ASP.Net Core İle Resim Boyutlandırma Uygulaması

2019-01-20T20:23:00+03:00

Contents

Giriş

Bu yazımda ImageSharp ile basit bir resim boyutlandırma uygulamasını nasıl yazıp geliştirdiğimden bahsedeceğim. Uygulamamızın ön yüzü olmayacak, API olarak çalışacak. Docker ile paketleyeğiz. Ve bence en önemlisi uygulamamızın özelliklerini HTTP istekleriyle test etmeye yarayan nodejs supertest ile test edeceğiz.

Tasarım Prensipleri

- Restful standartlara uygun bir API tasarlayacağız.
- Stateless tasarlamayı düşünüyordum, henüz tamamlayamadım. Şu anda dosyalar konteyner üzerine upload ediliyor ve bu **kesinlikle ve kesinlikle** canlı ortamda kullanılmaması gereken bir yöntem. Docker containerları üzerinde kalıcı veri tutmayınız. Host makine üzerinde bir klasöre, konteyner içindeki bir klasörü bağlamak çözümlerden biri. Dosyaları AWS S3'e yüklemek ise planladığım çözüm.
- Dosyaların sadece orijinal hallerini depolayacağız. Resimlerin boyutlandırılmış hallerini istek anında işleyip (`on-the-fly`) sunacağız, depolamayacağız.

- Olabildiğince basit ve kullanılabilir halde temel özelliklerden oluşan bir uygulama geliştireceğiz.
- Uygulamayı Docker ile derleyip ayağa kaldıracağız.
- Sadece temel fonksiyonları "yeteri" kadar test edeceğiz.

Uygulamanın Özellikleri

- Tekli ve çoklu resim yükleme
- Mevcut resmi değiştirme
- Resim silme
- Sabit genişlik ve sabit yükseklikte resimleri boyutlandırma.

Bu kadar.

Bu adrese dosyamızı gönderiyoruz.

`POST /api/upload`

Buradan alıyoruz. Yükseklik *0* olduğunda resmin orijinali döndürülür.

`GET /image/h{height}/{name}`

Nasıl Test Edeceğiz

Test kodları ise burada. Yazılmışı var.

Kısa bir örnek verelim:

Aşağıdaki kod, test için gerekli tanımlamaları yaptıktan sonra `'_data/ZY-IMG_0091-635px.jpg'` dosyasını yükler ve dönen sonucun 200 olmasını ve sonucun içinde *jpg* geçmesini bekler.

```
var should = require("chai").should(),
  expect = require("chai").expect,
  supertest = require("supertest"),
  API_URL = "http://app:5000",
  api = supertest(API_URL);

describe("JPEG File Upload", function() {
  it("single JPEG file upload", function(done) {
    api
      .post("/api/upload")
      .set("Content-Type", "multipart/form-data")
      .attach("image", "_data/ZY-IMG_0091-635px.jpg")
      .expect(200)
      .expect("Content-Type", /text\/plain/)
      .expect(/jpg/)
```

```
      .end(function(err, res) {
        if (err) {
          return done(err);
        }
        uploadedfile = res.text;

        done();
      });
  });
});
```

Bu testleri de Dockerize edelim:

```
FROM node:8
WORKDIR /usr/src/app
COPY package*.json ./

RUN npm install
COPY . .

CMD [ "node_modules/.bin/mocha", "--timeout", "25000", "--colors", "--reporter", "mocha-jenk
```

Peki Nasıl Çalıştıracağız?

Proje içinde bir Makefile var. İçinde ihtiyacınız olan komutlar hatta gereksiz derece fazlalık komutlar bile mevcut. `make up` demeniz yeterli. Veya docker-compose ile projeyi kolayca ayağa kaldırabilirsiniz.

Biri test, diğer uygulama olmak üzere iki konteyner ayağa kalkacak ve testler çalışacak. Şu an sekiz testten birinin başarısız olması gerekiyor.

```
docker-compose up --renew-anon-volumes --build
```

Kodlar

Kodlar Burada