Napravite API koji nećete mrzeti

Phil Sturgeon

Slaviša Petrović

Napravite API Koji Nećete Mrzeti

Svako sa svojim psom želi API, tako da biste verovatno trebali da naučite da ga napravite

Phil Sturgeon and Slaviša Petrović

This book is for sale at http://leanpub.com/build-apis-you-wont-hate-sr

This version was published on 2015-07-13

Садржај

HATEOAS

Uvod

HATEOAS je tema nezgodna za objašnjavanje, ali je u stvari jednostavna. Ona znači 'Hypermedia as the Engine of Application State' (hipermedija kao pokretač stanja aplikacije) i izgovara se kao 'hat-ee-os', 'hate O-A-S' ili 'hate-ee-ohs'; ovo poslednje zvuči kao žitarica za API programere.

Kako god da izgovarate, u osnovi znači dve stvari za vaš API:

1. Pregovaranje po pitanju sadržaja
2. Hipermedija kontrole

Po mom iskustvu, pregovaranje po pitanju sadržaja je prva stvar koju API programeri implementiraju. Kada sam gradio ekstenziju za CodeIgniter Rest-Server, to je bila prva osobina koju sam dodao, jer je bilo zabavno! Menjanje zaglavlja `Accept` i pregledanje zaglavlja `Content-Type` u promeni odziva sa JSON na XML ili CSV je odlično i super lako.

Pregovaranje po pitanju sadržaja

Neki samoproklamovani REST API interfejsi (Twitter, vas treba kriviti za ovo) upravljaju pregovaranjem po pitanju sadržaja preko ekstenzija datoteka. Njihove URL adrese često izgledaju ovako:

- `/statuses/show.json?id=210462857140252672`
- `/statuses/show.xml?id=210462857140252672`

Ovo je mala zloupotreba koncepta 'resursa' i prisiljava korisnika ne samo da zna da postoji krajnja tačka 'show', već da mora da odabere ekstenziju tipa sadržaja i koristi parametar `id`.

Dobar API bi jednostavno imao `/statuses/210462857140252672`. Ovo donosi dvostruku korist, dozvoljava da API odgovori podrazumevanim tipom sadržaja ili poštujući zaglavlje `Accept` i prikazivanjem tipa sadržaja zahteva ili izbacivanjem status koda `415`, ako ga API ne podržava. Druga korist je da korisnik ne mora da zna `?id=`.

URI adrese ne bi trebale da budu gomila fascikli ili imena datoteka, a API nije lista JSON ili XML datoteka. One su liste resursa, koji mogu da budu predstavljeni u različitim formatima u zavisnosti od zaglavlja `Accept`, **i ništa više**.

Jednostavan primera pregovaranja po pitanju sadržaja koji zahteva JSON

```
GET /places HTTP/1.1
Host: localhost:8000
Accept: application/json
```

Odziv bi onda sadržao JSON ako API podržava JSON kao izlazni format.

Skraćeni primer HTTP odziva koji sadrži JSON podatke

```
HTTP/1.1 200 OK
Host: localhost:8000
Connection: close

{
    "data": [
        {
            "id": 1,
            "name": "Mireille Rodriguez",
            "lat": -84.147236,
            "lon": 49.254065,
            "address1": "12106 Omari Wells Apt. 801",
            "address2": "",
            "city": "East Romanberg",
            "state": "VT",
            "zip": 20129,
            "website": "http://www.torpdibbert.com/",
            "phone": "(029)331-0729x4259"
        },
        ...
    ]
}
```

Najpopularniji API interfejsi će podrazumevano podržavati JSON ili *samo* JSON, kao što je to naša jednostavna aplikacija radila do sada. Ovo nije realistično, ali je zbog jednostavnosti, do sada tako rađeno u knjizi.

XML je još uvek težak jer morate da zahtevate datoteke za pregled, a to je van oblasti interesovanja ovog poglavlja.

YAML, je međutim, lakše postići, tako da možemo videti kako pregovaranje po pitanju sadržaja radi sa malim izmenama u našoj aplikaciji.

Pogledajte `~/apisyouwonthate/chapter12/` kako biste videli ažuriranu probnu aplikaciju.

Osim ubacivanja [Symfony YAML komponente] glavna promena je ubacivanje metoda `respondWithArray()` kako bi se proverilo zaglavlje `Accept` i odreagovalo u skladu sa njim.

Ažurirani metod respondWithArray() sa detekcijom zaglavlja accept

```
protected function respondWithArray(array $array, array $headers = [])
{
    // You will probably want to do something intelligent with charset if provid\
ed.
    // This chapter just ignores everything and takes the main MIME type value

    $mimeParts = (array) explode(';', Input::server('HTTP_ACCEPT'));
    $mimeType = strtolower($mimeParts[0]);

    switch ($mimeType) {
        case 'application/json':
            $contentType = 'application/json';
            $content = json_encode($array);
            break;

        case 'application/x-yaml':
            $contentType = 'application/x-yaml';
            $dumper = new YamlDumper();
            $content = $dumper->dump($array, 2);
            break;

        default:
            $contentType = 'application/json';
            $content = json_encode([
                'error' => [
                    'code' => static::CODE_INVALID_MIME_TYPE,
                    'http_code' => 406,
                    'message' => sprintf('Content of type %s is not supported.',\
 $mimeType),
                ]
            ]);
    }

    $response = Response::make($content, $this->statusCode, $headers);
    $response->header('Content-Type', $contentType);

    return $response;
}
```

Osnovne stvari, ali ako sada pokušamo drugačiji MIME tip, možemo očekivati drugačiji rezultat:

HTTP zahtev koji navodi željeni odzivni MIME tip

```
GET /places HTTP/1.1
Host: localhost:8000
Accept: application/x-yaml
```

Odziv će biti u YAML-u.

Skraćeni primer HTTP odziva sa YAML podacima

```
HTTP/1.1 200 OK
Host: localhost:8000
Connection: close

data:
    - { id: 1, name: 'Mireille Rodriguez', lat: -84.147236, lon: 49.254065, addr\
ess1: '12106 Omari Wells Apt. 801', address2: '', city: 'East Romanberg', state:\
 VT, zip: 20129, website: 'http://www.torpdibbert.com/', phone: (029)331-0729x42\
59 }
    ...
```

Programsko pravljenje ovih odziva je jednostavno.

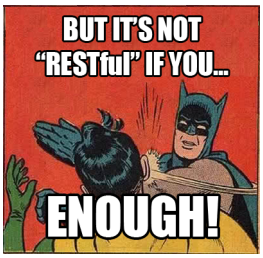
Korišćenje PHP i Guzzle paketa pri zahtevanju različitih tipova odziva

```
use GuzzleHttp\Client;

$client = new Client(['base_url' => 'http://localhost:8000']);

$response = $client->get('/places', [
        'headers' => ['Accept' => 'application/x-yaml']
]);

$response->getBody(); // YAML, ready to be parsed
```

Ovo nije kraj razgovora na temu pregovaranja po pitanju sadržaja, jer postoji još priča o MIME tipovima za resurse različitih vendora, koji se takođe mogu verzionisati. O ovome će biti reči u Poglavlju 13: API Verzionisanje.

Hipermedija Kontrole

Drugi deo HATEOAS, je međutim, drastično neiskorišćen, i poslednji je korak u postupku da vaš API bude u duhu REST-a.



Batman prečesto obezbeđuje standardni odziv na uzaludne primedbe "But it's not RESTful if you... (Ako nije dovoljno u duhu REST-a za tebe..)" Zasluga Troy Hunt-a (@troyhunt)

Iako često od mnogih ljudi čujete žalbe kao što je 'ali to nije dovoljno u duhu REST-a!' o šašavim stvarima, ovo je jedan od primera gde su potpuno u pravu. Roy Fielding kaže, pišući još 2008, da bez hipermedija kontrola API nije dovoljno u duhu REST-a[1]. Ljudi ovo od tada ignorišu, a poslednja procena kaže da 74% svih APi interfejsa koji tvrde da su u duhu REST-a ustvari i ne koriste hipermediju.

REST Nirvana

Postoji nešto što lebdi u REST/Hipermedija zajednici pod imenom ['Richardson Maturity Model'] (Richardson-ov Model Zrelosti), o kome je pisao ovde Martin Fowler[2], ali je originalno izumeo Leonard Richardson[3]. Govori o tome šta on smatra za 'četiri nivoa REST-a':

1. "**Močvara Boginja**." Koristite HTTP kako biste napravili RPC pozive. HTTP se jedino koristi kao tunel.
2. **Resursi**. Umesto da svaki poziv uputite krajnjoj tački servisa, imate više krajnjih tačaka koje predstavljaju resurse, a vi se obraćate njima. To je sam početak podrške za REST.
3. **HTTP Glagoli**. Ovo je nivo za koji vam Rails daje odmah podršku: interagujete ovim resursima koristeći HTTP glagole, umesto da uvek koristite POST.
4. **Hipermedija Kontrole**. HATEOAS. 100% REST kompatibilnosti.
   – **Izvor**: Steve Klabnik, "Haters gonna HATEOAS"[4]

Neki osporavaju ovaj model, kao što Roy kaže, osim ako imate hipermediju koja nije REST. Model je dobar sve dok razumete da koraci 1, 2 i 3 još uvek 'nisu REST', a korak 4 je 'REST'.

[1] http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven
[2] http://martinfowler.com/
[3] http://www.crummy.com/
[4] http://timelessrepo.com/haters-gonna-hateoas

Dakle, šta predstavljaju hipermedija kontrole? Oni su jednostavno veze (linkovi) ka drugim sadržajima, vezama i daljim akcijama. Ovo omogućava korisniku da predleda API i otkriva akcije u hodu.

U osnovi, vaši podaci trebaju da imaju 'hiperlinkove', koje ste verovatno godinama koristili u vašem HTML kodu. Ranije u ovoj knjizi, rekao sam da REST nije ništa drugo, do korišćenje istih konvencija kao i sam internet, umesto da se izmišljaju nove, stoga ima smisla da povezivanje sa drugim resursima treba da bude isto u API interfejsu kao i na web stranici.

Glavna stvar koju treba naglasiti u vezi hipermedije je, da API treba da u potpunosti imam smisla za klijentsku API aplikaciju i ljude koji gledaju odzive. bez gledanja dokumentacije.

Kroz ovu knjigu su mali HATEOAS koncepti tajno ubrizgani, počevši od sugerisanja kodova grešaka sa porukama čitljivim za ljude i vezama ka dokumentaciji, kako bi klijentu omogućili da izbegne matematiku prilikom interakcije sa straničenjem sadržaja. Osnovna tema je da se uvek naprave kontrole kao što su sledeće, prethodno (ili bilo koja druga vrsta interakcije) učine očiglednim računaru ili čoveku.

Razumevanje Hipermedija Kontrola

Ovo je najlakši deo pravljenja API interfejsa u duhu REST-a, tako da ću zaista pokušati da ne ostavim ovaj odeljak na 'samo dodaj linkove druže' (moj normalni savet za svakoga ko pita o pojmu HATEOAS).

Naši uobičajeni podaci su prikazani tako da predstavljaju jedan ili više resursa. Samo po sebi, jedan deo podataka je ostrvo, kompletno odsečeno od ostatka API interfejsa. Jedini način da se nastavni interakcija sa API interfejsom je da programer pročita dokumentaciju i razume koji podaci mogu biti povezani i da otkrije gde se ti podaci nalaze. Ovo je daleko od idealnog.

Povezivanje jednog `place` (mesto) sa resursima, podresursima ili kolekcijama je lako.

```
{
    "data": [
        "id": 1,
        "name": "Mireille Rodriguez",
        "lat": -84.147236,
        "lon": 49.254065,
        "address1": "12106 Omari Wells Apt. 801",
        "address2": "",
        "city": "East Romanberg",
        "state": "VT",
        "zip": 20129,
        "website": "http://www.torpdibbert.com/",
        "phone": "(029)331-0729x4259",
        "links": [
```

```
15                  {
16                          "rel": "self",
17                          "uri": "/places/2"
18                  },
19                  {
20                          "rel": "place.checkins",
21                          "uri": "/places/2/checkins"
22                  },
23                  {
24                          "rel": "place.image",
25                          "uri": "/places/2/image"
26                  }
27          ]
28      ]
29  }
```

Ovde imamo tri jednostavna unosa, gde prvi povezuje sa samim sobom. Oni svi sadrže `uri` (Universal Resource Indicator) i `rel` (Relationship).

URI protiv URL

Akronim 'URI' je često korišćen da bi se referenciralo samo na sadržaj koji dolazi posle protokola, imena hosta i porta (što znači da je URI putanja, ekstenzija i string upita), dok je 'URL' korišćen da opiše punu adresu. Iako ovo nije strogo tačno, podržavano je od strane mnogih softverskih projekata kao što je CodeIgniter. Wikipedia[5] i W3[6] govore gomilu kontradiktornih stvari, ali ja imam osećaj da je URI na jedan lak način opisan kao bilo koja vrsta identifikatora za lokaciju resursa na internetu.

URI može biti parcijalan ili apsolutni. Neki smatraju da je URL nepostojeći termin, ali ova knjiga koristi URL kako bi opisala apsolutni URI, što je ono što vidite u polju za adrese. Tačno ili netačno. Razumete?

Neki ljudi se rugaju relaciji `self`, sugerišući da je ona nepotrebna. Iako znate koju URL adresu ste upravo pozvali, ista se neće uvek poklapati sa `self` URI adresom. Na primer, ako ste upravo kreirali resurs `place`, upravo ste pozvali `POST /places`, a to je nešto što biste želeli da ponovo pozovete kako biste dobili ažurirane informacije o istom resursu. Nezavisno od sadržaja, prikazivanje resursa `place` uvek mora da ima relaciju `self`, a sam `self` ne bi trebao da prikazuje samo ono što je u polju za adresu. U osnovi, relacija `self` ukazuje na mesto gde resurs živi, a ne na trenutnu adresu.

Što se tiče ostalih `rel` stavki, oni su veze do podresursa koji sadrže povezane informacije, Sadržaj tagova može biti šta god želite, ali ipak zadržite konstantnost. Konvencija koja je korišćena u ovom primeru je vezana za imenske prostore, tako da su oni jedinstveni. Dva različita tipa resursa su

[5] http://wikipedia.org/wiki/Uniform_Resource_Identifier
[6] http://www.w3.org/TR/uri-clarification/

mogla imati relaciju `checkins` (npr: `users` i `places`), stoga njihova jedinstvenost može doprineti barem dokumentaciji. Možda biste želeli da uklonite imenski prostor, ali je to na vama.

Ove korisnički definisane relacije imaju relativno jedinstvena imena, ali za više generičke relacije možete da razmotrite korišćenje Registra Relacija Veza[7] definisanog od strane IANA, koju koristi Atom (RFC 4287[8]) i mnogo drugih stvari

Kreiranje Hipermedija Kontrola

Ovo je bukvalno prikazivanje linkova u vašim izlaznim podacima. Međutim, ako se odlučite da ovo sprovedete, može biti deo vašeg 'transformacionog' ili 'prezentacionog' sloja.

Ako koristite PHP komponentu Fractal - koja je korišćena kao primer kroz ovu knjigu - onda jednostavno možete da uradite sledeće:

Klasa PlaceTransformer sa ubačenim linkovima u odzivnim podacima.

```
1  public function transform(Place $place)
2  {
3      return [
4          'id'           => (int) $place->id,
5          'name'         => $place->name,
6          'lat'          => (float) $place->lat,
7          'lon'          => (float) $place->lon,
8          'address1'     => $place->address1,
9          'address2'     => $place->address2,
10         'city'         => $place->city,
11         'state'        => $place->state,
12         'zip'          => (float) $place->zip,
13         'website'      => $place->website,
14         'phone'        => $place->phone,
15
16         'links'        => [
17             [
18                 'rel' => 'self',
19                 'uri' => '/places/'.$place->id,
20             ],
21             [
22                 'rel' => 'place.checkins',
23                 'uri' => '/places/'.$place->id.'/checkins',
24             ],
25             [
```

[7] http://www.iana.org/assignments/link-relations/link-relations.xhtml
[8] http://atompub.org/rfc4287.html

```
                'rel' => 'place.image',
                'uri' => '/places/'.$place->id.'/image',
            ]
        ],
    ];
}
```

Ljudi pokušavaju da se prave pametni i da ubace različite relacije zasnovane na `$_SERVER` podešavanjima ili zasnovane na ORM relacijama, ali sve ovo može da vam donese samo probleme. Ako imate ove transformacije, onda ovu grupu podataka morate da napišete samo jednom. Ovo onda sprečava da se prikazuje logika vezana za bazu podataka, a kod ostaje čitljiv i razumljiv.

Sada kad ste uneli ove linkove, ljudi moraju da znaju kako da sa njima interaguju. Možda mislite 'svakako trebao bih da ubacim `GET` ili `PUT` kako bi ljudi znali šta rade'. pogrešno. Ovi linkovi su veze ka resursima, a ne akcijama. Slika postoji za mesto, a mi možemo ili da na slepo pretpostavimo da možemo da na njoj izvedemo neke akcije ili možemo pitati API koje su akcije dostupne i keširati rezultat.

Programsko Otkrivanje Resursa

Ako uzmemo skraćenu verziju ranijeg primera iz ovog poglavlja, možemo da očekujemo da vidimo izlaz kao ovaj:

```
{
    "data": [
            ...
        "links": [
                {
                        "rel": "self",
                        "uri": "/places/2"
                },
                {
                        "rel": "place.checkins",
                        "uri": "/places/2/checkins"
                },
                {
                        "rel": "place.image",
                        "uri": "/places/2/image"
                }
        ]
    ]
}
```

Možemo da pretpostavimoo da će `GET` raditi sa krajnim tačkama `self` i `place.checkins`, ali šta drugo možemo da uradimo sa njima? Iza toga, šta bi trebalo da radimo sa krajnjom tačkom `place.image`?

HTTP nas ovde pokriva, sa jednostavnim i efektivnim glagolom o kome još nije diskutovano: `OPTIONS`.

HTTP zahtev uz pomoć glagola OPTIONS

```
OPTIONS /places/2/checkins HTTP/1.1
Host: localhost:8000
```

Odziv za prethodni HTTP zahtev

```
HTTP/1.1 200 OK
Host: localhost:8000
Connection: close
Allow: GET,HEAD,POST
```

Pregledom zaglavlja `Allow`, mi kao ljudi (ili programski kao API klijent aplikacija) možemo da shvatimo koje opcije su nam dostupne za svaku krajnju tačku. Ovo je ono što Javascript radi često u vašim internet pregledačima prilikom AJAX zahteva, a da toga možda niste ni svesni.

Programski se ovo može uraditi takođe lako i većina HTTP klijenata u bilo kom programskom jeziku će vam omogućiti da napravite poziv `OPTIONS` jednako lako kao što to radite sa `GET` ili `POST`. Ako vam vaš HTTP klijent ovo ne dozvoljava, onda promenite vaš HTTP klijent.

Pravljenje HTTP zahteva OPTIONS koristeći PHP i paket Guzzle

```
use GuzzleHttp\Client;

$client = new Client(['base_url' => 'http://localhost:8000']);
$response = $client->options('/places/2/checkins');
$methods = array_walk('trim', explode(',', $response->getHeader('Accept'));
var_dump($methods); // Outputs: ['GET', 'HEAD', 'POST']
```

Dakle u ovom primeru, znamo da možemo da dobavimo listu čekiranja za jedno mesto koristeći `GET`, a možemo da tome pridodamo tako što ćemo da napravimo `POST` HTTP zahtev ka toj istoj URL adresi. Takođe možemo da proverimo `HEAD`, što je isto što i zahtev `GET`, ali bez HTTP tela. Verovatno ćete morati da ovo uradite drugačije u vašoj aplikaciji, ali ovo je jako korisno kada trebate da proverite da li postoji kolekcija ili resurs bez preuzmanja celokupnog tela sadržaja (t.j: samo tražite `200` ili `404`).

Možda deluje malo ludo da se pravi ovaj dodatni korak kako biste interagovali sa vašim API interfejsom, ali u stvari trebalo bi smatrati ovo lakšim nego pretraživanje kroz dokumentaciju. Razmislite: pokušaj da se pronađe mali link ‘Developers’ na websajtu, onda pretraživanje kroz

dokumentaciju kako biste pronašli pravi API (jer su oni toliko kul da imaju tri), onda pitanja da li imate pravu verziju…nije zabavno. Uporedite ovo sa programski samodokumentovanim API interfejsom, koji može da raste, menja se i širi tokom vremena, preimenovanje URL adresa i…pa to je pravi dobitak, Verujte mi.

Ako znate da se API pridržava REST principa onda *trebate* biti uvereni da se pridržava i HATEOAS - jer reklamirati da je nešto u duhu REST-a bez pridržavanja HATEOAS je velika smrdljiva laž. Na žalost, većina popularnih API interfejsa su veliki smrdljivi lažovi.

> GitHub daje odziv 500, Reddit 501 Not Implemented, Google mape 405 Method Not Allowed. Shvatate. Probao sam mnoge druge i rezultati su jako slični. Ponekad daje nešto identično GET odzivu. Ništa od ovoga nije u redu. – **Izvor**: Zac Stewart, "HTTP metod OPTIONS i mogućnost pravljenja samoopisnih API interfejsa u duhu REST-a"[9]

Ako pravite sopstveni API, ovo možete da uradite sami, onda će vašii klijenti znati da možete da napravite pristojan API.

To je sve što ima u vezi HATEOAS. Sada bi trebalo da znate da napravite API, koji u teoriji nećete mrzeti. Na žalost, verovatno ćete morati da napravite novu verziju za nekoliko meseci, stoga ćemo pogledati API verzionisanje.

[9] http://zacstewart.com/2012/04/14/http-options-method.html