

Swagger Nedir Nasıl Kullanılır?

Web API geliştirirken, en önemli şeylerden biri dokümantasyondur. API methodlarının ne işe yaradığını ve nasıl kullanılacağını anlatan bir dokümantasyon, geliştiriciler için hayati önem taşır. Ancak, bu dokümantasyonu el ile yazmak hem zor hem de güncel tutması zordur. İşte tam da burada Swagger devreye giriyor.

Swagger, API geliştiricilerinin web servislerini belgelemek, test etmek ve kullanmak için kullanabilecekleri açık kaynaklı bir çerçevedir. OpenAPI Specification (OAS) adıyla da bilinen Swagger, RESTful web servislerinizi tanımlamanıza, belgelendirmenize ve paylaşmanıza yardımcı olan bir dizi araç ve standarttır.

Swagger'ın önemli özelliklerinden bazıları şunlardır:

- 1. API Tanımı ve Belgeleme:** Swagger, API'lerinizin işlevlerini, parametrelerini, yanıtlarını ve hata durumlarını açık bir şekilde tanımlamanıza olanak tanır. Bu tanım, API kullanıcılarına API'nin nasıl kullanılacağı hakkında rehberlik eder.
- 2. Otomatik Dokümantasyon Üretimi:** Swagger, API tanımınızı kullanarak otomatik olarak API belgelerini oluşturur. Bu belgeler genellikle interaktif bir API konsolu, örnek istek ve yanıtlar, parametre açıklamaları gibi detayları içerir.
- 3. İstemci Kodu Üretimi:** Swagger, tanımladığınız API için istemci kodunu otomatik olarak oluşturabilir. Bu, API'yi kullanmak için gerekli olan istemci kodunu yazma zahmetinden kurtarır.
- 4. API Sürüm Yönetimi:** Swagger, farklı API sürümlerini ve değişiklikleri yönetmeye yardımcı olabilir. Bu, API'nizin gelişen gereksinimlerine uyum sağlamak için önemlidir.
- 5. API Testi:** Swagger, API'nin doğru çalıştığını doğrulamak için otomatik testler oluşturmanıza olanak tanır.
- 6. Topluluk ve Ekosistem:** Swagger, büyük bir geliştirici topluluğu tarafından desteklenmektedir ve çeşitli entegrasyonlar ve eklentilerle geniş bir ekosisteme sahiptir.

Swagger, API geliştiricilerine, API'lerini hızlı bir şekilde tanımlama, belgeleme ve paylaşma konusunda önemli bir yardımcıdır. Bu, hem geliştiricilerin daha verimli çalışmasına hem de API kullanıcılarının API'yi daha kolay anlamasına ve kullanmasına olanak tanır.

Swagger'ın Projeye Eklenmesi

Swagger'ın Gradle da projeye dahil edilmesi için build.gradle dosyası içerisine dependency olarak resimde ki gibi eklenmesi gerekir.

```
compile 'io.springfox:springfox-swagger2:2.9.2'  
compile 'io.springfox:springfox-swagger-ui:2.9.2'
```

Swagger'ın Maven da projeye dahil edilmesi için ise pom.xml dosyası içerisine yine dependency olarak aşağıdaki resimde görüldüğü gibi eklenmesi gerekir.

```
<dependency>  
  <groupId>io.springfox</groupId>  
  <artifactId>springfox-swagger2</artifactId>  
  <version>2.9.2</version>  
</dependency>  
  
<dependency>  
  <groupId>io.springfox</groupId>  
  <artifactId>springfox-swagger-ui</artifactId>  
  <version>2.9.2</version>  
</dependency>
```

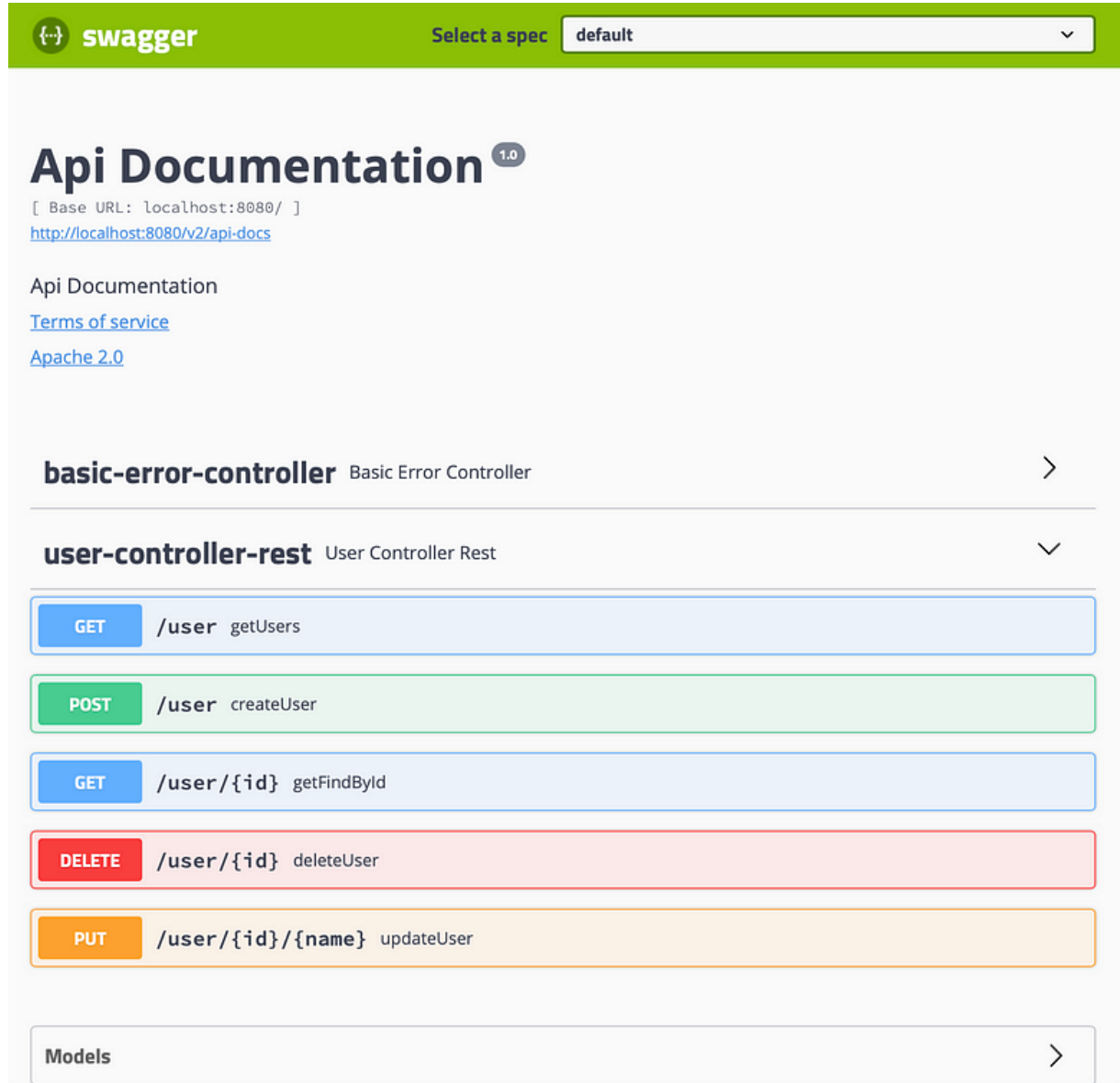
Swagger'ın Konfigürasyon İşleminin Yapılması

Swagger'ı projemize dahil ettikten sonra bazı konfigürasyon işlemlerini gerçekleştirmemiz gerekir.

```
@Configuration  
@EnableSwagger2  
public class SwaggerConfiguration {  
  
    @Bean  
    public Docket api(){  
        return new Docket(DocumentationType.SWAGGER_2)  
            .select()  
            .apis(RequestHandlerSelectors.any())  
            .paths(PathSelectors.any())  
            .build()  
            .pathMapping("/");  
    }  
}
```

@Configuration olduğunu belirttikten sonra **@EnableSwagger2** ile Swagger 2.0 spesifikasyonunu aktif hale getiriyoruz. `DocumentationType.SWAGGER_2` ile Swagger 2 kullanacağımızı belirtiyoruz, burayı 1.2 ya da WEB ile değiştirebilirsiniz. **select()** ile ApiSelectorBuilder instance'si döndürülüyor ve ayarlarımız bunun üzerine yapılıyor. **apis()** ile dokümana dahil edilecek paketleri seçebiliyoruz, şu an tüm paketler dahil isterseniz bunu değiştirerek belirlediğini paketleri kullanabilirsiniz. **paths()** ise yine aynı şekilde dokümana dahil edilecek adreslerimizi belirlediğimiz yerdir.

Şimdi yaptığımız projeyi ayağa kaldırıyoruz ve {{adres}}/swagger-ui.html adresine giderek swagger ile oluşturulmuş olan dokümantasyona ulaşabiliriz.

A screenshot of the Swagger UI interface. At the top, there's a green header with the Swagger logo and a dropdown menu labeled "Select a spec" with "default" selected. Below the header, the main content area has a title "Api Documentation" with a version badge "1.0". Underneath, it shows the base URL "[Base URL: localhost:8080/]" and a link to "http://localhost:8080/v2/api-docs". There are also links for "Terms of service" and "Apache 2.0". The interface is divided into sections. The first section is "basic-error-controller" with a right arrow. The second section is "user-controller-rest" with a dropdown arrow. Under "user-controller-rest", there are five API endpoints listed in colored boxes: a blue box for GET /user getUsers, a green box for POST /user createUser, a blue box for GET /user/{id} getFindByid, a red box for DELETE /user/{id} deleteUser, and an orange box for PUT /user/{id}/{name} updateUser. At the bottom, there's a "Models" section with a right arrow.

Yukarıda da görüldüğü gibi uygulama içerisinde gerçekleştirmiş olduğumuz Request-Response modellerinin bir listesini görüyoruz.

Swagger ile POSTMAN gibi uygulamalara gerek duymadan işlemlerimizi gerçekleştirebiliriz. Şimdi örnek olarak GET ve POST işlemlerinin nasıl yapıldığına bakalım.

GET İşlemi

localhost

user-controller-rest User Controller Rest

GET

/user getUsers

Parameters

No parameters

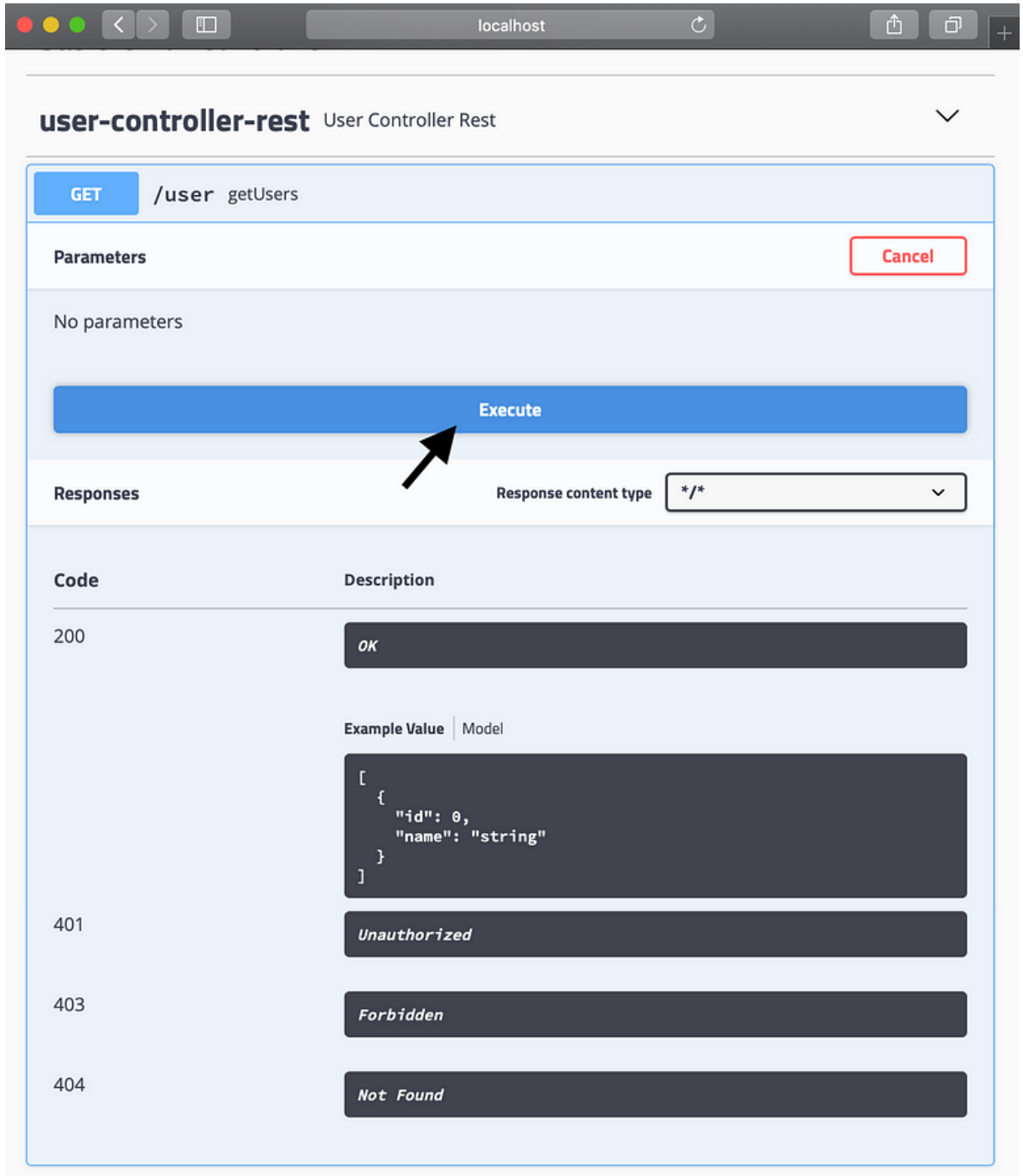
Try it out

Responses

Response content type */*

Code	Description
200	<div>OK</div> <div>Example Value Model</div> <div>[{ "id": 0, "name": "string" }]</div>
401	<div>Unauthorized</div>
403	<div>Forbidden</div>
404	<div>Not Found</div>

GET işleminin 1.Adımı



GET İşleminin 2.Adımı

localhost

Responses

Response content type */*

Curl

curl -X GET "http://localhost:8080/user" -H "accept: */*"

Request URL

http://localhost:8080/user

Server response

Code	Details
200	<div><div>Response body</div><div><pre>[{ "id": 1, "name": "User1" }, { "id": 2, "name": "User2" }, { "id": 3, "name": "User3" }, { "id": 4, "name": "User4" }]</pre></div><div>Download</div></div> <div><div>Response headers</div><div>content-type: application/json;charset=UTF-8 date: Mon, 19 Aug 2019 16:23:14 GMT transfer-encoding: Identity</div></div>

Responses

GET işleminin sonucu

POST İşlemi

localhost

Try it out

POST

/user create user

Parameters

Name	Description
user <small>★ required</small> (body)	<div>user</div> <div><div>Example Value</div><div>Model</div></div> <div><pre>{ "id": 0, "name": "string"} </pre></div> <div>Parameter content type application/json</div>

Responses

Response content type */*

POST İşleminin 1. Adımı

localhost

POST /user createUser

Parameters Cancel

Name	Description
user * required (body)	user

Example Value | Model

```
{  
  "id": 0,  
  "name": "DenemeString"  
}
```

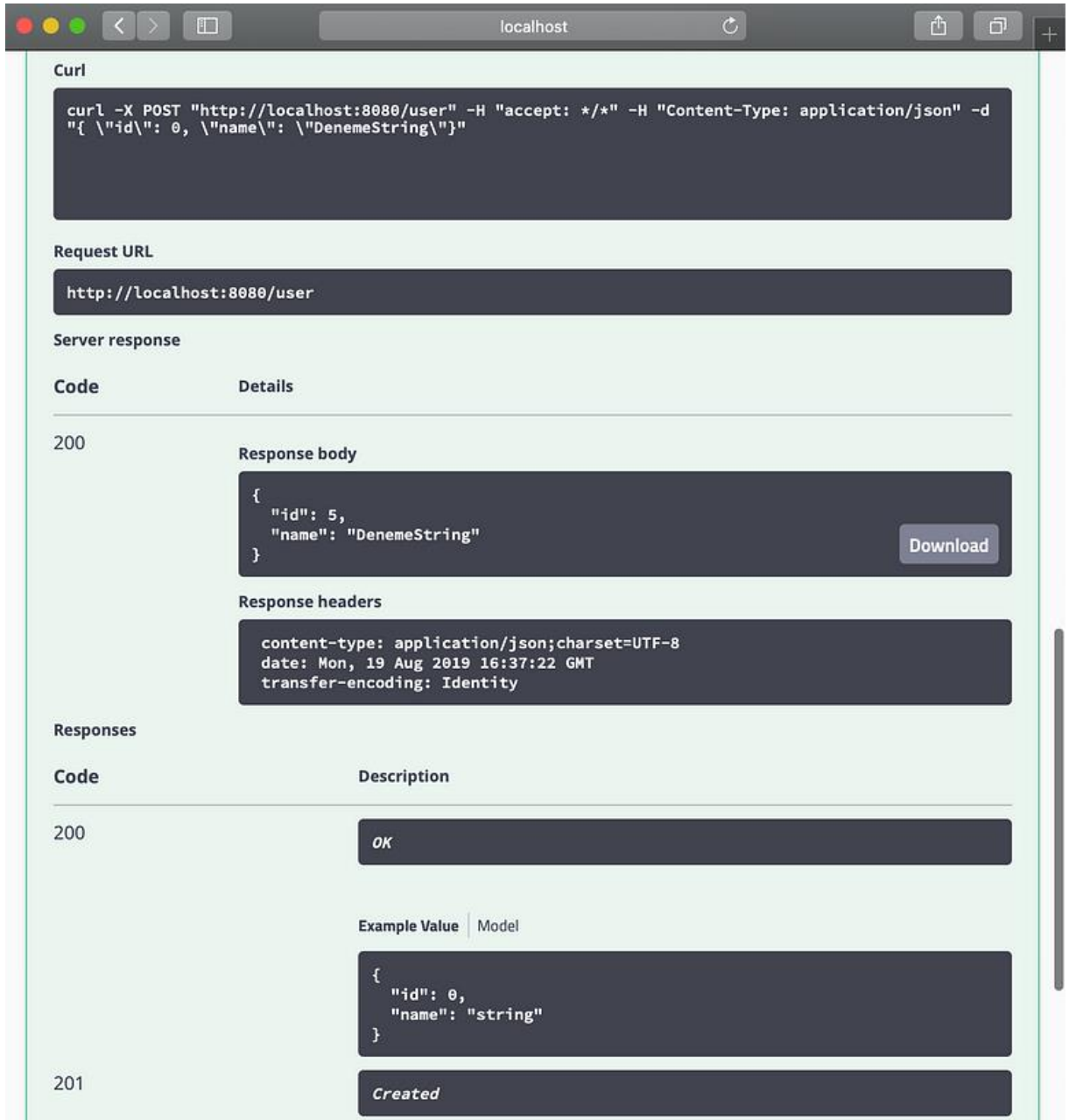
Cancel

Parameter content type
application/json

Execute

Responses Response content type */*

POST İşleminin 2.Adımı



POST İşleminin Sonucu

Resimlerde de görüldüğü gibi POST işlemini gerçekleştirdik ve bunu sonucunda sistem yine bize kayıt ettiğimiz değeri geri dönderdi.

KAYNAKLAR:

<https://medium.com/android-t%C3%BCrkiye/swagger-nedir-ne-i%C3%87%C5%9Fe-yarar-e8c12a9e9e7f>

OpenAI

