Proyecto CFDI

Versión de Documento: 0.01

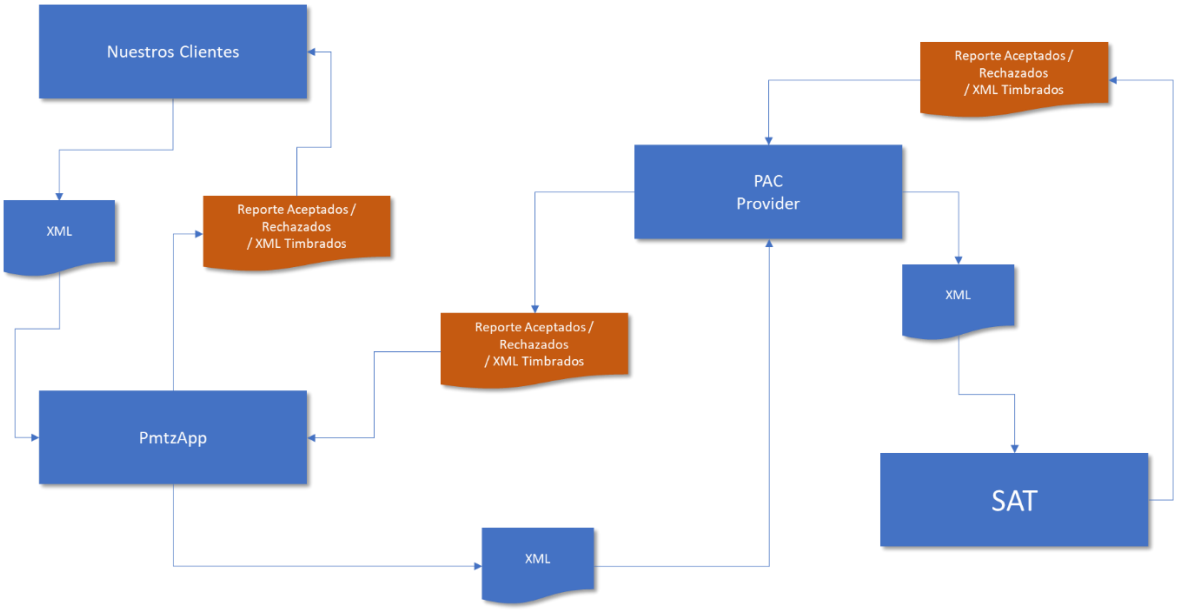
Autor: Luis Peña

Revisión: Jorge Peña

Proyecto CFDI 1
Visión General de Proyecto 3
Requerimiento Técnicos API 3
Autenticación/Autorización 3
Validación de entrada 4
Definir tipos de contenido 4
Codificación de la salida 4
Seguridad para datos en tránsito y almacenamiento 4
Requerimiento Técnicos Server 5

Visión General de Proyecto

<<Agregar texto descripción>>



<<Tipo de entrada desde el cliente (XML, Texto Plano, JSON, entrada manual u otro?)>>

<< Depende de nuestros clientes es definir en que tipo no enviarían los XML>>

Requerimiento Técnicos API

A continuación se presentan los conceptos claves que se deben considerar al diseñar las API REST.

- Autenticación/Autorización
- Validación de entrada
- Definir tipos de contenido
- Codificación de la salida
- Seguridad para datos en tránsito y almacenamiento
- Responder con los códigos de estado apropiados

Autenticación/Autorización

Aunque a veces estos términos se consideran como si fueran sinónimos en realidad no lo son.

Autenticación: La autenticación es el proceso de identificar si las credenciales pasadas juntos con la solicitud son válidas o no, aquí las credenciales se pueden pasar como ID de usuario y contraseña o un token asignado para la sesión de usuario.

Autorización: La autorización es el proceso de identificar o validar si la solicitud recibida puede acceder al punto final o método solicitado.

Nota: La Autorización debe procesarse después de la Autenticación y solo si esta última es correcta.

Validación de entrada

La validación de entrada se debe aplicar a los niveles sintáctico y semántico .

Sintáctico: Debe aplicar la sintaxis correcta de los campos estructurados (por ejemplo: RFC, CURP, fechas, etc.).

Semántico: Debe de validar si los datos de entrada cumplen las reglas de negocio (por ejemplo: cuando un campo es requerido o no, si debe existir o no, sumas de importes correctos, valores dentro de catálogos, etc.)

Definir tipos de contenido

Se debe de definir que tipo de contenido serán aceptados, JSON, XML, Texto Plano, etc. En caso de recibir un contenido fuera de los aceptados se deberá informar además de un mensaje descriptivo también se deberá devolver el código HTTP correcto: 406 Unacceptable o 415 Unsupported Media Type.

Codificación de la salida

Se deberá identificar correctamente el tipo de contenido de salida a fin de que sea aceptado e interpretado correctamente JSON, XML, Texto Plano, etc.

Seguridad para datos en tránsito y almacenamiento

Solo se debe de aceptar datos y peticiones a nuestra API utilizando el protocolo HTTPS en caso de que el cliente inicie la comunicación mediane HTPP se debe actualizarlo a HTTPS y manejar la solicitud.

Se recomienda que todo dato almacenado sea cifrado así como las copias de seguridad.

Responder con los códigos de estado apropiados para evitar la ambigüedad

A fin evitar la ambigüedad, deberá responder además de un mensaje descriptivo adicionar el código HTTP estándar correcto:

- 201 – Creado
- 200 – OK
- 202 – Aceptado y en cola para procesamiento
- 204 – Sin contenido
- 304 – No modificado
- 400 – Petición Incorrecta
- 401 – No autorizado
- 403 – Prohibido
- 404 – No encontrado
- 405 – Método no permitido
- 406 – No aceptable (utilizado con tipos de contenido)
- 415 – Tipo de medio no compatible
- 429 – Demasiadas solicitudes

Requerimiento Técnicos Server

Ignorar

Architectural Constraints by REST for RESTful Web Services.

REST is a guideline

RESTful is the application of this guideline, using APIs.

Design constrains to be a RESTful application.

1. Uniform Interface: Define consistence APIs.
2. Client Server Independent: The Client application must be independent of Server Application, this give us the ability to have clients for multiple platforms and/or expand our server to single server running all the APIs and later have multiple servers and each one running and specific API.
3. Stateless: Server do not store client context between requests. Client will manages its state. (many libraries exists Auth, etc.)
4. Cacheable: Caching applied on resource when applicable and declared by resource. Can be done in Client or Server.
5. Layered System: APIs (Server A), Data (Server B), etc. And Client not connect to each server just need to connect to Web Service.