



Back End Development 1 ○

+

Sesi 19



API + Documentation with Swagger + Javadoc

Introductions of Javadoc

Pertemuan sebelumnya kita sudah membahas step by step membuat REST API dengan springboot. Maka pada pertemuan kali ini kita akan membahas bagaimana membuat dokumentasi API dengan springboot.

Dokumentasi API yang baik adalah salah satu dari banyak faktor yang berkontribusi pada keberhasilan keseluruhan proyek pengembangan software.

Dan untuk mendukung itu Java sudah menyediakan semua dalam versi modern JDK menyediakan tools Javadoc – untuk menghasilkan dokumentasi API dari komentar yang ada dalam source code.

Prerequisites:

1. JDK 1.4 (JDK 7+ is recommended for the latest version of the Maven Javadoc plugin)
2. The JDK /bin folder added to the PATH environment variable
3. (Optional) an IDE that with built-in tools

Introductions

Mari kita mulai dengan comment

Struktur comment Javadoc terlihat sangat mirip dengan multi-line comment, tetapi perbedaan utamanya adalah tanda bintang tambahan di awal:

```
// This is a single line comment

/*
 * This is a regular multi-line comment
 */

/**
 * This is a Javadoc
 */
```

Introductions

Comment Javadoc dapat ditempatkan di atas class, method, atau field apa pun yang ingin kita dokumentasikan.

Comment ini biasanya terdiri dari dua bagian:

- Deskripsi dari apa yang kami komentari
- Tag blok mandiri (ditandai dengan simbol “@”) yang menjelaskan meta-data tertentu

*Untuk menggunakan beberapa tag blok yang lebih umum kunjungi panduan referensi (<https://docs.oracle.com/en/java/javase/11/tools/javadoc.html>).

API Documentation with Swagger + Javadoc - Sesi 19

Introductions

Mari kita lihat seperti apa comment Javadoc tingkat class :

```
/**
 * Hero is the main entity we'll be using to . . .
 *
 * Please see the {@link com.hacktiv8.javadoc.Person} class for true identity
 * @author Captain America
 *
 */
public class SuperHero extends Person {
    // fields and methods
}
```

Penjelasan :

Pada source diatas terlihat deskripsi singkat dan dua tag blok yang berbeda - standalone & inline:

Tag standalone muncul setelah deskripsi dengan tag sebagai kata pertama dalam satu baris, misalnya tag @author

Tag inline dapat muncul di mana saja dan dikelilingi tanda kurung kurawal, misalnya tag @link dalam deskripsi

Dalam contoh diatas kita juga dapat melihat dua jenis tag blok yang digunakan:

{@link} memberikan tautan inline ke bagian yang dirujuk dari source-code

@author nama author yang menambahkan class, method, atau field yang dikomentari

Introductions

Kita juga dapat menggunakan deskripsi tanpa tag blok seperti ini di dalam kelas SuperHero :

```
/**  
 * The public name of a hero that is common knowledge  
 */  
private String heroName;
```

Private Field tidak bisa dibuat Javadoc kecuali jika kita secara eksplisit meneruskan private-option ke command Javadoc.

API Documentation with Swagger + Javadoc - Sesi 19

Introductions

Method dapat berisi berbagai tag blok Javadoc.

```
/**
 * <p>This is a simple description of the method. . .
 * <a href="http://www.supermanisthegreatest.com">Superman!</a>
 * </p>
 * @param incomingDamage the amount of incoming damage
 * @return the amount of health hero has after attack
 * @see <a href="http://www.link_to_jira/HERO-402">HERO-402</a>
 * @since 1.0
 */
public int successfullyAttacked(int incomingDamage) {
    // do things
    return 0;
}
```


Method successAttacked berisi deskripsi dan banyak tag blok standalone

Ada banyak tag blok untuk membantu menghasilkan dokumentasi yang tepat dan dapat menyertakan semua jenis informasi yang berbeda bahkan dapat menggunakan tag HTML dasar di komentar.

Mari kita bahas tag yang kita temui pada contoh di atas:

@param memberikan deskripsi yang berguna tentang parameter method atau input yang diharapkan

@return memberikan deskripsi tentang apa yang akan atau dapat dikembalikan oleh suatu method

@see akan menghasilkan tautan yang mirip dengan tag {@link}, tetapi lebih dalam konteks referensi dan bukan inline

@since menentukan versi class, field, atau method mana yang ditambahkan ke proyek

@version menentukan versi software, biasanya digunakan dengan makros %I% dan %G%

@throws digunakan untuk menjelaskan lebih lanjut kasus-kasus yang diharapkan software sebagai pengecualian

@deprecated memberikan penjelasan mengapa kode tidak digunakan lagi, kapan kode itu mungkin sudah tidak digunakan lagi, dan apa alternatifnya



Introductions

Javadoc Command Line Tool sangat powerful tetapi memiliki beberapa kerumitan yang menyertainya.

Menjalankan perintah javadoc tanpa opsi atau parameter apa pun akan menghasilkan kesalahan dan parameter keluaran yang diharapkan.

Kita harus setidaknya menentukan package atau class apa yang kita ingin buat dokumentasinya.

Mari kita buka command line dan arahkan ke direktori proyek.

Contoh dengan asumsi semua class ada di folder src di direktori proyek:

```
user@hacktiv8:~$ javadoc -d doc src\*
```

Ini akan menghasilkan dokumentasi dalam direktori bernama doc sebagaimana ditentukan dengan flag -d. Jika ada beberapa package atau file dimana kita harus sediakan semuanya.

Memanfaatkan IDE dengan fungsionalitas bawaan, tentu saja, lebih mudah dan umumnya disarankan.

Kita juga dapat menggunakan plugin Maven Javadoc:

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-javadoc-plugin</artifactId>
      <version>3.0.0</version>
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
      </configuration>
      <tags>
        ...
      </tags>
    </plugin>
  </plugins>
</build>
```



Di direktori base proyek, kita jalankan perintah untuk menghasilkan Javadocs ke direktori di target\site:

```
user@hacktiv8:~$ mvn javadoc:javadoc
```

Sekarang mari kita lihat seperti apa halaman Javadoc yang dihasilkan:

[PACKAGE](#) [CLASS](#) [USE](#) [TREE](#) [DEPRECATED](#) [INDEX](#) [HELP](#)

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#)

[SUMMARY: NESTED | FIELD | CONSTR | METHOD](#) [DETAIL: FIELD | CONSTR | METHOD](#)


```
public class SuperHero
extends Person

Hero is the main entity we will be using to ...

Author:
Captain America
```


Field Summary

[Fields](#)

Modifier and Type	Field and Description
private int	defense
private int	health
private java.lang.String	heroName The public name of a hero that is common knowledge
private java.lang.String	uniquePower

Kita bisa melihat dalam analogi tree view dari class yang diperluas pada class SuperHero kita. Kita bisa melihat deskripsi, field, dan method dan kita dapat mengklik tautan untuk informasi lebih lanjut.

Tampilan mendetail dari method kita terlihat seperti ini:

Method Detail

successfullyAttacked

```
public int successfullyAttacked(int incomingDamage,  
                                java.lang.String damageType)  
    throws java.lang.Exception
```

Deprecated. *As of version 1.1, use ... instead*

This is a simple description of the method... Superman!

Parameters:
incomingDamage - the amount of incoming damage

Returns:
the amount of health hero has after attack

Throws:
java.lang.IllegalArgumentException - if incomingDamage is negative
java.lang.Exception

Since:
1.0

See Also:
[HERO-402](#)



Introductions of Swagger

Swagger banyak digunakan untuk memvisualisasikan API, dan dengan Swagger UI menyediakan sandbox online untuk visualisasi front-end.

Untuk mencoba nya , kita akan menggunakan implementasi Springfox dari spesifikasi Swagger 2.

Swagger adalah tools, spesifikasi, dan implementasi framework yang lengkap untuk menghasilkan representasi visual dari RESTful Web Services dimana memungkinkan dokumentasi diperbarui dengan kecepatan yang sama dengan server. Ketika didefinisikan dengan benar melalui Swagger, konsumen dapat memahami dan berinteraksi dengan layanan jarak jauh dengan sedikit logika implementasi.

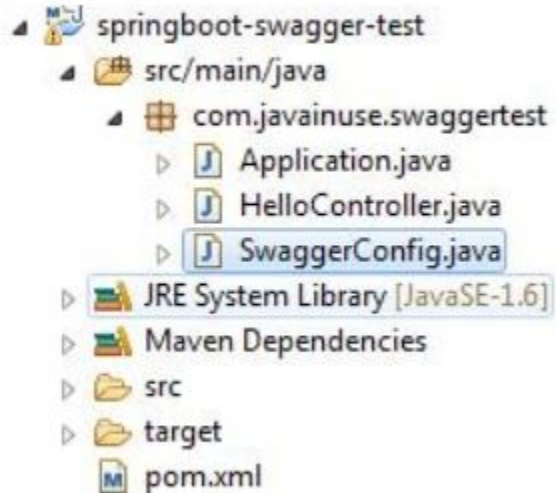
Jadi Swagger menghilangkan guesswork dalam memanggil services.



API Documentation with Swagger + Javadoc - Sesi 19

Structure Projects

The project will be as follows-



API Documentation with Swagger + Javadoc - Sesi 19

Introductions

Di Maven kita membutuhkan dependency swagger. Maven akan menjadi sebagai berikut :

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.hacktiv8</groupId>
  <artifactId>springboot-swagger-test</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.4.1.RELEASE</version>
  </parent>

  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
```




```
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.4.0</version>
</dependency>

<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>2.4.0</version>
</dependency>

</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
</project>
```



Buat Application.java seperti di bawah ini

```
package com.hacktiv8.swaggeritest;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }

}
```

API Documentation with Swagger + Javadoc - Sesi 19

Introductions

@RequestMapping maps /api/hacktiv8 request to sayHello() method.

```
package com.hacktiv8.swaggeritest;

import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HelloController {

    @RequestMapping(method = RequestMethod.GET, value = "/api/hacktiv8")
    public String sayHello() {
        return "Swagger Hello World";
    }
}
```



Penjelasan :

Untuk mengaktifkan Swagger 2 kita bisa menggunakan anotasi `@EnableSwagger2`.

Docket Bean didefinisikan dan menggunakan method `select()` nya, Lalu kita mendapatkan instance `ApiSelectorBuilder`. `ApiSelectorBuilder` yang kita konfigurasi dengan endpoint yang diekspose oleh Swagger.

Setelah Docket Bean didefinisikan maka method `select()` nya akan mengembalikan instance `ApiSelectorBuilder`, yang menyediakan cara untuk mengontrol endpoint yang diekspose oleh Swagger.

Dengan menggunakan `RequestHandlerSelectors` dan `PathSelectors`, kita mengonfigurasi predicates untuk selection `RequestHandlers`.



HACKTIV8

```
package com.hacktiv8.swaggeritest;  
  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
  
import com.google.common.base.Predicate;  
  
import springfox.documentation.builders.ApiInfoBuilder;  
import springfox.documentation.service.ApiInfo;  
import springfox.documentation.spi.DocumentationType;  
import springfox.documentation.spring.web.plugins.Docket;  
import springfox.documentation.swagger2.annotations.EnableSwagger2;  
import static springfox.documentation.builders.PathSelectors.regex;  
import static com.google.common.base.Predicates.or;
```



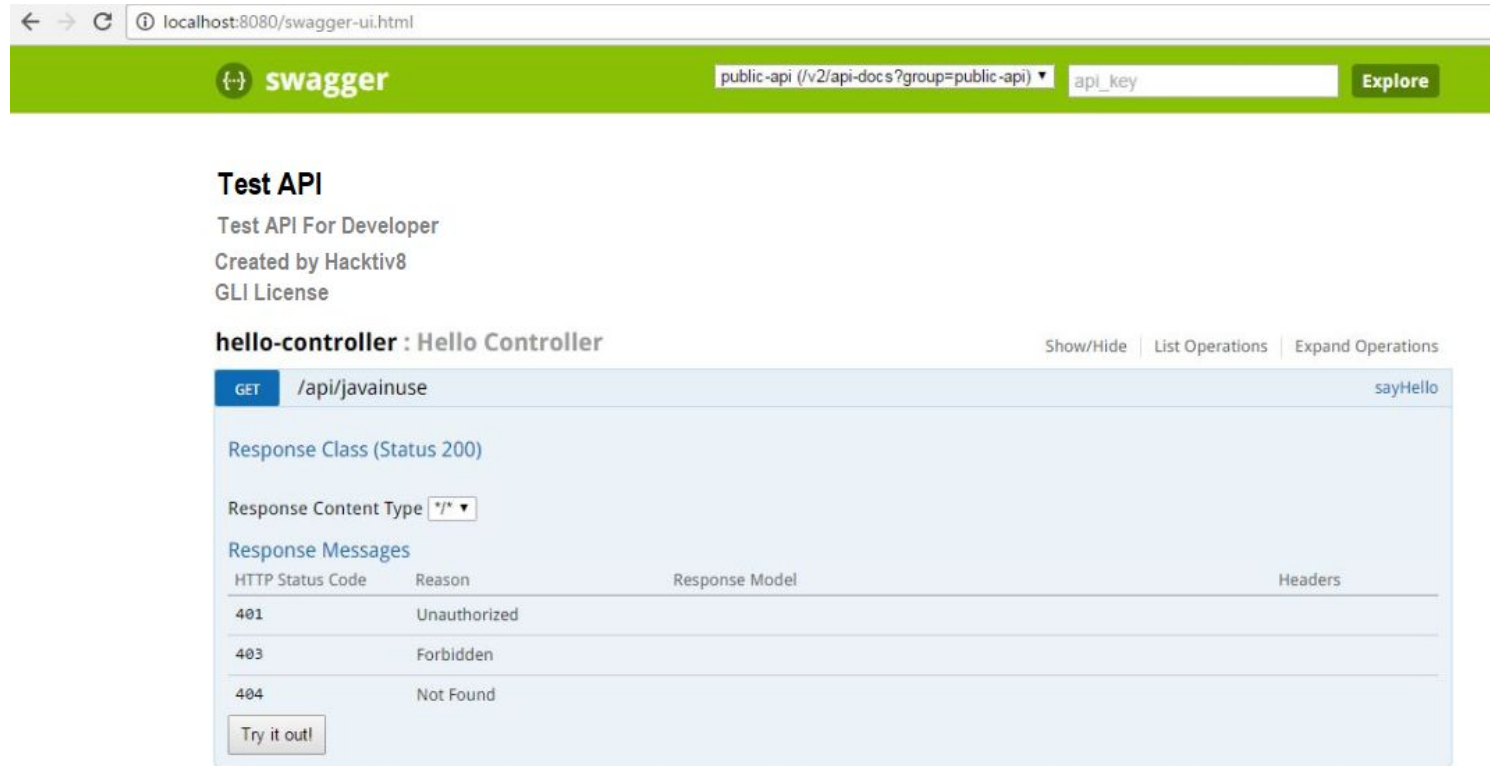
```
@Bean
public Docket postsApi() {
    return new Docket(DocumentationType.SWAGGER_2).groupName("public-api")
        .apiInfo(apiInfo()).select().paths(postPaths()).build();
}

private Predicate<String> postPaths() {
    return or(regex("/api/posts.*"), regex("/api/hacktiv8.*"));
}

private ApiInfo apiInfo() {
    return new ApiInfoBuilder().title("Test API")
        .description("Test API reference for developers")
        .termsOfServiceUrl("http://hacktiv8.com")
        .contact("hacktiv8@gmail.com").license("GLILicense")
        .licenseUrl("hacktiv8@gmail.com").version("1.0").build();
}
}
```



Sekarang buka <http://localhost:8080/swagger-ui.html>.



The screenshot shows the Swagger UI interface in a web browser. The address bar displays `localhost:8080/swagger-ui.html`. The Swagger logo is on the left, and the API title `public-api (/v2/api-docs?group=public-api)` is on the right, along with an `api_key` input field and an `Explore` button.

Test API
Test API For Developer
Created by Hacktiv8
GLI License

hello-controller : Hello Controller Show/Hide List Operations Expand Operations

GET `/api/javainuse` sayHello

Response Class (Status 200)

Response Content Type `*/*`

Response Messages

HTTP Status Code	Reason	Response Model	Headers
401	Unauthorized		
403	Forbidden		
404	Not Found		

[Try it out!](#)