# Tugas HTTP Method PUT



# JSON-SCHEMA

## **Overview**

Skema adalah metadata yang memberi tahu kita bagaimana data kita terstruktur. Sebagian besar database menerapkan beberapa bentuk skema yang memungkinkan kita untuk menalar tentang data kita dengan cara yang lebih terstruktur. WordPress REST API menggunakan Skema JSON untuk menangani penataan datanya. Anda dapat mengimplementasikan titik akhir tanpa menggunakan skema, tetapi Anda akan kehilangan banyak hal. Terserah Anda untuk memutuskan apa yang paling cocok untuk Anda.

1. **JSON**

Pertama, mari kita bicara sedikit tentang JSON. JSON adalah format data yang dapat dibaca manusia yang menyerupai objek JavaScript. JSON adalah singkatan dari JavaScript Object Notation. JSON semakin populer dan tampaknya menguasai dunia struktur data. REST API WordPress menggunakan spesifikasi khusus untuk JSON yang dikenal sebagai skema JSON. Untuk mempelajari lebih lanjut tentang Skema JSON, silakan periksa situs web Skema JSON dan pengenalan Skema JSON yang lebih mudah dipahami ini. Skema memberi kita banyak manfaat: pengujian yang lebih baik, kemampuan untuk ditemukan, dan struktur keseluruhan yang lebih baik. Mari kita lihat gumpalan data JSON.

|  |  |
| --- | --- |
|  | {      "shouldBeArray": 'LOL definitely not an array',      "shouldBeInteger": ['lolz', 'you', 'need', 'schema'],      "shouldBeString": 123456789  } |

Pengurai JSON akan melewati data itu tanpa masalah dan tidak akan mengeluh tentang apa pun, karena ini adalah JSON yang valid. Klien dan server tidak tahu apa-apa tentang data dan apa yang diharapkan mereka hanya melihat JSON. Dengan menerapkan skema, kita sebenarnya dapat menyederhanakan basis kode kita. Skema akan membantu menyusun data kita dengan lebih baik sehingga aplikasi kita dapat lebih mudah mempertimbangkan interaksi kita dengan WordPress REST API. API REST WordPress tidak memaksa Anda untuk menggunakan skema, tetapi dianjurkan. Ada dua cara di mana data skema dimasukkan ke dalam API; skema untuk sumber daya dan skema untuk argumen terdaftar kami.

1. **Sumber Schema**

Skema untuk sumber daya menunjukkan bidang apa yang ada untuk objek tertentu. Saat kami mendaftarkan rute kami, kami juga dapat menentukan skema sumber daya untuk rute tersebut. Mari kita lihat seperti apa skema komentar sederhana dalam representasi PHP dari skema JSON.

// Register our routes.

**function** prefix\_register\_my\_comment\_route() {

    register\_rest\_route( 'my-namespace/v1', '/comments', **array**(

        // Notice how we are registering multiple endpoints the 'schema' equates to an OPTIONS request.

**array**(

            'methods'  => 'GET',

            'callback' => 'prefix\_get\_comment\_sample',

        ),

        // Register our schema callback.

        'schema' => 'prefix\_get\_comment\_schema',

    ) );

}

jika Anda perhatikan, setiap sumber komentar sekarang cocok dengan skema yang kami tentukan. Kami membuat peralihan ini di **prefix\_rest\_prepare\_comment().** Dengan membuat skema untuk sumber daya kami, kami sekarang dapat melihat skema ini dengan membuat permintaan OPTIONS. Mengapa ini berguna? Jika kita menginginkan bahasa lain, JavaScript misalnya, untuk menginterpretasikan data kita dan memvalidasi data dari titik akhir kita, JavaScript perlu mengetahui bagaimana struktur data kita. Ketika kami menyediakan skema, kami membuka pintu bagi penulis lain, dan diri kami sendiri, untuk membangun di atas titik akhir kami secara konsisten.

Skema menyediakan data yang dapat dibaca mesin, sehingga kemungkinan apa pun yang dapat membaca JSON dapat memahami jenis data yang dilihatnya. Ketika kami melihat indeks API dengan membuat permintaan GET **ke https://ourawesomesite.com/wp-json/**, kami mengembalikan skema API kami, memungkinkan orang lain untuk menulis pustaka klien untuk menafsirkan data kami. Proses membaca data skema ini dikenal sebagai penemuan. Ketika kami telah menyediakan skema untuk sumber daya, kami membuat sumber daya itu dapat ditemukan melalui permintaan OPTIONS ke rute itu. Mengekspos skema sumber daya hanyalah satu bagian dari teka-teki skema kami. Kami juga ingin menggunakan skema untuk argumen terdaftar kami.

1. **Argument Schema**

Saat kami mendaftarkan argumen permintaan untuk titik akhir, kami juga dapat menggunakan Skema JSON untuk memberi kami data tentang argumen yang seharusnya. Ini memungkinkan kami untuk menulis pustaka validasi yang dapat digunakan kembali saat titik akhir kami berkembang. Skema lebih banyak pekerjaan di muka, tetapi jika Anda akan menulis aplikasi produksi yang akan berkembang, Anda pasti harus mempertimbangkan untuk menggunakan skema. Mari kita lihat contoh penggunaan skema argumen dan validasi.

// Register our routes.

**function** prefix\_register\_my\_arg\_route() {

    register\_rest\_route( 'my-namespace/v1', '/schema-arg', **array**(

        // Here we register our endpoint.

**array**(

            'methods'  => 'GET',

            'callback' => 'prefix\_get\_item',

            'args' => prefix\_get\_endpoint\_args(),

        ),

    ) );

}

// Hook registration into 'rest\_api\_init' hook.

Dan contoh di atas kita telah diabstraksikan dari menggunakan nama 'my-arg'. Kita dapat menggunakan fungsi validasi dan sanitasi ini untuk argumen lain yang seharusnya berupa string yang telah kita tentukan skemanya. Saat basis kode dan titik akhir Anda berkembang, skema akan membantu menjaga kode Anda tetap ringan dan dapat dipelihara. Tanpa skema Anda dapat memvalidasi dan membersihkan, namun akan lebih sulit untuk melacak fungsi mana yang harus memvalidasi apa. Dengan menambahkan skema untuk meminta argumen, kami juga dapat mengekspos skema argumen kami ke klien, sehingga perpustakaan validasi dapat dibangun di sisi klien yang dapat membantu kinerja dengan mencegah permintaan yang tidak valid dikirim ke API.

1. **API**

REST API mendefinisikan dua fungsi utama untuk menggunakan Skema JSON: **rest\_validate\_value\_from\_schema** dan **rest\_sanitize\_value\_from\_schema**. Kedua fungsi menerima data permintaan sebagai parameter pertama, definisi skema parameter sebagai parameter kedua, dan secara opsional nama parameter sebagai parameter ketiga. Fungsi validasi mengembalikan contoh true atau **WP\_Error** tergantung pada apakah data berhasil divalidasi terhadap skema. Fungsi sanitize mengembalikan bentuk data yang disanitasi yang diteruskan ke fungsi tersebut, atau instance **WP\_Error** jika data tidak dapat dibersihkan dengan aman.

Saat memanggil fungsi-fungsi ini, Anda harus berhati-hati untuk selalu memvalidasi data terlebih dahulu menggunakan **rest\_validate\_value\_from\_schema**, dan kemudian jika fungsi itu mengembalikan nilai true, bersihkan data menggunakan **rest\_sanitize\_value\_from\_schema**. Tidak menggunakan keduanya dapat membuka titik akhir Anda terhadap kerentanan keamanan.

Jika titik akhir Anda diimplementasikan menggunakan subkelas **WP\_REST\_Controller**, metode **WP\_REST\_Controller::get\_endpoint\_args\_for\_item\_schema** akan secara otomatis menandai argumen Anda sebagai menggunakan built-in memvalidasi dan membersihkan panggilan balik. Dengan demikian, tidak perlu menentukan callback secara manual.

Jika titik akhir Anda tidak mengikuti pola kelas pengontrol, argumen yang dikembalikan dari **WP\_REST\_Controller::get\_collection\_params()**, atau contoh lain di mana panggilan balik tidak ditentukan, objek **WP\_REST\_Request** akan menerapkan sanitasi dan validasi menggunakan fungsi rest\_parse\_request\_arg. Yang penting, ini hanya diterapkan ketika **sanitize\_callback** tidak ditentukan. Dengan demikian, jika Anda menentukan sanitize\_callback khusus untuk definisi argumen Anda, validasi Skema JSON bawaan tidak akan berlaku. Jika Anda memerlukan validasi ini, Anda harus secara manual menentukan rest\_validate\_request\_arg sebagai validasi\_callback dalam definisi argumen Anda.

1. **Strings**

Jenis string mendukung tiga kata kunci tambahan.

1. maxLenght dan minLenght

Kata kunci minLength dan maxLength dapat digunakan untuk membatasi panjang string yang dapat diterima. Yang penting karakter multi-byte dihitung sebagai satu karakter dan batas termasuk.

Misalnya, diberikan skema berikut, ab, abc, dan abcd valid, sedangkan a, dan abcde tidak valid.

**array**(

     'type'      => 'string',

      'minLength' => 2,

     'maxLength' => 4,

);

Kata kunci exclusiveMinimum dan exclusiveMaximum tidak berlaku, hanya berlaku untuk angka.

1. Pattern

Pola kata kunci Skema JSON dapat digunakan untuk memvalidasi bahwa bidang string cocok dengan ekspresi reguler.

Misalnya, dengan skema berikut, #123 akan valid, tetapi #abc tidak.

**array**(

    'type'    => 'string',

    'pattern' => '#[0-9]+',

);

Regex tidak secara otomatis berlabuh. Bendera regex, misalnya /i untuk membuat case case tidak sensitif tidak didukung.

Skema JSON RFC merekomendasikan untuk membatasi diri Anda pada fitur regex berikut sehingga skema dapat dioperasikan di antara sebanyak mungkin bahasa pemrograman yang berbeda.

* karakter Unicode individu, seperti yang didefinisikan oleh spesifikasi JSON [RFC4627].
* kelas karakter sederhana [abc], kelas karakter rentang [a-z].
* kelas karakter yang dilengkapi [^abc], [^a-z].quantifier sederhana: + (satu atau lebih), \* (nol atau lebih), ? (nol atau satu), dan versi malas mereka +?, \*?, ??.
* range quantifiers: {x} (tepat x kemunculan), {x,y} (setidaknya x, paling banyak y, kemunculan), {x,} (x kemunculan atau lebih), dan versi lambatnya.
* awal input ^ dan akhir input $ jangkar.pengelompokan sederhana (...) dan silih berganti |.
* Polanya harus valid menurut dialek regex ECMA 262.

1. **Numbers**

Jenis angka dan bilangan bulat mendukung empat kata kunci tambahan.

1. Minimum and maximum

Kata kunci minimum dan maksimum memungkinkan pembatasan kisaran angka yang dapat diterima. Misalnya, 2 akan valid menurut skema ini, tetapi 0 dan 4 tidak.

**array**(

    'type' => 'integer',

    'minimum' => 1,

    'maximum' => 3,

);

Skema JSON juga memungkinkan penggunaan kata kunci ExclusiveMinimum dan exclusiveMaximum untuk menunjukkan bahwa nilainya tidak dapat sama dengan minimum atau maksimum yang ditentukan masing-masing. Misalnya, dalam hal ini hanya 2 yang merupakan nilai yang dapat diterima.

**array**(

  'type'             => 'integer',

     'minimum'          => 1,

  'exclusiveMinimum' => true,

    'maximum'          => 3,

     'exclusiveMaximum' => true,

);

1. Multipleof

Kata kunci multipleOf memungkinkan untuk menyatakan integer atau tipe angka adalah kelipatan dari angka yang diberikan. Misalnya, skema ini hanya akan menerima bilangan bulat genap.

**array**(

    'type'       => 'integer',

    'multipleOf' => 2,

);

multipleOf juga mendukung desimal. Misalnya, skema ini dapat digunakan untuk menerima persentase dengan maksimum 1 titik desimal.

**array**(

    'type'       => 'number',

    'minimum'    => 0,

    'maximum'    => 100,

    'multipleOf' => 0.1,

);

1. **Arrays**

Menentukan tipe array membutuhkan data untuk menjadi array, tetapi itu hanya setengah dari cerita validasi. Anda juga ingin menerapkan format setiap item dalam array. Ini dilakukan dengan menentukan Skema JSON bahwa setiap item array harus sesuai dengan menggunakan kata kunci item.

Misalnya, skema berikut memerlukan larik alamat IP.

**array**(

    'type'  => 'array',

    'items' => **array**(

        'type'   => 'string',

        'format' => 'ip',

    ),

);

Ini akan melewati validasi.

[ "127.0.0.1", "255.255.255.255" ]

Sementara ini akan gagal validasi.

[ "127.0.0.1", 5 ]

Skema item bisa berupa skema apa pun, bahkan bisa berupa array itu sendiri!

**array**(

    'type'  => 'array',

    'items' => **array**(

        'type'  => 'array',

        'items' => **array**(

            'type'   => 'string',

            'format' => 'hex-color',

        ),

    ),

);

Ini akan melewati validasi.

[

  [ "#ff6d69", "#fecc50" ],

  [ "#0be7fb" ]

]

Meskipun ini akan gagal validasi

[

  [ "#ff6d69", "#fecc50" ],

  "george"

]

1. **Object**

Menentukan jenis objek memerlukan data untuk menjadi objek, tetapi itu hanya setengah dari cerita validasi. Anda juga ingin menerapkan format properti setiap objek. Ini dilakukan dengan menentukan peta nama properti ke Skema JSON yang harus dipatuhi oleh anggota objek menggunakan kata kunci properti.

Misalnya, skema berikut memerlukan objek di mana nama properti adalah string dan warna adalah warna hex.

**array**(

    'type'       => 'object',

    'properties' => **array**(

        'name'  => **array**(

            'type' => 'string',

        ),

        'color' => **array**(

            'type'   => 'string',

            'format' => 'hex-color',

        ),

    ),

);

Ini akan melewati validasi.

{

  "name": "Primary",

  "color": "#ff6d69"

}

Sementara ini akan gagal validasi.

{

  "name": "Primary",

  "color": "orange"

}

1. Required Property

Secara default, properti apa pun yang terdaftar untuk suatu objek adalah opsional, jadi meskipun mungkin tidak terduga, berikut ini juga akan lulus validasi untuk skema sebelumnya.

{

  "name": "Primary"

}

Ada dua mekanisme untuk meminta properti disediakan.

**Sintaks Versi 3**

Sementara WordPress sebagian besar mengikuti Skema JSON Versi 4, salah satu cara yang tidak dilakukan adalah dengan sintaks untuk mendefinisikan properti yang diperlukan. Cara utama menggunakan sintaks JSON Schema Versi 3 dengan menambahkan kata kunci yang diperlukan ke definisi setiap properti.

**array**(

    'type'       => 'object',

    'properties' => **array**(

        'name'  => **array**(

            'type'     => 'string',

            'required' => true,

        ),

        'color' => **array**(

            'type'     => 'string',

            'format'   => 'hex-color',

            'required' => true,

        ),

**Sintaks versi 4**

WordPress juga mendukung sintaks properti yang diperlukan Skema JSON Versi 4 di mana daftar properti yang diperlukan untuk suatu objek didefinisikan sebagai larik nama properti. Ini bisa sangat membantu ketika menentukan bahwa nilai meta memiliki daftar properti yang diperlukan.

Diberikan bidang meta berikut.

register\_post\_meta( 'post', 'fixed\_in', **array**(

    'type'         => 'object',

    'show\_in\_rest' => **array**(

        'single' => true,

        'schema' => **array**(

            'required'   => **array**( 'revision', 'version' ),

            'type'       => 'object',

            'properties' => **array**(

                'revision' => **array**(

                    'type' => 'integer',

                ),

                'version'  => **array**(

                    'type' => 'string',

Permintaan berikut akan gagal validasi.

{

    "title": "Check required properties",

    "content": "We should check that required properties are provided",

    "meta": {

        "fixed\_in": {

            "revision": 47089

        }

    }

}

Jika bidang **meta fixed\_in** dihilangkan seluruhnya, tidak akan ada kesalahan yang dihasilkan. Objek yang mendefinisikan daftar properti yang diperlukan tidak menunjukkan bahwa objek itu sendiri harus dikirimkan. Hanya saja jika objek disertakan, properti yang terdaftar juga harus disertakan.

**Sintaks versi 4** tidak didukung untuk skema tingkat atas titik akhir di **WP\_REST\_Controller::get\_item\_schema().** Dengan skema berikut, pengguna dapat berhasil mengirimkan permintaan tanpa judul atau properti konten. Ini karena dokumen skema itu sendiri tidak digunakan untuk validasi, melainkan diubah menjadi daftar definisi parameter.

**array**(

    '$schema'    => '[http://json-schema.org/draft-04/schema#](http://json-schema.org/draft-04/schema)',

    'title'      => 'my-endpoint',

    'type'       => 'object',

    'required'   => **array**( 'title', 'content' ),

    'properties' => **array**(

        'title'   => **array**(

            'type' => 'string',

        ),

        'content' => **array**(

            'type' => 'string',

        ),