Nodejs REST API

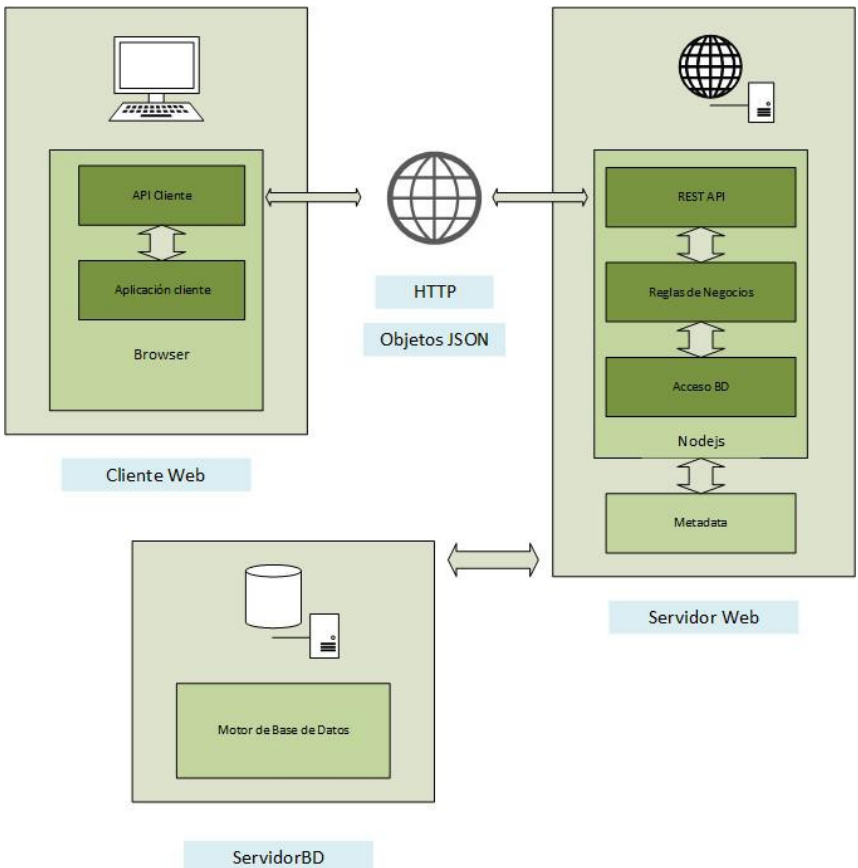
Índice general

Capítulo 1

Arquitectura

Este documento es una presentación de la arquitectura de la REST API.

La implementación de la REST API responderá a una arquitectura de diseño en tres capas expresada en el siguiente diagrama:



El diseño en tres capas es conceptual, lo que significa que en el modelo real podrá implementarse en dos o una capas.

1.1 Servidor Web

El servidor web se implementa sobre Nodejs con un middleware de router Express.

En Express se definirán los "end points" de la API, que en principio serán siete.

La programación del server Nodejs toma datos del modelo de la Base de Datos estructurados como "Recursos" a los cuales se les aplicarán los cuatro verbos REST.

Estos datos están almacenados como "metadata" (archivos JSON) que proveen la información necesaria para procesar la validación de la información recibida por la API, previo a su impacto en la Base de Datos, evitando tanto "round trips" a la base como interpretación de errores SQL.

La API genera en forma dinámica tanto las consultas a la base como las actualizaciones a la misma.

La API maneja las transacciones contra la base y provee control de concurrencia a nivel fila de las tablas de la base (recursos).

La programación del Nodejs es asincrónica con manejo de "promises" y secuenciado con "async/await".

Las reglas de negocios se deberán programar de acuerdo a este paradigma, devolviendo "promises".

Estas reglas (o su invocación) deberá insertarse en el código provisto para que tenga la capacidad de ser ejecutado en las distintas etapas del "ciclo de procesamiento".

El servidor Web expondrá una API en formato JSON para los verbos HTTP que soporten "body" (POST, PUT, DELETE) y un intérprete del "query string" para el verbo GET.

1.2 Cliente Web

El cliente web (fuera del alcance de este proyecto) deberá contemplar el procesamiento asincrónico del consumo de la API interpretando los códigos de retorno y el formato JSON del "response body". En el próximo envío de detallarán los "endpoints" y su funcionalidad, el formato de las interfaces y la estructura de la Metadata.

Capítulo 2

Metadata

La metadata le brinda la información necesaria a la API para realizar la validación individual y cruzada de los parámetros recibidos desde el cliente.

La API efectúa la validación individual de los parámetros respecto a tipo de dato, longitud y cantidad de decimales, y si el parámetro es requerido o no.

Efectúa también la consistencia de "uniqness" de las columnas, validación de existencia de valores de "foreign key" en insert y update, y chequeo de "primary key" como "foreign key" de otras tablas en delete contemplando, si así se indica, el "cascade delete".

Se denomina recurso a la unidad de afectación de la API. Normalmente un recurso tiene una relación biunívoca con una tabla de la base de datos. La excepción es el caso de una vista (view).

La metadata se compone de un grupo de archivos JSON (dentro de la carpeta "metadata" a partir del "root" de la aplicación). Existirá un archivo JSON por cada recurso. El servidor lee la metadata cada vez que se instancia. Al servidor se le debe indicar explícitamente que recursos debe leer (que archivos de metadata).

2.1 Estructura de un archivo de metadata

```
                {resource: <string> nombre del recurso,
table: <string> nombre de la tabla en la base de datos,
verbs: <char array> [G,P,U,D] lista de los código de verbos p
columns: <object array> array de objetos columna:
[{name: <string> nombre de la columna en la tabla,
```

`rol: <string> [P, F, D, V]` código de rol de la colu
`cascade: <string> [Y,N]` si el rol es “primary key” co
`type: <string> [S,I,N,T,D,M,F]` código de tipo de d
`length: <integer>` longitud de la columna aplicable a
`decimals: <integer>` cantidad máxima de decimales p
`required: <string> [Y,N]` si la columna es mandator
`unique: <string> [Y,N]` si el valor debe ser único en l
`table: <string>` para el caso de rol “foreign key” (cas
`}]}`