Yasin Kılıç Follow
Feb 23, 2021 · 6 min read

Symfony Framework ile Rest API ve Authentication

Symfony, güçlü bir PHP çerçevesi(framwork) dir. PHP frameworkleri içerisinde en çok kullanılanlardan biridir.
2005 yılından beri geliştirilen açık kaynaklı PHP MVC web uygulama çatısı olan Symfony, PHP 5 üzerinden Windows, Mac OS, Unix gibi platformlardan çalışmaktadır. Web siteleri ve web uygulamaları oluşturmak için önde gelen PHP çerçeveleri, Symfony bileşenlerinin üzerine inşa edilmiştir.

Teknik Gereksinimler

İlk Symfony uygulamamızı oluşturmadan önce ortamımız bazı teknik gereksinimlere ihtiyaç duymaktadır.

- PHP 7.2.5 veya üzeri sürümlerini ve PHP uzantılarını yükleyin: Ctype, iconv, JSON, PCRE, Session, SimpleXML and Tokenizer (Bu uzantılar, PHP 7 kurulumlarının çoğunda varsayılan olarak yüklenmiş ve etkinleştirilmiştir)
- PHP paketlerini kurmak için kullanılan Composer'ı kurun.

İsteğe bağlı olarak Symfony CLI'yi de kurabilirsiniz. Bu, symfony uygulamamızı yerel olarak geliştirmek ve çalıştırmak için ihtiyacımız olan tüm araçları sağlayan, symfony adlı bir ikili dosya oluşturur.

Symfony CLI'yi kurmak için tıklayınız.

Bu adımları da tamamladıktan sonra aşağıdaki komutu konsol terminalimizden çalıştırarak bilgisayarımızın tüm g arşılayıp karşılamadığını kontrol

edebiliriz.

```
symfony check:requirements
```

Symfony Uygulamaları Oluşturma

Konsol terminalimizi açalım ve yeni bir Symfony uygulaması oluşturmak için aşağıdaki komutlardan herhangi birini çalıştıralım:

```
# bir web uygulaması oluşturuyorsanız bunu çalıştırın
symfony new my_project_name --full

# bir mikro hizmet, konsol uygulaması veya API oluşturuyorsanız bunu çalıştırın
symfony new my_project_name
```

Bu iki komut arasındaki tek fark, varsayılan olarak kurulan paketlerin sayısıdır. — full seçeneği, genellikle web uygulamaları oluşturmak için ihtiyaç duyduğumuz tüm paketleri yükler, böylece yükleme boyutu daha büyük olur.

Symfony ikili dosyasını kullanmıyor isek, Composer'ı kullanarak yeni Symfony uygulamamızı oluşturmak için ise şu komutları çalıştırabiliriz:

```
composer create-project symfony/website-skeleton my_project_name

composer create-project symfony/skeleton my_project_name
```

Komutları çalıştırdıktan sonra oluşan projemizin yapısına göz atalım.

bin/ : En çok kullanacağımız bin/console ve diğer çalıştırılabilir dosyalar bu dizinde bulunur.
config/ : Rotalar, servisler ve paketler olmak üzere yapılandırma dosyalarını içerir.
public/ : Projemizin belge köküdür. Herkesin erişebileceği tüm dosyaları burada barındırabiliriz.
src/ : Tüm PHP kodlarımızın barındığı dizindir.
templates/ : Twig şablonlarının tutulduğu dizindir. (Bizim örneğimiz WEB sitesi olmayacağı için bu dizin projemizde mevcut değildir.)
var/ : Cache ve log gibi otomatik oluşturulan dosyaların depolandığı dizindir.

vendor/ : Third-party dediğimiz üçüncü parti yazılımların bulunduğu dizindir. Composer paket yöneticisi ile indirilen paketler burada tutulur.

Symfony ile ilgili daha detaylı bilgilere web sitesi üzerinden ulaşabilirsiniz.

Şimdi symfony ile örnek bir API geliştirelim.

Öncelikle projelerimizi tutmuş olduğumuz dizine giderek terminal ekranını açalım. Ardından aşağıdaki komutu çalıştıralım:

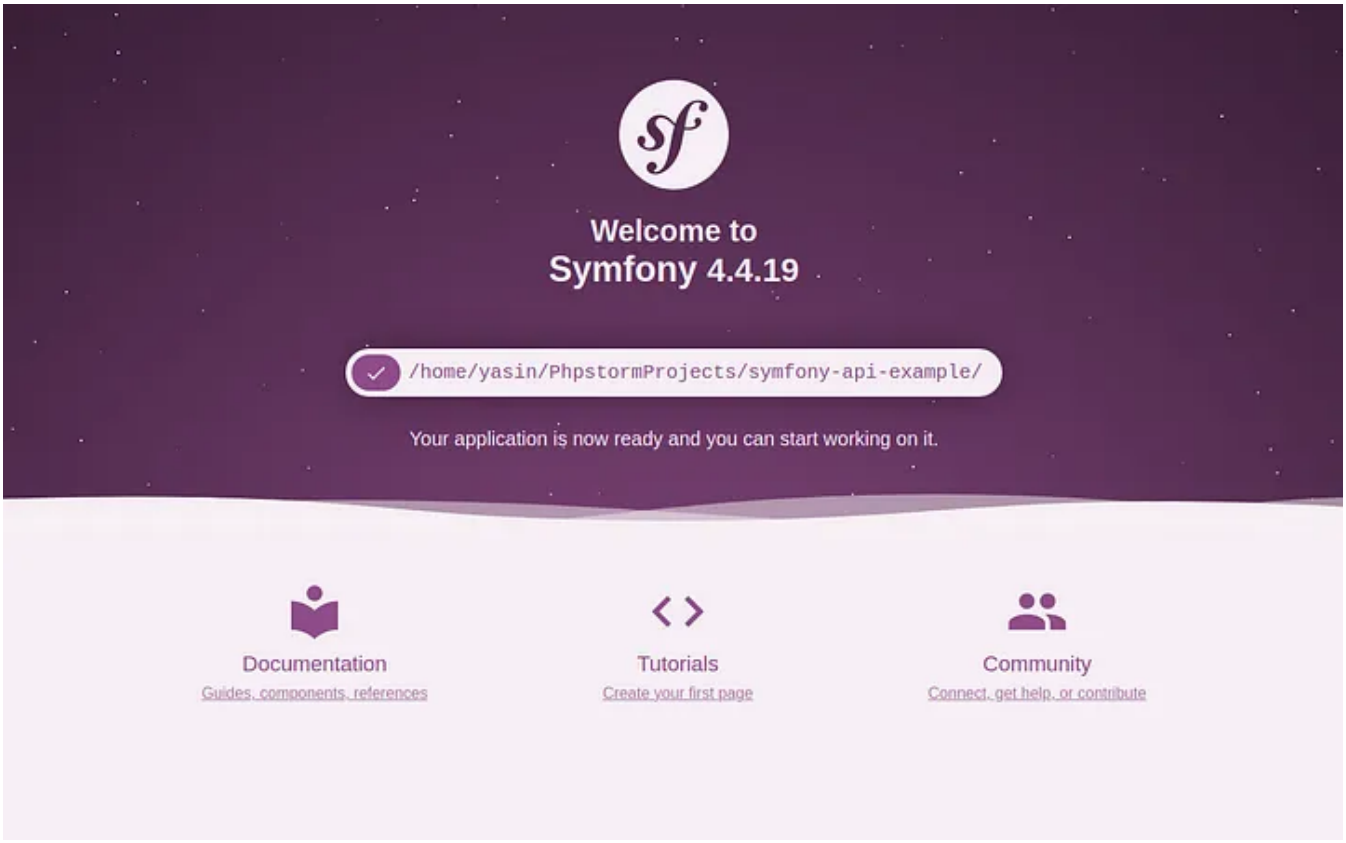
```
symfony new symfony-example --version=lts
```

— *version=lts* ile symfony frameworkünün en yeni LTS sürümünü kullanmak istediğimizi belirtmiş oluruz. Eğer farklı versiyonları kullanmak istiyorsak, örnek olarak — version=4.4 şeklinde kullanmak istediğimiz sürümü belirtebiliriz.

Kurulum tamamlandıktan sonra projemizin ana dizininde aşağıdaki komut çalıştırılarak projemizi ayağa kaldırabiliriz.

```
cd symfony-example
symfony server:start
```

Tarayıcımızda aşağıdaki gibi bir ekran geldiyse projemiz ayağa kalkmış demektir.



Şimdi ihtiyacımız olan paketleri yükleyeceğiz. Bunardan birincisi Doctrin ORM paketi, bir diğeri de komut satırında entity, controller, repository, form ve view oluşturmamızı sağlayan Maker Bundle paketidir. Aşağıdaki komutlar ile her iki paketi de projemize kuralım.

```
composer require symfony/orm-pack
composer require --dev symfony/maker-bundle
```

Paket kurulumlarımızı gerçekleştirdikten sonra *.env* dosyamızda veritabanı ayarlarını yapılandırmamız gerekecektir. MySql kullanıyorsanız aşağıdaki kod satırını örnek alabilirsiniz.

```
DATABASE_URL=mysql://username:password@127.0.0.1:3306/database_name?
serverVersion=8.0
```

Paket kurulumlarımızı ve veritabanı configürasyonlarımızı yapmış olduk. Örnek bir entity oluşturarak başlayalım.

Maker Bundle paketinin bize sağlayacağı kolaylıklardan faydalanarak **Subscriber** ve **City** adında iki ayrı entity oluşturalım. Bunun için terminal ekranından aşağıdaki

komutlar çalıştırmamız yeterli olacaktır.

```
php bin/console make:entity Subscriber
php bin/console make:entity City
```

Bu komutlar çalıştırıldıktan sonra *src/Entity* dosya yolunda **Subscriber** ve **City** sınıflarımız oluşur. Bu sınıflar üzerinde manuel olarak entity nize ait özellikleri (property) belirtebilir ya da yukarıdaki komutlar çalıştırıldıktan sonra komut satırındaki bize yöneltilen soruları cevaplayarak da oluşturabiliriz. Buradaki tercih tamamen bize kalmıştır.

Subscriber.php:

City.php:

Şimdi ise entity lerimize karşılık gelecek olan tablolarımız veritabanında migration yöntemini kullanarak oluşturalım. Bunun için ise aşağıdaki komutları sırası ile terminal ekranında çalıştıralım.

```
php bin/console make:migration
php bin/console doctrine:migrations:migrate
```

Veritabanımızı kontrol ettiğimizde ise *subscriber* ve *city* adında iki tablonun oluştuğunu göreceksiniz.
Tablomuz oluştu evet ama elimizde herhangi bir veri mevcut değil. Manuel olarak tablo üzerinden veri girmek yerine, Faker ile veriler oluşturalım.

Bunun için *orm-fixtures* ve *faker* paketlerini kuralım. Terminal ekranımızdan aşağıdaki komutları sırası ile çalıştıralım:

```
composer require --dev orm-fixtures
composer require fzaninotto/faker
```

İlk komut çalıştıktan sonra sonra *src/DataFixtures* altında *AppFixtures.php* dosyası oluşacaktır. Bu dosya üzerinde değil de farklı iki dosya üzerinde çalışacağız. Aşağıdaki iki komutu terminal ekranından çalıştıralım.

```
php bin/console make:fixtures SubscriberFixtures
php bin/console make:fixtures CityFixtures
```

src/DataFixtures dosya yolunda oluşan iki dosyamızı da aşağıdaki gibi güncelleyelim.

SubscriberFixtures.php:

CityFixtures.php:

Burada rastgele 10 adet abone kaydı ve 3 adet de şehir kaydı oluşturuyoruz. **CityFixtures** sınıfında **addReference** fonksiyonu ile de oluşturulan her şehir için bir de referans oluşturuyoruz. Bu referansları **SubscriberFixtures** sınıfında **getReference** fonksiyonu ile çağırarak ilgili alana set ediyoruz.

Aşağıdaki komutu çalıştırmamız ile beraber verilerimiz rastgele oluşacaktır.

```
php bin/console doctrine:fixtures:load
```

Şimdi **SubscriberRepository** ve **CityRepository** sınıflarımızı aşağıdaki gibi düzenleyelim.

İşlevlerimizi gerçekleştireceğimiz **SubscriberController** sınıfımızı oluşturalım. Bunun için çalıştıracağımız komut ise

```
php bin/console make:controller SubscriberController
```

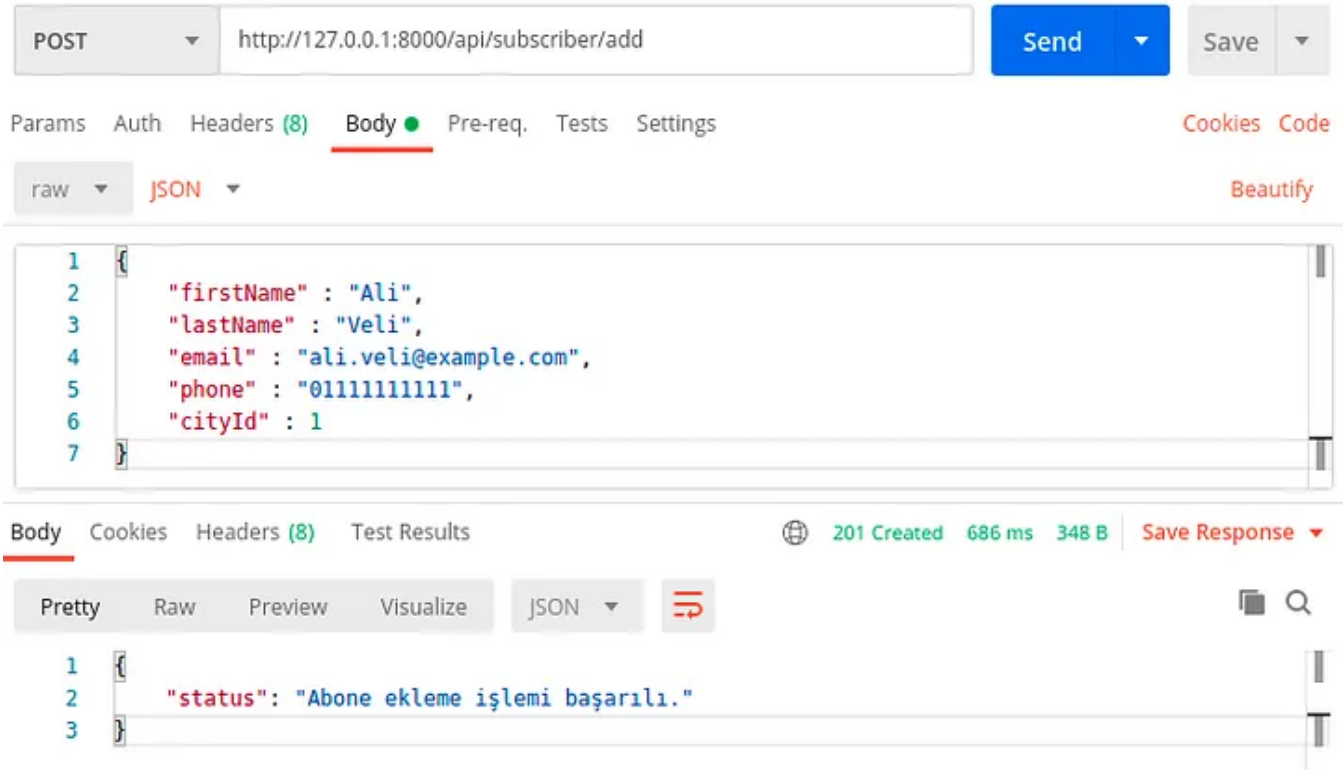
dır.

Bu komut ile beraber *src/Controller* dosya yolunda **SubscriberController** sınıfımız oluşmuştur.

SubscriberController sınıfımızı aşağıdaki gibi düzenleyelim.

Tüm işlemlerimizi tamamlamış bulunmaktayız. Şimdi Postman kullanarak CRUD işlemlerini gerçekleştirelim.

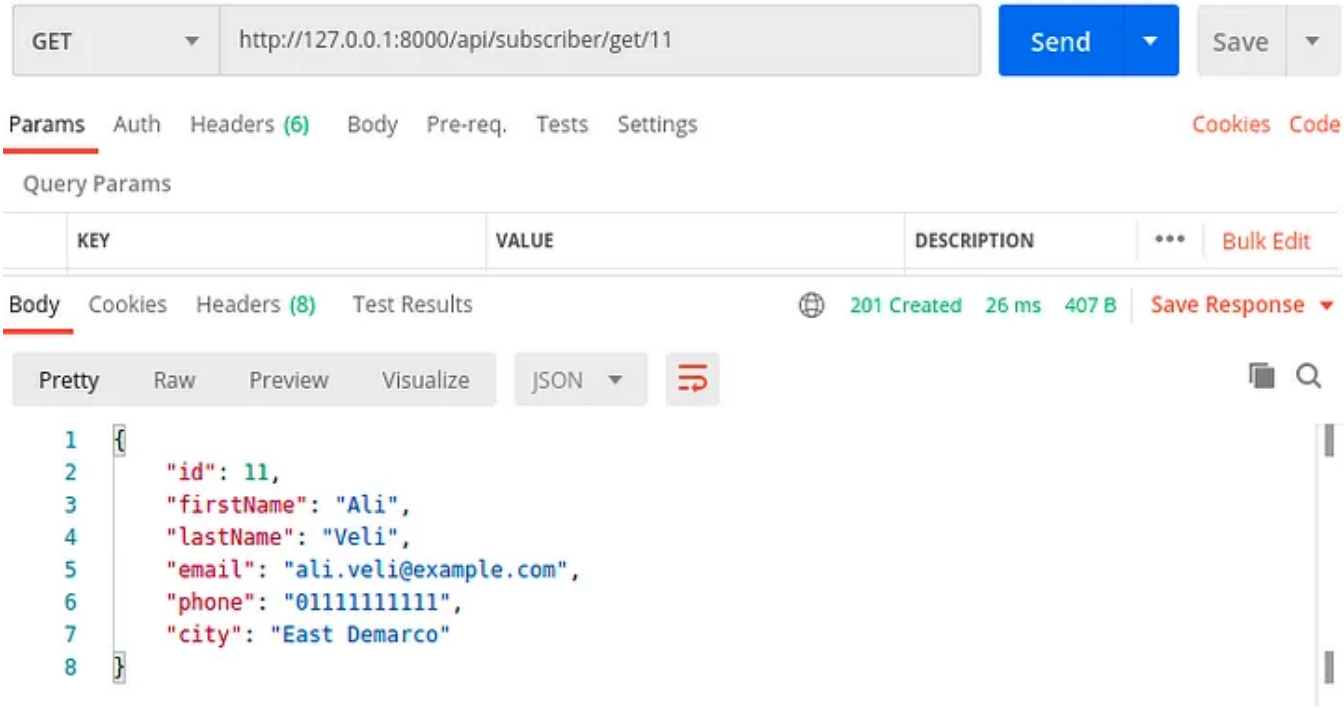
CREATE
Request Method : POST
URL : http://127.0.0.1:8000/api/subscriber/add



READ

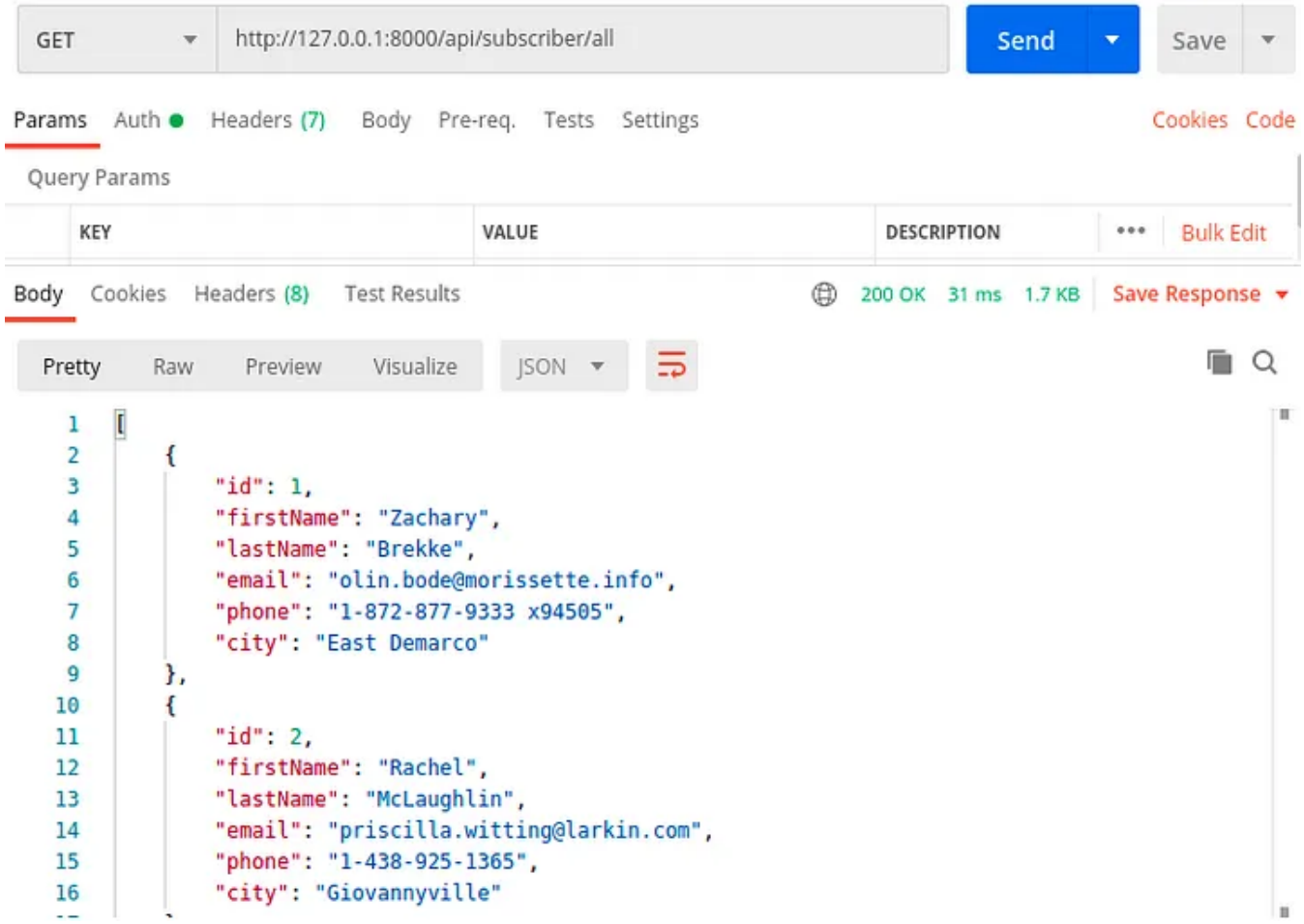
Request Method : GET

URL : http://127.0.0.1:8000/api/subscriber/get/11



Request Method : GET

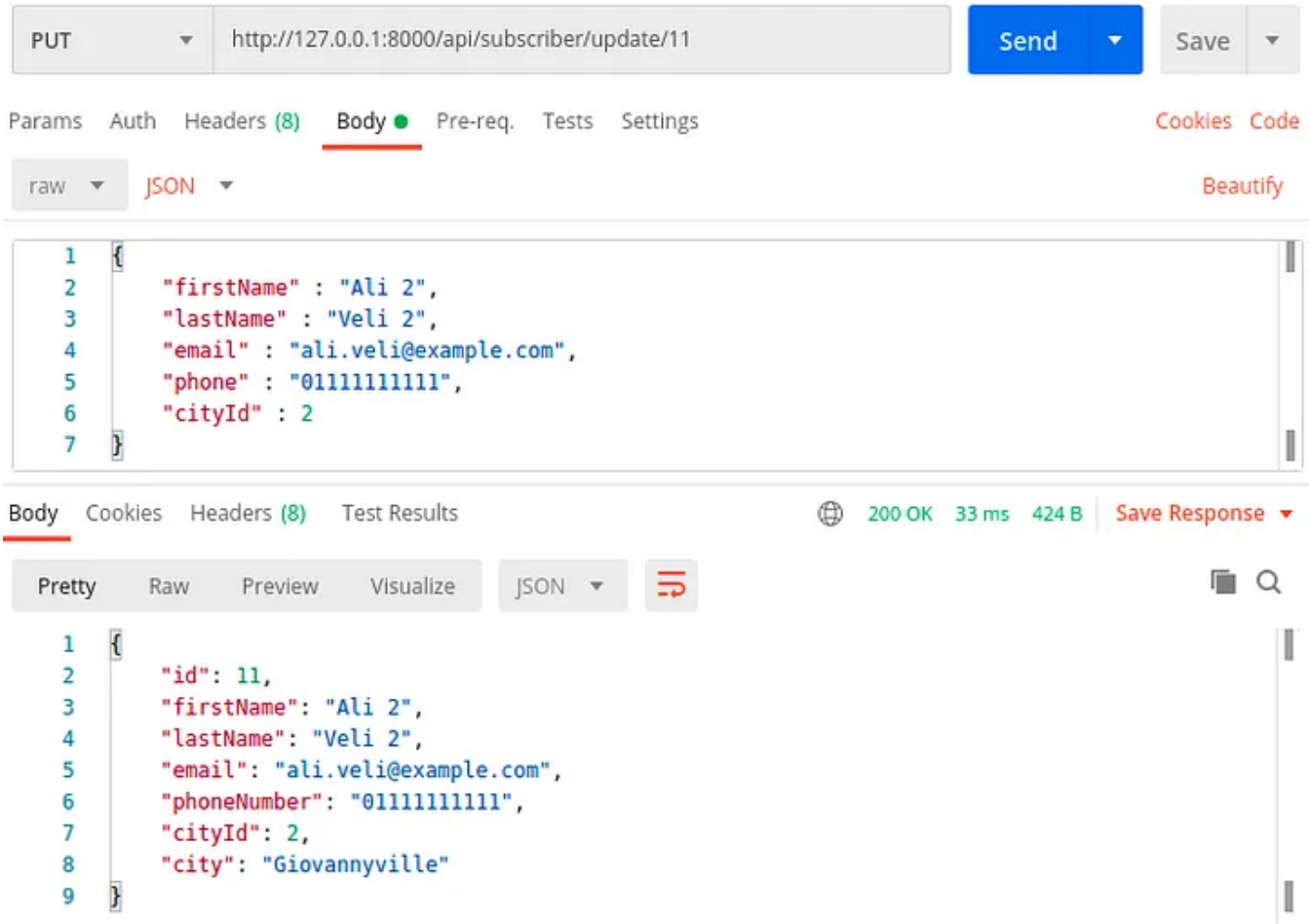
URL : http://127.0.0.1:8000/api/subscriber/all



UPDATE

Request Method : PUT

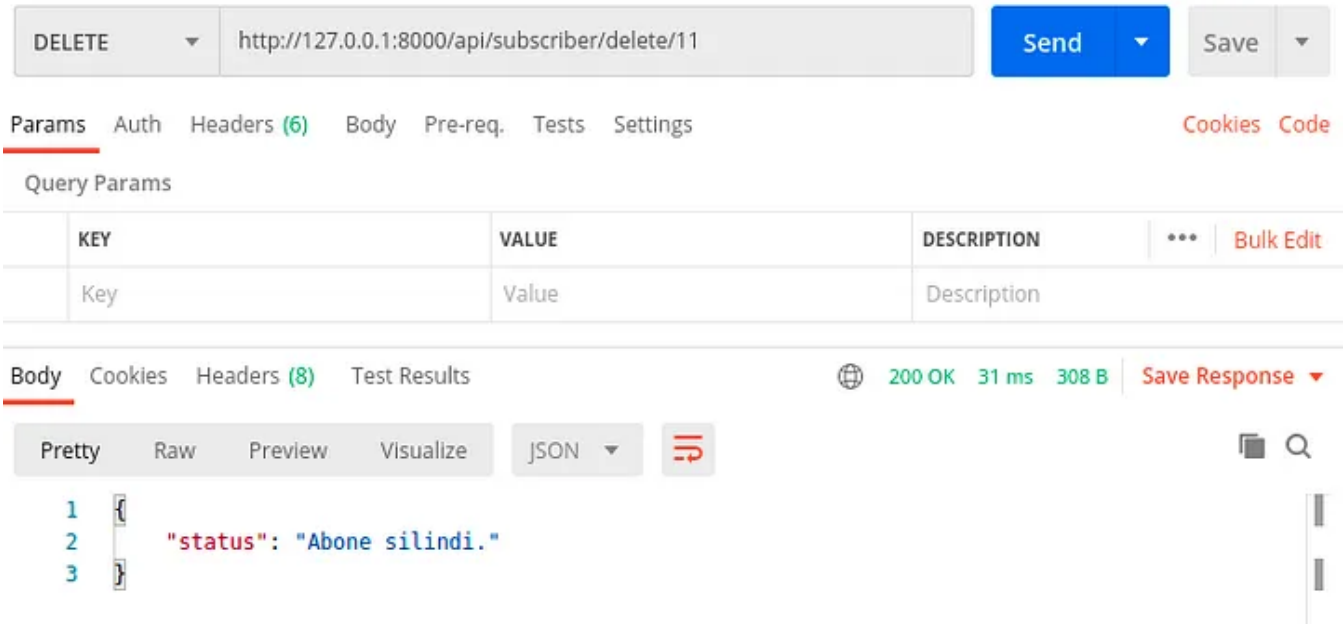
URL : http://127.0.0.1:8000/api/subscriber/update/11



DELETE

Request Method : DELETE

URL : http://127.0.0.1:8000/api/subscriber/delete/11



JWT Kimlik Doğrulaması (JWT Authentication)

JWT (JSON Web Token), istemci ve sunucu arasında veri taşımak için kullandığımız popüler bir teknolojidir. Sınırsız miktarda veri içerebilen bir kodlayıcı dizesidir. Symfony API'mizde bu kimlik doğrulamasını sağlayacak olan bir paket kullanacağız. Aşağıdaki komut ile paketimizi projemize dahil edebiliriz.

```
composer require lexik/jwt-authentication-bundle
```

Gizli anahtarımızı **(Secret Key)** oluşturmak için özel **(Private Key)** ve genel **(Public Key)** bir anahtar çifti oluşturmamız gerekir. Özel anahtar, JSON web belirteçlerini imzalamak için kullanılacaktır.

Private ve public anahtar çiftimizi oluşturmak için ağaşıdaki kodu çalıştıralım.

```
php bin/console lexik:jwt:generate-keypair
```

Bu kod, *config/jwt* dosya yolunda anahtar çiftimizi oluşturacaktır. Aynı zamanda bu anahtar çiftinin yappılandırılması için aşağıdaki dosyalarda ilgili satırları ekler:

.env:

```
JWT_SECRET_KEY=%kernel.project_dir%/config/jwt/private.pem
JWT_PUBLIC_KEY=%kernel.project_dir%/config/jwt/public.pem
JWT_PASSPHRASE=your_secret_passphrase
```

lexik_jwt_authentication.yaml:

```
lexik_jwt_authentication:
    secret_key: '%env(resolve:JWT_SECRET_KEY)%'
    public_key: '%env(resolve:JWT_PUBLIC_KEY)%'
    pass_phrase: '%env(JWT_PASSPHRASE)%'
```

Kurulum tamamlandı. Şimdi User entity mizi oluşturalım ve oluşan User sınıfını aşağıdaki gibi düzenleyelim.

```
php bin/console make:entity User
```

Yaptığımız değişiklikleri veritabanına oluşturmaya zorlama için aşağıdaki komutu çalıştıralım.

```
php bin/console doctrine:schema:update --force
```

UserRepository sınıfımızı da aşağıdaki gibi düzenleyelim.

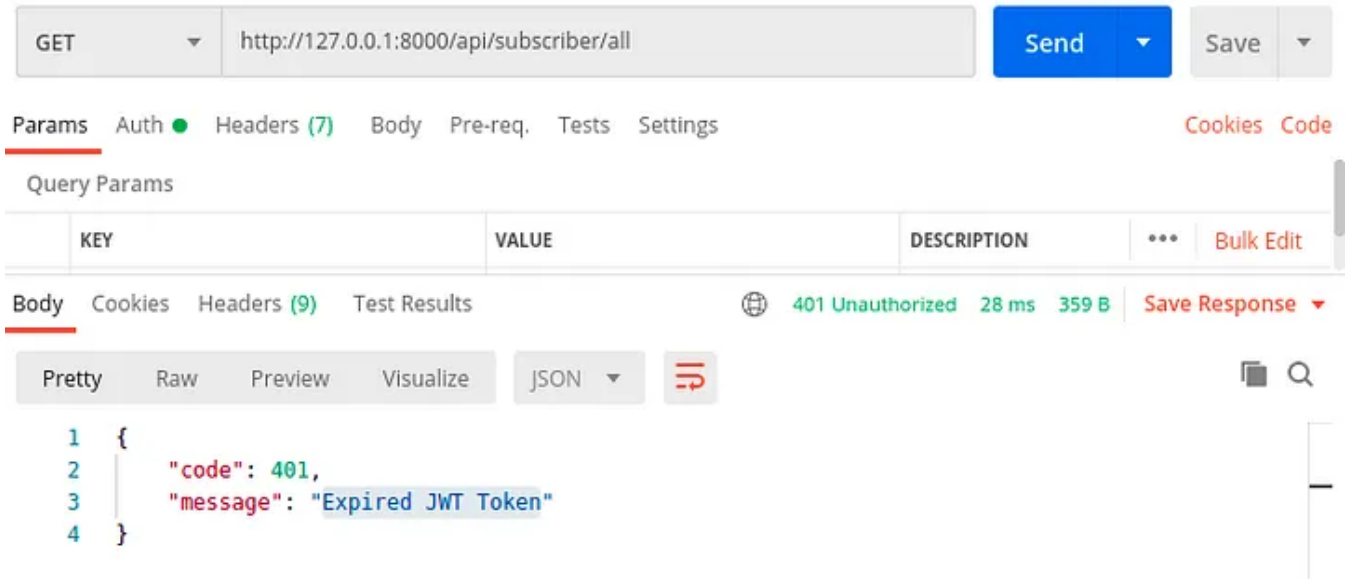
Sıra geldi Symfony nin güvenlik sistemindeki configürasyonlara. *config/packages* dosya yolunda bulunan *security.yaml* dosyasını aşağıdaki gibi düzenleyin.

Şimdi ise *src/Service* dosya yoluna **ResponseService** sınıfımızı, ardında da terminal üzerinden **AuthController** sınıfımızı oluşturalım. Oluşturduğumuz sınıfları ise aşağıdaki gibi düzenleyelim.

```
php bin/console make:controller AuthController
```

Son olarak da *routes.yaml* dosyamızda rotalarımızı tanımlayıp Postman üzerinden testlerimize başlayabiliriz.

Artık testlerimize geçebiliriz. İlk etapta önceden yazmış olduğumuz API lerden birini test edelim.

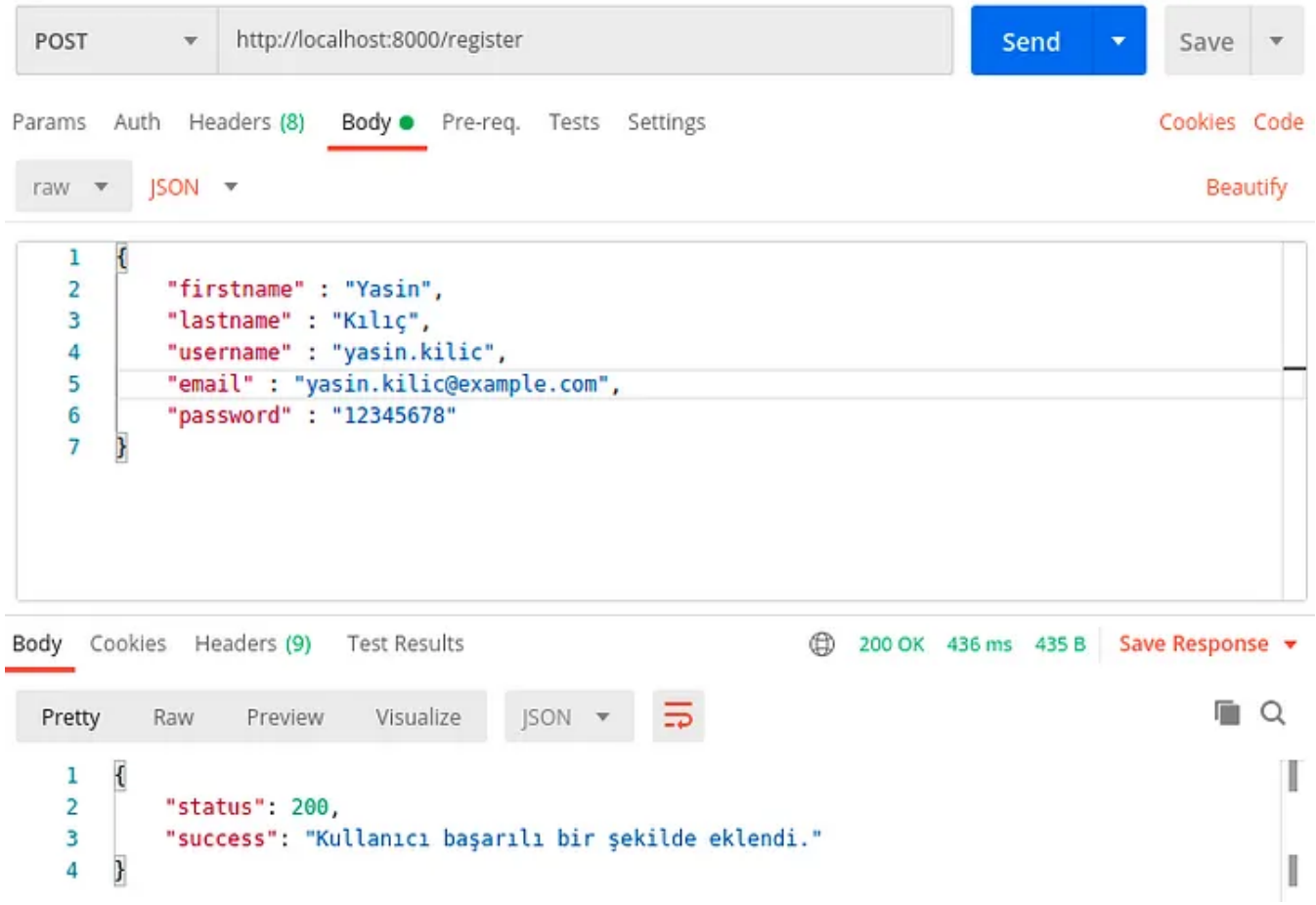


Görmüş olduğunuz gibi token ın süresinin dolduğu uyarısını aldık. Diğer API leri de çalıştırsak aynı hatayı oralarda da alacağız. Şimdi ise kimlik doğrulamasına adım adım gidelim. Öncelikle kullanıcı kaydı ile yani register ile başlayalım.

REGISTER

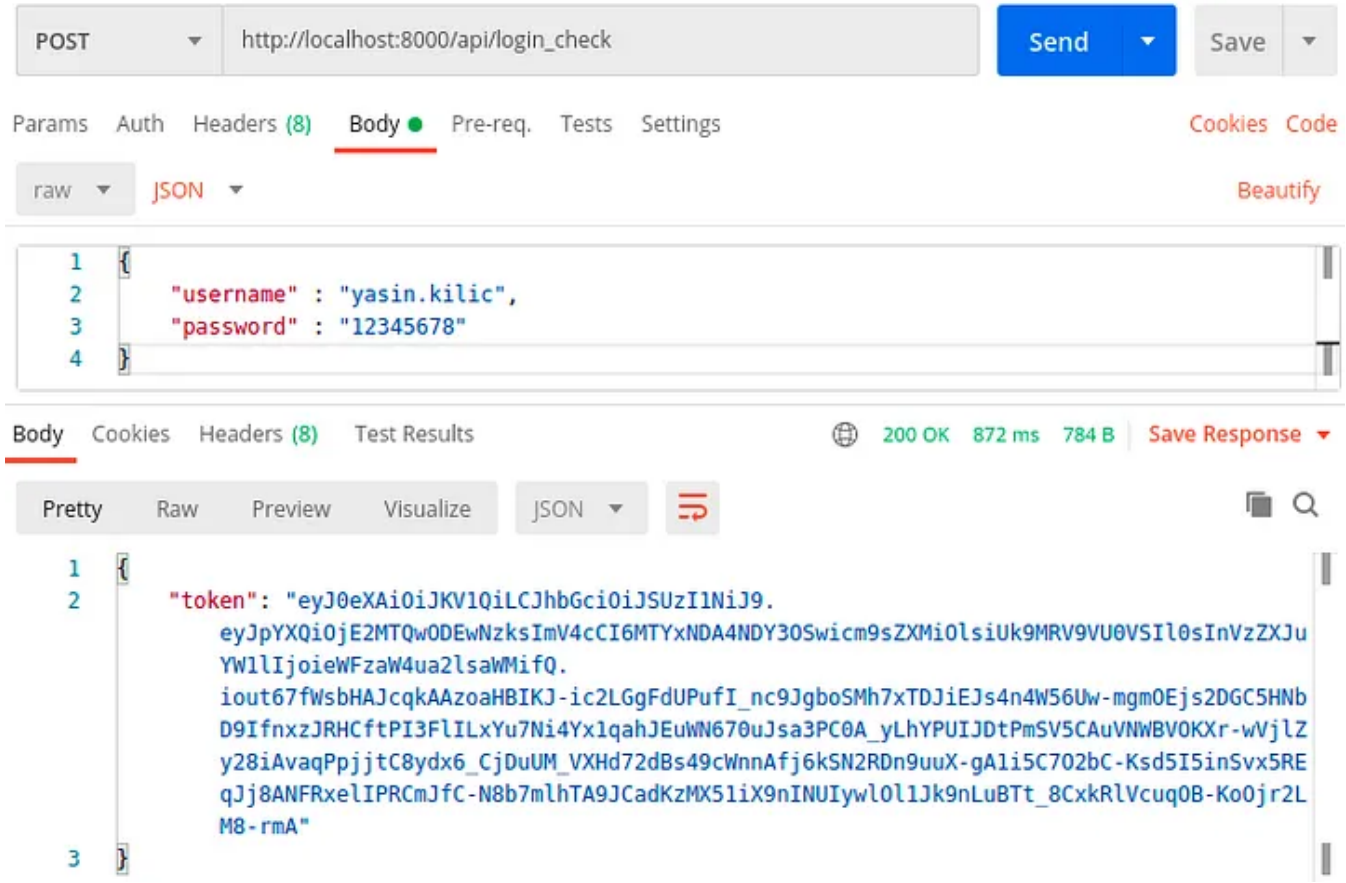
Request Method: POST

URL: http://localhost:8000/register



Başarılı bir şekilde kullanıcımızı oluşturmuş olduk. Sıra geldi oluşturmuş olduğumuz bu kullanıcıyı doğrulama işlemine. Bunun için *api/login_check* apisini kullanacağız.

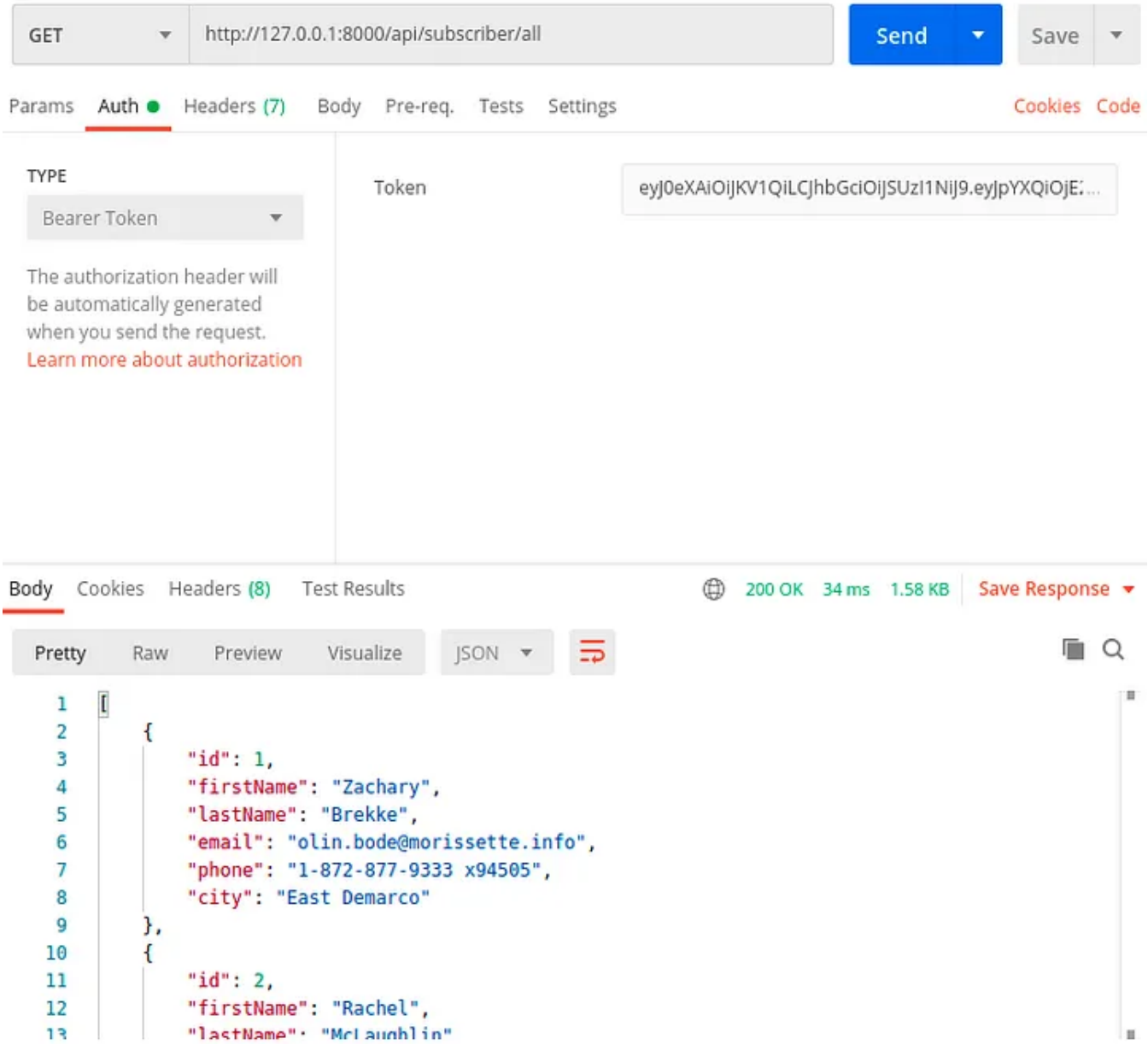
LOGIN CHECK
Request Method: POST
URL: http://localhost:8000/api/login_check



Görmüş olduğunuz username ve password bilgileri ile kullanıcı doğrulama işlemini gerçekleştirdik. *api/login_check* API si, doğrulama başarılı bir şekilde gerçekleştiği için bize token değerini geri döndürdü.

Az önce http://127.0.0.1:8000/api/subscriber/all API sini kullandığımızda token ımız olmadığı için hata almıştık. Şimdi aynı API yi token ı tanımlayarak tekrar çağıralım.

Öncelikler Authorization tab menüsünü açıyoruz. Burada Authorization türünü **Bearer Token** olarak seçiyoruz. Token ımızı da ilgili alana girdikten sonra API mize tekrar istek gönderiyoruz.



Ve sonuç bize başarılı bir şekilde geri döndü.

Bu yazımızda Symfony ile temel düzeyde API geliştirdik ve kimlik doğrulaması gerçekleştirdik.

Projeye buradan erişebilirsiniz.

Faydası olması dileğiyle.

Kaynakça:

https://symfony.com/

Symfony Rest Api Authentication