LARAVEL İLE REST API

Laravel Projesi Oluşturma:

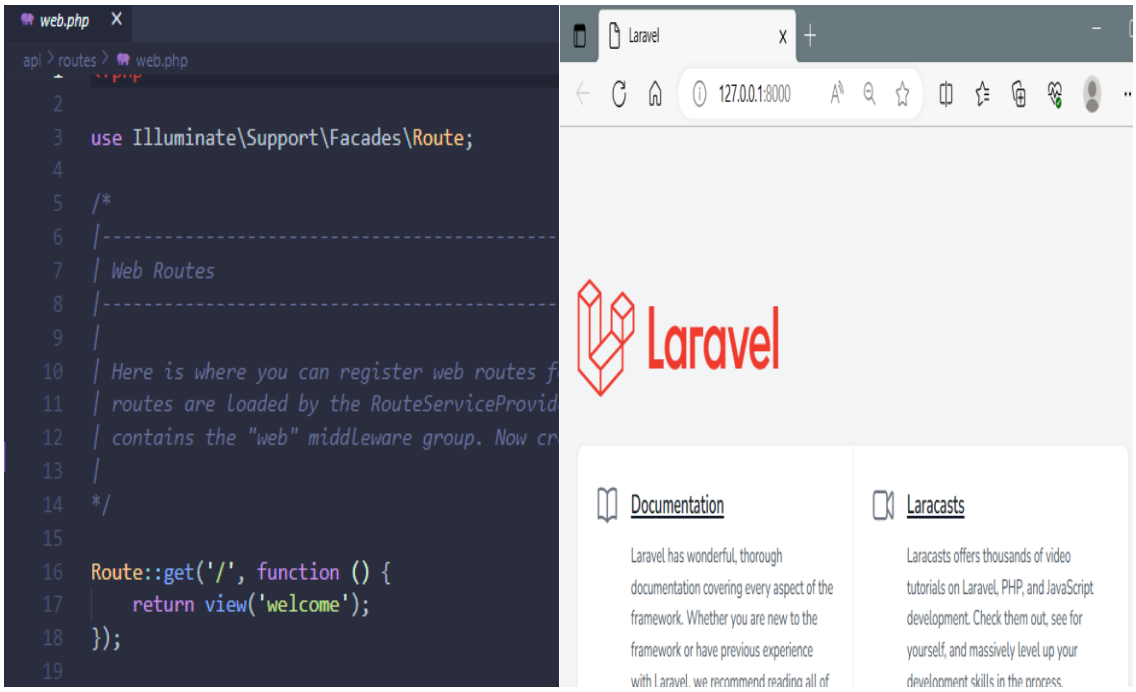
```
composer create-project laravel/laravel api
```

Laravel projesini çalıştırmak için:

```
C:\Users\User\Desktop\laravelapi\api>php artisan serve
```

Bu komut Laravel projesini ayağa kaldırır.

Web.php dosyasının altında gelen “welcome” View^’dan gelir.



CRUD işlemlerini gerçekleştirmek için migrations ve model dosyalarına ihtiyacımız var.

Item adında bir Model oluşturmak için:

```
C:\Users\User\Desktop\laravelapi\api>php artisan make:model Item -m
```

Bu komutun en sonundaki –m ; Item modeline bağlı olarak bir migration dosyası oluşturur.

```
C:\Users\User\Desktop\laravelapi\api>php artisan make:model Item -m

  INFO  Model [C:\Users\User\Desktop\laravelapi\api\app/Models/Item.php] created successf
ully.

  INFO  Migration [C:\Users\User\Desktop\laravelapi\api\database\migrations/2023_07_24_12
3658_create_items_table.php] created successfully.
```

Oluşan migration dosyalarının içeriğinde, tabloya ait sütunlar oluşturulur ve bu migrate edildiği zaman tablo oluşturulmuş olacak.

Database dosyasının altında bulunan migrations klasöründe item tablosu gelir.

```php
    */
    public function up()
    {
        Schema::create('items', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->string('desc'); //aciklama
            $table->integer('quantity'); //adet
            $table->timestamps(); //kayitlar uzerinde created_at ve upd
        });
    }
```

(keyword) function

Tablonun sütunları oluşturulur.

Model dosyasına aşağıdaki kod satırı eklenir:

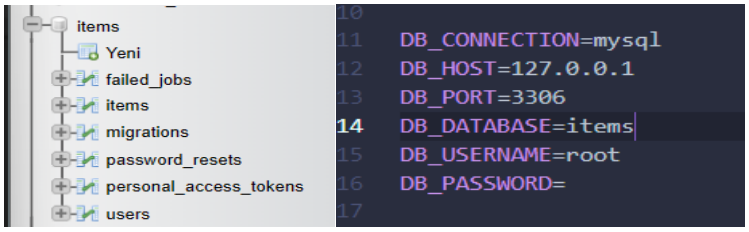
```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

0 references | 0 implementations
class Item extends Model
{
    0 references
    protected $fillable = ['name','desc','quantity']; //degisecek zorun
}
```

Burada ‘name’, ‘desc’, ‘quantity’ alanlarına veri eklenmesine izin verilirken diğer alanlara izin verilmez. Bu özellik, güvenlik açısından önemlidir ve bilinçli olarak hangi alanların dışarıdan doldurulabileceğini belirtmek için kullanılır. Bu sayede ‘Item’ modeli veritabanı işlemleri için kullanılabilir ve güvenli bir şekilde dışarıdan gelen verilerle doldurulabilir.

```
C:\Users\User\Desktop\laravelapi\api>php artisan migrate
```

Bu komut ile oluşturduğumuz tablonun database’de oluşmasını sağlarız.

Phpmyadmin’de önceden oluşturulan “items” adlı database’e bu tabloların gelmesi için .env dosyasında database adını “items” olarak değiştirilir. “php artisan migrate” komutu çalıştırıldığında migrations klasöründe bulunan tabloların “items” database’ine geldiği görülür.



Laravel, MVC (Model-View-Controller) mimarisi temel alınarak geliştirilen bir framework'tür.

MVC mimarisi, web uygulamalarının yapılandırılması ve kodun düzenlenmesini kolaylaştıran bir tasarım desenidir.

Model: Uygulamanın veritabanı işlemlerini ve iş mantığını yönetir. Veritabanı tablolarıyla etkileşime girer ve veri alma, ekleme, güncelleme ve silme gibi veritabanı işlemlerini yürütür. Aynı zamanda, uygulama mantığını da içeren veri işleme süreçlerini bu katmanda tanımlar.

View: Kullanıcı arayüzünü temsil eder. Kullanıcıya sunulan HTML, CSS ve JavaScript kodlarını içerir. Bu katmanda, sunum ve kullanıcı etkileşimi ile ilgili işlemler gerçekleştirilir. Kullanıcının gördüğü sayfaların ve içeriklerin düzeni ve görünümü burada tanımlanır.

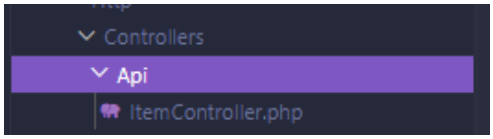
Controller: Model ve View arasındaki iletişimi sağlar. Kullanıcı tarafından yapılan istekleri alır, gerekli işlemler için Model'i kullanır ve sonuçları View ile birleştirerek kullanıcıya sunar. Yani, işlem yapılacak veri ve işlem sonucu View ile birleştirilerek kullanıcıya sunulur.

Bu projede Model oluşturuldu. API yazıldığı için View'a gerek yok. Controller dosyası oluşturulmalı.

Bu Controller, veritabanına bağlanıp gerekli işlemleri yapacak.

```
C:\Users\User\Desktop\laravelapi\api>php artisan make:controller Api\ItemController --api
```

Bu komut controller klasörüne "Api" adlı bir klasör oluşturur ve bunun içine yazar. –api demesek de Laravel bir controller oluşturur fakat –api dediğimiz zaman Laravel, bunun bir api controller olduğunu anlar.



api komutuyla oluşturulduğu için çeşitli fonksiyonları hazır olarak sunar (index(item döndürme), store(kaydetme/ekleme), show(detay gösterme), update(güncelleme), destroy(silme)).

Burada veritabanına bağlanıp işlemler gerçekleştirilecek.

→ "index" fonksiyonunda tüm item'ları json formatında döndürür.

```
     *
     * @return \Illuminate\Http\Response
     */
    0 references | 0 overrides
    public function index()
    {
        //tüm item'lari dondurme
        $items = Item::all(); //ilgili tabloda ne kadar veri varsa dondurur
        return $items; //json formatinda dondurur
    }
```

→ “store” fonksiyonunda kayıt işlemi yapılır.

```
     * @param  \Illuminate\Http\Request  $request
     * @return \Illuminate\Http\Response
     */
    0 references | 0 overrides
    public function store(Request $request)
    {
        $item = new Item();
        $item->name = $request->name; //istekten gelen name'i model aracılığıyla tablo
        $item->desc = $request->desc;
        $item->quantity = $request->quantity;

        $item->save(); //kaydetme islemi
    }
```

“store” fonksiyonu, HTTP isteğini alır ve istekteki verileri kullanarak yeni bir “Item” kaydını veritabanına ekler.

`public function store(Request $request)`: `store` fonksiyonu bir http isteğini kabul eder ve ‘$request’ parametresi aracılığıyla bu isteğe erişir. ‘$request’, bir nesnedir ve isteğin tüm bilgilerini içerir.

`$item = new Item();` Yeni bir ‘Item’ modeli oluşturur ve bu model, veritabanında ‘items’ tablosuna karşılık gelir.

`$item->name = $request->name;`: `$request` Nesnesi, HTTP isteğindeki tüm verilere erişim sağlar. http isteğindeki ‘name’ alanını alır ve oluşturulan ‘Item’ modelinin ‘name’ özelliğine atar. Aynı işlemler desc ve quantity için de yapılır.

Verileri modele atadıktan sonra ‘save’ fonksiyonunu kullanarak bu yeni ‘Item’ kaydını veritabanına ekler. Yani, yeni bir ‘Item’ kaydı oluşturulur ve veritabanına kaydedilir.

Bu kod, API üzerinden gelen verilerin işlenmesi gibi senaryolarda kullanılarak, istemci tarafından gönderilen verileri alır, bir ‘Item’ modeli oluşturur ve bu verileri veritabanına ekler. Böylece, veritabanında yeni bir öğe oluşturulmuş olur.

→ “show” fonksiyonunda, bir HTTP isteği ile veritabanında belirli bir ‘Item’ kaydı görüntülenir.

```
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    0 references | 0 overrides
    public function show($id)
    {
        $item = Item::find($id);
        return $item;
    }
```

Bir $id parametresi alır. Bu $id parametresi, görüntülemek istediğimiz ‘Item’ modelinin veritabanındaki benzersiz kimliğini (primary key) temsil eder.

`$item = Item::find($id);`

Veritabanında, $id parametresine sahip olan 'Item' kaydını bulur ve bu kaydı $item değişkenine atar.

`return $item;`: `$item` değişkeninde bulunan 'Item' modelini geri döndürür. Bu, istemci tarafına belirtilen $id değerine sahip 'Item' modelinin bilgilerini döndüren bir HTTP yanıtı oluşturur. Eğer veritabanında böyle bir $id değerine sahip kayıt yoksa 'null' dönecektir.

→ "update" fonksiyonu, HTTP isteği ile veritabanında belirli bir 'Item' kaydını güncellemek için kullanılır.

```
     * @param  \Illuminate\Http\Request  $request
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    0 references | 0 overrides
    public function update(Request $request, $id)
    {
        $item = Item::findOrFail($request->id);
        $item->name = $request->name;
        $item->desc = $request->desc;
        $item->quantity = $request->quantity;

        $item->save();
        return $item;
    }
```

`public function update(Request $request, $id)`: `update` fonksiyonu, bir HTTP isteğini kabul eder ve $request parametresi aracılığıyla bu isteğe erişir. Aynı zamanda $id parametresi de alır. $id, güncellenmek istenen 'Item' kaydının veritabanındaki primary key'ini temsil eder.

`$item = Item::findOrFail($request->id);` veritabanında, $request nesnesinden gelen $id değerine sahip olan 'Item' kaydını bulur ve bu kaydı $item değişkenine atar.
`Item::findOrFail($request->id)` ile veritabanından ilgili modeli bulmak için Eloquent ORM'ni 'findOrFail' metodu kullanılır. Eğer veritabanında belirtilen $id değerine sahip bir kayıt yoksa, ModelNotFoundException hatası fırlatılır.

`$item->name = $request->name;`: `$request` nesnesi, HTTP isteğindeki tüm verilere erişim sağlar. Bu kod satırı, HTTP isteğindeki 'name' alanını alır ve $item modelinin 'name' özelliğine atar. Aynı işlemler desc ve quantity için de yapılır.

`$item->save();` Verileri modele atadıktan sonra, 'save' fonksiyonu kullanılarak değişiklikler veritabanında güncellenir. Yani, veritabanında belirtilen $id değerine sahip olan 'Item' kaydı güncellenir.

`return $item;`: `$item` değişkeninde bulunan güncellenmiş 'Item' modelini geri döndürür. Bu, istemci tarafına güncellenmiş 'Item' modelinin bilgilerini içeren bir HTTP yanıtı oluşturur.

→ “destroy” fonksiyonu bir HTTP isteği ile veritabanında belirli bir ‘Item’ kaydını silmek için kullanılır.

```
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    0 references | 0 overrides
    public function destroy($id)
    {
        $item = Item::destroy($id);
        return response()->json(['message'=>'silme başarılı']);
    }
}
```

`public function destroy($id)`: `destroy` fonksiyonu, $id parametresi alır. Bu $id parametresi silmek istediğimiz ‘Item’ modelinin veritabanındaki primary key’ini temsil eder.

`$item = Item::destroy($id);`: `Item::destroy($id)` ile veritabanından belirtilen $id değerine sahip ‘Item’ kaydı silinir.

`return response()->json(['message'=>'silme başarılı']);` silme işlemi başarıyla gerçekleştirildikten sora, response()->json fonksiyonu ile bir JSON yanıtı oluşturulur ve istemciye gönderilir. JSON yanıtında message adında bir alan bulunur ve değer “silme başarılı” olarak atanır.

Controller fonksiyonları yazıldı fakat çeşitli API rotalarının, endpointlerin olması gerekir. Bu endpointler kullanılarak controller’daki fonksiyonların tetiklenmesi gerekiyor.

Bu işlem “routes” klasöründe bulunan “api.php” dosyasında yapılır. Burada kendi API rotalarımızı yazacağız.

→ api.php dosyası

Bu kod, bir Laravel uygulamasının rotalarını tanımlayan ve ‘ItemController’ adlı bir controller sınıfıyla ilişkilendirilen kodları içerir.

`Route::middleware('auth:sanctum')->get('/user', function (Request $request) { ... } );` Bu kod satırı, '/user' yolunda bir GET isteği alındığında çalışacak bir rota tanımlar. Ancak, önce 'auth:sanctum' orta yazılımını çalıştırarak bu isteğin yetkilendirilip yetkilendirilmediğini kontrol eder. Eğer kullanıcı yetkilendirilmişse, anonim fonksiyon içindeki kod çalışır ve '$request->user()' metodu ile oturum açmış olan kullanıcıyı temsil eden bilgileri döndürür. Bu durumda, istemciye kullanıcı bilgileri JSON formatında döndürülür.

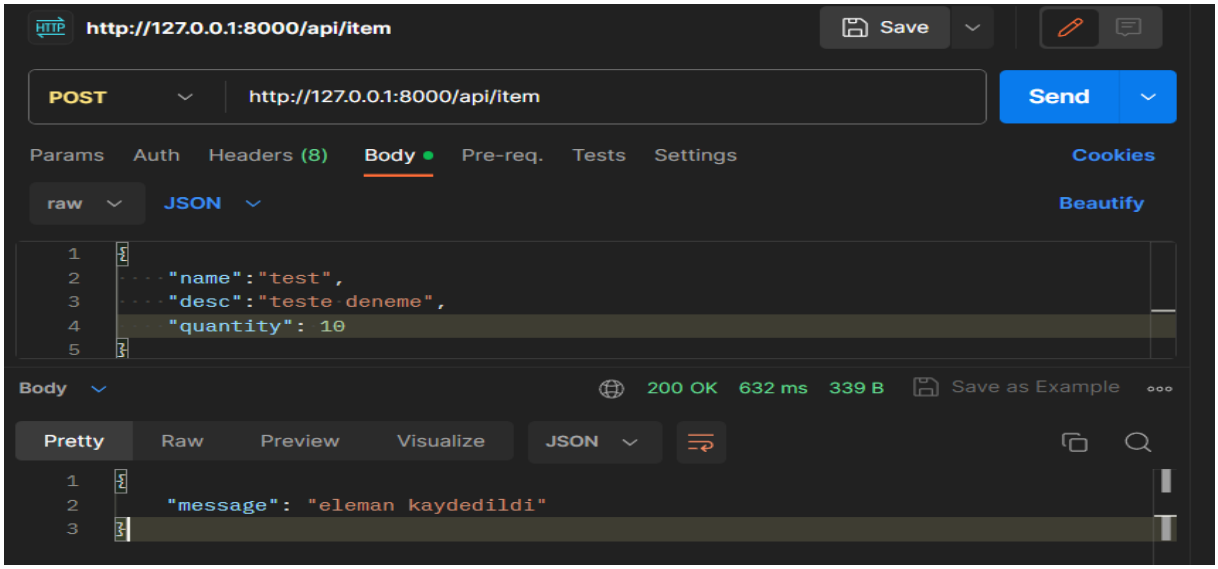
`Route::controller(ItemController::class)->group(function(){ ... });` Bu kod, 'ItemController' adlı controller sınıfı tanımlanan rotalara ilişkilendirilir. 'ItemController' sınıfındaki yöntemlerin isimleri, rotalarda belirtilen HTTP metodlarına (GET, POST, PUT, DELETE) göre otomatik olarak eşleştirilir.

Bu şekilde 'ItemController' sınıfındaki ilgili yöntemler belirtilen rotalara ilişkilendirilir ve ilgili http isteklerine göre doğru işlemlerin gerçekleştirilmesi sağlanır.

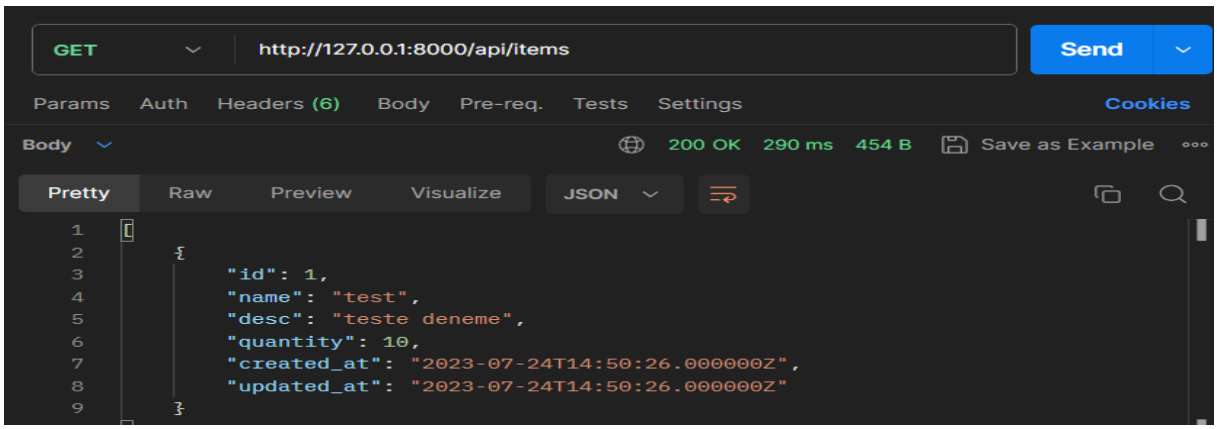
Localhost'a bu rotayı yazdığımda "127.0.0.1:8000/api/items" boş bir array döner çünkü veritabanında herhangi bir değer bulunmaktadır.

Bunun için Postman'den veri eklenebilir.

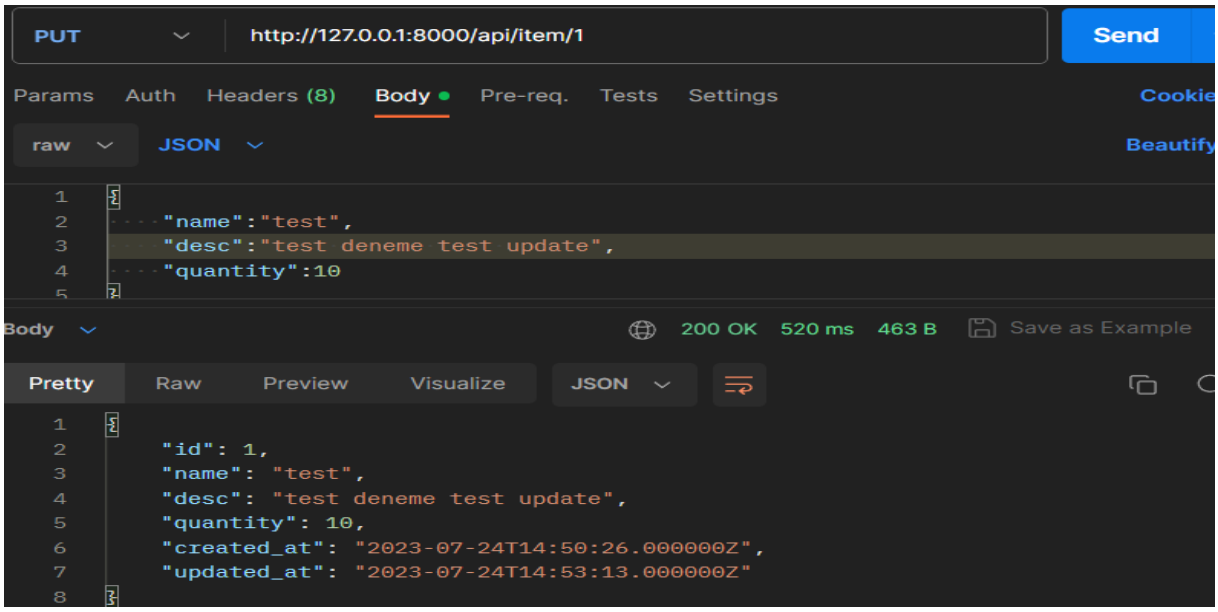
→post işlemi



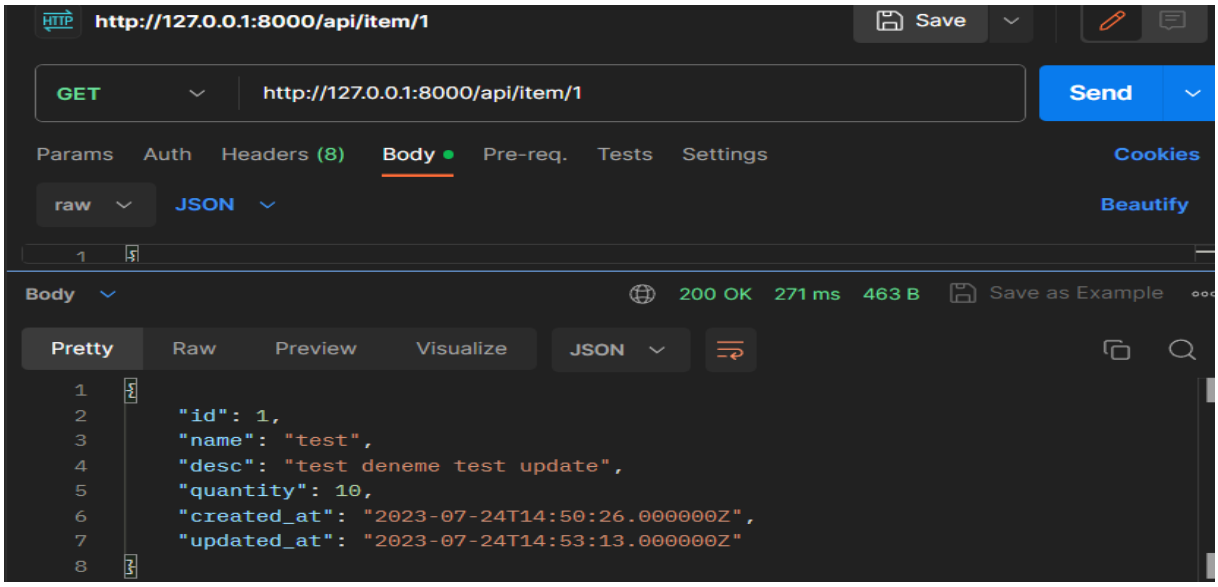
→get işlemi



→put işlemi



→id'ye göre getirme işlemi



→silme işlemi