1. وحدة التحكم (Controller Layer):

هي واجهة الخادم (Server) التي من خلالها يمكن استقبال الطلبات عبر البوابات وارسال النتيجة، ولكي يقوم بمهمته لابد أن تحتوي كل بوابة على ثلاثة أشياء:

1.تحديد نوع الدالة: تحديد نوع الطلب GET, POST, PUT , DELETE باستخدام التعليق التوضيحي @....mapping

2. تحديد URL: الذي من خلاله يمكن طلب الصفحة

3. تحديد End Point: الدالة المسؤولة عن استقبال الطلب وارسال النتيجة

```
@RestController
@RequestMapping("/api/v1/message")
public class TodoController {

@GetMapping("/message")
public String getMessage(){
  return “Hey from Spring Boot”;
 }
```

@RestController	يوضح أن هذا class نوعه controller ويقوم بإرجاع نتيجة من نوع JSON
@RequstMapping	لوضع API محدد لهذا controller
@GetMapping	لتحديد نوع Method
(“/message”)	تحديد URLللوصول لهذه الصفحة

2. مفهوم REST :

يعتبرREST ، أوREpresentational State Transfer ، أسلوباً معمارياً لتوفير معايير بين أنظمة الكمبيوتر على الويب، مما يسهل على الأنظمة التواصل مع بعضها البعض. تتميز الأنظمة المتوافقة معREST ، والتي غالباً ما تسمى أنظم ةRESTful ، بكيفية كونها عديمة الحالة وتفصل اهتمامات العمي ل (Client) والخادم .(Server) سوف ندرس ما تعنيه هذه المصطلحات ولماذا تعتبر خصائص مفيدة للخدمات على الويب. فصل العميل (Client) والخاد م (Server)في النمط المعماريREST ، يمكن تنفيذ العميل (Client) وتنفيذ الخاد م (Server) بشكل مستقل دون معرفة كل م نهما بالآخر. هذا يعني أنه يمكن تغيير الكود الموجود على جانب العمي ل (Client) في أي وقت دون التأثير على تشغيل الخاد م(Server) ، ويمكن تغيير الكود الموجود على جانب الخادم (Server) دون التأثير على عمل العميل .(Client) طالما أن كل جانب يعرف تنسيق الرسائل التي سي تم إرسالها إلى الآخر، فيمكن الاحتفاظ بها معيارية ومنفصلة. بفصل مخاوف واجهة المستخدم عن مخاوف تخزين البيانات(Data) ، نقوم بتحسين مرونة الواجهة عبر الأنظمة الأساسية وتحسين قابلية التوسع من خلال تبسيط مكونات الخادم .(Server) بالإضافة إلى ذلك، يتيح الفصل لكل مكون القدرة على التطور بشكل مستقل. باستخدام واجه ةREST ، يصل العملاء المختلفون إلى نفس نقاط نهاية REST، ويقومون بنفس الإجراءات، ويتلقون نفس الاستجابات. الأنظمة التي تتبع نموذج REST هي أنظمة عديمة الحال ة(Stateless) ، مما يعني أن الخادم (Server) لا يحتاج إلى معرفة أي شيء عن حالة العمي ل (Client) والعكس صحيح. بهذه الطريقة، يمكن لكل من الخاد م (Server) والعميل (Client) فهم أي رسالة مستلمة، حتى بدون رؤية الرسائل السابقة.

هناك 4 أفعال HTTP أساسية نستخدمها في طلبات التفاعل مع الموارد في نظام Rest

1. GET : استرداد مورد معين
2. POST : إنشاء مورد جديد
3. PUT: تحديث مورد معين
4. DELETE: إزالة مورد معين بواسطة المعرف

3. مفهوم CRUD (Create,Read,Update,Delete) :

الإنشاء والقراءة والتحديث والحذف (CRUD) هي الوظائف الأساسية الأربع التي يجب أن تكون النماذج قادرة على القيام بها ،على الأكثر. عندما نبني واجهات برمجة التطبيقات، نريد أن توفر نماذجنا أربعة أنواع أساسية من الوظائف.
يجب أن يكون النموذج قادرًا على إنشاء الموارد، وقراءتها وتحديثها وحذفها. غالباً ما يشير علماء الكمبيوتر إلى هذه الوظائف بالاختصا ر .CRUD يجب أن يتمتع النموذج بالقدرة على أداء هذه الوظائف الأربع على الأكثر حتى يكتمل.
إذا كان لا يمكن وصف إجراء من خلال إحدى هذه العمليات الأربع، فمن المحتمل أن يكون نموذجًا خاصًا به. يعد نموذج CRUDشائعاً في إنشاء تطبيقات الويب، لأنه يوفر إطارًا لا ينُسى لتذكير المطورين بكيفية إنشاء نماذج كاملة قابلة للاستخدام.

مثال: لنتخيل نظامًا لتتبع كتب المكتبة.

في قاعدة بيانات المكتبة الافتراضية هذه يمكننا أن نتخيل أنه سيكون هناك مورد كتب، والذي سيخزن أشياء من الكتب.
لنفترض أن كائن الكتاب يشبه هذا:

```
“book”: {
     "id": <Integer>,
     “title”: <String>,
     “author”: <String>,
  }
```

لجعل نظام المكتبة هذا قابلاً للاستخدام، نود التأكد من وجود آليات واضحة لإكمال عمليات CRUD :

عملية الإنشاء Create : سيتكون هذا من وظيفة نسميها عند إضافة كتاب مكتبة جديد إلى الفهرس. البرنامج الذي يستدعي الوظيفة سيوفر قيم ”title“ و ”author“ ، بعد استدعاء هذه الوظيفة، يجب أن يكون هناك إدخال جديد في مورد الكتب المقابل لهذا الكتاب الجديد. بالإضافة إلى ذلك، يتم تعيين معرّف فريد للإدخال الجديد، والذي يمكن استخدامه للوصول إلى هذا ال مورد لاحقاً .

عملية القراءة Read : سيتكون هذا من وظيفة سيتم استدعاؤها لرؤية جميع الكتب الموجودة حالياً في الفهرس. لن يؤدي استدعاء الوظيفة هذا إلى تغيير الكتب الموجودة في الكتالوج - بل سيؤدي ببساطة إلى استرداد المورد وعرض النتائج. سيكون لدينا أيضًا وظيفة لاسترداد كتاب واحد، يمكننا توفير العنوان أو المؤلف ومرة أخرى، لن يتم تعديل هذا الكتاب، بل سيتم استرجاعه فقط .

عملية التعديل أو التحديث Update: يجب أن تكون هناك وظيفة لاستدعاء عندما يجب تغيير المعلومات حول الكتاب. سيقوم البرنامج الذي يستدعي الوظيفة بتزويد القيم الجديدة لكل من ”title“ و ”author“ بعد استدعاء الوظيفة، سيحتوي الإدخال المقابل في مورد الكتب على الحقول الجديدة المتوفر ة .

عملية الحذف Delete: يجب أن تكون هناك وظيفة لاستدعاء لإزالة كتاب مكتبة من الفهرس. سيوفر البرنامج الذي يستدعي الوظيفة قيمة واحدة أو أكثر من title و author لتحديد الكتاب، ثم تتم إزالة هذا الكتاب من مصدر الكتب.
بعد استدعاء هذه الوظيفة، يجب أن يحتوي مورد الكتب على جميع الكتب التي كان لديه من قبل، باستثناء الكتاب الذي تم حذفه للتو.

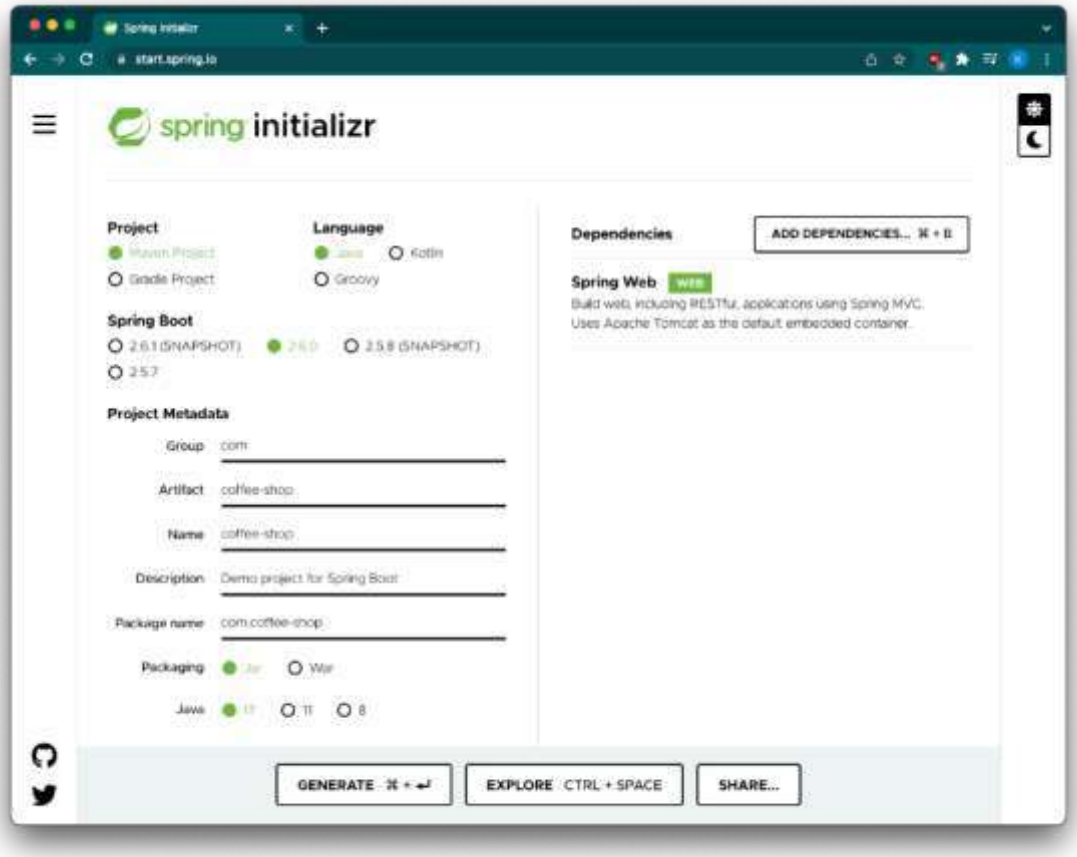
4. بناء REST Api :

المتطلبات: برنامج Postman لاختبار الروابط.

في هذا الجزء سنقوم ببناء مشروع يحتوي على الأربع عناصر الأساسية في أي مشروع Api وهي:

- getوهو النوع الخاص باستقبال البيانات.
- postوهو النوع الخاص بإضافة بيانات جديدة.
- putوهو النوع الخاص بتعديل البيانات.
- deleteوهو النوع الخاص بحذف البيانات.

بناء مشروع Spring Boot عن طريق Spring Boot Initializer



مثال: إنشاء مجلد خاص بالقهوة وصفاتها:

في البداية سنقوم بإنشاء المجلد الخاص بالقهوة والذي يحتوي على جميع الملفات الخاصة بالقهوة، ولكن قبل البدء بكتابته سنتعرف على أفضل الممارسات في تنظيم المشروع.

```
com  +-
example
     +- myapplication
         +- MyApplication.java
         |
         +- class-name
         |   +- ClassName.java
         |   +- ClassNameController.java
         |
```

إنشاء ملف coffee.java الخاص بالقهوة:

بداخل هذا المجلد سنقوم بإنشاء ملف javaجديد وسيكون باسم Coffee.javaوبداخله الصفات المتعلقة بالقهوة مثل الاسم و السعر... إلى آخره.

```
package
com.coffeeshop.coffee; public
class Coffee {     private
final long id;     private
String name;     private
double price;
    public Coffee(long id, String name, double price)
{         this.id = id;         this.name = name;
this.price = price;
    }
    public long getId() {
return id;
    }
    public String getName() {
return name;
    }
    public void setName(String name) {
this.name = name;
    }
    public double getPrice() {
return price;
    }
    public void setPrice(double price) {
this.price = price;
    }
    @Override
    public String toString() {
        return "Coffee [id=" + id + ", name=" + name + ", price=" +
price + "]";
    }
}
```

يحتوي ملف coffee.java على ثلاثة صفات :

- صفة :id وهي ترمز إلى رقم خاص بكل قهوة ونوعها.long
- صفة :name وهي صفة خاصة باسم القهوة ونوعها.String
- صفة :price وهي صفة خاصة بسعر القهوة ونوعها.double

وتم إضافة setter و getter للصفات وكذلك تم إضافةtoString .

إنشاء ملف CoffeeController.java

coffeeController.java هو الملف المسؤول عن إدارة طلبات Rest Api Class الخاصة، وسوف يحتوي على requests الأساسية: Get , Post ,Put , Delete

من المهم استخدام @RestController عند بدايته وذلك لتحديد أنه كلاس خاص بمعالجة Rest Requests

```
package com.coffeeshop.coffee;

import org.springframework.web.bind.annotation.RestController;
    @RestController
    public class CoffeeController {

    }
```

الآن نضيف بيانات coffee وسوف تكون من نوع ArrayList <Coffee> وباسم coffeesوسوف نسند له قيمة new ArrayList<>()

```
Array<Coffee> coffees = new ArrayList<>();
```

4.1 طلبات Get :

سنقوم الآن بإنشاء أول Api لنا وسيكون من نوع get وسيكون خاص باستعراض أنواع القهوة التي لدينا ، حالياً سيقوم بإرجاع مصفوفة فارغة.

4.2 طلبات Post :

تختلف طلبات Post عن Get بسبب أن Post تحتوي على Body ويمكننا من وضع بيانات فيها وتقوم Post بإضافة وإرسال البيانات. الآن سنقوم بإضافة postApi آخر وسيكون وظيفته إضافة قهوة إلى مجموعة القهوة التي نملكها. في البداية سوف نقوم بكتابة method الخاصة بهذه الوظيفة:

```
@PostMapping()
public ResponseEntity<Coffee> addCoffee(@RequestBody Coffee c) {
coffees.add(c);
    return ResponseEntity.status(201).body(new ApiResponse("Coffee
Added"));
}
```

استخدمنا @PostMapping()من أجل تحديد أن هذا الطلب ه و Post أو الرابط الخاص بهذه الوظيفة، كما قمنا بإضافة

واستخدام @RequestBodyوذلك لإضافة بيانات القهوة الجديدة. وبداخل هذه الدالة قمنا بإضافة القهوة cالموجودة بداخل @RequestBody .

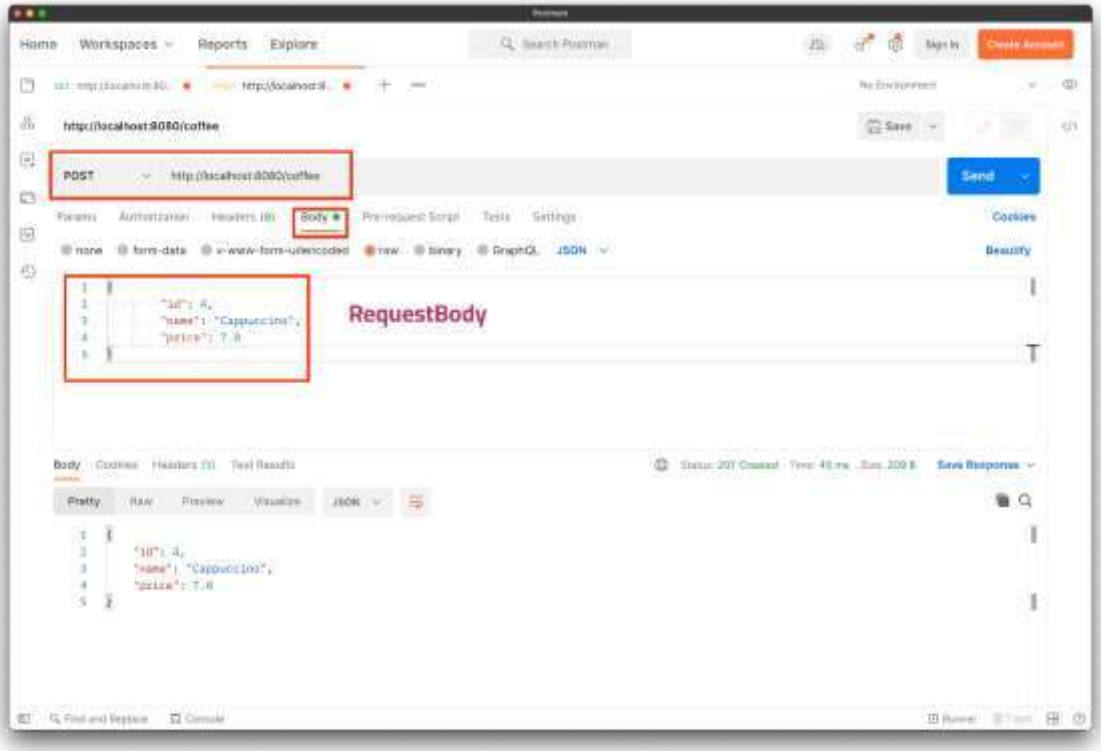
ولتحسين Api سنقوم باستخدام ResponseEntity وذلك لكي نعيد إشارة Httpststus والتي سنتعرف عليها بشكل أكبر في وقت لاحق .

وكذلك قمنا بإنشاء كلاس ApiResponse ومعرف بداخله message من نوع String لاستخدامه في إرجاع message داخل ال Body

يمكننا طلب Api عن طريق الرابط http://localhost:8080

ولكن سنقوم بتعديل المسار إلى http://localhost:8080/coffee وذلك لتمييز أن هذا هو المسار الخاص بجلب المعلومات الخاصة بالقهوة. ويمكننا ذلك عن طريق إضافة

```
@PostMapping("/coffee)
```



4.3 طلبات PUT :

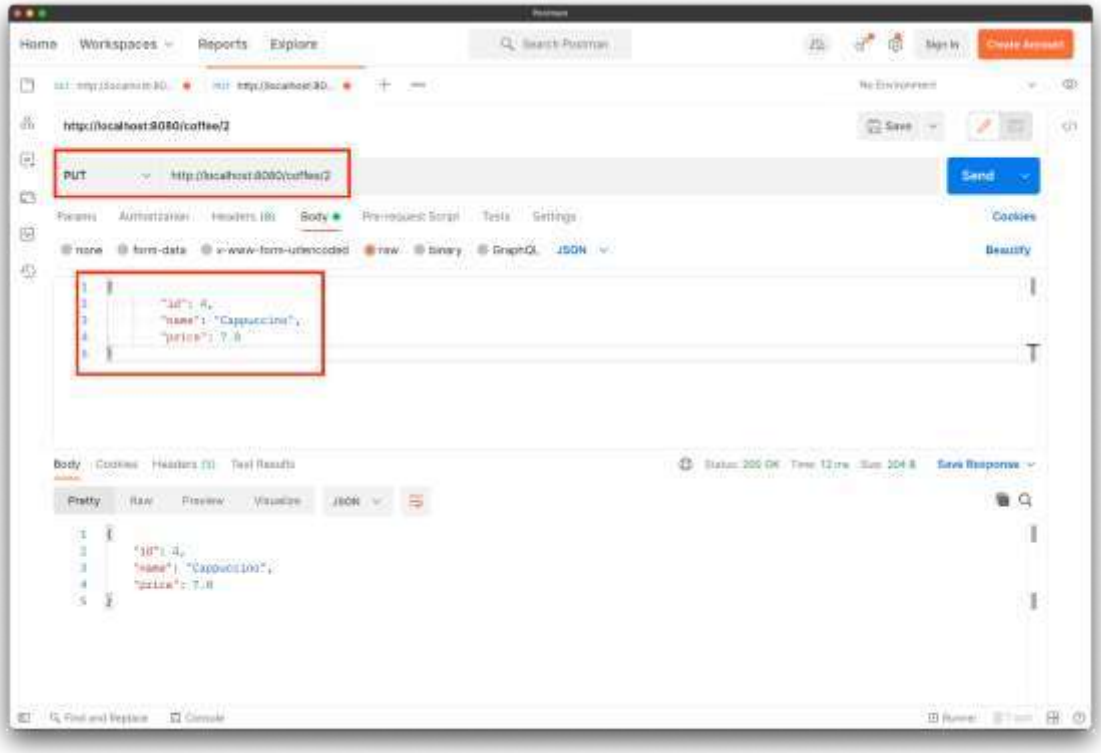
تقوم طلبات Put بتعديل البيانات الموجودة. سنقوم الآن بإضافة putApi وسيكون وظيفته التعديل على بيانات القهوة الموجودة. في البداية سنقوم بكتابة method الخاصة بهذه الوظيفة:

```
public ResponseEntity updateCoffee(@PathVariable Integer
id,@RequestBody Coffee c) {
coffees.set(id,c);
return ResponseEntity.status(200).body(new ApiResponse("Coffee
Updated"));
```

- استخدمنا @PutMapping("/{id}") من أجل تحديد المسار أو الرابط الخاص بهذه الوظيفة وقمنا بإضافة {id}/وذلك لإرسال هذه القيمة إلى Parameterالخاص بهذه الدالة،

مثال http://localhost:8080/coffee/1 :ويمثل رقم ١ idالخاص بالقهوة.

- قمنا باستخدام @ PathVariableوذلك لجعل id يأخذ قيمته من الرابط.
- قمنا باستخدام @RequestBody وذلك لإضافة بيانات القهوة الجديدة.

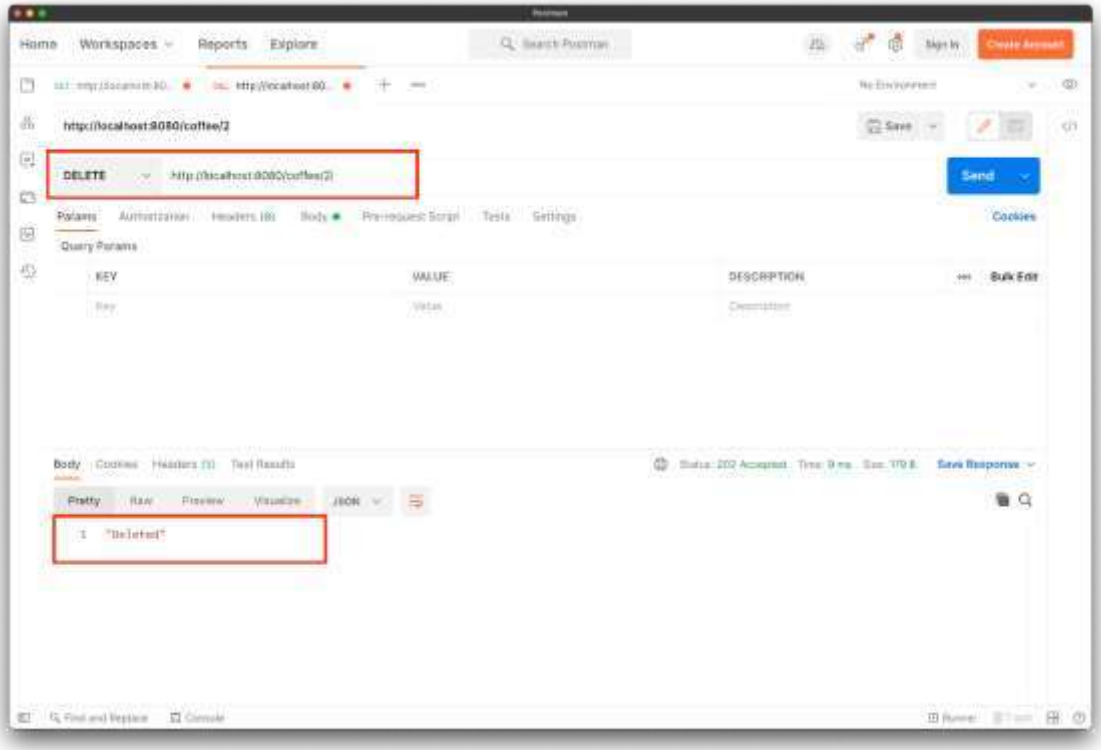


4.4 طلبات Delete :

تقوم طلبات Delete بحذف البيانات. سنقوم الآن بإضافة deleteApi وسيكون وظيفته حذف قهوة موجودة بين القهوة التي لدينا. في البداية سنقوم بكتابة method الخاصة بهذه الوظيفة:

```
public ResponseEntity deleteCoffee(@PathVariable Integer id) {
        coffees.remove(id);          return
ResponseEntity.status(200).body(new
ApiResponse("Deleted"));
```

- ستخدمنا @DeleteMapping("/{id}")من أجل تحديد المسار أو الرابط الخاص بهذه الوظيفة وقمنا بإضافة {id}/وذلك لإرسال هذه القيمة إلى Parameterالخاص بهذه الدالة
مثال http://localhost:8080/coffee/1 :ويمثل رقم ١ idالخاص بالقهوة.
- قمنا باستخدام @PathVariableوذلك لجعل idيأخذ قيمته من الرابط.



5. مكتبة Lombok :

هي مكتبة جافا تستُخدم لتقليل / إزالة الشفرة المعيارية وحفظ الوقت للمطورين أثناء التطوير عن طريق استخدام بعض التعليقات التوضيحية فقط. بالإضافة إلى ذلك ، فإنه يزيد أيضًا من قابلية قراءة الكود المصدري ويوفر المساحة.

تعليقات Lombok التوضيحية: توفر Lombok مجموعة من التعليقات التوضيحية لجعل حياتنا البرمجية أسهل .

لنلقي نظرة على التعليقات التوضيحية القليلة الأكثر استخدامًا فيLombok :

@Getter and @Setter: : ينشئ التعليق التوضيحي Getter طريقة getter بنوع وصول كعام والذي يقوم ببساطة بإرجاع الحقل وباسم getName() إذا كان اسم الحقل هو ."Name" ينشئ التعليق التوضيحي Setter طريقة تعيين بنوع الوصول كعموم والذي يعيد الفراغ ويأخذ معلمة واحدة لتعيين القيمة للحقل. سيكون للمُع يّن الافتراضي اسم setName() إذا كان اسم الحقل هو"Name" .

@NoArgsConstructor:يتم استخدام هذا التعليق التوضيحي لإنشاء مُنشئ بدون وسيطات. له جسم فارغ ولا يفعل شيئاً. يتم استخدامه بشكل عام مع بعض المُنشئ ذي المعلمات الأخرى قيد الاستخدام. يكون مطلوبًا عندما تريد إنشاء كائن للفئة من خلال عدم تمرير أي وسائط في المنشئ.

@AllArgsConstructor:يتم استخدام هذا التعليق التوضيحي لإنشاء مُنشئ ذي parameters إنه مطلوب عندما تريد إنشاء كائن للفئة عن طريق تمرير القيم الأولية للحقول في المنشئ.

@Data:هذا التعليق التوضيحي عبارة عن تعليق توضيحي مختصر لمجموعات التعليقات التوضيحية ToStringو Getterو Setterو RequiredArgsConstructorفي تعليق توضيحي واحد.