PHP-FPM, Nginx, MySQL) y replicar el entorno de producción en las máquinas de desarrollo (Docker Inc., 2020). Eliminó la clásica brecha “works on my machine” y simplificó la orquestación en staging y despliegue continuo.
- **WSL2 + Ubuntu 22.04:** Subsistema de Windows para Linux que proporcionó un entorno de desarrollo homogéneo con el servidor. Ubuntu 22.04 se usó dentro de WSL2 (Canonical Ltd, 2025) para instalar dependencias, ejecutar scripts de despliegue y administrar contenedores Docker desde la línea de comandos.
- **Terminal de Windows**: Consola unificada para PowerShell y Bash, empleada para ejecutar comandos Git, Docker, Composer y migraciones de base de datos. Su soporte para pestañas y perfiles mejoró la productividad sin cambiar de ventana.

- **GitHub**: Plataforma de control de versiones (Hosting Cloud, 2022) y colaboración donde se gestionaron ramas, solicitudes de extracción y tableros Kanban. Las GitHub Actions automatizaron pruebas unitarias y despliegues continuos hacia los entornos de staging y producción.
- **Postman:** Suite para pruebas, documentación y monitoreo de APIs RESTful (Postman Inc, 2025). Se crearon colecciones parametrizadas que validan cada endpoint, automatizan casos de prueba y generan documentación interactiva compartida con el equipo de desarrollo móvil y web.

**RESULTADOS**

A continuación, se presentan los resultados obtenidos.

En las siguientes imágenes, se muestran el despliegue y muestra de resultados de cada una de las APIs creadas.

***Figura 3***. *Despliegue de productos desde Postman. Fuente: Creación propia.*

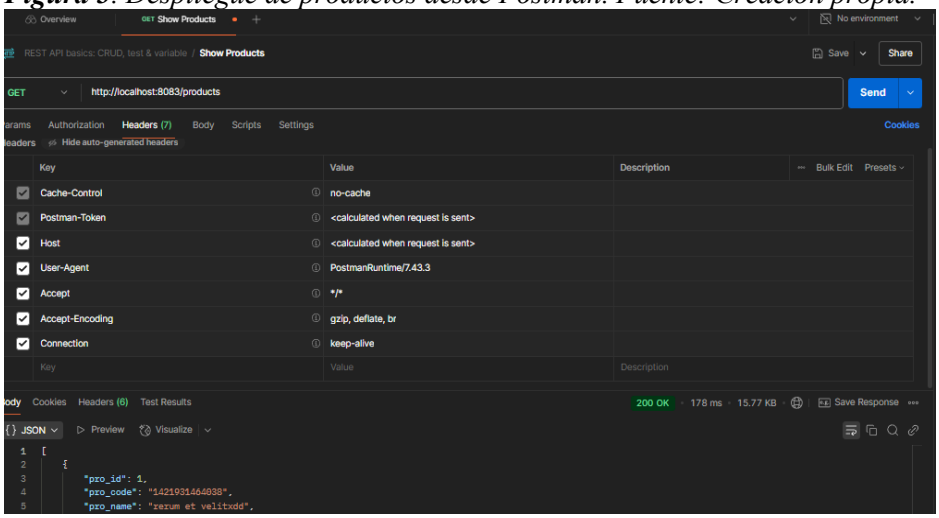



***Figura 4***. *Resultados devueltos en formato JSON por la API que muestra productos.*

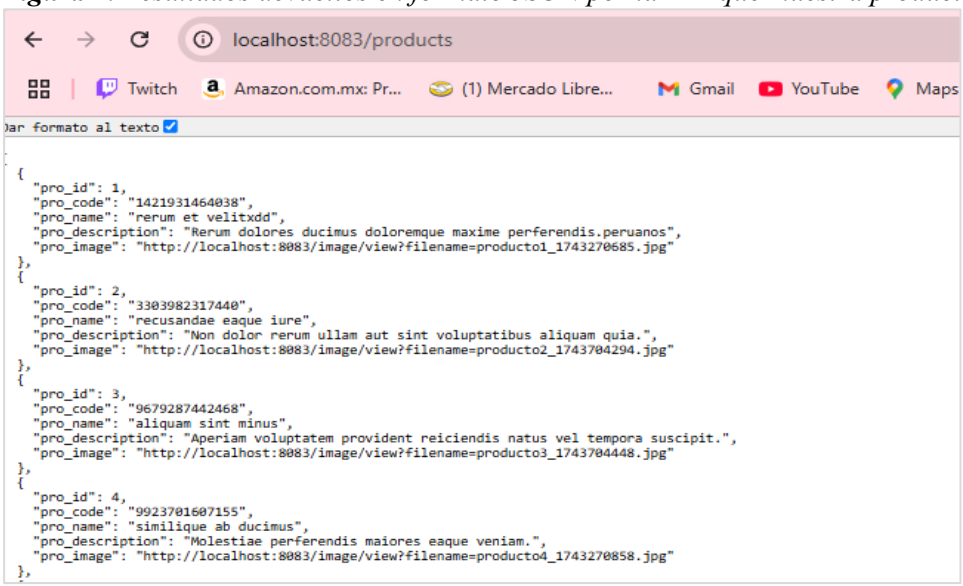


***Figura 5**. Despliegue de lista de proveedores desde Postman. Fuente: Creación propia*

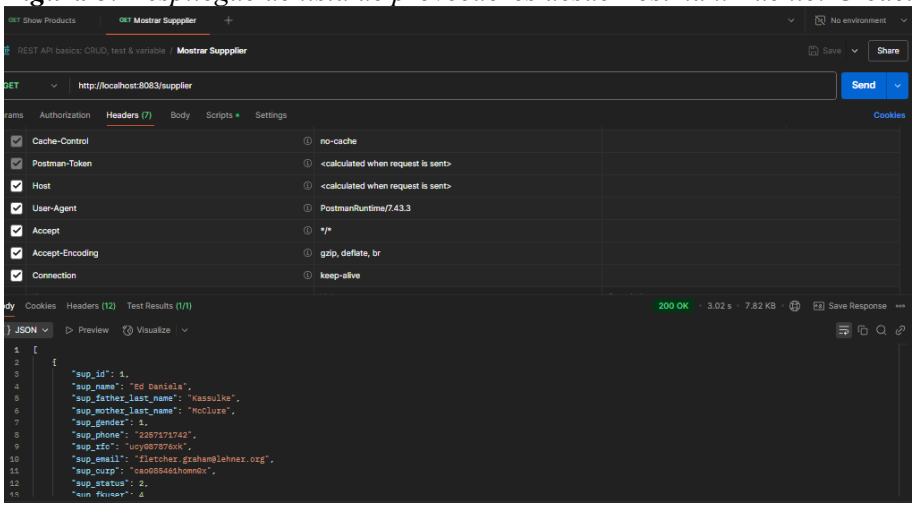
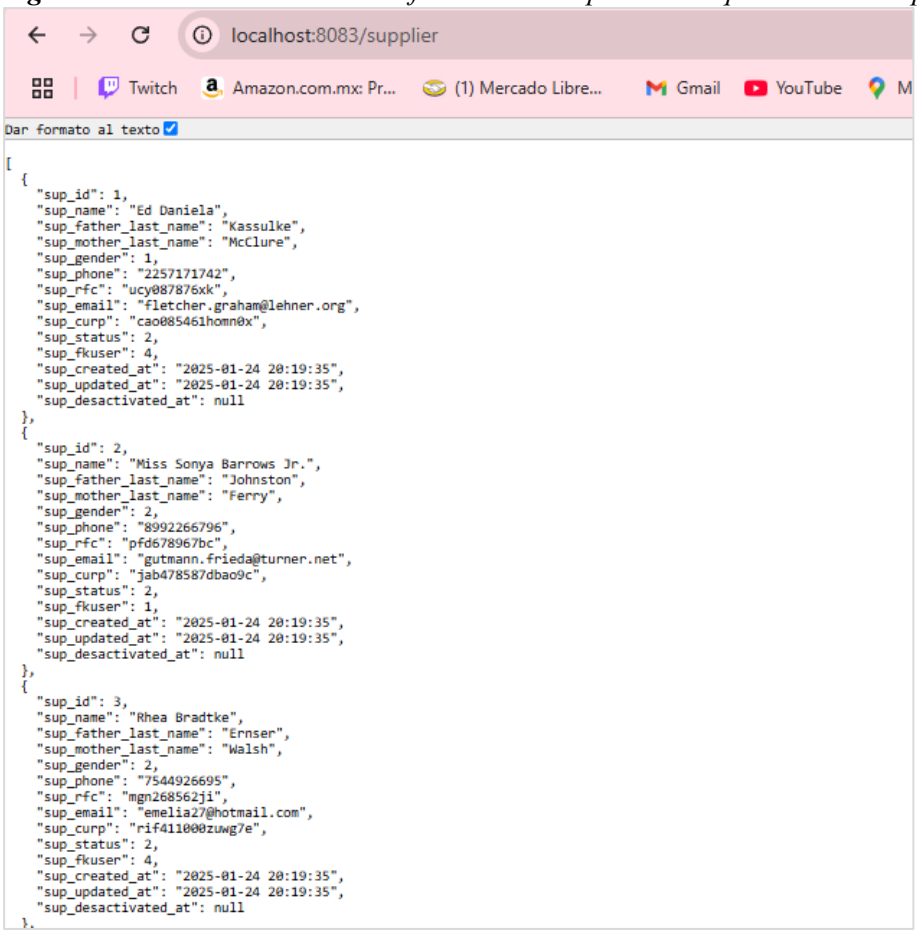



***Figura 6**. Resultados devueltos en formato JSON por la API que muestra los proveedores*




*Fuente: Creación propia.*

***Figura 7.*** *API para mostrar puntos de venta. Fuente: Creación propia.*

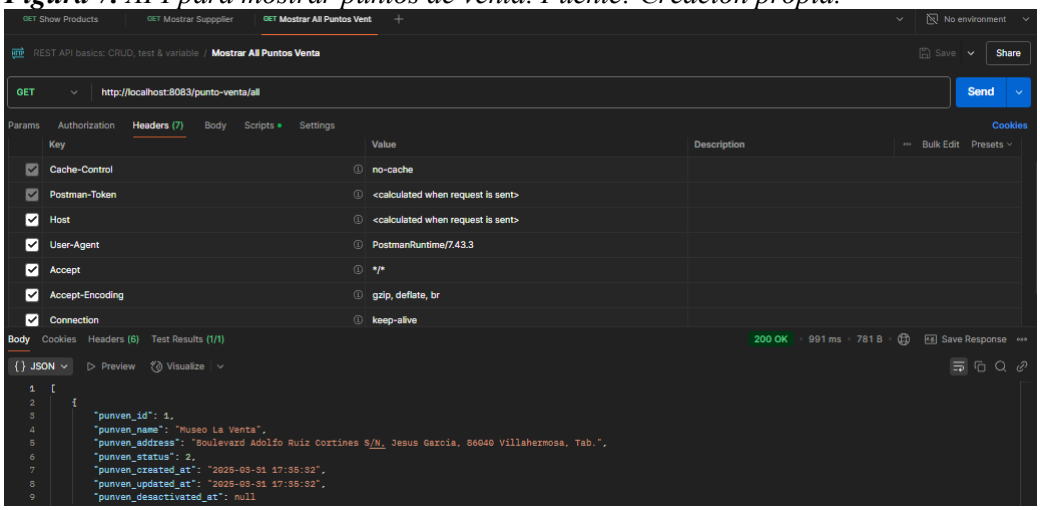



***Figura 8****. Resultados en formato JSON de la API puntos de venta. Fuente: Creación propia.*

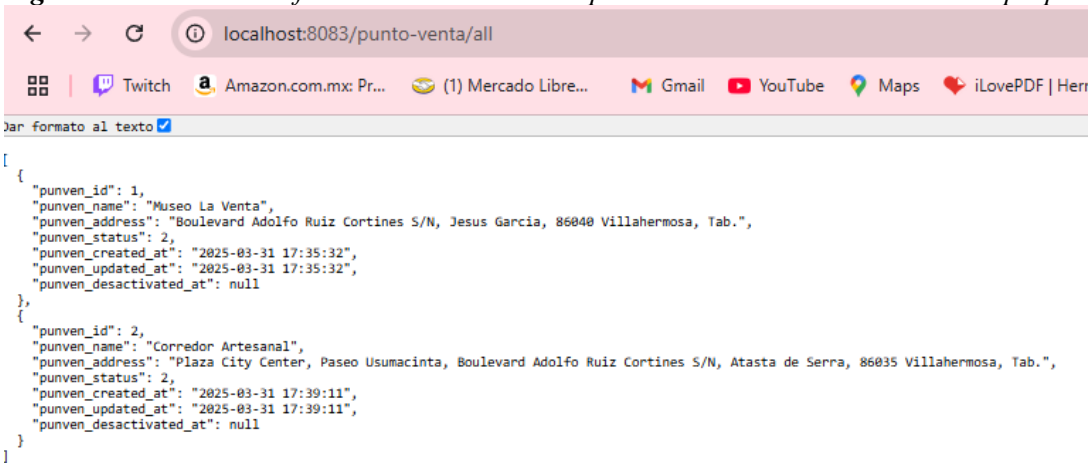
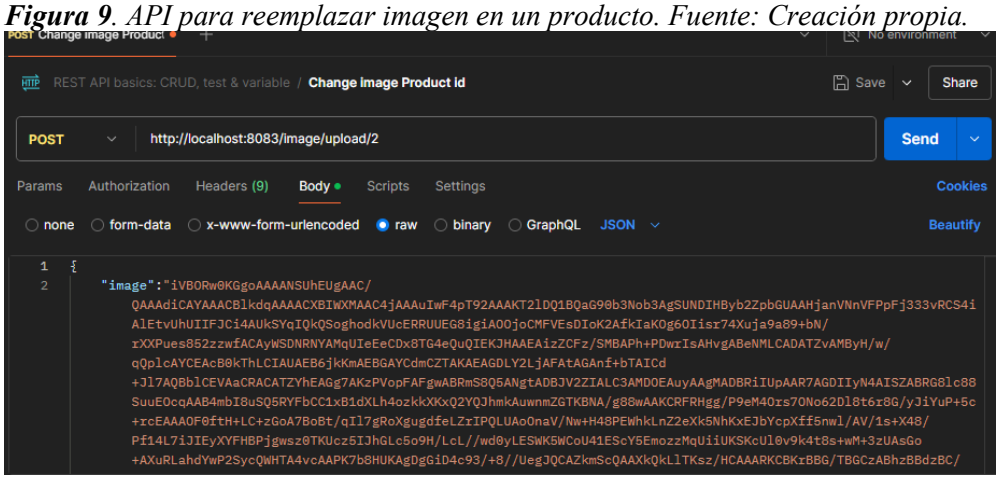



***Figura 9****. API para reemplazar imagen en un producto. Fuente: Creación propia.*




En el apartado de *Body* > *raw* dentro del código "image" de la **Figura 9** se agrega la imagen en formato base64 y en el momento que se pegue en el código, se procede a darle "send" y la imagen será cargada/actualizada. Para confirmar que el procedimiento fue elaborado correctamente en la consola se despliega el siguiente mensaje:

***Figura 10**. Mensaje de sustitución de imagen satisfactorio.*


Como se mencionó en la metodología, para el uso correcto de las APIs diseñadas previamente fue necesario realizar ciertas configuraciones de lado de la aplicación web, específicamente en archivos de configuración del framework Yii2.

En la **Figura 11** se muestra el archivo en el cual se realizaron dichos ajustes. El componente **urlManager** de Yii 2 actúa como "central de enrutamiento":

- Traduce cada URL entrante en una ruta interna (controller/action + parámetros) y
- Genera URLs salientes coherentes cuando se llama a Url::to() o createUrl() dentro del código.

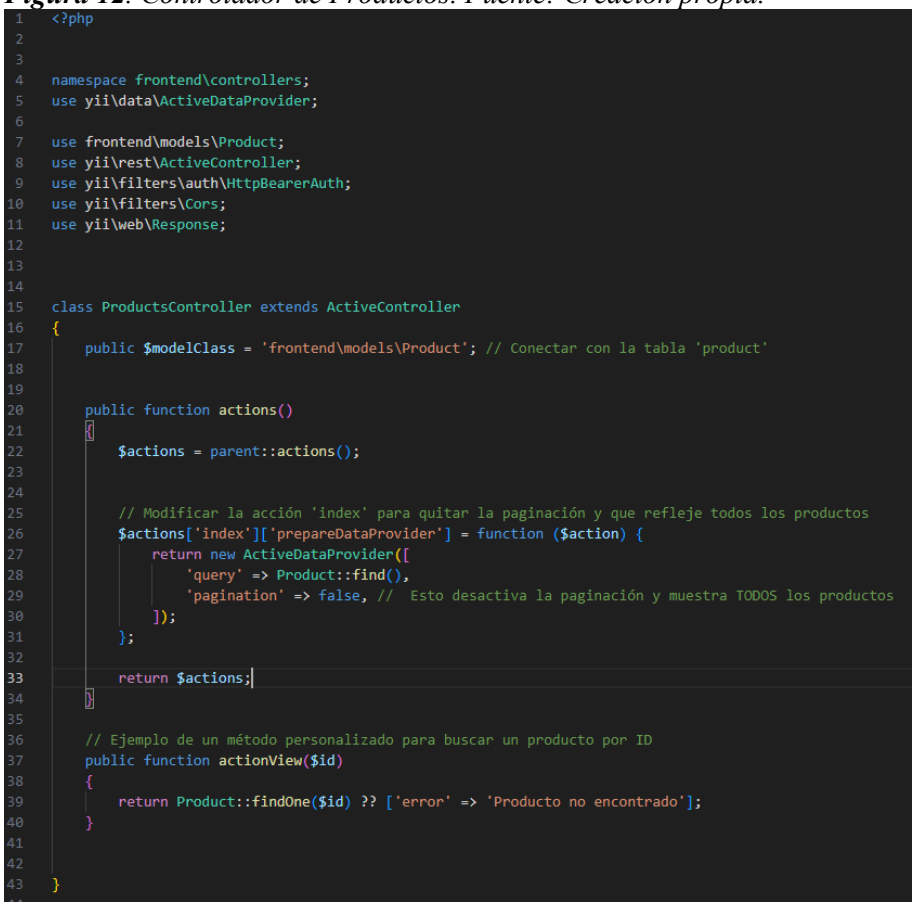
***Figura 11**. Configuración urlManager del main.php. Fuente: Creación propia.*

```
        'urlManager' => [
            'enablePrettyUrl' => true,
            'showScriptName' => false,
            'rules' => [
                //Productos
                'GET,HEAD products' => 'products/index',
                'POST products' => 'products/create',
                //Imagenes
                'POST image/upload/<id:\d+>' => 'image/upload',
                'GET image/view' => 'image/view',

                //Fallback
                '<controller:\w+>/<id:\d+>' => '<controller>/view',
                '<controller:\w+>/<action:\w+>/<id:\d+>' => '<controller>/<action>',
                '<controller:\w+>/<action:\w+>' => '<controller>/<action>',


                //Proveedores
                ['class' => 'yii\rest\UrlRule', 'controller' => 'supplier'],
                'DELETE supplier/<id:\d+>' => 'supplier/delete',

                //Punto de venta
                'GET punto-venta/all' => 'punto-venta/all', // Regla para la acción personalizada
                'GET punto-venta/get-punto-venta' => 'punto-venta/get-punto-venta',

            ],
        ],
    ],
];
```

A continuación, se describe el Modelo y Controlador de productos.

- ***Controlador productos:***

***Figura 12**. Controlador de Productos. Fuente: Creación propia.*



```php
<?php


namespace frontend\controllers;
use yii\data\ActiveDataProvider;

use frontend\models\Product;
use yii\rest\ActiveController;
use yii\filters\auth\HttpBearerAuth;
use yii\filters\Cors;
use yii\web\Response;



class ProductsController extends ActiveController
{
    public $modelClass = 'frontend\models\Product'; // Conectar con la tabla 'product'


    public function actions()
    {
        $actions = parent::actions();


        // Modificar la acción 'index' para quitar la paginación y que refleje todos los productos
        $actions['index']['prepareDataProvider'] = function ($action) {
            return new ActiveDataProvider([
                'query' => Product::find(),
                'pagination' => false, //  Esto desactiva la paginación y muestra TODOS los productos
            ]);
        };

        return $actions;
    }

    // Ejemplo de un método personalizado para buscar un producto por ID
    public function actionView($id)
    {
        return Product::findOne($id) ?? ['error' => 'Producto no encontrado'];
    }


}
```

En la **Tabla 2** se describen las líneas del archivo controlador. Esta explicación es para comprender que líneas se añadieron a las ya existentes de la aplicación web.

***Tabla 2.** Descripción del archivo controlador.*

| Línea / Bloque | Descripción |
|---|---|
| namespace frontend\controllers; | Ubica el controlador dentro del módulo **frontend**; permite a Yii2 autoloadear la clase. |
| use yii\data\ActiveDataProvider;<br>use frontend\models\Product;<br>use yii\rest\ActiveController;<br>use yii\filters\auth\HttpBearerAuth; | Importa las clases necesarias:<br>ActiveDataProvider para paginación/colecciones.<br>Product modelo ActiveRecord que mapea la tabla product.<br>ActiveController base REST que ya incluye acciones CRUD.<br>• Filtros de autenticación (HttpBearerAuth) y CORS (no se usan explícitamente en el fragmento, pero suelen añadirse en behaviors).<br>• Response para formatear salidas si se requiere. |

| | |
|---|---|
| use yii\filters\Cors;<br>use yii\web\Response; | |
| class ProductsController extends ActiveController | Declara el controlador heredando de ActiveController, lo que habilita automáticamente las acciones REST estándar (index, view, create, update, delete, options). |
| $modelClass = 'frontend\\models\\Product'; | Indica a ActiveController qué modelo ActiveRecord utilizar para las operaciones CRUD. |
| public function actions() | Sobrescribe el método para personalizar las acciones predefinidas. |
| $actions = parent::actions(); | Obtiene el arreglo de acciones generadas por ActiveController. |
| **Sobrecarga de** index | $actions['index']['prepareDataProvider'] = function ($action) {<br>return new ActiveDataProvider(['query' => Product::find(), 'pagination' => false, ]);};<br>• Reemplaza la función que construye el *data provider* de la acción index.<br>• Se desactiva la paginación ('pagination' => false) para que la API devuelva **todos** los productos en una sola respuesta.<br>• La consulta base es Product::find(), que selecciona todas las filas de la tabla product. |
| return $actions; | Devuelve el arreglo de acciones ya modificado; Yii las registrará enrutando GET /products → index, POST /products → create, etc. |
| public function actionView($id) | Define una **acción personalizada** que busca un producto por ID. Product::findOne($id) devuelve el registro o null. En caso de no existir, retorna un arreglo con el mensaje ['error' => 'Producto no encontrado'] (que Yii serializará a JSON). |

- ***Modelo productos:***

***Figura 13.*** *Modelo de Productos. Fuente: Creación propia.*

```php
namespace frontend\models;

use Yii;
use yii\db\ActiveRecord;

class Product extends ActiveRecord
{
    public static function tableName()
    {
        return 'product';
    }

    public function rules()
    {
        return [
            [['pro_image'], 'safe'], // Asegura que se puede guardar/actualizar la imagen
        ];
    }

    public function fields()
    {
        return [
            'pro_id',
            'pro_code',
            'pro_name',
            'pro_description',
            'pro_image' => function () {
                return $this->pro_image
                ? Yii::$app->request->hostInfo . '/image/view?filename=' . $this->pro_image
                : '';
            },
        ];
    }
}
```

En la **Tabla 3** se describen las líneas del archivo correspondiente al modelo.

***Tabla 3.*** *Descripción del modelo Productos.*

| Sección / Bloque | Función y descripción |
|---|---|
| namespace frontend\models; | Ubica la clase dentro del módulo *frontend*, facilitando el autoload y la organización del código. |
| use Yii;<br>use yii\db\ActiveRecord; | Importa el alias de la aplicación (Yii) y la clase base ActiveRecord, que proporciona el mapeo objeto-relacional (ORM). |
| class Product extends ActiveRecord | Declara el modelo; cada instancia representa una fila de la tabla **product**. Hereda métodos CRUD (find(), save(), delete(), etc.). |
| public static function tableName() | Devuelve 'product', indicando explícitamente a qué tabla se vincula el modelo. |
| public function rules() | Define reglas de validación:<br>• [['pro_image'], 'safe'] → permite asignar masivamente el |

| | |
|---|---|
| | campo pro_image sin validaciones adicionales (útil cuando la imagen se gestiona aparte). |
| public function fields() | Personaliza los atributos que se enviarán al serializar el modelo a JSON:<br>• pro_id, pro_code, pro_name, pro_description se exponen tal cual.<br>• pro_image se genera mediante un *closure* que: Si existe pro_image, concatena el dominio (hostInfo) con /image/view?filename= para devolver la URL pública. Si no existe, devuelve cadena vacía. |
| **Campo calculado** pro_image | Ventajas:<br>1. Oculta la lógica de rutas al frontend.<br>2. Evita exponer la ruta física del archivo.<br>3. Devuelve siempre un string (no null), simplificando el consumo en Flutter/web. |
| **Flujo típico** | 1. Product::findOne(10) recupera el registro.<br>2. Al devolverlo desde un controlador REST, Yii llama a fields() y genera un JSON limpio.<br>3. Cualquier cambio en atributos pasa por rules() antes de save(). |
| **Relación con otros componentes** | • Consumido por ProductsController para listar (index) y mostrar (view).<br>• Trabaja con el endpoint POST /image/upload/<id> que actualiza pro_image.<br>• Los clientes (app Flutter, web) reciben la misma estructura JSON, garantizando interoperabilidad. |

- ***Controlador de imágenes***

- ***Figura 14.*** *Controlador de imágenes.*

```php
namespace frontend\controllers;

use Yii;
use yii\rest\Controller;
use yii\web\Response;
use frontend\models\Product;

class ImageController extends Controller
{
    public function behaviors()
    {
        $behaviors = parent::behaviors();

        // Habilitar CORS para este controlador
        $behaviors['corsFilter'] = [
            'class' => \yii\filters\Cors::class,
            'cors' => [
                'Origin' => ['*'],
                'Access-Control-Allow-Origin' => ['*'],
                'Access-Control-Allow-Methods' => ['GET', 'POST', 'OPTIONS'],
                'Access-Control-Allow-Headers' => ['*'],
            ],
        ];

        return $behaviors;
    }
    public function actionView()
    {
        $filename = Yii::$app->request->get('filename');
        if (!$filename) {
            Yii::$app->response->statusCode = 400;
            return ['error' => 'Parámetro "filename" no proporcionado'];
        }

        $path = Yii::getAlias('@frontend/web/uploads/') . $filename;

        if (file_exists($path)) {
            Yii::$app->response->format = Response::FORMAT_RAW;
            Yii::$app->response->headers->set('Content-Type', 'image/jpeg');
            // El filtro Cors ya maneja Access-Control-Allow-Origin
            return file_get_contents($path);
        }

        Yii::$app->response->statusCode = 404;
        return ['error' => 'Imagen no encontrada'];
    }
```

***Figura 15.*** *Continuación del controlador de imágenes.*

```
    /**
     * Subir (o reemplazar) una imagen para el producto con ID = $id.
     * Espera {"image": "<base64>"} en el body.
     */
    public function actionUpload($id)
    {
        $product = Product::findOne($id);
        if (!$product) {
            return ['error' => 'Producto no encontrado'];
        }

        $imageBase64 = Yii::$app->request->getBodyParam('image');

        if ($imageBase64) {
            // 1. Si había una imagen previa, la eliminamos
            if ($product->pro_image) {
                $oldPath = Yii::getAlias('@frontend/web/uploads/') . $product->pro_image;
                if (file_exists($oldPath)) {
                    @unlink($oldPath); // borrar la anterior
                }
            }

            // 2. Generar nuevo nombre (por ejemplo con time())
            $imageName = 'producto' . $id . '_' . time() . '.jpg';
            $imagePath = Yii::getAlias('@frontend/web/uploads/') . $imageName;

            // 3. Guardar archivo
            if (file_put_contents($imagePath, base64_decode($imageBase64))) {
                // 4. Actualizar pro_image en la BD
                $product->pro_image = $imageName;
                if ($product->save()) {
                    return [
                        'message' => 'Imagen subida (o reemplazada) correctamente',
                        'pro_image' => $product->pro_image
                    ];
                } else {
                    return ['error' => 'No se pudo guardar la imagen en la base de datos'];
                }
            } else {
                return ['error' => 'No se pudo guardar la imagen en el servidor'];
            }
        }

        return ['error' => 'No se recibió una imagen'];
    }
}
```

En la **Tabla 4** se describe el funcionamiento del controlador anterior.

***Tabla 4.*** *Funcionamiento de "ImageController".*

| Sección / Bloque | Función y Descripción |
|---|---|
| namespace frontend\controllers; | Ubica este controlador dentro del módulo *frontend*, lo que permite el autoload de Yii 2 para las clases aquí definidas. |
| use Yii;<br>use yii\rest\Controller;<br>use yii\web\Response;<br>use frontend\models\Product; | Importa:<br>• Yii: acceso a la configuración y componentes globales.<br>• Controller: clase base para endpoints REST, distinta a ActiveController (no implementa CRUD automático). |

| | |
|---|---|
| | • Response: manipulación de cabeceras y formatos.<br>• Product: modelo asociado a los productos cuyas imágenes se administran. |
| class ImageController extends Controller | Clase que centraliza la lógica de **visualizar** y **subir** imágenes, proveyendo rutas personalizadas en lugar de CRUD estándar. |

***Tabla 5.*** *Configuración del CORS global mediante behaviors()*

| Elemento | Descripción |
|---|---|
| parent::behaviors() | Recupera la configuración básica de comportamientos (formato JSON, etc.). |
| corsFilter (filtro CORS) | Permite peticiones **cross-origin** desde cualquier dominio:<br>• Origin = *<br>• Methods = [GET, POST, OPTIONS]<br>• Headers = *<br>Esto resulta clave para que la app móvil (o cualquier otro cliente) pueda acceder a estos endpoints sin restricciones de dominio. |

***Tabla 6.*** *Configuración del actionView().*

| Paso / Detalle | Función |
|---|---|
| Ruta esperada | GET /image/view?filenam e=… |
| Validación de filename | Si no se recibe, responde con **HTTP 400** ({\"error\":\"Parámetro 'filename' no proporcionado\"}). |
| Construcción de la ruta del archivo | @frontend/web/uploads/<filename>: alias de Yii que apunta a la carpeta donde se alojan las imágenes. |
| Verificación de existencia del archivo | • Si el archivo existe, se retorna su contenido binario con **Response: FORMAT_RAW** y cabecera Content-Type: image/jpeg.<br>• Si no existe, regresa **404** ({\"error\":\"Imagen no encontrada\"}). |

***Tabla** 7. Configuración del actionUpload($id)*

| Paso / Detalle | Función |
|---|---|
| Ruta esperada | POST /image/upload/<id> |
| Obtención de producto | Product::findOne($id); si no lo encuentra, responde con {\"error\":\"Producto no encontrado\"}. |
| Lectura de la imagen en base64 | Se obtiene mediante Yii::$app->request->getBodyParam('image'). |
| Eliminación de imagen previa | Si el producto ya poseía pro_image, se elimina para evitar archivos obsoletos. |
| Generación de un nuevo nombre | Combina el ID del producto con la marca de tiempo time() para asegurarse de que el archivo sea único (ej. producto5_1691234567.jpg). |
| Escritura del archivo | file_put_contents($imagePath, base64_decode($imageBase64)); si falla, responde con un mensaje de error ({\"error\":\"No se pudo guardar la imagen en el servidor\"}). |
| Actualización del modelo en la BD | Se asigna el nuevo nombre a $product->pro_image, y se llama a $product->save(). Si se guarda con éxito, retorna {\"message\":\"Imagen subida…\",\"pro_image\":\"…\"}; en caso contrario, un error genérico. |
| Uso en la aplicación | • Se envía la imagen desde Flutter u otro cliente en formato base64 junto con la petición POST.<br>• El controlador decodifica, guarda y asigna la ruta al producto, simplificando el manejo de imágenes en el frontend. |

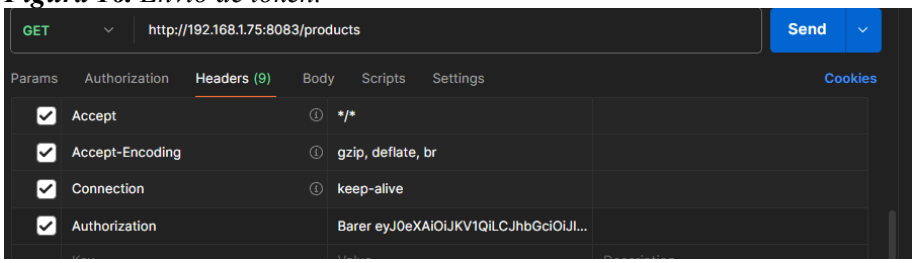
Las APIs desarrolladas adoptaron JSON Web Tokens (JWT) como núcleo de su esquema de autenticación y autorización. Tras un inicio de sesión exitoso, el servidor genera un token firmado (compuesto por header, payload y firma HMAC-SHA-256) que encapsula la identidad del usuario y los permisos (claims) necesarios para acceder a los recursos protegidos.

Este enfoque sin estado (stateless) presenta varias ventajas:

- **Escalabilidad**: Al no almacenarse sesiones en la base de datos, los nodos backend pueden escalar horizontalmente sin replicar información de sesión
- **Seguridad integrada**: La firma impide la alteración del token; cualquier intento de falsificación es detectado al validar la firma con la clave secreta del servidor.
- **Portabilidad**: El mismo token funciona en todas las plataformas (web, Android, iOS), simplificando la autenticación multiplataforma.
- **Control de vida útil:** Cada token expira en un intervalo definido (por ejemplo, 60 min). La renovación controlada (refresh tokens) reduce el riesgo de uso indebido.

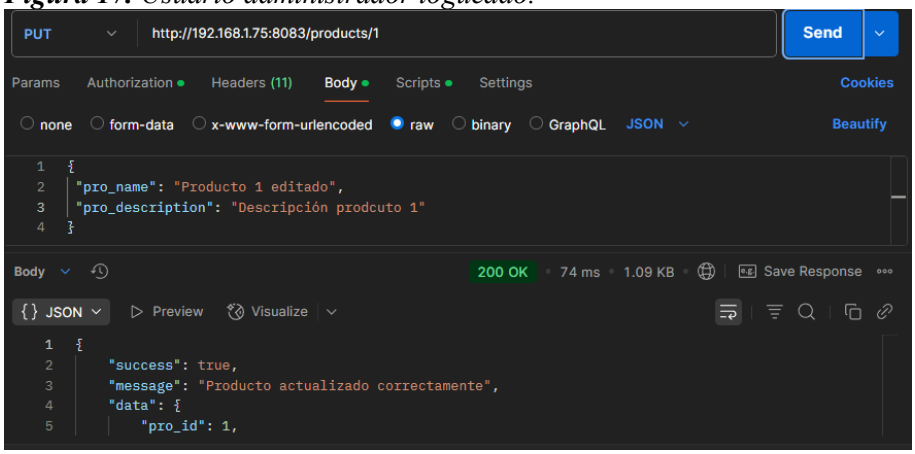
Sin embargo, en cada petición posterior, el cliente envía el token en la cabecera HTTP. Aquí se presenta un ejemplo de cómo se vería desde el postman tomando de ejemplo la API ***/products*** :

***Figura 16.*** *Envío de token.*



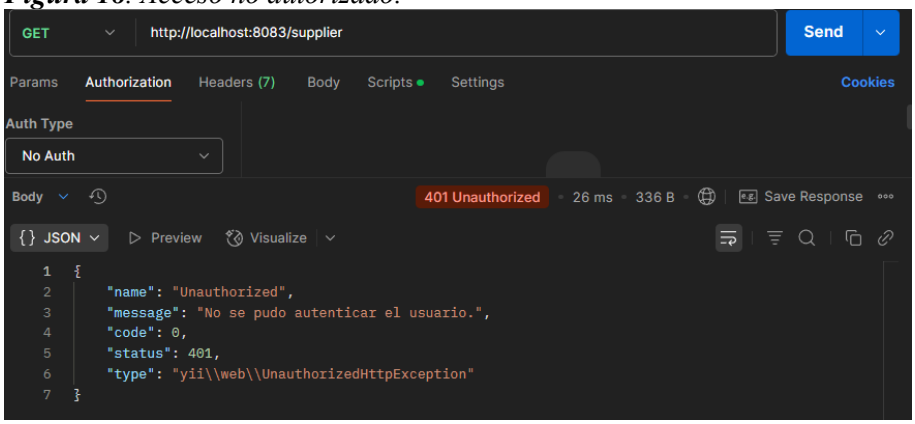

Así mismo, se implementaron algunas restricciones en la aplicación en donde solamente el usuario administrador tenga total control de ellas. Una de esas consultas restringidas seria la modificación de los datos de los productos ya que dicha consulta exclusivamente el administrador tiene acceso a ella. Aquí un ejemplo con el api para editar el producto:

***Figura 17.*** *Usuario administrador logueado.*



Como parte de las medidas de seguridad implementadas, si algún usuario desea consultar información a la cual no tiene permisos, la API lanzará un error 401 Unauthorized ya que se necesitará la sesión del usuario para que este mismo tenga acceso a la información. En la **Figura 18** se muestra un ejemplo con el api /supplier:

***Figura 18***. *Acceso no autorizado.*




El middleware de Yii2 (HttpBearerAuth) intercepta la solicitud, verifica firma y vigencia, reconstruye la identidad del usuario y autoriza o rechaza la operación según los roles y scopes definidos. Si el token carece de validez o ha expirado, la API responde con 401 Unauthorized, evitando el acceso no autorizado a los endpoints críticos como creación, edición o eliminación de productos y proveedores.

Con este diseño, la capa de servicios garantiza confidencialidad, integridad y control de acceso de forma eficiente, manteniendo la experiencia de usuario sencilla mientras cumple los requisitos de seguridad del IFAT.

**DISCUSIÓN**

A partir de las APIs diseñadas y probadas, se obtuvo una conexión exitosa entre la aplicación web y móvil.

***Figura 19.*** *Vista final de la aplicación móvil del usuario administrador con la API de productos.*

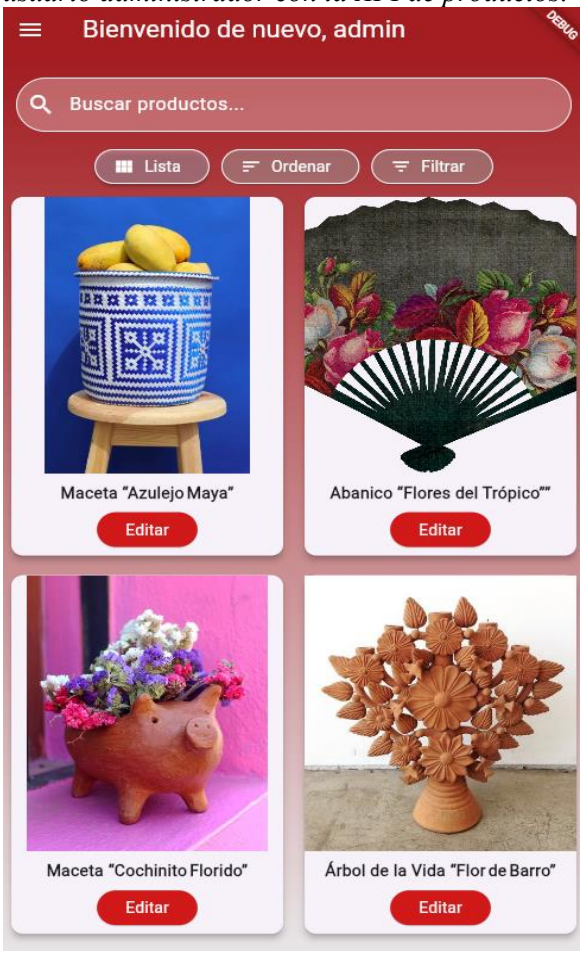



***Figura 20***. *Vista final de la aplicación móvil de un usuario estándar con la API de productos.*


Las APIs desarrolladas permiten obtener toda la información de la aplicación web en una aplicación móvil, lo que facilita la movilidad y la accesibilidad a datos en tiempo oportuno desde cualquier punto geográfico con un acceso a internet. Se demuestra que la implementación de APIs RESTful y arquitecturas de microservicios pueden transformar la manera en que las instituciones se conectan digitalmente con las comunidades.

En términos prácticos, con la implementación de las APIs se obtuvo una mejora notable en la experiencia de usuarios y sentó las bases para una plataforma que puede seguir creciendo sin complicaciones. A nivel conceptual, el trabajo reafirma que las APIs son claves para lograr sistemas digitales flexibles, seguros y sostenibles, incluso en contextos sociales y culturales.

A pesar de los avances logrados, el uso de APIs también implicó ciertos desafíos. Uno de ellos fue la complejidad técnica que implica mantener y actualizar estas interfaces, sobre todo en un entorno donde los recursos humanos y tecnológicos son limitados. Además, no todos los usuarios finales —en este

caso, los artesanos— cuentan con acceso continuo a internet o a dispositivos móviles adecuados, lo cual limita el alcance y el impacto inmediato de la solución. También se identificó la necesidad de mejorar la documentación técnica para que otros desarrolladores puedan darle continuidad al proyecto sin dificultad.

Para el futuro, se sugiere reforzar la gestión técnica de las APIs mediante buenas prácticas como el control de versiones y la automatización de pruebas, lo cual facilitaría futuras mejoras sin interrumpir el servicio. También sería útil incorporar una capa de gestión mediante un API Gateway que ofrezca mayor seguridad y control. También se recomienda integrar herramientas de análisis que podrían ayudar a comprender cómo se están utilizando estas APIs, qué aspectos pueden mejorarse y cómo seguir fortaleciendo el ecosistema digital del IFAT.

**CONCLUSIÓN**

Este proyecto representó un paso importante en la transformación digital del IFAT, al implementar un sistema basado en arquitecturas de microservicios y APIs RESTful que permitió conectar de forma eficiente su plataforma web con una futura aplicación móvil. A través de un enfoque metodológico ágil y aplicado, se logró diseñar e integrar una serie de APIs funcionales que no solo facilitan el acceso en tiempo real a información clave —como productos, puntos de venta, inventario y usuarios—, sino que también sientan las bases para el crecimiento sostenido de la plataforma.

Los resultados demuestran que las APIs son una herramienta poderosa para mejorar la interoperabilidad y accesibilidad de los sistemas digitales, especialmente en iniciativas sociales y culturales como la promoción de la artesanía tabasqueña. A pesar de los desafíos técnicos y contextuales enfrentados —como la disponibilidad de internet o la complejidad en el mantenimiento del sistema—, los beneficios superan ampliamente las limitaciones.

Este trabajo reafirma que, con la implementación adecuada, las APIs pueden convertirse en un motor de inclusión tecnológica y desarrollo social. Su correcta gestión, documentación y evolución continua serán claves para garantizar un ecosistema digital sólido, seguro y accesible para todos los usuarios, especialmente para aquellos que buscan visibilizar y comercializar su trabajo artesanal en un entorno cada vez más digitalizado.

**REFERENCIAS BIBLIOGRÁFICAS**


Amazon Web Services, I. (2024). *¿Qué es la interoperabilidad?* Https://Aws.Amazon.Com/Es/What-Is/Interoperability/.

Chen, M., Zhang, D., & Zhou, L. (2005). Providing web services to mobile users: The architecture design of an m-service portal. *International Journal of Mobile Communications*, *3*(1). https://doi.org/10.1504/IJMC.2005.005870

Christensen, J. H. (2009). Using RESTful web-services and cloud computing to create next generation mobile applications. *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA*, 627–633. https://doi.org/10.1145/1639950.1639958

Coelho, F. (2020). Metodología de la investigación. *Significados.Com*, *September*.

Dart. (2025). *Dart*. Https://Dart.Dev/.

Docker Inc. (2020). Docker Documentation | Docker Documentation. *Docker Documentation*.

Fette, I., & Melnikov, A. (2011). *The WebSocket Protocol*.

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures* [DISSERTATION]. UNIVERSITY OF CALIFORNIA, IRVINE.

Flutter. (2025). *Flutter*. Https://Flutter.Dev/.

Google Inc. (2025). *Android Studio*. Https://Developer.Android.Com/.

Hosting Cloud. (2022). What Is GitHub? A Beginner's Introduction to GitHub. *Kinsta.Com*.

Manual, M. R. (2023). MySQL 8.0 Reference Manual. *MySQL 8.0 Reference Manual*, *1*.

McFaddin, S., Coffman, D., Han, J. H., Jang, H. K., Kim, J. H., Lee, J. K., Lee, M. C., Moon, Y. S., Narayanaswami, C., Paik, Y. S., Park, J. W., & Soroker, D. (2008). Modeling and managing mobile commerce spaces using RESTful data services. *Proceedings - IEEE International Conference on Mobile Data Management*. https://doi.org/10.1109/MDM.2008.38

Newman, S. (2021). *Building Microservices: Designing Fine-Grained Systems* (Second). O'Reilly Media, Inc.

Postman Inc. (2025). *Postman* . Https://Www.Postman.Com/.

Red Hat, Inc. (2018). *El concepto de las interfaces de programación de aplicaciones*. Https://Www.Redhat.Com/Es/Topics/Api.

Seobility Wiki. (2024). *API REST*. Https://Www.Seobility.Net/Es/Wiki/API_REST.

Canonical Ltd. (2025, February 24). *Ubuntu en WSL*. Https://Documentation.Ubuntu.Com/Wsl/Latest/.

SOAINT. (2024, July 31). *Interoperabilidad en la Era Digital: Desafíos y Soluciones en Seguridad de Datos*. Https://Soaint.Com/Blog/Interoperabilidad-En-La-Era-Digital-Desafios-y-Soluciones-En-Seguridad-de-Datos/.

Soligo, P., Ierache, J. S., & Witold Martínez, P. (2023). Informe técnico, telemetría satelital de tiempo real sobre websockets y framework Django. *ReDDI: Revista Digital Del Departamento de Ingeniería*, *7*(2). https://doi.org/10.54789/reddi.7.2.5

Tarkar, M., & Parker, A. (2018). APIs and Restful APIs. *International Journal of Trend in Scientific Research and Development*, *Volume-2*(Issue-5), 319–322. https://doi.org/10.31142/ijtsrd15797

The PHP Group. (2025). *¿Qué es PHP?* Https://Www.Php.Net/Manual/Es/Introduction.Php.

Vizcaíno Zúñiga, P. I., Cedeño Cedeño, R. J., & Maldonado Palacios, I. A. (2023). Metodología de la investigación científica: guía práctica. *Ciencia Latina Revista Científica Multidisciplinar*, *7*(4). https://doi.org/10.37811/cl_rcm.v7i4.7658

Wheaton, R. (2024). *WebSocket vs. HTTP – A Comparative Look with Other Key Network Protocols*. Https://Www.Cel.Com/Blog/Websocket-vs-Http-Comparison/.

Yii. (2025). *Guía Definitiva de Yii 2.0*. https://www.yiiframework.com/doc/guide/2.0/es