Nefret Etmeyeceğiniz Uygulama Programlama Arayüzleri (API) İnşa Edin

Phil Sturgeon

Sinan Eldem

Nefret Etmeyeceğiniz Uygulama Programlama Arayüzleri (API) İnşa Edin

Herkes API istiyor, öyleyse artık nasıl inşa edildiğini öğrenmenizin zamanı geldi.

Phil Sturgeon ve Sinan Eldem

Bu kitap şu adreste satılmaktadır http://leanpub.com/build-apis-you-wont-hate-tr

Bu versiyon şu tarihte yayımlandı 2014-05-12

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

©2014 Phil Sturgeon ve Sinan Eldem

Kitabı tweetleyin!

Phil Sturgeon ve Sinan Eldem'a kitabını şu adresten Twitter tanıtarak yardımcı olun!

Kitap için önerilen hashtag #build-apis-you-wont-hate-tr.

Kitap için diğerleri ne demiş merak ediyorsanız bağlantıya tıklayarak hashtagları arayabilirsiniz:

https://twitter.com/search?q =#build-apis-you-wont-hate-tr

Bu kitap, "Build APIs You Won't Hate" kitabının Türkçe çevirisidir.

API inşa etmek için bilmeniz gereken tüm detayları, teknolojileri ve yazılım bilgisini size sunmaktadır.

Yazılımı inşa ederken de günümüzün en popüler Php Framework'ü Laravel 4'ü kullanmaktadır.

Çeviri sürecinde meydana gelmiş olası yazım hataları da düzeltilerek kitap sürekli güncelliğini koruyacaktır.

Sonraki tüm sürümleri, kitabı bir defa satın alarak ücretsiz edinebileceksiniz.

İçindekiler

Giriş

Uzun bir süredir API'lar inşa ediyorum ve front-end JavaScript frameworklerinin, iPhone uygulamalarının ve API merkezli mimarilerin artması sayesinde API'lar sunucu taraflı geliştiriciler için de giderek daha yaygın bir hale gelmektedir. Bir yanıyla siz sadece bir veri kaynağından bir şeyleri alıp onu JSON olarak dışarıya verirsiniz ama iş mantığında, veritabanı şeması güncellemelerinde, yeni özellikler veya artık önerilmeyen uç noktalardaki değişikliklerin hayata geçirilmesi süper zordur.

Ben çoğu kaynakların korkunç derecede eksik ya da özel olarak tek bir frameworke dönük olduğunu görmüşümdür. Bu konudaki birçok ders ya da kitap yeterince somut olmayan elmalar ve armutlar örneklerini kullanmaktadır veya ihtiyacınız olan tek uç nokta /users ve /users/1 imiş gibi bahsetmektedir. Son bir yılımı Kapture adlı bir şirkette çalışarak geçirdim ve oradaki esas fonksiyonum çok farklı kullanım durumları olan birçok farklı uç noktalara sahip oldukça büyük bir API'ı devralmak, yeniden inşa etmek ve daha da geliştirmek idi.

Ben şirkete katıldığım zaman söz konusu API v2 idi ve orijinal geliştirici tarafından öldürülesiye hacklenmiş, günümüzde artık önerilmeyen bir ORM kullanıyordu ve FuelPHP'de yazılmıştı. Kapture şirketi iPhone uygulamalarının yeni işlevsellikler gerçekleştirmesi için yeniden inşa edilmesi sürecindeydi, bu nedenle ben bunu dağınıklığı yok etmek ve Laravel 4'ün basit (başlangıçta Symfony tabanlı) Routing, Database Migration'ları, Schema, Seeding ve benzeri özelliklerinden yararlanarak, v3'ü Laravel 4'de inşa etmek için bir fırsat olarak kullandım. Şimdi aynısını v4 için yapıyoruz ama bu sefer yeniden yazmak gerekmiyor, hatta bazı farklı işlevsellikler eklenen v3 reposu v4 için fork edilmiştir ve her ikisi de aynı "API" sunucularında yan yana yaşamakta ve aktif olarak geliştirilmektedir.

API geliştirme konusunda yeni iseniz, birtakım en iyi uygulamalar ve genel iyi önerileri gözden geçirmekle çalışma zeminine varabilirsiniz. Diğer taraftan bazı korku hikayeleri (ve onların nasıl üstesinden geldiklerini/önlediklerini/geri çevirdiklerini) dinleyerek umarım içine düştüğünüz veya düşeyazdığınız yahut da başkalarının düştüğünü gördüğünüz tuzakların birçoğundan kaçınabilirsiniz. Bu kitap herhangi bir dilde veya frameworkte API'lar tasarlama ve inşa etmenin teorisini tartışacaktır. Bu teori çoğunlukla PHP'de inşa edilmiş örneklerle uygulanacaktır ve bazen Ruby ve Python da verilecektir. Kod okuma pek eğlenceli olmadığı için, bu kitap çok fazla kod yoğun olmayacaktır.

Bu kitabın sonunda, iyi bir RESTful API'ın yapması gereken oluşturma, okuma, güncelleme, silme, listeleme, arama ve diğer her şeyi halledebilen bir API inşa edeceksiniz. Burada anlatılan daha ileri konuların bir kısmı şunlardır: uç nokta testleri, hata ayıklama, veri nesnelerini tutarlı ve ölçeklenebilir bir biçimde gömme/içiçe geçirme, cevapların sayfalandırılması (gömülü nesneler dahil olmak üzere) ve HATEOAS linkler.

Bilgi

Bu kitabın özgün ismi “Build APIs You Won’t Hate”‘dir.

Bu kitapta kullanılan bazı kelimelerin okunuşları aşağıda belirtilmiştir. Bu, kitabın daha rahat okunabilmesine olanak sağlayacaktır.

Kısaltmaların Okunuşları

API = Ey Pi Ay Seeder = Siidır Controller = Kontrollır Faker = Feykır Kapture = Kepçır Feature = Fiiçır Scenario = Sinıri JSON = Ceysın Transformer = Transformır URI = Yu Ar Ay URL = Yu Ar El Token = Tokın

Teşekkür

Öncelikle sevgili eşim Bilge ve gözümün ışığı kızım Tuana Şeyma'ya teşekkürler. İyi ki varsınız!

Kitapların çevirisinde tüm süreç boyunca yanımda olan ve çok katkı sağlayan değerli Sergin Arı[1]'ya, kattıklarından dolayı minnettarım. Sen olmadan olmazdı!

[1] http://www.serginari.com

Yararlı Veritabanı Ekimi

Giriş

Her türlü uygulama oluşturmak için ilk adım veritabanını oluşturmaktır. İster bir ilişkisel platform, ister MongoDB, ister Riak ya da başka bir şey kullanıyor olsanız da verilerinizi nasıl saklayacağınız konusunda bir fikriniz olması gerekecektir.

İlişkisel veritabanları için planlamaya bir antite-ilişkiler çizelgesi ile başlayacaksınız ve MongoDB, CouchDB veya ElasticSearch gibi belge tabanlı veritabanları için uygulamanızın bir şema oluşturmasına izin vermeniz yeterli olacaktır. Ama hangi yol olursa olsun, bir peçete üzerinde bile olsa, bir plan oluşturmanız gerekir. Bu kitap, verilerinizi geleneksel ilişkisel bir veritabanının sakladığını kabul edecektir ama bu ilkeler NoSQL sistemlere de kolaylıkla adapte edilebilir.

Bu bölüm sizin bir veritabanını zaten tasarlamış ve inşa etmiş olduğunuzu varsayar. Bu bölüm "Bir veritabanı planlanması" kesimini atlayacaktır, zira bu konuda çok sayıda kitap bulunmaktadır.

Veritabanı Ekimine Giriş

Bir veritabanı şeması tasarlanıp gerçekleştirilmesiyle, sonraki adım bir miktar veri saklamaktır. Gerçek verilerinizi girmek yerine, şemanızın API uygulamanız için uygun olup olmadığını test etmek için "sahte veri" kullanmak çok daha kolaydır. Bu size verilerinizin korunması konusunda kaygı duymadan veritabanınızı didikleyebilme ve tekrar deneyebilme yararı verir.

Bir veritabanının doldurulması süreci "seeding" (ekim) olarak bilinir.

Bu veriler şunlar olabilir:

- test kullanıcılar
- bir demet yorumları olan içerik girişleri
- check-in yapılabilecek uyduruk konumlar
- bir iPhone uygulamasında göstermek için uyduruk bildirimler (her bir tipten)
- çeşitli işlem evrelerindeki kredi kartı ödemeleri - bir kısmı tamamlanmış, bir kısmı yarım ve bir kısmı süper hileli gözüken olmak üzere.

Seeding scriptleri oluşturulması işlemi, sizin bunu elle tekrar tekrar oluşturmak için zaman harcamamanız demektir. Sonuç olarak, API'ınızı geliştirme sırasında işlemleri ne kadar otomatikleştirirseniz, uygulamanız için çok daha fazla düşünülmesi gereken karmaşıklıkları ele almak için daha fazla zamanınız olacaktır.

Sahte veriler gerçekçi kabul testleri için, yararlı içerikle hızlanmak amacıyla serbest çalışanlar/yeni personeller almak için, gerçek müşteri verilerinizi şirketiniz dışındakilere mahrem tutmak için ve canlı verileri geliştirme ortamınıza kopyalama sıkıntısının önlenmesi için gereklidir.

Geliştirme sırasında üretim verilerinin kullanılması niçin kötüdür?

Siz ömrünüzde hiç e-postalar gönderen bir script yazdınız mı ve onu inşa ederken bazı uyduruk kopyalar kullandınız mı? Bu içerikte bazı küstah kelimeler kullandınız mı? Kazara bu e-postayı 10,000 gerçek müşterinin email adresine gönderdiniz mi? Bir şirketin £200,000 üzerinde değer kaybetmesi yüzünden kovuldunuz mu?

Ben yapmadım ama yapan bir adamı biliyorum. Bu adam gibi olmayın.

Ne tür veri kullanmanız gerekir?

Çer çöp! Geliştirme veritabanınız için kesinlikle anlamsız ama doğru veri tipi, boyutu ve biçiminde veriler kullanın. Bunu François Zaninotto[2] tarafından yazılan eğlenceli küçük bir kitaplık olan Faker[3] ile yapabilirsiniz, bu harika bir kitaplıktır.

Seeder'ların İnşası

Benim çalıştığım Kapture şirketi, bünyesinde Database Seeding / Veri Ekme[4] barındıran Laravel frameworkü kullanmaktadır. Bu aslında neredeyse her modern PHP frameworkünün sahip olduğu (veya olması gereken) bir görevidir, dolayısıyla bu ilkeler hepsine uygulanabilir.

Veritabanı ekicilerininizi mantıklı gruplara bölün. "Her tablo için bir seeder" olması gerekmez ama öyle yapabilirsiniz. Bu kurala yapışmamaya çalışmamın nedeni bazen verilerinizi başka tipteki verilerle aynı zamanda inşa etmeniz gerekeceği içindir, bu nedenle bizim Users verilerimiz, bunların ayarları, OAuth tokenları ve arkadaşlık verilerinin yapıldığı aynı "seeder"da oluşturulur. Bu işleri sırf işleri küçük tutmak için birden çok seeder'lar içine koymak boşu boşuna bir egzersiz olacak ve hiçbir neden yokken her şeyin yavaşlamasına yol açacaktır.

Bu Bölümde, bir örnek olarak bir check-in uygulaması kullanacağım. Bu uygulama kullanıcıları ("users") işler ve bunların tüccarlara ("merchants") veya mekanlara ("venues") "check-in"lerini takip eder. "Merchants" ayrıca kampanyalar ("campaigns") veya fırsatlar ("opportunities") da sağlamaktadır.

Bu nedenle, Laravel'e özgü yapıyı göz ardı ederek user seeder'ımız hepsi bir arada şeklinde çok basitleştirilmiş haliyle şöyledir. *Eğer Laravel 4 kullanıyorsanız, `run()` metodunuzda bunu kullanmanız yeterlidir.*

[2] https://twitter.com/francoisz/

[3] https://github.com/fzaninotto/Faker

[4] http://laravel.gen.tr/docs/migrations#database-seeding

Faker ve Eloquent ORM ile bir user oluşturulması

```
$faker = Faker\Factory::create();

for ($i = 0; $i < Config::get('seeding.users'); $i++) {

    $user = User::create([
        'name'               => $faker->name,
        'email'              => $faker->email,
        'active'             => $i === 0 ? true : rand(0, 1),
        'gender'             => rand(0, 1) ? 'male' : 'female',
        'timezone'           => mt_rand(-10, 10),
        'birthday'           => rand(0, 1) ? $faker->dateTimeBetween('-40 years',\
 '-18 years') : null,
        'location'           => rand(0, 1) ? "{$faker->city}, {$faker->state}" : \
null,
        'had_feedback_email' => (bool) rand(0, 1),
        'sync_name_bio'      => (bool) rand(0, 1),
        'bio'                => $faker->sentence(100),
        'picture_url'        => $this->picture_url[rand(0, 19)],
    ]);
}
```

Peki burada ne yapıyoruz? En iyisi kesimin üzerinden bir geçelim:

```
$faker = Faker\Factory::create();
```

Bir Faker olgusu, istihdam ettiğimiz sahtekar sanatçı.

```
for ($i = 0; $i < Config::get('seeding.users'); $i++) {
```

Belirli sayıda kullanıcı istiyoruz, ancak zamandan kazanmak için geliştirme sırasında test işlemi ve evreleme sürecindekinden daha az olmasını öneriyorum.

```
    $user = User::create([
        'name'               => $faker->name,
        'email'              => $faker->email,
```

Rastgele bir isim ve rastgele bir email yap. Onun kullanması için bir rastgele veri havuzu tanımlamak zorunda değiliz, çünkü O SİHİRLİDİR!

```
        'active'             => $i === 0 ? true : rand(0, 1),
```

Tamam yalan söyledim, çer çöpümüz % 100 rastgele değildir. Daha sonra test etmek amacıyla 1 numaralı kullanıcının aktif olmasını istiyoruz.

```
        'gender'             => $faker->randomElement(['male', 'female']),
```

Cinsiyet eşitliği önemlidir.

```
        'timezone'           => mt_rand(-10, 10),
```

Zaman dilimini bir tam sayı olarak saklamaya karar veren orijinal geliştiricimizin yaptığı çok akıllıca bir şeydi.

Uzaklıkları Değil Zaman Dilimlerini Saklayın

Bazı zaman dilimlerinin tam saatler olmadığını bilmiyor musunuz? Nepal'in is UTC/GMT +05:45 olduğunu biliyor muydunuz? Chatham Adalarının (Yeni Zelanda) yaz aylarında UTC/GMT +12:45'ten UTC/GMT +13:45'e geçtiğini biliyor muydunuz? Bazı yerlerin gün ışığından yararlanma zamanında 30 dakika eklediklerini biliyor muydunuz? Zaman damgaları olarak tam sayılar kullanmayın. PHP bir endüstri standardı olan IANA[5] zaman dilimi veritabanını kullanmaktadır. Eğer siz kullanıcılar için `America/New_York` veya `Asia/Khandyga` saklarsanız, uzaklıklar ve gün ışığından yararlanma zamanı otomatik olarak hesaplanacaktır.

```
        'birthday'           => rand(0, 1) ? $faker->dateTimeBetween('-40 years',\
 '-18 years') : null,
```

Tüm kullanıcılarımız hedef yaş grubunda olacak.

```
        'location'           => rand(0, 1) ? "{$faker->city}, {$faker->state}" : \
null,
```

Bir şehir ve bir eyalet/devlet ismi verecek. Bu yabancı ülkeler için de iyi iş görür.

```
        'had_feedback_email' => $faker->boolean,
        'sync_name_bio'      => $faker->boolean,
```

Çok da umurumuzda olmayan bazı kullanıcı flagları. True veya false, ikisi için de.

[5] http://www.iana.org/time-zones

```
        'bio'                => $faker->sentence(100),
```

İçinde 100 karakter olan bir cümle yap.

İşte bu kadar

Bu dosyalardan çok sayıda oluşturacaksınız ve verilerinizin olduğu her tabloyu güzelce doldurmak isteyeceksiniz. Ayrıca, Veritabanı Seeder'ınıza, doldurulacak tüm tabloları silmesini de söyleyeceksiniz. Bunu işlemin başında global olarak yapın, tabloları her seeder'ın başında silmeyin ya da diğer seeder'lardaki tablo içeriklerini aynı işlemde sileceksiniz.

Laravel 4'te genel bir sistem örneği

```
class DatabaseSeeder extends Seeder
{
    public function run()
    {
        if (App::environment() === 'production') {
            exit('Ben kovulmana engel oldum. Sevgiler, Phil');
        }

        Eloquent::unguard();

        $tables = [
            'locations',
            'merchants',
            'opps',
            'opps_locations',
            'moments',
            'rewards',
            'users',
            'oauth_sessions',
            'notifications',
            'favorites',
            'settings',
            'friendships',
            'impressions',
        ];

        foreach ($tables as $table) {
            DB::table($table)->truncate();
        }
```

```
30
31          $this->call('MerchantTableSeeder');
32          $this->call('PlaceTableSeeder');
33          $this->call('UserTableSeeder');
34          $this->call('OppTableSeeder');
35          $this->call('MomentTableSeeder');
36      }
37  }
```

Bu her şeyi siler, sonra da kendi işlerini yapacak diğer seeder'ları çalıştırır.

N> ### Yabancı Anahtarlar N> Yabancı anahtar sınırlamaları zorlandığı zaman bir veritabanını silmek zor olabilir, bu nedenle böyle bir senaryoda N> veri tabanı ekiciniz tabloları truncate etmeden önce `DB::statement('SET FOREIGN_KEY_CHECKS =0;');` ve sonra da kontrolü yeniden etkinleştirmek için `DB::statement('SET FOREIGN_KEY_CHECKS =1;');` çalıştırmalıdır.

İkincil Veriler

Daha önceden de söylediğim gibi, bir diğeriyle ilişkili veriler eklemeniz oldukça muhtemeldir. Bunu yapmak için hangi verinin birincil (users gibi) olacağını çalışmanız ve bir check-in sisteminde sisteminizin isimlendirmesine bağlı olarak belki "venues" (mekanlar) veya "merchants" (tüccarlar) olabilir.

Bu örnek için ben nasıl merchant oluşturulacağını göstereceğim, sonra da "opportunities" (fırsatlar) ekleyeceğim, bunlar esasında "kampanyalardır".

Merchant Tablosu için Birincil Seeder

```
1   <?php
2
3   class MerchantTableSeeder extends Seeder
4   {
5       /**
6        * Run the database seeds.
7        *
8        * @return void
9        */
10      public function run()
11      {
12          $faker = Faker\Factory::create();
13
14          // Birçok tüccar oluştur
15          for ($i = 0; $i < Config::get('seeding.merchants'); $i++) {
```

```
            Merchant::create([
                'name'        => $faker->company,
                'website'     => $faker->url,
                'phone'       => $faker->phoneNumber,
                'description' => $faker->text(200),
            ]);
        }
    }
}
```

Opp Tablosu için Birincil Seeder

```
<?php

use Carbon\Carbon;
use Kapture\CategoryFinder;

class OppTableSeeder extends Seeder
{
    /**
     * İnşa et
     *
     * @param Place
     */
    public function __construct(CategoryFinder $finder, Place $places)
    {
        $this->categoryFinder = $finder;
        $this->places = $places;
    }

    /**
     * Images.
     *
     * @var string
     */
    protected $imageArray = [
        'http://example.com/images/example1.jpg',
        'http://example.com/images/example2.jpg',
        'http://example.com/images/example3.jpg',
        'http://example.com/images/example4.jpg',
        'http://example.com/images/example5.jpg',
    ];
```

```
31
32      /**
33       * Run the database seeds.
34       *
35       * @return void
36       */
37      public function run()
38      {
39          $faker = Faker\Factory::create();
40
41          foreach (Merchant::all() as $merchant) {
42
43              // Bu tüccar için birçok fırsat oluştur
44              foreach (range(1, rand(2, 4)) as $i) {
45
46                  // Eklenecek üç image var
47                  $image = Image::create([
48                      'name' => "{$merchant->name} Image #{$i}",
49                      'url' => $faker->randomElement($this->imageArray),
50                  ]);
51
52                  // Onu hemen başlat ve 2 ay ömür biç
53                  $starts = Carbon::now();
54
55                  // En az birinin kontrolümüzde olması lazım
56                  if ($i === 1) {
57                      // BİR olanın vadesi birazdan dolacak
58                      $ends = Carbon::now()->addDays(2);
59                      $teaser = 'Something about cheese';
60
61                  } else {
62                      $ends = Carbon::now()->addDays(60);
63                      $teaser = $faker->sentence(rand(3, 5));
64                  }
65
66                  $category = $this->categoryFinder->setRandom()->getOne();
67
68                  $opp = Opp::create([
69                      'name'          => $faker->sentence(rand(3, 5)),
70                      'teaser'        => $teaser,
71                      'details'       => $faker->paragraph(3),
72                      'starts'        => $starts->format('Y-m-d H:i:s'),
```

```
                    'ends'           => $ends->format('Y-m-d H:i:s'),
                    'category_id'    => $category->id,
                    'merchant_id'    => $merchant->id,
                    'published'      => true,
                ]);

                // Bu fırsata bu konumu ekle
                $opp->images()->attach($image, [
                    'published' => true
                ]);
            }

            echo "$merchant->name için $i Opps oluşturuldu \n";
        }
    }
}
```

Bu biraz çılgınca görünebilir ve kesinlikle controllerdaki lazy-statik ORM kullanımı ve bazı bağımlılık enjeksiyonunun bir karışımıdır ancak bu seederlar büyük bir miktarda sevgi almamışlardır. Bunlar işlerini tam olarak yaparlar, temeller şunlardır:

```
        foreach (Merchant::all() as $merchant) {
```

Tüm merchant'ları dolaş.

```
            // Bu tüccar için birçok fırsat oluştur
            foreach (range(1, rand(2, 4)) as $i) {
```

Bir merchant için 1 ve 4 arasında opportunity oluştur.

```
                // Eklenecek üç image var
                $image = Image::create([
                    'name' => "{$merchant->name} Image #{$i}",
                    'url' => $faker->randomElement($this->imageArray),
                ]);
```

Örnek images dizimizden veya web sitenizdeki başka bir yerden bir image ekle. Fazlası daha iyi.

```
                $category = $this->categoryFinder->setRandom()->getOne();
```

Kitabın daha sonraki bir bölümünde Finder'lerden bahsedeceğim ama şimdilik bunun rastgele tek bir kategori elde etmenin bir yolu olduğunu bilmeniz yeterlidir.

Geri kalan kısım nispeten açıktır.

Eğer Laravel 4 kullanıyorsanız, komut satırında `$ php artisan db:seed` komutu ile yukarıdaki şeyleri çalıştırabilirsiniz.

Bu ne zaman çalıştırılacak?

Bu çoğu kere elle çalıştırılır ve duruma bağlı olarak otomatik olarak çalıştırılır.

Örneğin, eğer yeni verileri olan yeni bir uç nokta eklemişseniz, ekip üyelerinin son kodu çekmeleri gerektiğini bilmelerini istersiniz ve migrasyonları çalıştırın ve db seed çalıştırın.

Bu aynı zamanda iş yapmak için serbest çalışan biri geldiğinde veya yeni bir geliştirici işe başladığında veya iPhone geliştirmeniz bazı veriler kullanmak istediğinde de harika bir iştir. Tüm bu durumlarda gereken tek şey bu komutun komut satırında çalıştırılmasıdır.

Bu ayrıca evreleme sunucusunda zaman zaman elle çalıştırılır ve API'ın yeni buildlerini dağıttığınız zaman Jenkins test sunucusunda otomatik olarak da çalıştırılabilir.