Flask ile Resim Alma Servisi Oluştur

Gerekli tahmini süre: 90 dakika

Flask ile Resim Alma Servisi Oluştur uygulamalı laboratuvarına hoş geldiniz. Bu laboratuvar, nihayetinde IBM Code Engine'a dağıtacağınız servisi oluşturmanıza başlayacak. Laboratuvar, başlangıç yapmanız için bir GitHub şablon deposu sağlar. Depo ayrıca Python birim testleri içerir. Tüm testleri geçebilmesi için kodu tamamlamanız istenecektir.

Hedefler

Bu laboratuvar sırasında şunları yapacaksınız:

- Bir Flask sunucusu oluşturun
- Resim URL kaynağı üzerinde RESTful API'ler yazın
- API'lerin verilen **pytest** testlerini geçmesini kontrol edin

Not: Önemli Güvenlik Bilgisi

Not: Önemli Güvenlik Bilgisi

Cloud IDE'ye hoş geldiniz. Tüm geliştirmelerinizin burada gerçekleşeceği yerdir. Kullanmanız gereken tüm araçlar burada, **Python** ve **Flask** dahil.

Laboratuvar ortamının geçici olduğunu anlamak önemlidir. Kısa bir süre yaşar ve ardından yok edilir. Kendi GitHub deposunuza yaptığınız tüm değişiklikleri itmeniz zorunludur, böylece gerektiğinde yeni bir laboratuvar ortamında yeniden oluşturulabilir.

Ayrıca, bu ortamın paylaşıldığını ve dolayısıyla güvenli olmadığını unutmayın. Bu ortamda herhangi bir kişisel bilgi, kullanıcı adı, şifre veya erişim belirteci saklamamalısınız.

Göreviniz

Eğer bir GitHub Kişisel Erişim Belirteci oluşturmadıysanız, şimdi oluşturmalısınız. Bu, kodunuzu deponuza itmek için gerekecek. `repo` ve `write` izinlerine sahip olmalı ve `60` gün içinde süresi dolacak şekilde ayarlanmalıdır. Cloud IDE ortamında Git sizden bir şifre istediğinde, bunun yerine Kişisel Erişim Belirtecinizi kullanın. Ayrıntılı talimatlar için Git Token Oluşturma Laboratuvarındaki adımları izleyin.

Ekran Görüntüleri Hakkında Not

Bu laboratuvar boyunca, ekran görüntüleri almanız ve bunları cihazınıza kaydetmeniz istenecektir. Bu ekran görüntülerine ya notlandırılan quiz sorularını yanıtlamak için ya da bu kursun sonunda akran değerlendirmesi için gönderim olarak yüklemek için ihtiyacınız olacak. Ekran görüntünüzün .jpg veya .png uzantısına sahip olması gerekmektedir.

Ekran görüntüleri almak için çeşitli ücretsiz ekran yakalama araçlarını veya işletim sisteminizin kısayol tuşlarını kullanabilirsiniz. Örneğin:

- Mac: Tüm ekranınızı yakalamak için klavyenizde `Shift + Command + 3 (⇧ + ⌘ + 3)` tuşlarını kullanabilir veya bir pencere veya alanı yakalamak için `Shift + Command + 4 (⇧ + ⌘ + 4)` tuşlarına basabilirsiniz. Ekran görüntüleri, Masaüstünüzde .jpg veya .png dosyası olarak kaydedilecektir.
- Windows: Aktif pencerenizi yakalamak için klavyenizde `Alt + Print Screen` tuşlarına basabilirsiniz. Bu komut, aktif pencerenizin bir görüntüsünü panoya kopyalar. Ardından, bir resim düzenleyici açın, panonuzdaki görüntüyü resim düzenleyiciye yapıştırın ve resmi .jpg veya .png dosyası olarak kaydedin.

Geliştirme Ortamını Başlat

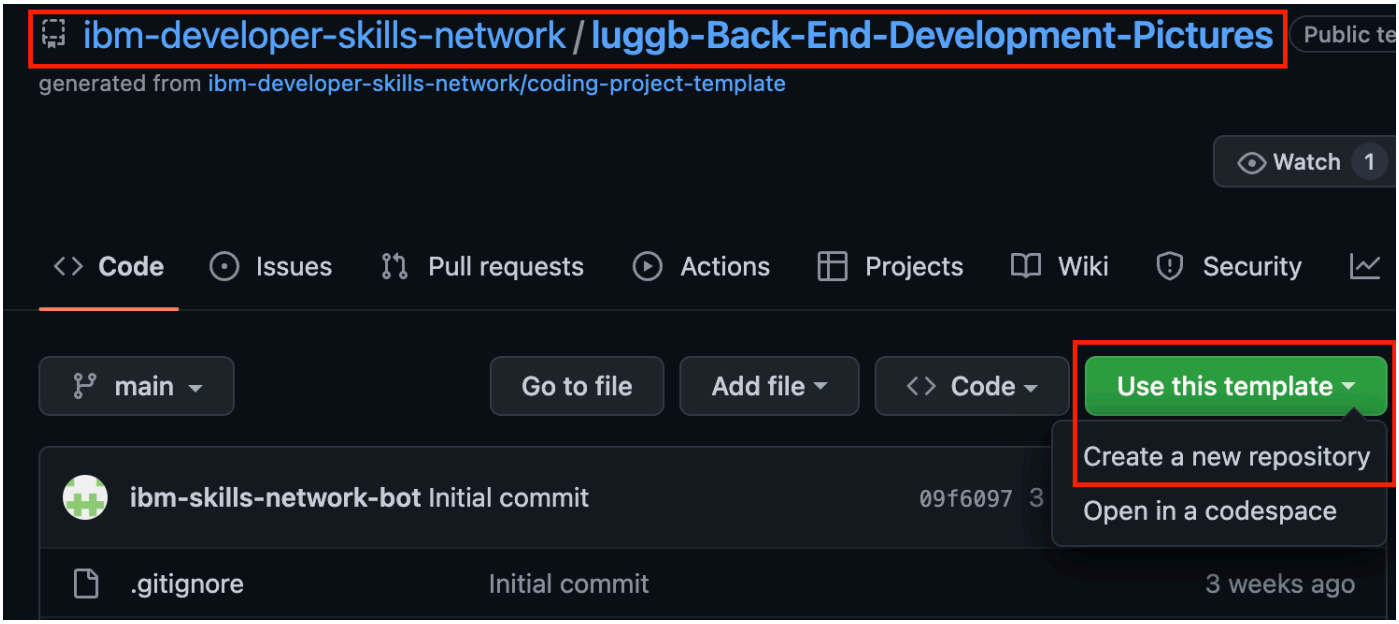
Cloud IDE ortamı geçici olduğu için herhangi bir zamanda silinebilir. Laboratuvara bir sonraki gelişinizde yeni bir ortam oluşturulabilir. Ne yazık ki, bu, her seferinde geliştirme ortamınızı yeniden başlatmanız gerektiği anlamına gelir. Bu sık sık gerçekleşmemelidir çünkü ortam birkaç gün boyunca sürebilir, ancak silindiğinde, yeniden oluşturma prosedürü aşağıdaki gibidir.

Genel Bakış

Şablondan yeni depo oluştur

1. Başlangıç kodu projesini açmak için bu URL'ye tıklayın: https://github.com/ibm-developer-skills-network/luggb-Back-End-Development-Pictures
2. Bu depoyu özel GitHub hesabınıza klonlamak için yeşil **Bu şablonu kullan** butonuna tıklayın.

   Fork kullanmayın; Şablon butonunu kullanın.



3. Depo isminizi `Back-End-Development-Pictures` olarak verin. Bu, değerlendiricilerin çalışmanızı değerlendirmek için arayacağı isimdir.
4. Depo için Genel seçeneğini seçtiğinizden emin olun ve ardından oluşturun.

Geliştirme Ortamını Başlat

Laboratuvar geliştirme ortamınızı kurmanız gerektiğinde her seferinde üç komut çalıştırmanız gerekecek.

Her bir komut, birer birer, aşağıdaki bölümde daha ayrıntılı olarak açıklanacaktır.

`{your_github_account}` sizin GitHub hesabı kullanıcı adınızı temsil eder.

Komutlar şunları içerir:

- GitHub deponuzu hesabınızdan klonlamak
- `Back-End-Development-Pictures` dizinine geçmek

- kurulum bash betiğini çalıştırmak
- terminalden çıkmak

Şimdi, bu komutların her birini tartışalım ve neler yapılması gerektiğini açıklayalım.

Görev Detayları

Aşağıdaki adımları kullanarak ortamınızı başlatın:

1. Eğer açık değilse, `Terminal -> Yeni Terminal` ile bir terminal açın.
2. Ardından, GitHub hesabınızın adını içeren bir ortam değişkenini dışa aktarmak için export GITHUB_ACCOUNT komutunu kullanın.

   Not: Aşağıdaki {your_github_account} yer tutucusunu gerçek GitHub hesabınız ile değiştirin:

   ```
   export GITHUB_ACCOUNT={your_github_account}
   ```

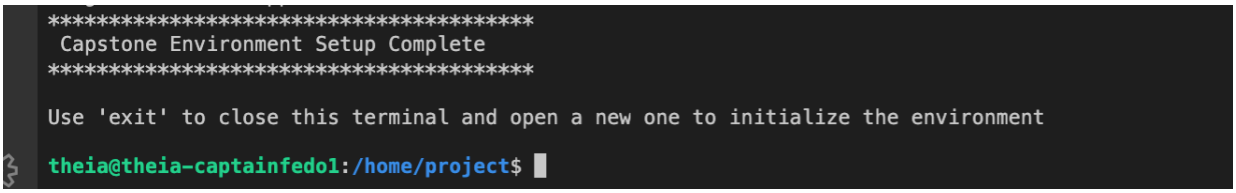
3. Ardından, depoinizi klonlamak için aşağıdaki komutları kullanın.

   ```
   git clone https://github.com/$GITHUB_ACCOUNT/Back-End-Development-Pictures.git
   ```

4. devops-capstone-project dizinine geçin ve `./bin/setup.sh` komutunu çalıştırın.

   ```
   cd /home/project/Back-End-Development-Pictures
   bash ./bin/setup.sh
   ```

5. Kurulum işleminin sonunda aşağıdakileri görmelisiniz:

6. Son olarak, mevcut terminali kapatmak için `exit` komutunu kullanın. Ortam, bir sonraki adımda yeni bir terminal açana kadar tam olarak aktif olmayacaktır.

   ```
   exit
   ```

Doğrulama

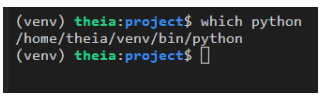
Ortamınızın doğru çalıştığını doğrulamak için yeni bir terminal açmalısınız çünkü Python sanal ortamı yalnızca yeni bir terminal oluşturulduğunda etkinleşecektir. Önceki görevinizde terminalden çıkmak için `exit` komutunu kullandığınızdan emin olun.

1. `Terminal -> Yeni Terminal` komutunu kullanarak bir terminal açın. Terminal istemcisinin önünde Python sanal ortamının `(backend-pics-venv)` eklendiğini görmelisiniz. Her şeyin doğru çalıştığını kontrol etmek için `which python` komutunu kullanın:

   Hangi Python'u kullandığınızı kontrol edin:

   ```
   which python
   ```

   Geri almanız gereken:

   Python sürümünü kontrol edin:

   ```
   python --version
   ```

Geri almanız gereken Python 3.9.18'in bazı yaman düzeyleri:

```
(venv) theia:project$ python --version
Python 3.9.18
(venv) theia:project$
```

Kanıt

1. Akış değerlendirmesi için göndermek üzere GitHub havuzunuzun URL'sini (şablon değil) not edin. Değerlendiricilerin hesabınızda `Back-End-Development-Pictures` adında bir havuz aradığını hatırlayın.

Geliştirme ortamının kurulumu tamamlandı. Ortamınız her yeniden oluşturulduğunda yukarıdaki prosedürü takip etmeniz gerekecek.

Artık çalışmaya başlamaya hazırsınız.

Proje Genel Görünümü

Müşteriniz, ünlü bir grup için bir web sitesi oluşturmanızı istedi. Projedeki arka uç geliştiricisi yakın zamanda ayrıldı ve web sitesinin yayına girebilmesi için kodu tamamlamanız gerekiyor. Uygulama, birlikte çalışan birkaç farklı mikro hizmetten oluşmaktadır.

Bu laboratuvar çalışmasında, **Resimleri Al** mikro hizmetini tamamlamanız isteniyor. Bu mikro hizmet, geçmiş etkinliklerden fotoğrafların URL'lerini saklar. Önceki geliştirici, Python Flask tabanlı bir REST API başlattı ve TDD veya test odaklı geliştirme sürecine uygun bazı testler yazdı. Kodunuzu GitHub'dan almanız ve kodun tüm testleri geçebilmesi için eksik parçaları tamamlamanız gerekecek.

REST API Kılavuzları Gözden Geçirme

Mimar, uç noktalar için aşağıdaki şemayı sağlamıştır:

RESTful API Uç Noktaları

Eylem	Yöntem	Dönüş kodu	Gövde	URL Uç Noktası
Listele	GET	200 OK	Resim URL'lerinin dizisi `[{...}]`	`GET /picture`
Oluştur	POST	201 OLUŞTURULDU	Bir resim kaynağı json olarak `{...}`	POST `/picture`
Oku	GET	200 OK	Bir resmi json olarak `{...}`	GET `/picture/{id}`
Güncelle	PUT	200 OK	Bir resmi json olarak `{...}`	PUT `/picture/{id}`
Sil	DELETE	204 İÇERİK YOK	""	DELETE `/picture/{id}`

Önceki geliştirici tarafından tamamlanan ve referans olarak kullanılabilecek uç noktalar:

Eylem	Yöntem	Dönüş kodu	Gövde	URL Uç Noktası
Sağlık	GET	200 OK	""	GET `/health`
Sayı	GET	200 OK	""	GET `/count`

Egzersiz 1: Sağlık ve sayım uç noktalarını test et

`Get Pictures` API'sini uygulamadan önce, önceki geliştiricinin uyguladığı iki uç noktanın testini yapalım.

- `/health`
- `/count`

Uç noktayı test etmenin bir yolu, sunucuyu başlatmak ve ardından uç noktalara bir istek göndermek için `curl` komutunu kullanmaktır. Terminali açın eğer zaten açık değilse ve `` dizinine geçin.

```
cd /home/project/Back-End-Development-Pictures
```

Sonra, flask sunucusunu geliştirme modunda çalıştırmak için aşağıdaki komutu çalıştırın:

```
flask --app app run --debugger --reload
```

Ana uygulamanız `app.py` adında bir dosyada bulunduğundan, bunu belirtmenize gerek yok. Aşağıdaki komut aynı sonucu verir:

```
flask run --debugger --reload
```

Aşağıdaki çıktıyı terminalde görebilmeniz gereken flask sunucusu çalışıyor olmalıdır:

```
$ flask --app app run --debugger --reload
 * Serving Flask app 'app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 132-341-814
```

Artık `health` ve `count` uç noktalarından çıktıyı görmek için aşağıdaki `curl` komutunu çalıştırabilirsiniz. **Sunucuyu çalıştıran terminalin yanına başka bir terminal oluşturmak için terminaldeki bölme butonunu kullanın.** Komutu çalıştırmadan önce doğru dizine geri dönmeniz gerekecek:

```
cd /home/project/Back-End-Development-Pictures
```

Yukarıdaki komutları çalıştırın:

```
curl --request GET --url http://localhost:5000/health
```

ve

```
curl --request GET --url http://localhost:5000/count
```

Aşağıdaki sonuçları görmelisiniz:
`/health`

```
$ curl --request GET --url http://localhost:5000/health
{"status":"OK"}
```

`/count`

```
$ curl --request GET --url http://localhost:5000/count
{"length":10}
```

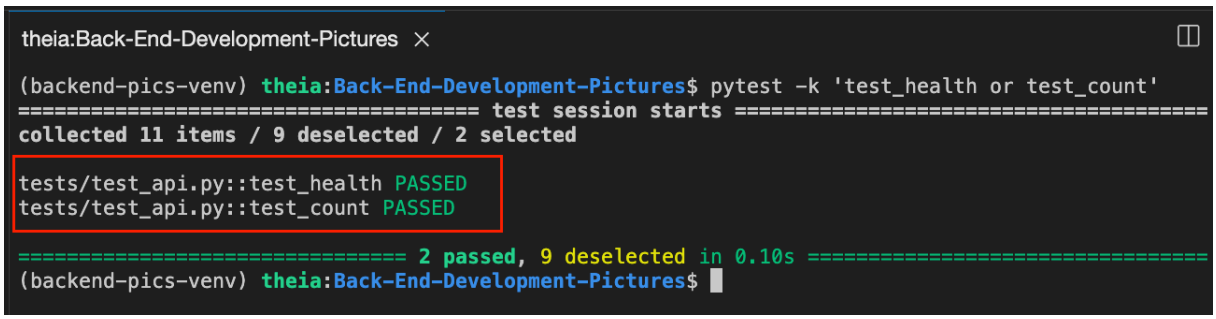
Geliştirme sırasında kodu test etmenin ikinci ve tercih edilen yolu TDD yöntemini izlemektir. Daha önce belirtildiği gibi, önceki geliştirici kod için testleri yazmıştır. `pytest` komutunu kullanarak kodun testleri geçip geçmediğini kontrol edebilirsiniz. Kodun `/count` ve `/health` uç noktaları için geçmesi gerekir.

Göreviniz

1. `health` ve `count` uç noktaları için iki testi çalıştırmak üzere pytest komutunu çalıştırın. Aşağıdaki komutu kullanabilirsiniz:

```
pytest -k 'test_health or test_count'
```

Aşağıdaki çıktıyı görmelisiniz:



Eğer `-k` bayrağı olmadan `pytest` komutunu çalıştırırsanız, tüm testler çalıştırılacak ve diğer testlerin başarısız olduğunu göreceksiniz. Çıktıyı yalnızca iki uç nokta testine daraltmak için `-k` bayrağını kullanırsınız.

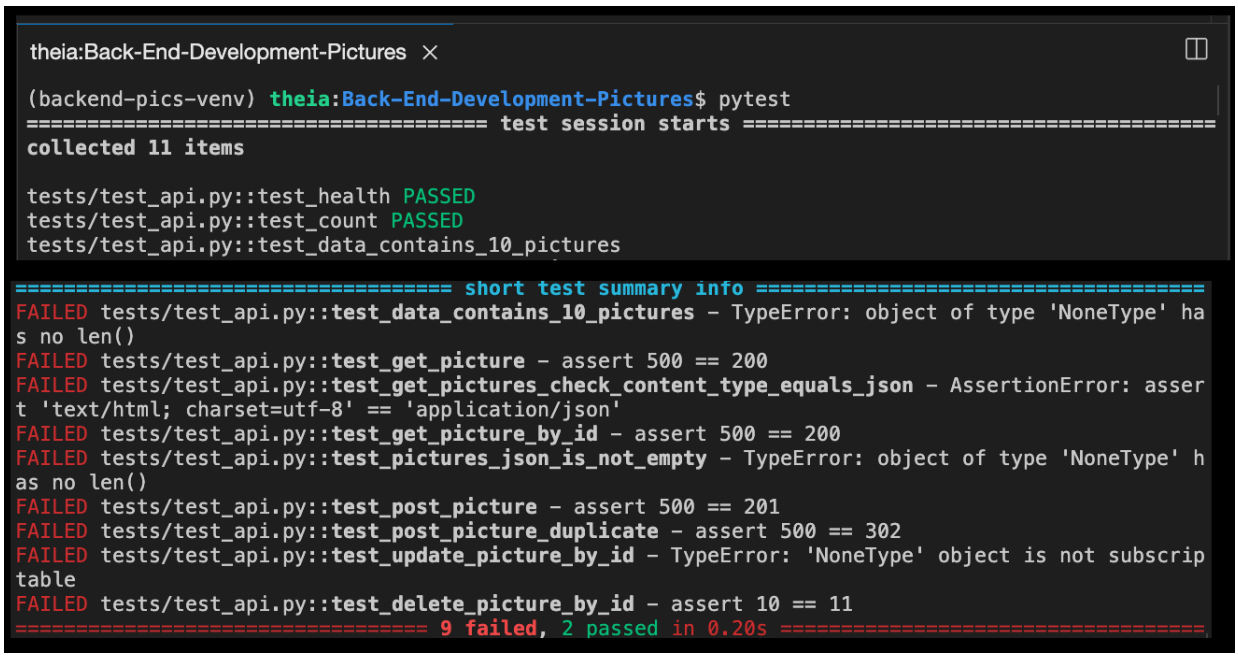
Kanıt

1. Yukarıda listelenen pytest komutunu çalıştırın ve terminalin ekran görüntüsünü alın. Kırmızı kutuları eklemenize gerek yok. Ekran görüntüsünü `exercise1-count-health-passing.jpg` (veya `.png`) olarak kaydedin. Ekran görüntüsü, iki testin geçtiğini göstermelidir.

Tebrikler! İlk hikayenizi tamamladınız.

Egzersiz 2: GET /picture uç noktasını uygulayın

Artık geri kalan uç noktaları uygulama zamanı. Eğer şimdi `pytest` komutunu çalıştırırsanız, 9 testin başarısız olduğunu göreceksiniz. Çıktınız ekran görüntüsünden biraz farklı görünebilir çünkü başarısız testlerin tüm günlüklerini kısaltmak için kaldırdık.



Laboratuvarın geri kalan kısmındaki göreviniz, başarısız testleri geçmek için kalan kodu tamamlamaktır. Öncelikle `GET /picture` uç noktasına başlayalım.

Göreviniz

Uç nokta için kodu yazmadan önce, kodunuzu GitHub'a geri gönderebilmeniz için bir dal oluşturalım.

`Back-End-Development-Pictures` dizinine geçin ve aşağıdaki adımları uygulayın:

```
cd /home/project/Back-End-Development-Pictures
git checkout main
git pull
git checkout -b backend-rest
```

Bu, ana dalı değiştirecek, en son değişiklikleri çekecek ve yeni bir dal oluşturacaktır. Tüm değişikliklerinizi GitHub reposuna itmeniz ve tüm kodu ana dalınıza bir pull request ile birleştirmeniz istenecektir.

Görev 2 : Uç noktanın kodunu tamamlayın

Tüm kodu `Back-End-Development-Pictures/backend/routes.py` dosyasında yazacaksınız.

Open **routes.py** in IDE

Not: Dosya Gezgini'nde açmak için bu konuma gidin:
`Back-End-Development-Pictures/backend/routes.py`

1. `/picture` uç noktası için GET metoduna yanıt veren bir Flask rotası oluşturun.
2. Uygulamayı barındıracak `get_pictures()` adında bir fonksiyon oluşturun.
3. URL'ler `data` adında bir listeye yüklenir. Bu metot içinde bunu döndürmeniz gerekiyor.
4. Aşağıdaki fonksiyonlar geçene kadar pytest'i çalıştırın:

```
tests/test_api.py::test_health PASSED
tests/test_api.py::test_count PASSED
tests/test_api.py::test_data_contains_10_pictures PASSED
tests/test_api.py::test_get_picture PASSED
tests/test_api.py::test_get_pictures_check_content_type_equals_json PASSED
```

Kanıt

1. Fonksiyonlar geçtikten sonra, geçen fonksiyonların bir ekran görüntüsünü alın ve adını `exercise2-get-pictures-passing.jpg` (veya `.png`) olarak koyun.

Tebrikler! Arka uçunuza ilk REST uç noktasını eklediniz.

Alıştırma 3: GET /picture/id uç noktasını uygulayın

Daha önce olduğu gibi, uç nokta için kodu `./backend/routes.py` dosyasında yazacaksınız.

Open **routes.py** in IDE

Not: Dosya Gezgini'nde açmak için bu konuma gidin:
`Back-End-Development-Pictures/backend/routes.py`

1. `/picture/<id>` uç noktası için GET yöntemine yanıt veren bir Flask rotası oluşturun.
2. Uygulamayı içerecek `get_picture_by_id(id)` adında bir fonksiyon oluşturun.
3. URL'ler `data` adında bir listeye yüklenir. Verilen id ile URL'yi bulmak ve geri döndürmek için listeyi ayrıştırmanız gerekecek.
4. Aşağıdaki fonksiyonlar geçene kadar pytest'i çalıştırın:

```
tests/test_api.py::test_health PASSED
tests/test_api.py::test_count PASSED
tests/test_api.py::test_data_contains_10_pictures PASSED
tests/test_api.py::test_get_picture PASSED
tests/test_api.py::test_get_pictures_check_content_type_equals_json PASSED
tests/test_api.py::test_get_picture_by_id PASSED
```

```
tests/test_api.py::test_pictures_json_is_not_empty PASSED
```

Tebrikler! Backend'inize ikinci REST uç noktasını eklediniz.

Egzersiz 4: POST /picture uç noktasını uygulama

Daha önce olduğu gibi, uç noktanın kodunu `./backend/routes.py` dosyasında yazacaksınız.

Open **routes.py** in IDE

Not: Dosya Gezgini'nde açmak için şu konuma gidin:
`Back-End-Development-Pictures/backend/routes.py`

1. `/picture/<id>` uç noktası için POST yöntemine yanıt veren bir Flask rotası oluşturun. Uygulama dekoratörünüzde `methods=["POST"]` kullanın.
2. Uygulamanın içeriğini tutmak için `create_picture()` adında bir fonksiyon oluşturun.
3. Öncelikle istek gövdesinden resim verilerini çıkarmanız ve ardından `data` listesine eklemeniz gerekecek.
4. Eğer belirtilen id ile bir resim zaten mevcutsa, kullanıcıya `{"Message": "id {picture['id']} olan resim zaten mevcut"}` mesajı ile birlikte `302` HTTP kodu gönderin.
5. Aşağıdaki fonksiyonlar geçene kadar pytest'i çalıştırın:

```
tests/test_api.py::test_health PASSED
tests/test_api.py::test_count PASSED
tests/test_api.py::test_data_contains_10_pictures PASSED
tests/test_api.py::test_get_picture PASSED
tests/test_api.py::test_get_pictures_check_content_type_equals_json PASSED
tests/test_api.py::test_get_picture_by_id PASSED
tests/test_api.py::test_pictures_json_is_not_empty PASSED
tests/test_api.py::test_post_picture {'id': 200, 'pic_url': 'http://dummyimage.com/230x100.png/dddddd/000000', 'event_country': 'United States', 'event_state': 'California', 'event_city': 'Fremont', 'eve
PASSED
tests/test_api.py::test_post_picture_duplicate {'id': 200, 'pic_url': 'http://dummyimage.com/230x100.png/dddddd/000000', 'event_country': 'United States', 'event_state': 'California', 'event_city': 'Frem
PASSED
```

Kanıt

1. Fonksiyonlar geçtikten sonra, geçen fonksiyonların ekran görüntüsünü alın ve adını `exercise4-post-picture-passing.jpg` (veya `.png`) olarak belirleyin.

Egzersiz 5: PUT /picture uç noktasını uygulayın

PUT uç noktası, mevcut bir resim kaynağını güncellemek için kullanılacaktır. Daha önce olduğu gibi, uç nokta için kodu `./backend/routes.py` dosyasında yazacaksınız.

Open **routes.py** in IDE

Not: Dosya Gezgini'nde açmak için bu konuma gidin:
`Back-End-Development-Pictures/backend/routes.py`

1. `/picture/<int:id>` uç noktası için POST yöntemine yanıt veren bir Flask rotası oluşturun. Uygulama dekoratörünüzde `methods=["PUT"]` kullanın.
2. Uygulamanın içeriğini tutmak için `update_picture(id)` adında bir fonksiyon oluşturun.
3. Öncelikle, resim verilerini istek gövdesinden çıkarmanız gerekecek.
4. Ardından, resmi `data` listesinden bulacaksınız. Resim mevcutsa, gelen istekle güncelleyeceksiniz.
5. Resim mevcut değilse, `{"message": "resim bulunamadı"}` mesajıyla birlikte `404` durumu döndüreceksiniz.
6. Aşağıdaki fonksiyonlar geçene kadar pytest'i çalıştırın:

```
tests/test_api.py::test_health PASSED
tests/test_api.py::test_count PASSED
tests/test_api.py::test_data_contains_10_pictures PASSED
tests/test_api.py::test_get_picture PASSED
tests/test_api.py::test_get_pictures_check_content_type_equals_json PASSED
tests/test_api.py::test_get_picture_by_id PASSED
tests/test_api.py::test_pictures_json_is_not_empty PASSED
tests/test_api.py::test_post_picture {'id': 200, 'pic_url': 'http://dummyimage.com/230x100.png/dddddd/000000', 'event_country': 'United States', 'event_state': 'California', 'event_city': 'Fremont', 'eve
PASSED
tests/test_api.py::test_post_picture_duplicate {'id': 200, 'pic_url': 'http://dummyimage.com/230x100.png/dddddd/000000', 'event_country': 'United States', 'event_state': 'California', 'event_city': 'Frem
PASSED
tests/test_api.py::test_update_picture_by_id PASSED
```

Egzersiz 6: DELETE /picture uç noktasını uygulama

Görev 1 : Silme uç noktasını uygulama

DELETE uç noktası, mevcut bir resim kaynağını silmek için kullanılır. Daha önce olduğu gibi, uç nokta için kodu `./backend/routes.py` dosyasında yazacaksınız.

Open **routes.py** in IDE

Not: Dosya Gezgini'nde açmak için bu konuma gidin:
`Back-End-Development-Pictures/backend/routes.py`

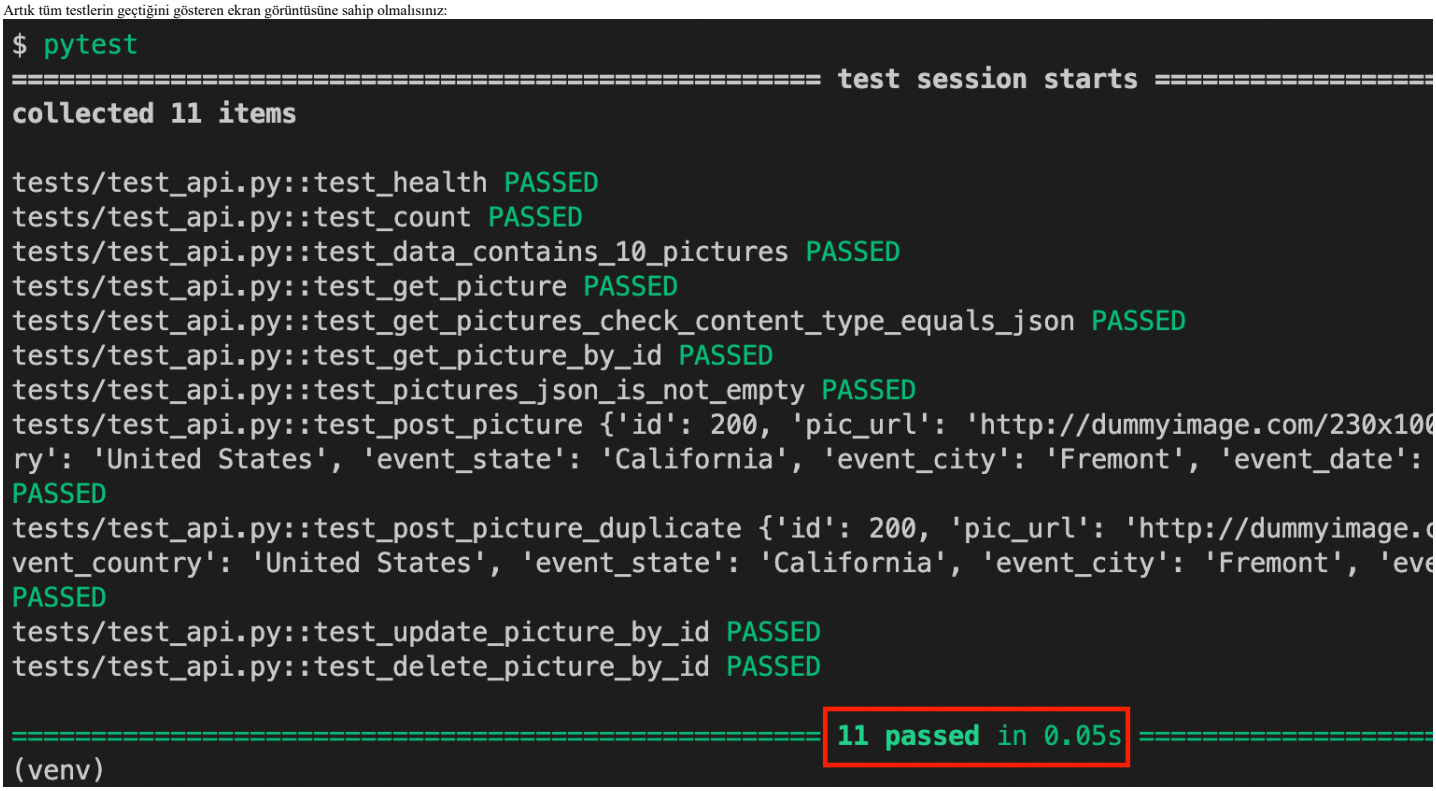
1. `/picture/<int:id>` uç noktası için POST yöntemine yanıt veren bir Flask rotası oluşturun. Uygulama dekoratörünüzde `methods=["DELETE"]` kullanın.
2. Uygulamanın içeriğini tutacak `delete_picture(id)` adında bir fonksiyon oluşturun.
3. Öncelikle URL'den id'yi çıkaracaksınız.
4. Ardından, `data` listesini id ile resmi bulmak için gezin. Resim mevcutsa, listedeki öğeyi silip HTTP_204_NO_CONTENT durumu ile boş bir yanıt döneceksiniz.
5. Resim mevcut değilse, `{"message": "resim bulunamadı"}` mesajı ile `404` durumu döndüreceksiniz.
6. Aşağıdaki fonksiyonlar geçene kadar pytest'i çalıştırın:

```
tests/test_api.py::test_health PASSED
tests/test_api.py::test_count PASSED
tests/test_api.py::test_data_contains_10_pictures PASSED
tests/test_api.py::test_get_picture PASSED
tests/test_api.py::test_get_pictures_check_content_type_equals_json PASSED
tests/test_api.py::test_get_picture_by_id PASSED
tests/test_api.py::test_pictures_json_is_not_empty PASSED
tests/test_api.py::test_post_picture {'id': 200, 'pic_url': 'http://dummyimage.com/230x100.png/dddddd/000000', 'event_country': 'United States', 'event_state': 'California', 'event_city': 'Fremont', 'eve
PASSED
tests/test_api.py::test_post_picture_duplicate {'id': 200, 'pic_url': 'http://dummyimage.com/230x100.png/dddddd/000000', 'event_country': 'United States', 'event_state': 'California', 'event_city': 'Frem
PASSED
tests/test_api.py::test_update_picture_by_id PASSED
tests/test_api.py::test_delete_picture_by_id PASSED
```

Kanıt

1. Fonksiyonlar geçtikten sonra, geçen fonksiyonların ekran görüntüsünü alın ve adını `exercise6-delete-picture-passing.jpg` (veya `.png`) olarak kaydedin.

Artık tüm testlerin geçtiğini gösteren ekran görüntüsüne sahip olmalısınız:



Görev 2 : Dalı GitHub'a gönderin ve bir PR oluşturun

Artık mikro hizmet için kodu tamamladığınıza göre, `backend-rest` dalını GitHub fork'unuza geri gönderebilirsiniz. Bu projede tek başınıza çalıştığınız için PR'yi birleştirip dalı silin. Bir sonraki laboratuvara geçmeden önce tüm kod değişikliklerinizin ana dalınıza gönderildiğinden emin olun.

1. İlk kez gönderim yaptığınızda git kullanıcı adınızı ve e-posta adresinizi ayarlamanız istenecektir:

    ```
    git config --local user.name "{GitHub adınızı buraya yazın}"
    git config --local user.email {GitHub e-posta adresinizi buraya yazın}
    ```

2. `git commit -am` komutunu kullanarak "resim hizmeti uygulandı" mesajı ile değişikliklerinizi taahhüt edin ve bu değişiklikleri depounuza göndermek için `git push` komutunu kullanın.

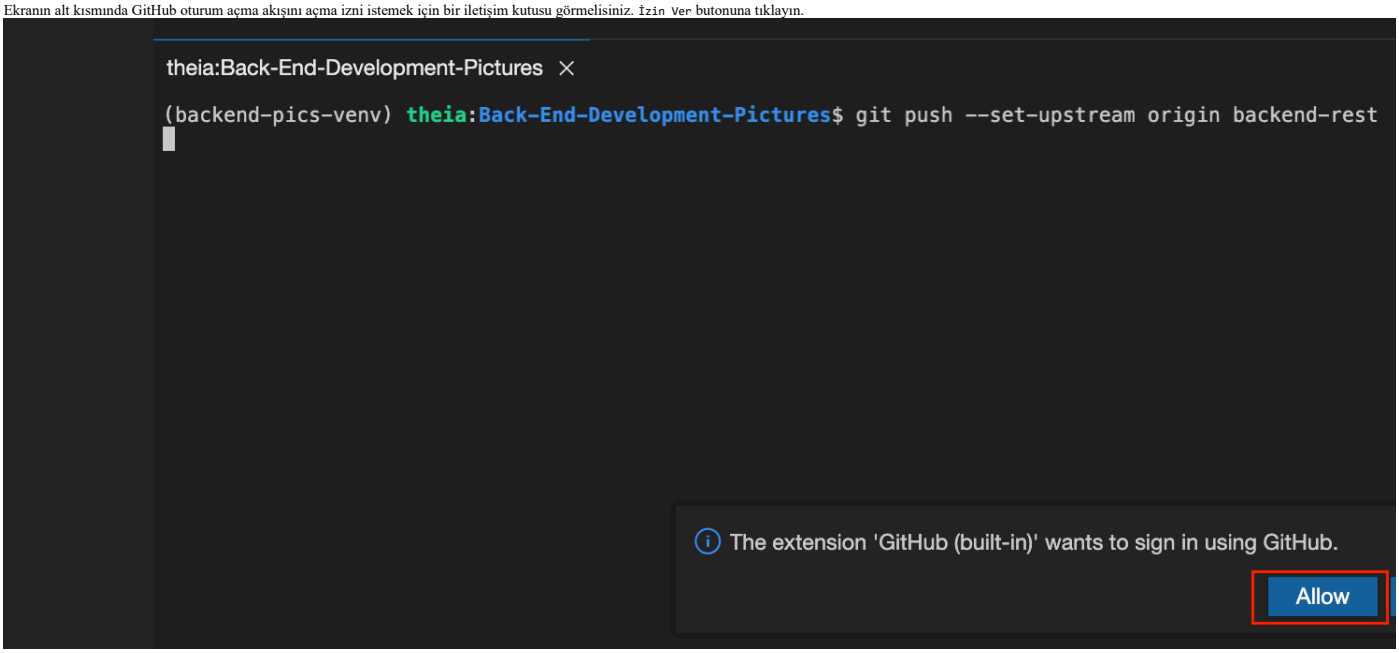
    ▼ İpucu için buraya tıklayın.

    ```
    git commit -am "{buraya mesaj yazın}"
    git push --set-upstream origin {buraya dal adı yazın}
    ```

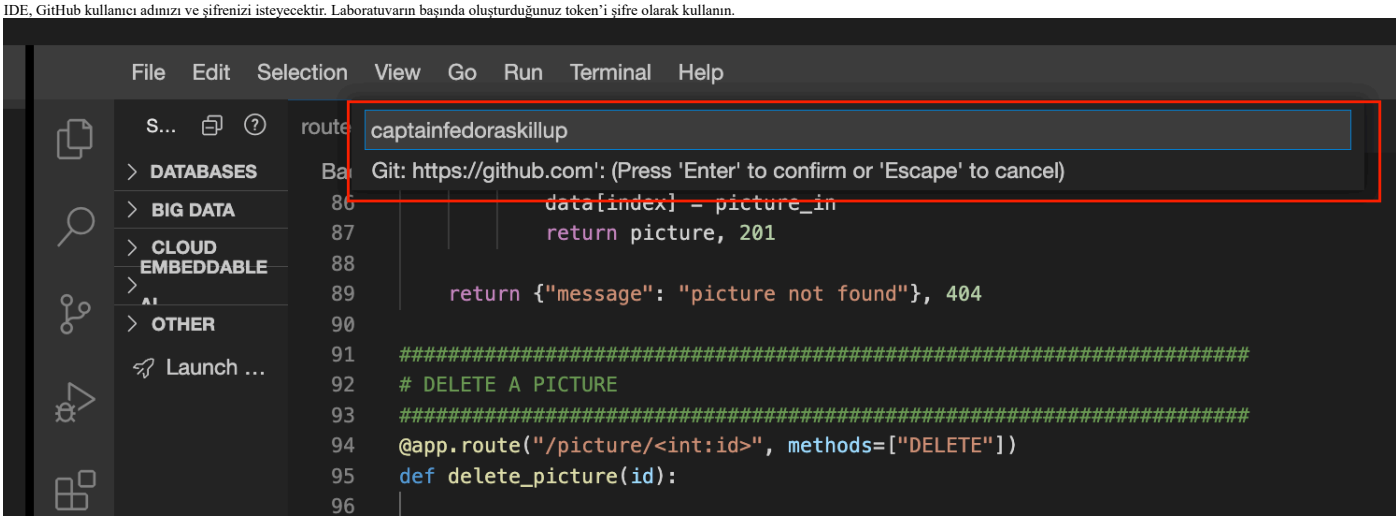
    ▼ İpucu için buraya tıklayın.

    ```
    git commit -am "resim hizmeti uygulandı"
    git push --set-upstream origin backend-rest
    ```

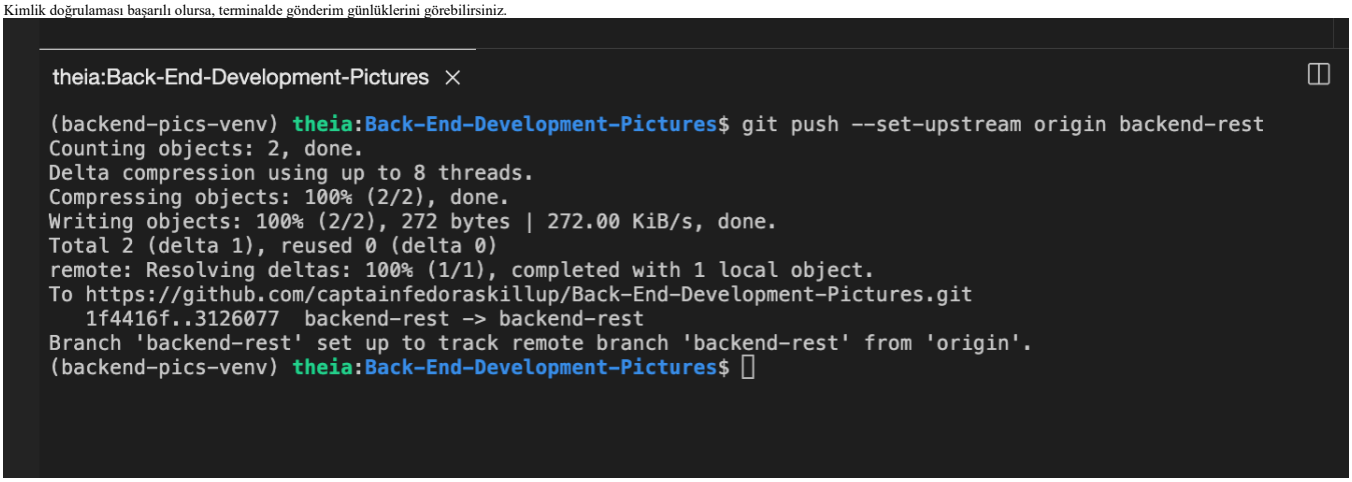
3. Ekranın alt kısmında GitHub oturum açma akışını açma izni istemek için bir iletişim kutusu görmelisiniz. `İzin Ver` butonuna tıklayın.



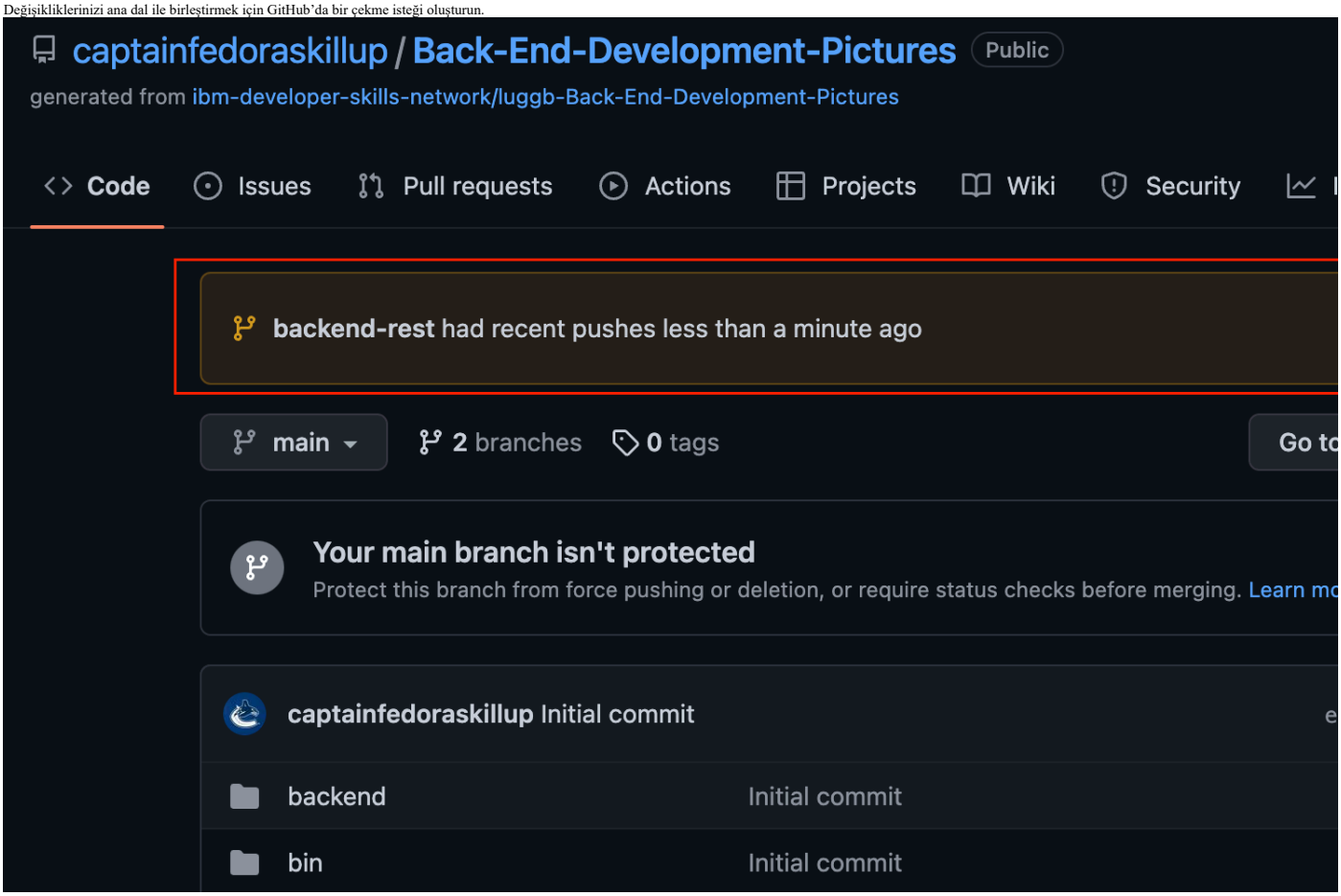
4. IDE, GitHub kullanıcı adınızı ve şifrenizi isteyecektir. Laboratuvarın başında oluşturduğunuz token'i şifre olarak kullanın.



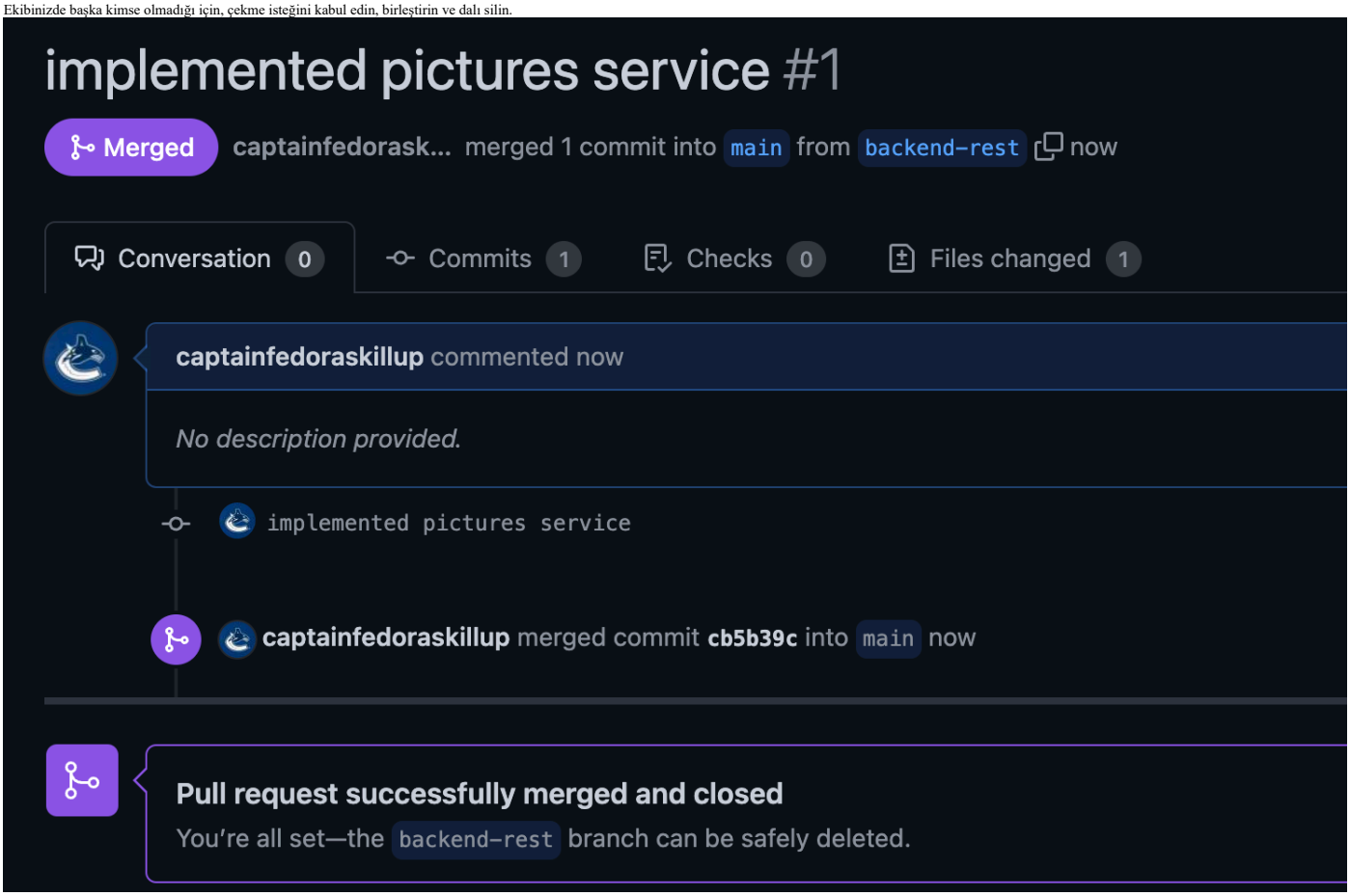
5. Kimlik doğrulaması başarılı olursa, terminalde gönderim günlüklerini görebilirsiniz.



6. Değişikliklerinizi ana dal ile birleştirmek için GitHub'da bir çekme isteği oluşturun.



7. Ekibinizde başka kimse olmadığı için, çekme isteğini kabul edin, birleştirin ve dalı silin.



Bu noktada, ana dal tamamlanmış kodunuzu içermelidir.

Referans: RESTful Servis

Her bir uç noktanın RESTful davranışı hakkında bazı ipuçları burada bulunmaktadır.

Liste

- Liste, yalnızca resimlerin sözlük listesini geri göndermeli ve HTTP_200_OK dönüş kodunu döndürmelidir. Sadece `data` yapısını geri döndürün.
- Asla 404_NOT_FOUND göndermemelidir.

Oku

- Oku, bir resim kimliği almalı ve `data` içinde kimliği bulmak için gezmelidir.
- Resim bulunamazsa, HTTP_404_NOT_FOUND döndürmeli ve `{"message": "resim bulunamadı"}` mesajını iletmelidir.
- Resim bulunursa, resmi bir Python sözlüğü olarak HTTP_200_OK dönüş kodu ile geri döndürmelidir.

Oluştur

- Oluştur, yalnızca POST yöntemi ile gelen istekleri kabul etmelidir.
- Gelen istekte resmi arayacaktır.
- Resim `data` listesinde zaten varsa, HTTP_302_FOUND döndürmelidir.
- Aksi takdirde, gelen resmi `data` listesine eklemeli ve `{"Message": f"resim id {picture_in['id']} zaten mevcut"}` mesajı ile birlikte HTTP_201_CREATED döndürmelidir.

Güncelle

- Güncelle, bir account_id ve PUT HTTP yöntemini kabul etmelidir.
- Resim bulunamazsa, HTTP_404_NOT_FOUND döndürmelidir.
- Resim bulunursa, resmi istekteki içerikle değiştirmeli ve HTTP_201_CREATED kodu ile güncellenmiş resmi döndürmelidir.
- Resim bulunamazsa, HTTP_404_NOT_FOUND kodu ve `{"message": "resim bulunamadı"}` mesajını döndürmelidir.

Sil

- Sil, bir resim kimliği almalı ve resmi `data` listesinde aramalıdır.
- Resim bulunamazsa, HTTP_404_NOT_FOUND kodu ve `{"message": "resim bulunamadı"}` mesajını döndürmelidir.
- Resim bulunursa, resmi `data` listesinden silmelidir.
- Boş bir dize “” ile birlikte HTTP_204_NO_CONTENT dönüş kodunu döndürmelidir.

Yukarıda gösterildiği gibi test durumlarını geçmek için kodu yazın.

İpuçları ve Çözümler

Bu sayfa, Listeleme, Oluşturma, Güncelleme ve Silme REST API’leri için kalan ipuçlarını ve çözümleri içermektedir.

İpuçları

Listele

▼ İpucu için buraya tıklayın.

```
@app.route("{insert URL here}", methods="{insert HTTP method name here}")
def {insert method name here}():
    return jsonify({insert data list here})
```

Oku

▼ İpucu için buraya tıklayın.

```
@app.route("{insert URL here}", methods=["GET"])
def {insert method name here}(id):
    {enumerate the data list}:
        if picture["id"] == id:
            return picture
    return {"message": "{insert error message here}"}, {insert HTTP_NOT_FOUND_STATUS}
```

Oluştur

▼ İpucu için buraya tıklayın.

```
@app.route("{insert URL here}", methods="insert list of correct method here")
def {insert method name here}():
    # get data from the json body
    picture_in = {insert code to get json from the request here}
    # if the id is already there, return 303 with the URL for the resource
    {enumerate the picture in data list}:
        if picture_in["id"] == picture["id"]:
            return {
                "Message": f"{insert message here}"
            }, {insert HTTP code here}
    data.append(picture_in)
    return picture_in, {insert HTTP content created code here}
```

Güncelleme

▼ İpucu için buraya tıklayın.

```
@app.route("{insert URL here}", methods={insert List of HTTP method here})
def {insert method name here}(id):
    # get data from the json body
    picture_in = {insert code to get json from request here}
    {insert code to enumerate picture in data list with index}:
        if picture["id"] == id:
            data[index] = picture_in
            return picture, {insert HTTP code here}
    return {"message": "insert error message here"}, {insert HTTP NOT FOUND code here}
```

Sil

▼ İpucu için buraya tıklayın.

```
@app.route("{insert URL here}", methods={insert List of HTTP method here})
def {insert method name here}(id):
    {insert code to enumerate pictures in data}:
        if picture["id"] == id:
            {insert code to delete picture from data}
            return "", {insert code to return HTTP code}
    return {"message": "{insert error message here}"}, {insert code to return HTTP code}
```

Çözümler

Liste

▼ Çözümünüzü kontrol etmek için buraya tıklayın.

```
@app.route("/picture", methods=["GET"])
def get_pictures():
    return jsonify(data)
```

Oku

▼ Çözümünüzü kontrol etmek için buraya tıklayın.

```
@app.route("/picture/<int:id>", methods=["GET"])
def get_picture_by_id(id):
    for picture in data:
        if picture["id"] == id:
            return picture
    return {"message": "picture not found"}, 404
```

Oluştur

▼ Çözümünüzü kontrol etmek için buraya tıklayın.

```
@app.route("/picture", methods=["POST"])
def create_picture():
    # get data from the json body
    picture_in = request.json
    print(picture_in)
    # if the id is already there, return 303 with the URL for the resource
    for picture in data:
        if picture_in["id"] == picture["id"]:
            return {
                "Message": f"picture with id {picture_in['id']} already present"
            }, 302
    data.append(picture_in)
    return picture_in, 201
```

Güncelleme

▼ Çözümünüzü kontrol etmek için buraya tıklayın.

```
@app.route("/picture/<int:id>", methods=["PUT"])
def update_picture(id):
    # get data from the json body
    picture_in = request.json
    for index, picture in enumerate(data):
        if picture["id"] == id:
            data[index] = picture_in
            return picture, 201
    return {"message": "picture not found"}, 404
```

Sil

▼ Çözümünüzü kontrol etmek için buraya tıklayın.

```
@app.route("/picture/<int:id>", methods=["DELETE"])
def delete_picture(id):
    for picture in data:
        if picture["id"] == id:
            data.remove(picture)
            return "", 204
    return {"message": "picture not found"}, 404
```

▼ Details

::page{title="Sonuç"}

Tebrikler! Resim almak için ilk mikroservisi uygulamayı tamamladınız. Bu mikroservis, projenin son laboratuvarında ana site tarafından kullanılacaktır.

Sonraki Adımlar

Bu noktada kursa devam edebilirsiniz. Bir sonraki modülde başka bir mikroservis oluşturmanız istenecek.

Author(s)

CF

Changelog

Date	Version	Changed by	Change Description
2023-02-04	0.1	CF	İlk versiyon oluşturuldu
2023-02-09	0.2	SH	Düzenlemelerle QA geçişi
2024-01-30	0.3	Manvi Gupta	Python versiyonu ve routes.py güncellendi