FREE EBOOK

Memulai Java Enterprise dengan Spring Boot

Langkah mudah menguasai
Basic Spring Boot

Teten Nugraha

Kata Pengantar

Sebelum ada Spring, sebenarnya ada tools untuk pemrograman enterprise yaitu Java Enterprise atau JEE pada tahun 2000. Namun, ketika membuat aplikasi menggunakan J2EE sangatlah tidak mudah dan sangat rumit. J2EE sangat *tightly coopling* sehingga tidak memungkinkan untuk melakukan unit testing dengan koneksi eksternal misalnya koneksi ke database. Bahkan, untuk menguji fitur sederhana sekalipun harus menggunakan / mengerahkan seluruh aplikasi dalam sebuah container. Karena hal seperti itu lah J2EE mulai banyak ditinggalkan oleh programmer Java khususnya programmer Java di era tahun 2000 an.

Spring dengan *Dependency Injection* nya bisa menjawab permasalahan programmer-programmer khususnya yang sudah lama berkecimpung menggunakan *Java Enterprise Edition* sehingga dengan itu bisa melakukan unit testing dan antar *bean* menjadi modular dan tidak ada keterkaitan dengan *bean* yang lainnya.
Jika kita melihat dokumentasi nya secara langsung, Spring Framework adalah sebuah platform Java yang sangat handal yang menyediakan arsitektur yang komprefensif yang mendukung sepenuhnya untuk membangun aplikasi menggunakan Java Enterprise.
Pertama kali ditulis oleh Rod Jhonson dengan lisensi dari Apache 2.0 pada bulan April 2003. Spring seakan menjadi semacam standar wajib untuk programmer Programmer Java dalam sebuah perusahaan.
Spring Framework menggunakan teknik programan yang sederhana, didukung dengan komunitas yang besar dan dokumentasi yang lengkap, banyak programmer java yang pindah menggunakan Spring Framework. Akhirnya pemrogram bisa membuat aplikasi Enterprise atau Big Data sekalipun dalam waktu yang relative cepat.

Dalam buku ini akan dibahas mengenai, pengenalan Java Enterprise menggunakan Spring Boot dari fundamental sampai tingkat lanjut. Topik-topik yang akan dibahas dalam buku ini adalah :

- Konsep *Dependency Injection* dan penerapannya dalam *Spring*
- Menggunakan *Spring Data JPA* untuk berkomunikasi dengan database
- Spring Web dengan menggunakan *template engine Thymeleaf*
- Membuat RESTFul Webservice berbasis JSON
- Penerapan *Basic Authentication* dengan *Spring Security*
- Membuat Unit Testing dengan JUnit dan Mockito agar *source code* yang sudah dibuat bisa menjadi berkualitas.
- Monitor kualitas *Source code* yang ditulis dengan Jacoco dan Sonarqube
- Belajar menggunakan *Abstract Oriented Programming*
- Deploy aplikasi RESTful ke dalam Docker menggunakan Docker-Compose

Semoga *ebook* ini sedikitnya dapat membantu teman-teman dalam mempelajari Spring Boot dasar.

Bandung, 11 Maret 2020

Teten Nugraha

Daftar Isi

Konsep Object Oriented Programming

Berikut adalah beberapa konsep OOP (*Object Oriented Programming*) yang dimiliki Java :

- ***Class*** merupakan sebuah kerangka/model (blueprint) atau bentuk awal (prototype) yang berfungsi untuk menaruh dan mendeskripsikan perilaku (behavior) dsari sebuah objek nantinya. Penamaan nama class deprogram harus sama dengan struktur file extension .java.

- ***Object*** merupakan sebuah representasi dari instance dari Class. Object adalah sebuah inti dan wujud real dari sebuah Class. Object didefinisikan sebagai state dan behavior dari Class.

- ***Attribute*** merupakan sebuah unsur data yang ada di class, atribut biasanya terdiri dari sebuah data, variable, property dan field. Atribut bisa juga disebut state dari object tersebut. Missal Manusia mempunyai atribut nama, tinggi, berat dan Jenis Kelamin

- ***Method*** merupakan perilaku dari sebuah class, misalkan untuk class Manusia mempunyai method: Berjalan, Berbicara, Tidur.

- ***Encapsulation*** adalah suatu memkanisme membungkus suatu data (variable) agar tidak dapat diakses oleh class lain, dengan menggunakan modifier *private* atau *protected*. Didalam konsep ini beberapa variable akan disembunyikan oleh class lain dan hanya bisa diakses di main Class dengan menggunakan method yang mempunyai modifier *public*.

- ***Inheritance*** (Pewarisan) adalah suatu proses dimana, suatu class yang bsia disebut *super class* dapat mewarisi sifat atau ciri-ciri seperti atribut dan method ke dalam class turunannya yaitu sub class. Super class akan mewarisi nilai dan atribut atau behavior dari class turunannya .

- ***Polymorphisme*** adalah suatu kemampuan yang dimiliki oleh sebuah method dengan nama yang sama tapi dengan perilak yang berbeda-beda.

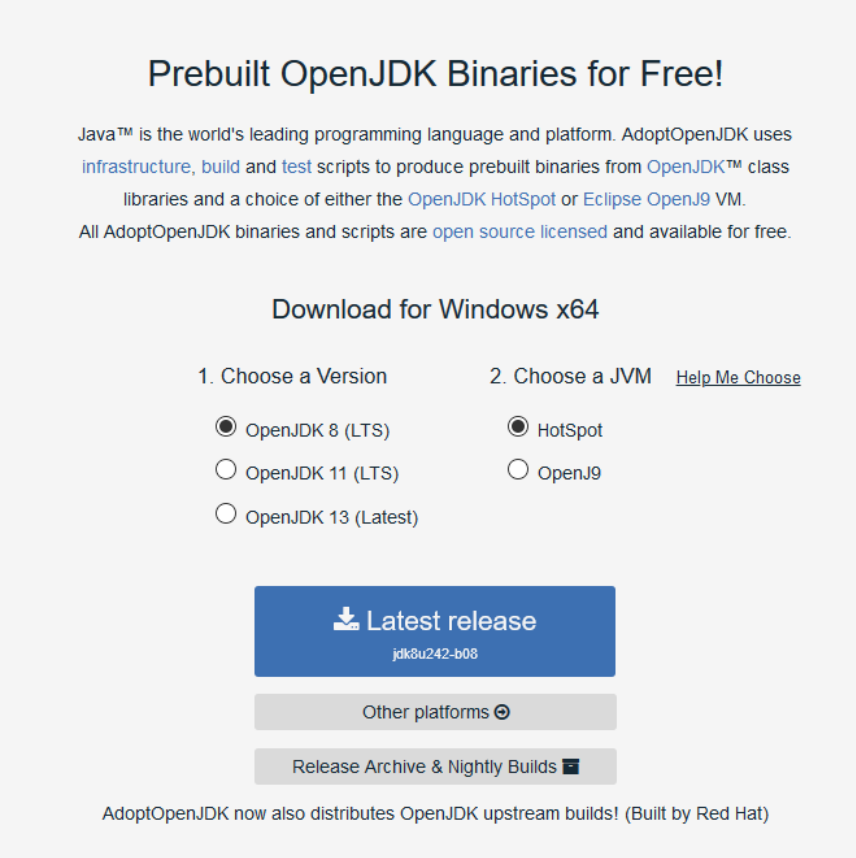
- ***Abstraksi*** adalah proses menyembunyikan detail implemtasi dan hanya menampilkan fungsionalitas kepada pengguna, jadi user atau pengguna tidak tahu menahu proses pembuatannya.

Tools yang digunakan

1. JDK (*Java Development Kit*) dan setting JAVA_HOME

JDK (*Java Developmentkit*) adalah sebuah perangkat lunak yang digunakan untuk membuat program berbasis Bahasa Java. Didalam nya sudah *incude* dengan JRE (Java Runtime Environtment) yang berfungsi untuk menjalankan program yang sudah dibuat.
Saat ini terdapat dua versi JDK yang sering dipakai yaitu yang bernaung dibawah Oracle Corporation dan OpenJDK yang berbasis komunitas. Keduanya sama baiknya digunakan sesuai kebutuhan, namun dalam buku ini penulis menggunakan versi Openjdk 8, kalian juga bisa menggunakan versi 11 nya.

- Download OpenJDK
  Kunjungi situs ini https://adoptopenjdk.net/ untuk mendownloadnya. Situs tersebut secara otomatis mendeteksi system operasi apa yang digunakan pada laptop sehingga tidak perlu mengisi pilihan system operasi.

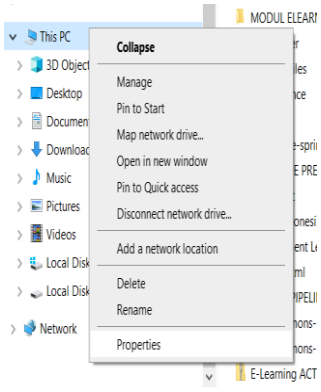


  Pada situs tersebut kalian bisa memilih versi 8, 11 atau 13 untuk JDK yang akan digunakan. Tapi dalam buku ini menggunakan Microsoft Windows 10 64 bit dengan versi JDK 11.

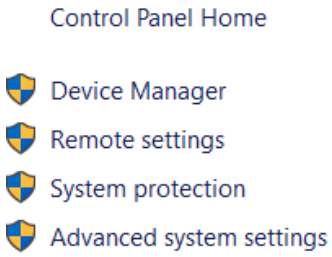
- Install OpenJDK
  Karena AdoptOpenJDK berbasis GUI, maka user dapat dengan mudah melakukan proses instalasi nya :
  a. Masuk ke direktori tempat mengunduh JDK kemudian double klik
  b. Baca dan setujui prasayat
  c. Dalam *Custom Setup*, kalian bisa memilih tempat dimana jdk akan diinstall. Tapi secara default, system akan menyimpan pada direktori `c:\Program Files\AdoptOpenJDK\<package>`
  d. Klik Install dan Tunggu sampai instalasi beres

- Setting Path dan JAVA_HOME
  Masuk ke dalam system properties laptop, kemudian setting PATH dan masukan direktori tempat OpenJDK diinstall

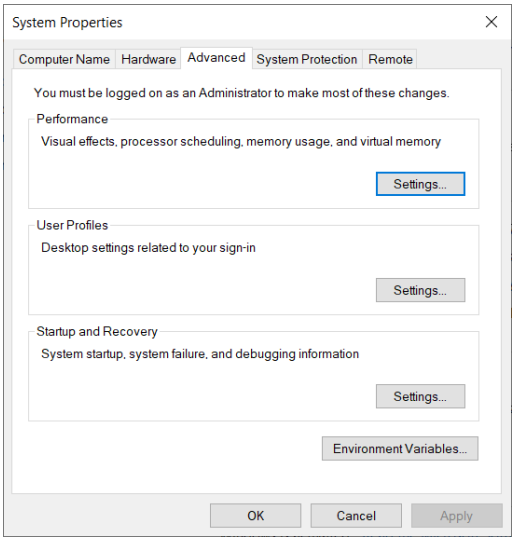
a. Klik kanan This PC dalam explorer, kemudian pilih **Properties**



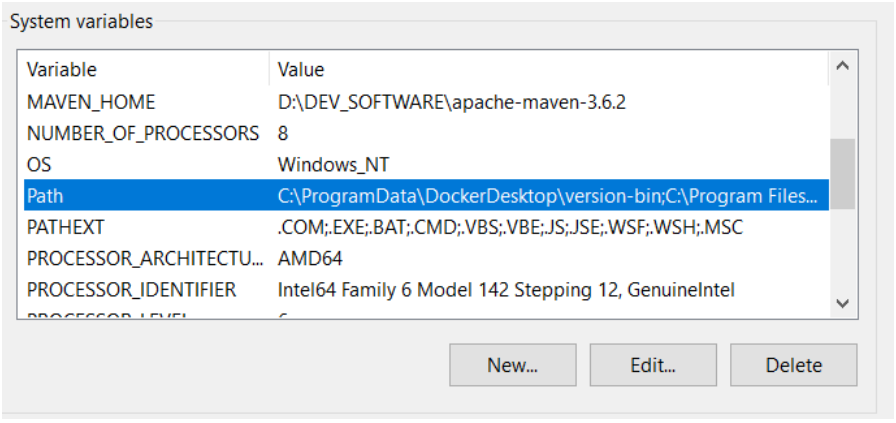
b. Muncul window System, kemudian pilih **Advance System Setting**



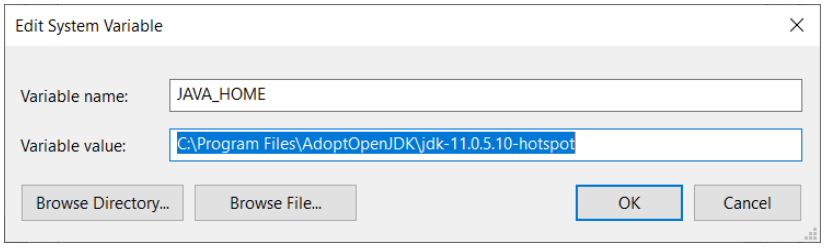
c. Kemudian pilih **Environtment Variables**



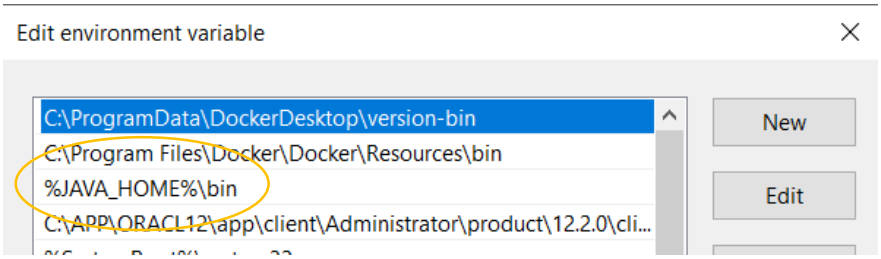
d. Pada System Variables, kemudian klik button **New**



e. Masukan Path JAVA_HOME pada local computer anda sesuai dengan direktori tempat instalasi OpenJDK



f. Kemudian Masukan ke PATH System Variable nya



g. Klik Apply kemudian Ok.

- Testing dalam Command Prompt
  Jika sudah beres, buka command prompt, kemudian ketik

```
java -version
```

jika berhasil maka tampilannya akan seperti dibawah ini

```
Select C:\Windows\system32\cmd.exe
Microsoft Windows [Version 10.0.18362.592]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\tetennugraha>java -version
openjdk version "11.0.5" 2019-10-15
OpenJDK Runtime Environment AdoptOpenJDK (build 11.0.5+10)
OpenJDK 64-Bit Server VM AdoptOpenJDK (build 11.0.5+10, mixed mode)
```

2. Maven 3

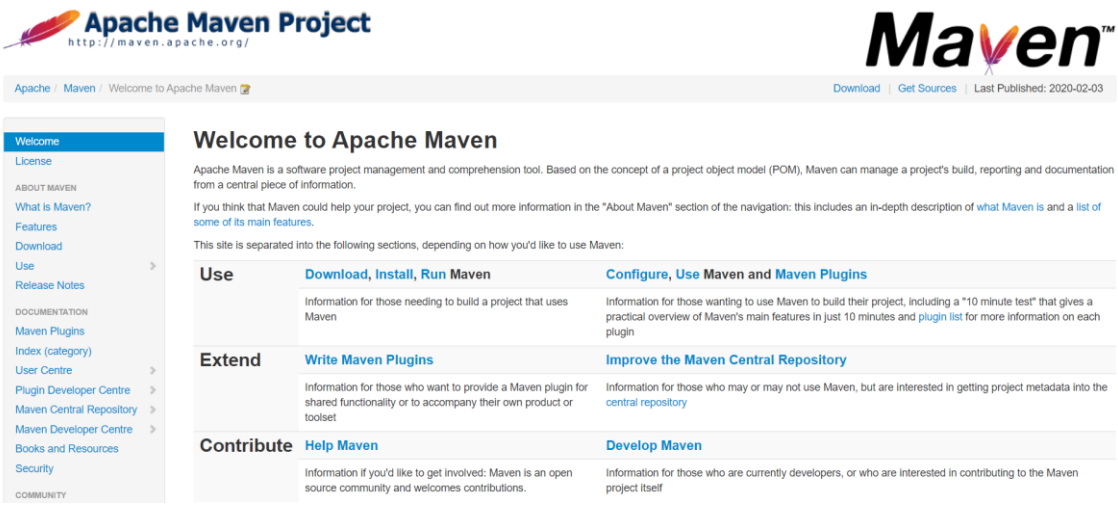
Maven adalah sebuah *build tools* yang berdasarkan pada POM (*Project Object Model*) digunakan untuk mem-*build, deploy* dan dokumentasi semua project yang berbasiskan Java. Dengan Maven, banyak memudahkan pekerjaan programmer Java.

Beberapa keunggulan menggunakan maven diantaranya :

- Mem-*build* project dapat dilakukan secara cepat
- Kita dapat dengan mudah menambah / mengurangi atau bahkan memperbaharui library *dependency* dalam sebuah file *pom*
- Maven dapat menyediakan informasi suatu projek (Log Dokumen, Daftar *Dependencies, Codecoverage, Unit Test Report* dan lain-lain).
- Maven dapat dengan mudah membuild projek ke dalam bentuk JAR atau WAR.

a. Mengunduh Maven

Untuk mengunduh maven, kalian bisa langsung mengunduh nya di situs resmi maven yaitu https://maven.apache.org/ . Disitus tersebut terdapat banyak versi Maven yang bisa digunakan. Namun penulis menggunakan versi Maven 3.6.2



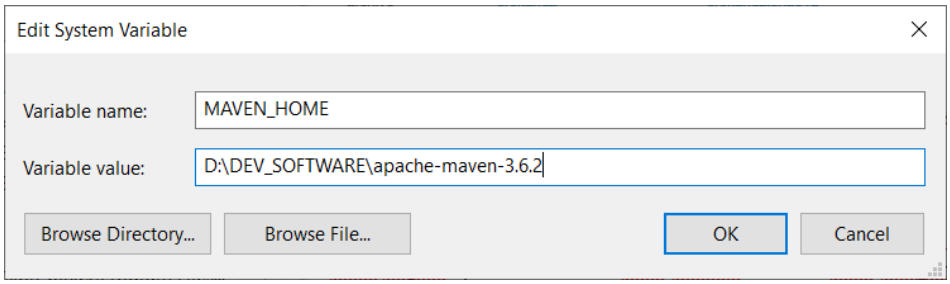
Maven tersedia dalam paket yang sudah dikompresi (.tar.gz / zip). Silahkan unduh menggunakan versi .zip nya

	Link	Checksums	Signature
Binary tar.gz archive	apache-maven-3.6.3-bin.tar.gz	apache-maven-3.6.3-bin.tar.gz.sha512	apache-maven-3.6.3-bin.tar.gz.asc
Binary zip archive	apache-maven-3.6.3-bin.zip	apache-maven-3.6.3-bin.zip.sha512	apache-maven-3.6.3-bin.zip.asc
Source tar.gz archive	apache-maven-3.6.3-src.tar.gz	apache-maven-3.6.3-src.tar.gz.sha512	apache-maven-3.6.3-src.tar.gz.asc
Source zip archive	apache-maven-3.6.3-src.zip	apache-maven-3.6.3-src.zip.sha512	apache-maven-3.6.3-src.zip.asc

Kemudian unzip paket maven yang sudah diunduh ke dalam sebuah folder.

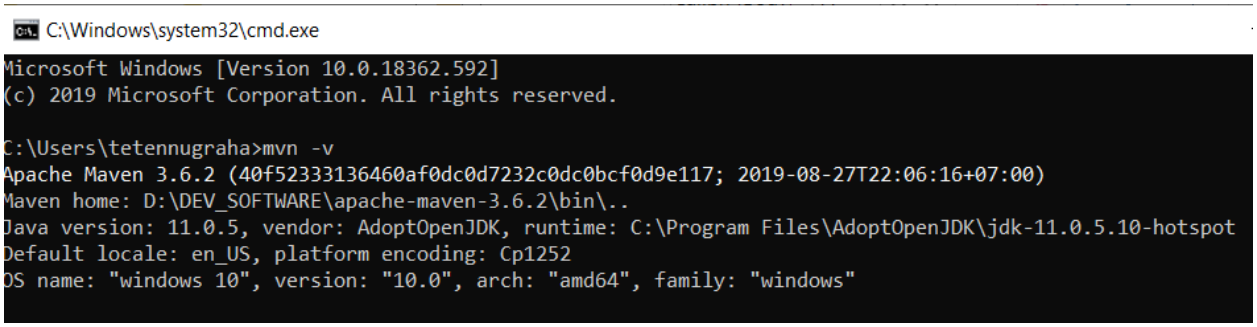
b. Setting Path Maven

Masuk ke *Environtment Variables* (Seperti yang dilakukan pada saat setting PATH Java Home). Pada bagian *System Variables* buat sebuah variable baru dengan nama *MAVEN_HOME* dengan nilai yaitu direktori tempat maven disimpan difolder diatas.



Kemudian klik Ok, dan masukan MAVEN_HOME yang sudah dibuat ke dalam *Path system variables* sama seperti pada saat melakukan instalasi JAVA_HOME. Jika selesai maka kita akan test apakah maven nya sudah berjalan di service dengan mengetikan sintak berikut pada command prompt.

mvn -v

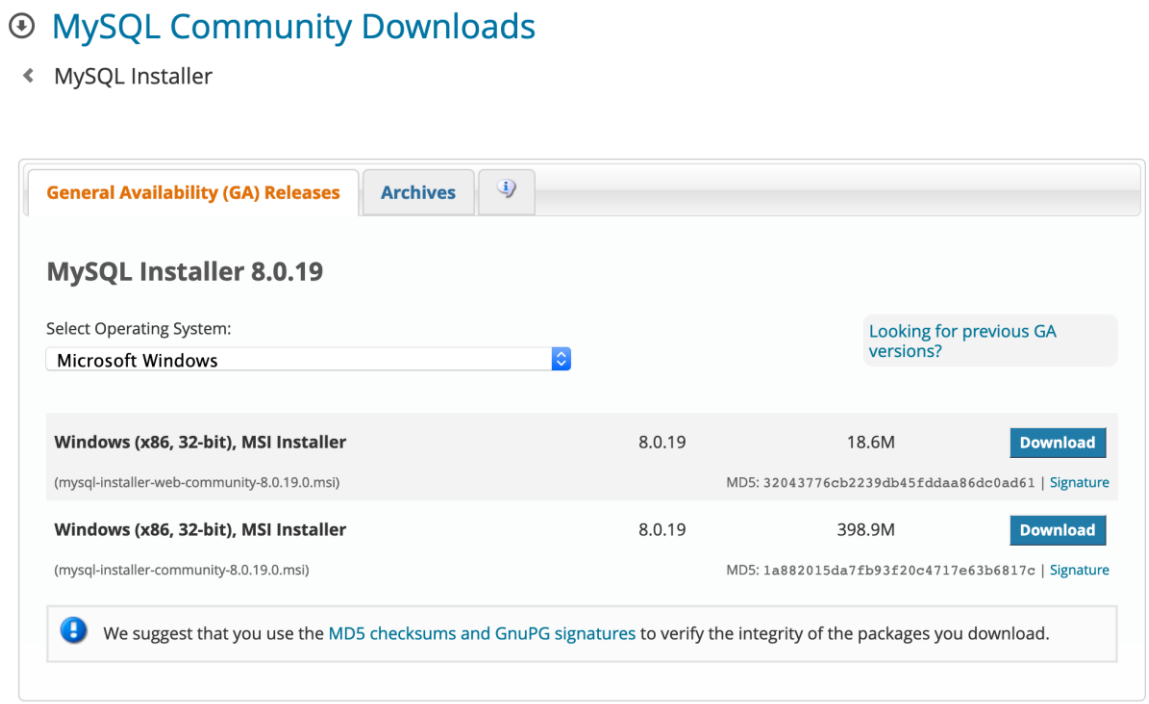


3. MySQL Database

Pada buku ini kita menggunakan MySQL Database untuk penyimpanan datanya. MySQL adalah engine database yang bersifat RDBMS (*Relational Database Management System*) yang biasanya digunakan oleh banyak programmer baik untuk development atau bahkan ada yang sudah dipakai di level production.

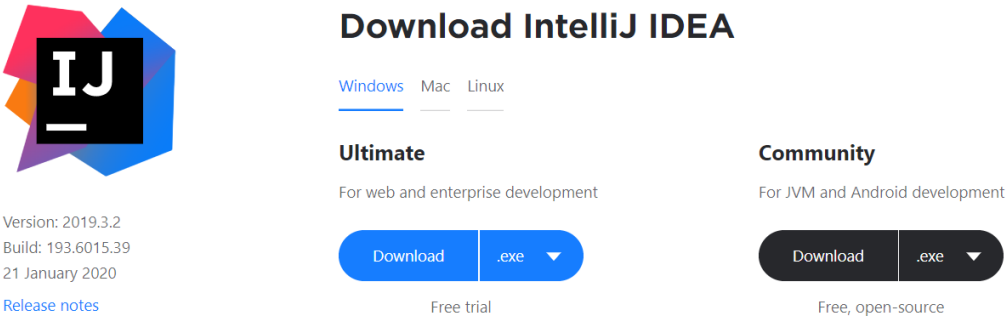
MySQL AB membuat MySQL tersedia gratis untuk komunitas dibawah lisensi *GNU General Public License*, tetapi mereka juga membuat versi yang premium atau komersial untuk kasus-kasus penggunanya tidak cocok dengan GPL.

Kalian bisa mengundu di situs https://dev.mysql.com/downloads/installer/ dan bisa dipilih berdasarkan system operasi yang kalian gunakan dilaptop. Untuk buku ini penulis menggunakan versi MySQL GPL atau freesoftware.



4. IntelliJ IDEA Community IDE

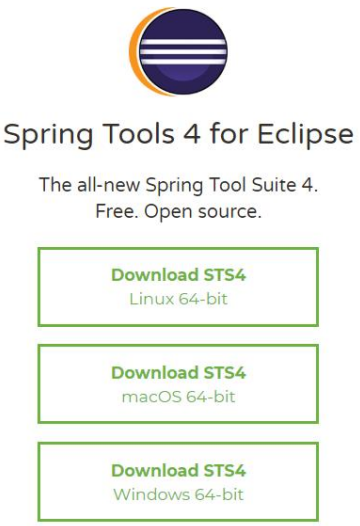
Untuk IDE (*Integrated Development Environtment*), bisa menggunakan IntelliJ IDEA. Menurut penulis, IDE ini merupakan salah satu IDE yang paling baik untuk membuat program khususnya berbasis Java. Karena sangat cepat, ringan, memiliki tools-tools yang sangat digunakan oleh programmer Java tetapi menggunakan memory yang sangat banyak. Terdapat dua versi, yaitu versi Commercial dan versi Community. Kalian bisa langsung mengunduh disitus resmi nya https://www.jetbrains.com/idea/download/ .

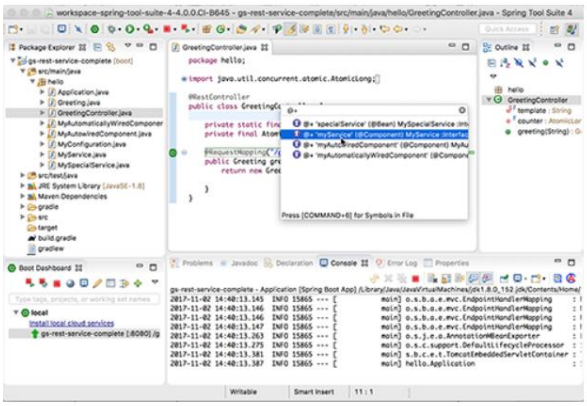


5. Spring Tool Suite

Jika kalian terbiasa menggunakan IDE Eclipse dalam membuat program Java. Saya sarankan untuk menggunakan IDE ini. STS atau kepanjangan dari *Spring Tool Suite* adalah sebuah Eclipse IDE yang di modifikasi atau di desain khusus untuk framework Spring.

Untuk lisensi, IDE ini adalah *Opensource* jadi kita tidak perlu khawatir mengenai lisensi dan bisa dipakai sampai kapanpun tanpa ada batasan waktu. Untuk mengunduhna dapat mengunjungi laman resmi nya di https://spring.io/tools .

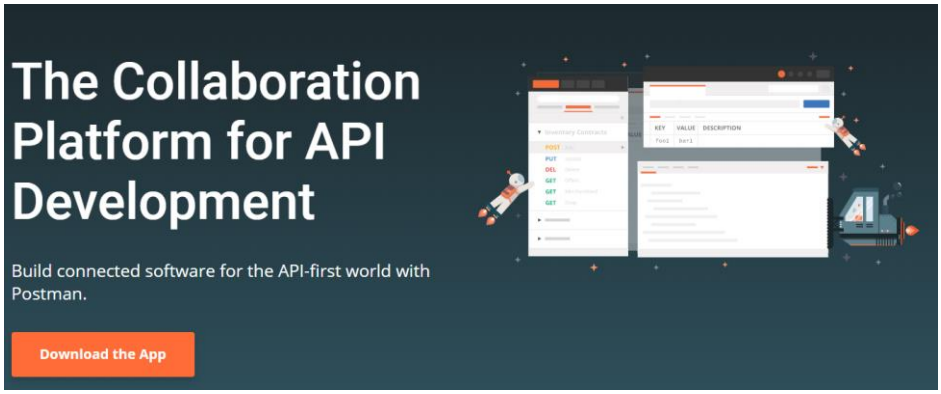




6. Postman REST Api Client

Postman adalah sebuah perangkat lunak atau *tools* yang digunakan untuk melakukan uji coba RestAPI dari aplikasi yang sedang kita buat. Karena di bab selanjutnya postman akan digunakan ketika membangun webservice berbasis Rest API.

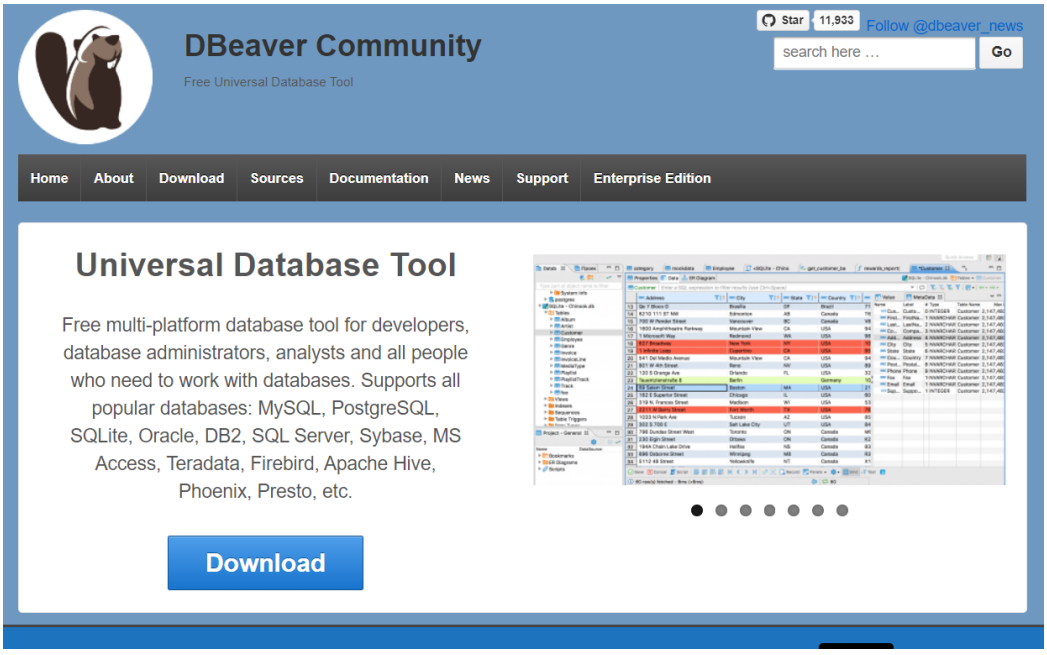
Postman dapat kalian unduh dalam laman resmi nya disini https://www.postman.com/postman.



7. DBeaver

DBeaver adalah salah satu *SQL Client* yang bisa digunakan untuk manajemen database dari aplikasi yang sedang kita bangun. Mendukung multiplatform database yaitu MySQL, Postgresql, Oracle, SQLServer, DB2, SyBase, MS Access, Teradata, Firebird, Apache Hive dan lain-lain.

Software ini bersifat *open source* dan kalian bisa mengunduh langsung dilaman resmi nya yaitu https://dbeaver.io/download/.



Konsep *Dependency Injection*

Jika kita berbicara mengenai *Spring Framework* maka tidak akan lepas dengan apa yang Namanya *Dependency Injection* atau yang disebut juga *Inversion of Control.* Dijelaskan bahwa *Dependency Injection* itu adalah :

> *IoC is also know as dependency injection (DI). It is a process whereby object define their dependencies, that is, the other objects they work with, only through constructor arguments, arguments to a factory method, or properties that are set on the object instance after it is contructed or returned from a factory method. The container then injects those dependencies when it creates the bean. This is process is fundamentally the inverse, hence the name Inverstion of Control (IOC), of the bean itself controlling the instantiation or location of its dependencies by using direct contructor of classes, or a mechanism such as the Service Locator pattern.*

Dalam setiap aplikasi java pasti berisi sekumpulan *class-class* Java, atau yang disebut juga dengan *bean.* Dari sinilah muncul istilah *Java Bean*. Idealnya sebuah bean itu bersifat mandiri dan tidak terikat pada bean yang lain (*depend*) atau yang disebut dengan *loose coupling* sehingga pada *Spring* sangatlah menekankan akan hal itu dan kebalikan dari *loose coupling* itu adalah *tight coupling* dimana suatu kondisi *bean* yang mempunyai ketergantuangan kuat pada *bean* yang lain.

```
class Trip {
   Car c = new Car();
   void startTrip() {
       c.go();
   }
}
```

```
class Car {
    void go () {
        // logic …
    }
}
```

Jika kita lihat pada *bean* atau *class* diatas class Trip memiliki ketergantungan kepada class Car dengan adanya deklarasi Car c = new Car() . Hal ini lah yang ingin dihindari oleh *Spring*. Sehingga jika kita ingin mengubah kode diatas menjadi *loose coupling* maka kita edit menjadi sebagai berikut .

1. Buatlah sebuah interface karena interface sangat lah mendukung penggunaan sifat *loose coupling* karena hanya berisi defisni dan tidak ada bisnis logic.

   ```
   interface Vehicle () {
       void go ();
   }
   ```

2. Buat class logic dengan mengimplementasi interface *Vehicle* diatas

   ```
   class Car implements Vehicle {
   ```

```
    @Override
    public void go() {
       System.out.println("Go with car")
    }
}


class Bike implements Vehicle {

    @Override
    pulic void go() {
        System.out.println("Go with Bike");
    }
}
```

3. Edit class Trip yang memiliki sifat *tight coupling* dengan mendeklarasikan interface Vechile dan bukan dari class implementasinya. Dan buatlah setter untuk menginisiasi interface tersebut.

```
class Trip {

   Vehicle vehicle;

   public void setVehicle(Vehicle v) {
       this.v = v;
   }

   void startTrip() {
        v.go();
   }

}
```

Pada method setVehicle diatas itu kita menempatkan atau menginject class Car atau Bike nya sehingga program diatas sudah bisa dikatakan sebagai potongan program yang *loose coupling*.

Pengenalan Spring Framework

Java Enterprise Edition

Sebelum ada Spring, sebenarnya ada tools untuk pemrograman enterprise yaitu Java Enterprise atau JEE pada tahun 2000. Namun, ketika membuat aplikasi menggunakan J2EE sangatlah tidak mudah dan sangat rumit. J2EE sangat *tightly coopling* sehingga tidak memungkinkan untuk melakukan unit testing dengan koneksi eksternal misalnya koneksi ke database. Bahkan, untuk menguji fitur sederhana sekalipun harus menggunakan / mengerahkan seluruh aplikasi dalam sebuah container. Karena hal seperti itu lah J2EE mulai banyak ditinggalkan oleh programmer Java khususnya programmer Java di era tahun 2000 an.

Spring Framework

Spring dengan *Dependency Injection* nya bisa menjawab permasalahan programmer-programmer khususnya yang sudah lama berkecimpung menggunakan *Java Enterprise Edition* sehingga dengan itu bisa melakukan unit testing dan antar *bean* menjadi modular dan tidak ada keterkaitan dengan *bean* yang lainnya.
Jika kita melihat dokumentasi nya secara langsung, Spring Framework adalah sebuah platform Java yang sangat handal yang menyediakan arsitektur yang komprefensif yang mendukung sepenuhnya untuk membangun aplikasi menggunakan Java Enterprise.
Pertama kali ditulis oleh Rod Jhonson dengan lisensi dari Apache 2.0 pada bulan April 2003. Spring seakan menjadi semacam standar wajib untuk programmer Programmer Java dalam sebuah perusahaan.
Spring Framework menggunakan teknik programan yang sederhana, didukung dengan komunitas yang besar dan dokumentasi yang lengkap, banyak programmer java yang pindah menggunakan Spring Framework. Akhirnya pemrogram bisa membuat aplikasi Enterprise atau Big Data sekalipun dalam waktu yang relative cepat.

Adapun Kelebihan Spring Framework :

a. Pengembangan Cepat
   Spring Framework menawarkan kecepatan dalam membangun aplikasi menggunakan Java, karena sudah *built-in* dengan tomcat, sehingga pemrogram bisa langsung mendebug pada laptop nya tanpa perlu deploy terlebih dahulu ke server.

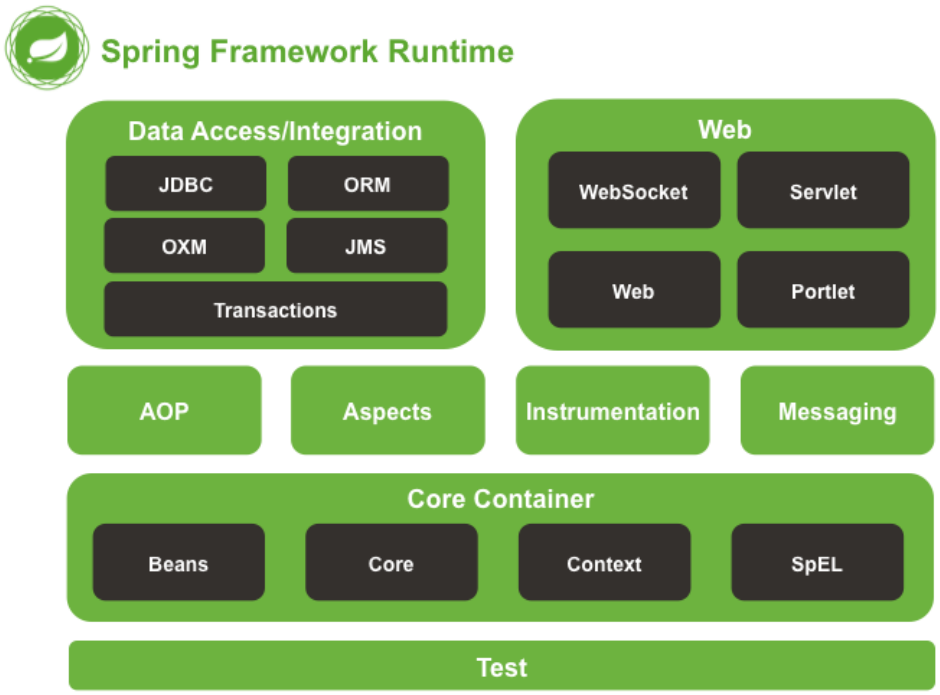
b. Dokumentasi yang lengkap
   Dari Versi awal sampai sekarang, strutur project Spring tidak mengalami perubahaan secara besar-besaran tapi ada beberapa code baru di setiap upate nya.
   Dalam situs nya juga, Spring mempunyai dokumentasi yang lengkap dan komprehensif. Hal ini sangat bermanfaat bagi pemrogram karena menurut saya baik / buruk nya suatu framework bisa dilihat dari kualitas Dokumentasinya.
   Kalian bisa membaca nya, dalam dokumentasi nya diterangkan juga secara fundamental dan secara praktikal.

c. Dukungan komunitas yang Besar
   Di Indonesia dan dunia Internasional juga sudah banyak grup yang membahas tentang Spring Framework ini. Dengan adanya komunitas ini, programmer tidak sendirian ketika mengalami kendala dalam menggunakan Spring sehingga programmer bisa bertukar pikiran, berdiskusi atau bahkan bisa melakukan kontribusi dalam projek Spring sendiri.

d. Menjadi standar perushaan Enterprise
   Pada saat ini, seakan sudah menjadi standar bagi perusahaan-perusahaan Enterprise untuk mencari programmer Java. Salah satunya harus bisa menggunakan Spring Framework.
   Mulai dari segmen industry Perbankan, Asuransi, telekomunikasi dan startup khususnya dalam bidang financial technology.

Spring Modules

Spring mempunyai banyak fitur, semua nya itu di bungkus dalam 20 module yang digroup kan dalam diagram dibawah ini.



- *Core Container*
  Dalam layer *Core Container* ini terbagi dua, yaitu :
  a. Core and Beans

Sesuai dengan namanya, ini adalah inti dari Spring Framework sendiri. Modul ini menyediakan IoC atau DI */ Dependency Injection* dan menyediakan implementasi factory pattern yang canggih menggunakan *BeanFactory.*

b. Context
Berisi dukungan untuk beberapa fitur Java EE seperti EJB (*Enterprise Java Beans*), JMX (*Java Management Extension*) dan dukungan layanan *remote*.

- *Data Access / Integration*

Pada Layer ini , khusus untuk dapat berkomunikasi dengan database, integration system dan modul transaksi.

1. JDBC
Atau yang disebut dengan *Java Database Connectivity* adalah sebuah API (*Application Programming Interface*) yang memungkinkan program java dapat berkomunikasi dengan Relasional Database.

2. ORM
Object Relational Mapping adalah sebuah teknik pemrograman yang memetakan sebuah object dengan database sehingga dapat berkomunikasi menggunakan paradigm Object Oriented Programming.
Banyak sekali produk-produk ORM, namun yang sering banyak dipakai sekarang yaitu JPA, Hibernate dan MyBatis. Dari setiap produk tersebut mempunyai keunggulan masing-masing tergantung dengan kebutuhan proses bisnis yang akan dibuat nantinya.

3. OXM
Menyediakan layer abstraksi untuk menggunakan sebuah nilai dari implementasi Object/XML seperti JAXB, Castor, XMLBeans, JiBX, XStream.

4. JMS
Menyediakan layanan untuk mendukung pengiriman pesan (*messaging)* menggunakan teknologi JMS (*Java Messaging Service*). Dengan menggunakan JMS seorang programmer dapat melakukan komunikasi pesan dari MQSeries IBM, SonicMQ dan beberapa produk *messaging* lainnya. Selain itu, JMS mendukung pesan yang berisi objek Java dan pesan yang berisi halaman *Extensible Markup Language (XML)*.

5. Transaction
Menyediakan abstraksi yang konsisten untuk manajemen transaksi tidak hanya untuk *class* yang mengimplementasi special *interface* tapi untuk semua POJOs (*plain old Java Object*).

- *Web*

Dalam layer *web* ini terdiri dari tiga bagian yang terpisah yaitu, *Spring Web, Web Servlet* dan *Web Portlet* modul.
Spring Web, modul ini menyediakan keperluan dan fungsionalitas dasar dari *web* seperti upload data modul ini juga berelasi dengan dukungan layanan jarak jauh Spring (*Spring remoting support*).

Web Servlet, jika kalian sudah menggunakan pola MVC (*Model View Controller*) dalam sebuah framework maka hal ini juga di dukung oleh *Web Servlet*. *Web Servlet* mendukung pola MVC sehingga menyediakan sebuah pemisah yang sangat bersih antara kode untuk Tampilan (VIew), logic dari aplikasi (Controller) dan layer yang berhubungan dengan database (*Model*).

- *AOP*

  Adalah singkatan dari *Aspect Oriented Programming*. AOP lebih menekankah pada *cross cutting concern* yaitu seperti fitur *authentication, logging, security* dan lain-lain. Untuk AOP akan kita bahas lebih lanjut pada bab tersendiri.

- *Test*

  Modul ini mendukung bahwa semua *Spring Component* bisa menggunakan JUnit atau TestNG.

Spring Projects

Disamping Spring Modules, Spring juga mengembangkan beberapa projek yang dapat membantu programmer dalam membuat aplikasi *enterprise* . Salah satu projek nya yaitu Spring Boot yang akan kita gunakan dalam buku ini .

- Spring Cloud

  Spring Cloud adalah sebuah framework aplikasi *cloud* yang sangat handal. Framework ini menyediakan apa saja yang dibutuhkan untuk membuat aplikasi dalam lingkungan terdistribusi (*distributed environtment*).

  Untuk buku versi ke-2 mengenai Microservices, akan banyak menggunakan fitur *Spring Cloud* ini.

- Spring Data

  Misi dari Spring Data adalah menyediakan model pemrograman yang berbasis Spring yang konsisten tapi tetap mempertahankan sifat-sifat khusus dari manipulasi data.

  Memberikan kemudahan dalam mengatur data yang bersifat *relational* dan *non-relational.* Projek Spring Data ini dikembangkan dengan bekerja sama dengan beberapa perusahaan dan para pengembang.

- Spring Integration

  Turunan dari model pemrograman Spring atau yang lebih dikenal dengan *Enterprise Integration Pattern.* Misi projek ini adalah menyediakan model yang sederhana untuk mengintegrasikan solusi *enterprise* sambil mempertahankan *separation of concern* untuk mengahasilkan kode yang dapat dipertahankan dan dapat diuji.

- Spring Batch

  Batch Processing adalah sebuah proses yang melibatkan banyak *jobs* tanpa interaksi dengan manusia. Sebuah *batch* dapat menangani data dalam jumlah besar dan *running* dalam waktu yang lama. Spring Batch membantu untuk menangani *batch processing* tersebut.

- Spring Security
  Sesuai dengan nama nya, projek ini digunakan biasanya untuk menangani *Authentication* (siapa yang akan masuk ke sistem) dan *Authorization* (menu /fitur apa saja yang bisa diakses oleh seseorang) .

- Spring Rest Docs
  JIka Anda sudah bekerja menggunakan REST Api, Spring menyediakan REST Client khusus menggunakan Swagger. Fungsi nya sama seperti Postman yaitu untuk mengetest aplikasi REST yang sudah dibuat.

- Spring AMQP
  Menyediakan *core concept* untuk development berbasis AMQP *Messaging*. Projek ini terbagi menjadi dua abstraksi yaitu spring-amqp sebagai dasar abstraksi dan spring-rabbit sebagai implementasi.

- Spring Mobile
  Framework yang memberikan solusi untuk mendeteksi *device* yang melakukan request ke server.

- Spring Webservice
  Adalah salah satu produk dari *Spring Community* yang focus untuk membuat layanan web berbasis *document-driven*.

- Spring LDAP
  Projek Spring yang memberikan kemudahan dalam management LDAP (*Lighweight Directory Access Protocol*) yang salah satu nya terkait dengan *Active Directory.*

- Spring Shell
  Projek Spring yang memberikan kemudahan programmer dalam mengembangkan aplikasi berbasis Spring dengan *command line* yang dapat berkomunikasi langsung dengan REST API atau bagaimana berinteraksi dengan konten file local.

- Spring Flo
  Spring Flo adalah sebuah library yang ditulis dalam bahasa Javascript yang meng-*embedd* HTML5 Visual Builder untuk *pipeline* dan graph yang sederhana.

- Spring Kafka
  Jika anda pernah menggunakan product *messaging* maka tidak asing lagi dengan kafka. Dengna projek ini spring memberikan komudahan dalam berkomunikasi dengan kafka. Lebih tepat nya menyediakan semacam *template* sebagai *high level abstraction* untuk mengirim pesan.

Semua project diatas adalah *main project* dari Spring Framework tapi dalam buku ini, kita tidak akan membahas semua. Kita focus hanya membahas Spring Framework, Spring Boot, Spring Data JPA, Spring Web dan Spring Security. Untuk lebih detail nya kalian bisa mempelajarinya langsung dalam situs ini https://spring.io/projects/.

Pengenalan Spring Boot

Sejak Spring Framework diperkenalkan untuk menjawab permasalahan-permasalahan umum dari Java Enterprise Edition maka dengan itu perkembangan dan popularitas Spring Framework sangatlah cepat. Banyak aplikasi yang dibangun menggunakan fondasi Spring khususnya di lingkungan *enterprise*. Karena konfigurasi Spring banyak menggunakan XML maka hal itu agak menjadi sebuah masalah karena pada jaman sekarang Spring harus dapat bersaing dengan framework-framework yang mengedepankan kecepatan dalam proses *development* seperti Rails, Laravel atau Django. Untuk menjawab tantangan itu maka di perkenalkan lah Spring Boot V1.0 pada tahun 2014.

Spring Boot didesain untuk tujuan sebagai berikut :

- Mencegah konfigurasi berbasis XML yang kompleks
- Memberikan kemudahan kepada programmer untuk membuat program
- Dapat berjalan secara *independent*

Kenapa harus menggunakan Spring Boot ?

- Sudah menyediakan fleksibilitas untuk mengatur *Java Beans*, Konfigurasi dan *Database Transaction*
- Mendukung *batch processing* dan mendukung REST endpoint (untuk aplikasi berbasis webservice API).
- Dalam Spring Boot, semua sudah di *auto configuration* sehingga tidak perlu melakukan manual konfigurasi
- Sudah mendukung Annotation Based Konfigurasi menggunakan annotasi @SpringBootApplication
- Karena sudah terintegrasi dengan Maven, maka dapat dengan mudah mengatur plugin / dependency aplikasi.
- *Embedded Servlet Container* sudah tertanam.

Java Based Configurations

Spring Boot banyak menggunakan *Java Based Annotations* untuk konfigurasi nya untuk menggantikan pendahulunya yang masih menggunakan *XML Based*. Berikut annotasi-annotasi berbasis Java yang sering digunakan dalam Spring Boot:

- @Autowired
- @Service
- @Component
- @Bean
- @Configuration

XML Configuration VS Java Configuration

Berikut ini akan kita bandingkan penggunakan konfigurasi menggunakan XML dan Java Based. Ambil contoh kasus akan membuat *bean* dari sebuah class.

XML Config

```
<beans>
   <bean id = "helloWorld" class = "id.belajar.HelloWorld" />
</beans>
```

Java Config

```
@Configuration
public class HelloWorldConfig {

   @Bean
   public HelloWorld helloWorld(){
      return new HelloWorld();
   }
}
```

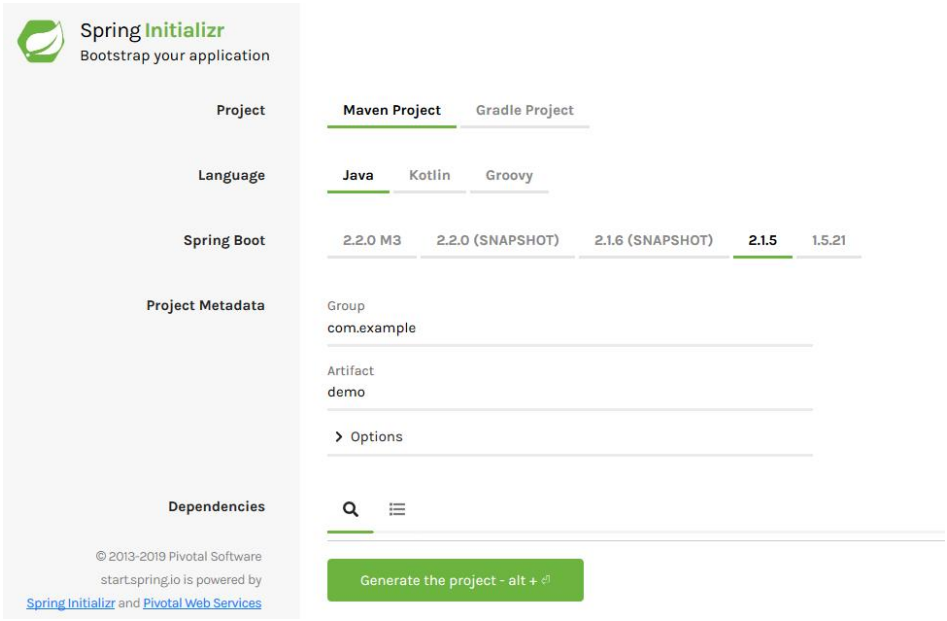
Anda bisa bandingkan kode diatas, lebih nyaman menggunakan XML Config atau Java Config.

Membuat Projek di Spring Boot

Spring Initialzr adalah sebuah webtool yang disediakan oleh Spring untuk membantu programmer dalam membuat projek berbasis spring secara cepat. Didalamnya, kalian juga bisa memilih plugin apa saja yang akan digunakan nantinya. Bisa diakses langsung ke sini https://start.spring.io/ .

Spring initialzr juga dapat digunakan dalam beberapa cara yaitu :

- Membuat projek langsung di dalam IDE nya yaitu di Spring tool Suite atau Intelljidea
- Atau menggunakan Spring Boot CLI



Project, pilih build tools mana yang akan digunakan antara Maven atau Gradle. Karena dalam buku ini menggunakan maven sebagai build tools nya maka pilihlah Maven Project.

Language, pilih bahasa pemrograman apa yang akan digunakan pilihannya yaitu Java, Kotlin atau Groovy.

Spring Boot, pilih versi Spring Boot sebagai base versi framework yang akan dibuatkan aplikasi nantinya.

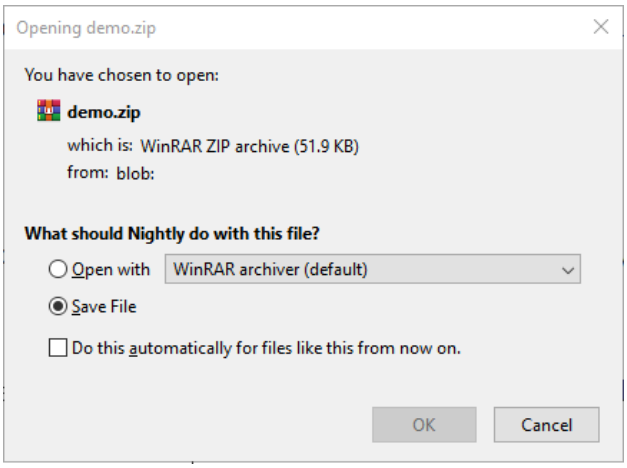
Project Metadata, simpan informasi projek khususnya tentang *Group* dan *Artifact* dari aplikasi.

Dependencies, kalian bisa memilih plugin / dependencies yang diperlukan. Misalnya Spring Web, Spring Data JPA, Spring Security dan lain-lain.

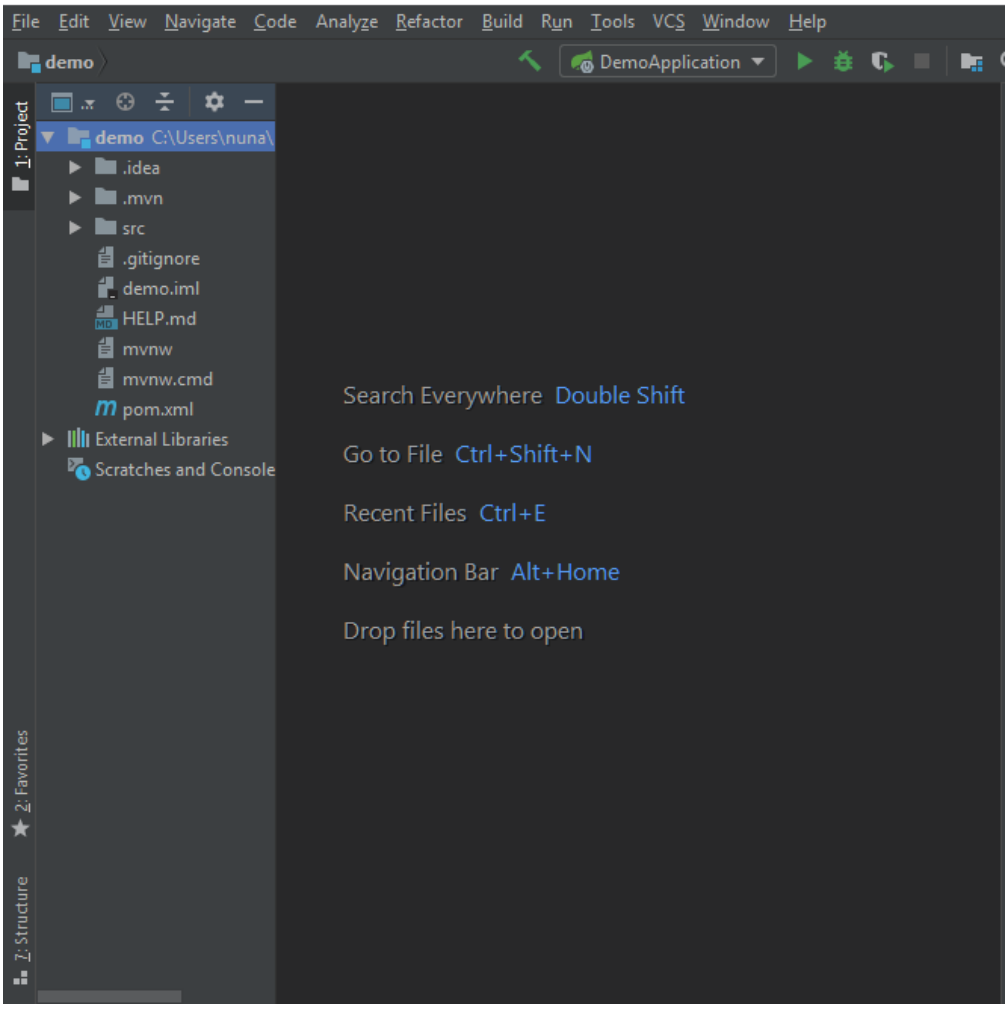
Options, terdapat beberapa informasi dalam menu ini diantaranya :

- **Name** : sama dengan artifact
- **Description** : adalah deskripsi singkat mengenai aplikasi yang akan dibuat
- **Packaging** : Pilih format output dari aplikasi jika berhasil di build yaitu **Jar** atau **WAR**.
- **Java** : kita dapat memilih versi Java , bisa menggunakan versi 8, 11 atau 13.

Setelah selesai kemudian klik button ‘ Generate the project‘. Dan web akan secara otomatis mengunduh projek yang telah dibuat tadi ke dalam bentuk file .zip.



Ekstrak projek tersebut ke dalam suatu folder kemudian buka IDE (IntelijIDEA atau Spring Tool Suite), namun saya menggunakan Intelijidea pada buku ini. Biasanya maven akan terlebih dahulu mengunduh dependency-dependency yang telah dipilih tadi jika di laptop belum terdapat dendency nya. jadi pastikan laptop kalian sudah terhubung dengan jaringan internet.



Spring Data JPA

Sebelumnya kita bahas apa itu *JPA, Hibernate* karena kadang-kadang kita suka kebingungan mengenai dua hal ini. Penulis khususnya, sebelum mengenal JPA, untuk membuat aplikasi yang bisa berkomunikasi dengan database perlu banyak hal yang harus dilakukan diantaranya membuat koneksi database, membuat *try-catch* manual untuk masing-masing method yang mengoperasian *CRUD (Create, Read, Update, Delete)* kemudian membuat sintak-sintak SQL secara native dalam class secara manual dengan menggunakan *Statement* atau *PreparedStatement* seakan harus membuat semua nya itu dari nol.

JPA atau singkatan *Java Persistence API* adalah sebuah spesifikasi standar bagaimana Java yang notabene nya adalah *Object Oriented Programming* dapat berkomunikasi dengan Database khususnya RDBMS (*Relational Database Management System*) yang tidak ada kaitannya dengan OOP sehingga menggunakan JPA kita tidak perlu lagi menyiapkan konfigurasi ke database dari awal seperti di atas dan kita bisa focus hanya ke dalam *class-class* yang akan dipetakan ke dalam table-table yang ada di database karena JPA sudah mengimplementasikan teknologi ORM (*Obejct Relational Mapping*).

Hibernate adalah sebuah framework salah satu implementasi atau produk dari JPA itu sendiri. Dibuat oleh Gavin King pada tahun 2002 sebagai sebuah alternative penggunaan *entity beans* pada layer *persistence*. Selain *Hibernate* ada banyak vendor yang telah mengimplementasikan JPA diantaranya :

- Toplink
- iBatis
- OpenJPA
- Spring Data JPA
- Dan lain-lain

Pengenalan Spring Data JPA

Spring data JPA adalah salah satu yang mengimplementasikan JPA juga dan yang di custom sehingga operasi-operasi standar seperti Create Read Update Delete sudah disediakan. Spring data JPA menggunakan interface *Repository* yang mana menggunakan ID dari *class* domain sebagai parameter nya. Disamping itu Spring data JPA mendukung *query method, native query, JPQL* dan lain sebagainya.

Membuat Projek Spring Data JPA

Akses ke https://start.spring.io/ dan buat projek Spring Boot baru dengan spesifikasi di bawah ini :

Project	**Maven Project**
Versi Spring Boot	2.1.12
Group	id.belajar
Artifact	spring-data-jpa
Dependencies	Spring Data JPA
	MySQL Driver

Jika kita lihat di file pom.xml, terlihat kita menggunakan dependencies

pom.xml

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

Dengan plugin ini, kita tidak perlu menambahkan lagi *hibernate-core, spring-orm* ke dalam projek secara manual. Semua nya sudah di bundle dalam plugin ini.

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<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
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
       <groupId>org.springframework.boot</groupId>
       <artifactId>spring-boot-starter-parent</artifactId>
       <version>2.2.4.RELEASE</version>
       <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>id.belajar</groupId>
    <artifactId>spring-data-jpa</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>spring-data-jpa</name>
    <description>Demo project for Spring Boot</description>

    <properties>
       <java.version>1.8</java.version>
    </properties>

    <dependencies>
       <dependency>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-starter-data-jpa</artifactId>
       </dependency>
       <dependency>
          <groupId>com.zaxxer</groupId>
          <artifactId>HikariCP</artifactId>
          <version>3.1.0</version>
       </dependency>
       <dependency>
          <groupId>mysql</groupId>
          <artifactId>mysql-connector-java</artifactId>
          <scope>runtime</scope>
       </dependency>
       <dependency>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-starter-test</artifactId>
          <scope>test</scope>
          <exclusions>
             <exclusion>
                <groupId>org.junit.vintage</groupId>
                <artifactId>junit-vintage-engine</artifactId>
             </exclusion>
```

```
            </exclusions>
        </dependency>
    </dependencies>

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

</project>
```

Membuat Entity Class

Spring JPA bekerja bekerja dengan class yang merepresentasikan entitas dalam sebuah system. Sebagai demo, buatkan sebuah class Entitas Book sebagai Berikut.

id.belajar.springdatajpa.entity.Book

```
@Entity
public class Book {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String title;

    @Column(nullable = false)
    private String writer;

    @Column(nullable = false)
    private String isbn;

    public Book() {
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }
```

```
    public String getWriter() {
        return writer;
    }

    public void setWriter(String writer) {
        this.writer = writer;
    }

    public String getIsbn() {
        return isbn;
    }

    public void setIsbn(String isbn) {
        this.isbn = isbn;
    }
}
```

Penjelasan dari class Book diatas :

@Entity : merepresentasikan bahwa class tersebut adalah sebuah entitas, yang nantinya akan ada sebuah Tabel di database dengan nama Book

@Id : menunjukan bahwa variable itu adalah sebuah ID atau *Primary Key* dari suatu table. Penggunaan ***@Id*** di Spring Data JPA sangat lah mandatory /wajib. Setiap class yang diberi annotasi @Entity harus mempunyai @Id.

@GeneratedValue : sebuah mekanisme dalam memberikan nilai. Karena disini menggunakan `strategy = GenerationType.IDENTITY` yang artinya nilai akan diberikan secara increment atau fungsi nya sama dengan AUTO_INCREMENT pada sebuah table.

`@Column(nullable = false)` : Memberikan info bahwa variable tersebut yang akan menjadi kolom di table Book, tidak boleh null atau *not null.*

Membuat JPA Properties dan Hikari Connection Pool

Sebenarnya Spring Boot versi 2 sudah secara default menggunakan HikariCP sebagai *connection pool* nya sebelumnya menggunakan *tomcat pool*. HIkariCP sangatlah cepat, ringan, dapat diandalkan dalam fase production.

Buka file *application.properties* di dalam path **src/main/resources** kemudian tambahkan beberapa konfigurasi dibawah ini dan pastikan Anda sudah membuat database dengan nama **spring-data-jpa.**

src.main.resources.application.properties

```
#MySQL Connection
spring.datasource.url = jdbc:mysql://localhost:3306/spring-data-jpa
spring.datasource.username = root
spring.datasource.password = root

#HikariCP
spring.datasource.hikari.connection-timeout=20000
spring.datasource.hikari.minimum-idle=5
spring.datasource.hikari.maximum-pool-size=12
spring.datasource.hikari.idle-timeout=300000
```

```
spring.datasource.hikari.max-lifetime=120000
spring.datasource.hikari.auto-commit=true
```

```
#JPA Properties
spring.jpa.database-platform=org.hibernate.dialect.MySQL5InnoDBDialect
spring.jpa.hibernate.ddl-auto=update
spring.datasource.driverClassName=com.mysql.jdbc.Driver
```

Penjelasan dari potongan kode diatas :

spring.datasource.url : adalah alamat dari database MySQL yang sudah diinstall. Default nya yaitu localhost:3306

spring.datasource.username : adalah username untuk dapat mengakses database yang dalam hal ini kita menggunakan username sebagai *root*.

spring.datasource.password : adalah password yang digunakan untuk mengakses database dengan username diatas.

Tidak disarankan untuk menggunakan username *root* dan password *root* jika aplikasi kalian sudah release

spring.datasource.hikari.connection-timeout=20000 : adalah nilai maksimum dalam hitungan mili detik yang dibutuhkan client menunggu untuk sebuah *connection* dari *connection pool*.

spring.datasource.hikari.minimum-idle=5 : nilai atau jumlah dari koneksi *idle*
spring.datasource.hikari.maximum-pool-size=12 : adalah jumlah maksimum dari ukuran *connection pool*

spring.datasource.hikari.idle-timeout=300000 : jumlah maksimum dalam hitungan mili detik yang mengijinkan koneksi berada dalam status idle dalam *connection pool*

spring.datasource.hikari.max-lifetime=120000 : adalah maksimum waktu hidup (*life time*) dalam hitungan mili detik dari sebuah koneksi dalam *connection pool* setelah keluar.

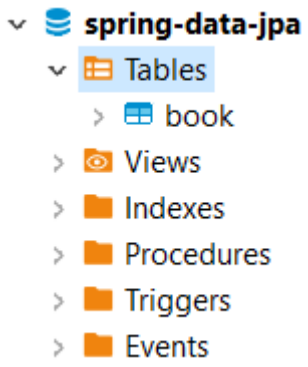
spring.datasource.hikari.auto-commit=true : mengatur *setting* auto-commit dengan nilai nya *true*

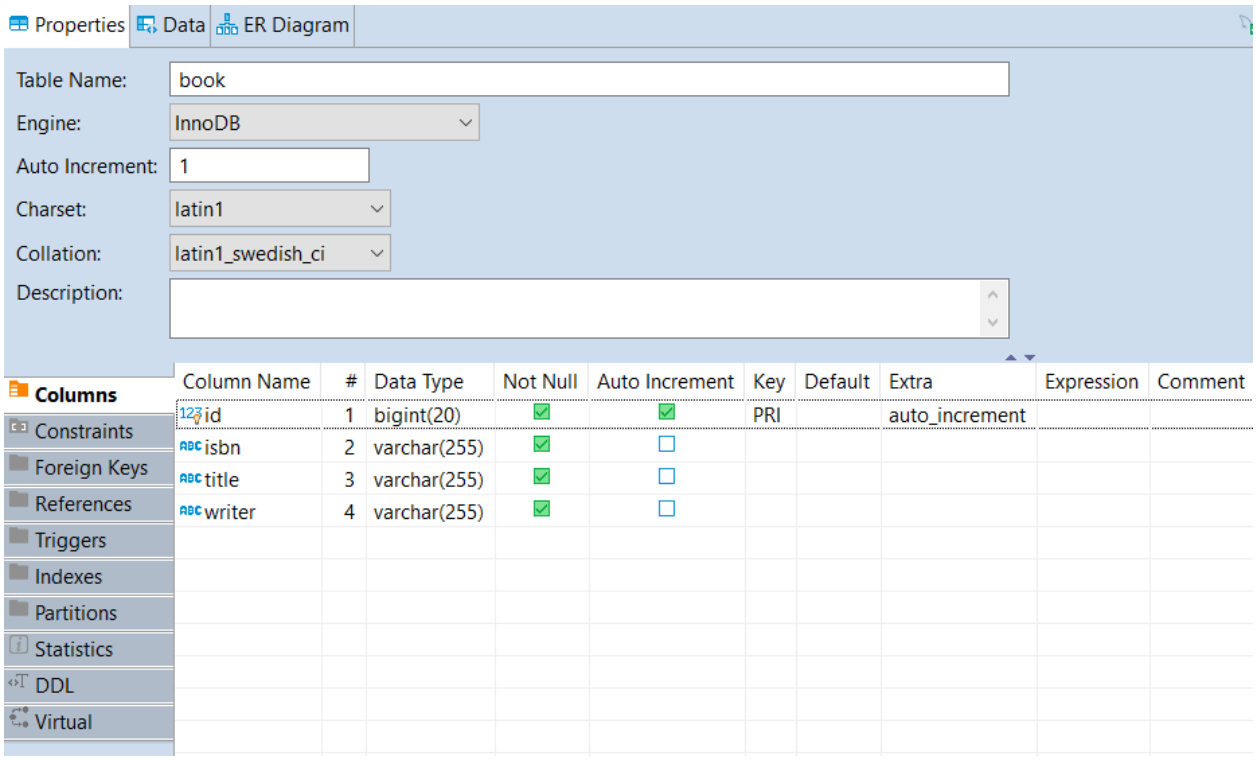
spring.jpa.database-platform : dialek yang digunakan nantinya oleh JPA. Disini kita mengunakan dialek MySQL dengan tipe InnoDB.

spring.jpa.hibernate.ddl-auto : property ini bertugas untuk membuatkan table-tabel didatabase tergantun dari value nya. Disini kita menggunakan value *update* yang artinya Spring Data JPA akan secara otomatis membuatkan table Book di database.

`spring.datasource.driverClassName` : sesuai dengan nama nya, property ini digunakan untuk menentukan driver yang digunakan sebagai jembatan komunikasi dengan database.

Jalankan program ini jika tidak terdapat error, kita bisa lihat di DBeaver pada database spring-data-jpa akan terdapat Book table.





Kita bisa lihat apa yang kita sudah deklarasikan di class Book maka akan di representasikan menjadi sebuah table. Untuk kolom Id menjadi primary key dengan nilai auto_increment. Dan untuk kolom isbn, title, writer mempunyai atribut Not Null.

Eksekusi *Query Method*

Untuk dapat menggunakan fitur *query method* bawaan Spring Data JPA maka kita harus membuat Layer Repository untuk class Book.

`id.belajar.springdatajpa.repository.BookRepository`

```
import id.belajar.springdatajpa.entity.Book;
import org.springframework.data.jpa.repository.JpaRepository;

public interface BookRepository extends JpaRepository<Book, Long> {

}
```

Dari kode diatas dapat kita lihat bahwa BookRepository adalah turunan JPARepository dimana JPARepository menyediakan fungsi untuk *create, retrieve, update, delete, findAll* dan lain-lain sehingga kita tidak perlu lagi untuk membuat fungsi-fungsi CRUD dari awal. Supaya lebih paham kita akan coba menyimpan dua buah object Book ke dalam database.

id.belajar.springdatajpa.SpringDataJpaApplication

```
@SpringBootApplication
public class SpringDataJpaApplication implements CommandLineRunner {

   private final Logger LOG =
LoggerFactory.getLogger(SpringDataJpaApplication.class);

   @Autowired
   private BookRepository bookRepository;

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

   @Override
   public void run(String... args) throws Exception {

      Book book1 = new Book();
      book1.setTitle("Belajar Spring Boot");
      book1.setWriter("Teten Nugraha");
      book1.setIsbn("IS-90908");

      Book book2 = new Book();
      book2.setTitle("Belajar Spring Boot 2");
      book2.setWriter("Teten Nugraha");
      book2.setIsbn("IS-9090890");

      bookRepository.save(book1);
      bookRepository.save(book2);

      LOG.info("Berhasil menyimpan "+book1);
      LOG.info("Berhasil menyimpan "+book2);

   }
}
```

Jika kita jalankan kode diatas, maka dalam console LOG aka nada keterangan mengenai dua object Book yang sudah kita simpan ke database.

```
INFO 8976 --- [           main] i.b.s.SpringDataJpaApplication           : Berhasil menyimpan Book{id=1, title='Belajar Spring Boot', writer='Teten Nugraha', isbn='IS-90908'}
INFO 8976 --- [           main] i.b.s.SpringDataJpaApplication           : Berhasil menyimpan Book{id=2, title='Belajar Spring Boot 2', writer='Teten Nugraha', isbn='IS-9090890'}
```

Kemudian kita cek lagi di database, terdapat dua nilai dari table Book.

id	isbn	title	writer
1	IS-90908	Belajar Spring Boot	Teten Nugraha
2	IS-9090890	Belajar Spring Boot 2	Teten Nugraha

Disamping method save kita akan coba untuk me*retrieve* data. Edit kembali file BookRepository dan tambahkan query methodnya sebagai berikut.

id.belajar.springdatajpa.repository.BookRepository

```
List<Book> findAll();
```

Kita ubah kembali class SpringDataJpaApplication dan panggil method findAll diatas.

id.belajar.springdatajpa.SpringDataJpaApplication

```
@SpringBootApplication
public class SpringDataJpaApplication implements CommandLineRunner {

   private final Logger LOG =
LoggerFactory.getLogger(SpringDataJpaApplication.class);

   @Autowired
   private BookRepository bookRepository;

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

   @Override
   public void run(String... args) throws Exception {

      List<Book> books = bookRepository.findAll();

      LOG.info("Books : "+books);

   }
}
```

Kita jalankan kode nya, terlihat di consle log terdapat dua buah nilai dari hasil method *findAll.*

```
: Books : [Book{id=1, title='Belajar Spring Boot', writer='Teten Nugraha', isbn='IS-90908'}, Book{id=2, title='Belajar Spring Boot 2', writer='Teten Nugraha', isbn='IS-9090890'}]
```

Disamping findAll, kita juga bisa membuat query misalnya untuk mendapatkan data book berdasarkan penulis atau *writer.*

id.belajar.springdatajpa.repository.BookRepository

```
List<Book> findAllByWriter(String writer);
```

id.belajar.springdatajpa.SpringDataJpaApplication

```
@Override
public void run(String... args) throws Exception {

    final String writer = "Teten Nugraha";

    List<Book> books = bookRepository.findAllByWriter(writer);

    LOG.info("Books : "+books);

}
```

Kita jalankan kembali dan lihat log nya.

```
: Books : [Book{id=1, title='Belajar Spring Boot', writer='Teten Nugraha', isbn='IS-90908'}, Book{id=2, title='Belajar Spring Boot 2', writer='Teten Nugraha', isbn='IS-9090890'}]
```

Terakhir, misalkan kita ingin melakukan pencarian data Book berdasarkan nomor ISBN. Nomor ISBN itu unik dan pasti akan ada satu data Book dengan satu nomor ISBN, oleh karenanya kita buat method seperti ini.

id.belajar.springdatajpa.repository.BookRepository

```
Book findByIsbn(String isbn);
```

id.belajar.springdatajpa.SpringDataJpaApplication

```
@Override
public void run(String... args) throws Exception {

    final String isbn = "IS-90908";

    Book book = bookRepository.findByIsbn(isbn);

    LOG.info("Book : "+book);

}
```

Jalakan, kita cek kembali di log nya

```
2020-02-12 14:05:20.568  WARN 30352 --- [           main] com.zaxxer.hikari.util.DriverDataSource  : Registered driver with driverClassName=com.mysql.jdbc.Driver was not found, trying direct ins
2020-02-12 14:05:21.581  INFO 30352 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
2020-02-12 14:05:21.593  INFO 30352 --- [           main] org.hibernate.dialect.Dialect            : HHH000400: Using dialect: org.hibernate.dialect.MySQL5InnoDBDialect
2020-02-12 14:05:22.337  INFO 30352 --- [           main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.i
2020-02-12 14:05:22.345  INFO 30352 --- [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2020-02-12 14:05:22.827  INFO 30352 --- [           main] i.b.s.SpringDataJpaApplication           : Started SpringDataJpaApplication in 4.477 seconds (JVM running for 5.977)
2020-02-12 14:05:23.007  INFO 30352 --- [           main] i.b.s.SpringDataJpaApplication           : Book : Book{id=1, title='Belajar Spring Boot', writer='Teten Nugraha', isbn='IS-90908'}
2020-02-12 14:05:23.012  INFO 30352 --- [extShutdownHook] j.LocalContainerEntityManagerFactoryBean : Closing JPA EntityManagerFactory for persistence unit 'default'
2020-02-12 14:05:23.017  INFO 30352 --- [extShutdownHook] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Shutdown initiated...
2020-02-12 14:05:23.024  INFO 30352 --- [extShutdownHook] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Shutdown completed.
```

Berikut kode lengkap dari BookRepository.

id.belajar.springdatajpa.repository.BookRepository

```
public interface BookRepository extends JpaRepository<Book, Long> {

    List<Book> findAll();

    List<Book> findAllByWriter(String writer);

    Book findByIsbn(String isbn);

    Book findByTitle(String title);
}
```

Eksekusi *Native Query*

Untuk beberapa kasus, kadang kita ingin menggunakan *native sql syntax* secara langsung seperti yang kita pernah gunakan jika menggunakan Statement atau PreparedStatement. Spring Data juga sudah menyediakan untuk fungsi ini. Masih di file yang sama yaitu di file BookRepository.

id.belajar.springdatajpa.repository.BookRepository

```
public interface BookRepository extends JpaRepository<Book, Long> {

    …

    @Query(
        nativeQuery = true,
        value = "select * from book"
    )
    List<Book> findAllQueryNative();

    …
}
```

id.belajar.springdatajpa.SpringDataJpaApplication

```
@Override
public void run(String... args) throws Exception {

   List<Book> books = bookRepository.findAllQueryNative();

   LOG.info("Book : "+books);

}
```

Jika kita running maka dalam log akan ada dua data book yang kita panggil.

```
DriverDataSource  : Registered driver with driverClassName=com.mysql.jdbc.Driver was not found, trying direct instantiation.
iDataSource       : HikariPool-1 - Start completed.
ialect            : HHH000400: Using dialect: org.hibernate.dialect.MySQL5InnoDBDialect
rmInitiator       : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
anagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
.ication          : Started SpringDataJpaApplication in 3.974 seconds (JVM running for 5.684)
.ication          : Book : [Book{id=1, title='Belajar Spring Boot', writer='Teten Nugraha', isbn='IS-90908'}, Book{id=2, title='Belajar Spring Boot 2', writer='Teten Nugraha', isbn='IS-9090890'}]
anagerFactoryBean : Closing JPA EntityManagerFactory for persistence unit 'default'
iDataSource       : HikariPool-1 - Shutdown initiated...
iDataSource       : HikariPool-1 - Shutdown completed.
```

nah sekarang bagaimana cara nya jika kita ingin melempar sebuah parameter, katakanlah kita ingin mendapatkan semua data buku berdasarkan nama penulis ?.

id.belajar.springdatajpa.repository.BookRepository

```
public interface BookRepository extends JpaRepository<Book, Long> {

    …

    @Query(
        nativeQuery = true,
        value = "select * from book where writer = ?1"
    )
    List<Book> findAllByWriterQueryNative(String writer);


    …
}
```

id.belajar.springdatajpa.SpringDataJpaApplication

```
@Override
public void run(String... args) throws Exception {

   final String writer = "Teten Nugraha";

   List<Book> books = bookRepository.findAllByWriterQueryNative(writer);

   LOG.info("Book : "+books);

}
```

```
riverDataSource  : Registered driver with driverClassName=com.mysql.jdbc.Driver was not found, trying direct instantiation.
DataSource       : HikariPool-1 - Start completed.
alect            : HHH000400: Using dialect: org.hibernate.dialect.MySQL5InnoDBDialect
mInitiator       : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
nagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
cation           : Started SpringDataJpaApplication in 3.677 seconds (JVM running for 5.237)
cation           : Book : [Book{id=1, title='Belajar Spring Boot', writer='Teten Nugraha', isbn='IS-90908'}, Book{id=2, title='Belajar Spring Boot 2', writer='Teten Nugraha', isbn='IS-9090890'}]
nagerFactoryBean : Closing JPA EntityManagerFactory for persistence unit 'default'
DataSource       : HikariPool-1 - Shutdown initiated...
DataSource       : HikariPool-1 - Shutdown completed.
```

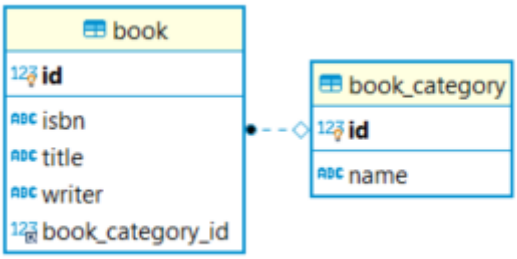
Untuk file BookRepository jika digabungkan akan menjadi seperti ini

id.belajar.springdatajpa.repository.BookRepository

```java
package id.belajar.springdatajpa.repository;

import id.belajar.springdatajpa.entity.Book;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;

import java.util.List;

public interface BookRepository extends JpaRepository<Book, Long> {

    List<Book> findAll();

    List<Book> findAllByWriter(String writer);

    Book findByIsbn(String isbn);

    Book findByTitle(String title);

    @Query(
            nativeQuery = true,
            value = "select * from book"
    )
    List<Book> findAllQueryNative();

    @Query(
            nativeQuery = true,
            value = "select * from book where writer = ?1"
    )
    List<Book> findAllByWriterQueryNative(String writer);
}
```

One to Many Relationship

One to Many relationsip adalah salah satu relasi yang menghubungkan antara dua entitas atau dua table yang pada umumnya sering disebut hubungan *parent-child* dimana atribut *parent* selalu ada pada *child.* Pada subbab ini, kita akan membuat perubahan *enhancement* pada class *Book* dan membuat satu class baru dengan nama *BookCategory.*



Buat Sebuah class BookCategory pada package ***entity***. *Class* ini nantinya akan menjadi *class parent* untuk *class Book* beserta class BookCategoryRepositorynya.

id.belajar.springdatajpa.entity.BookCategory

```java
package id.belajar.springdatajpa.entity;

import javax.persistence.*;
import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;


@Entity
public class BookCategory {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;

    @OneToMany(
            mappedBy = "bookCategory",
            cascade = CascadeType.ALL
    )
    private List<Book> books;

    public BookCategory() {
    }

    public BookCategory(String name, Book... books) {
        this.name = name;
        this.books = Stream.of(books).collect(Collectors.toList());
        this.books.forEach(x -> x.setBookCategory(this));
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public List<Book> getBooks() {
        return books;
    }

    public void setBooks(List<Book> books) {
        this.books = books;
    }

    @Override
    public String toString() {
        return "BookCategory{" +
```

```
                "id=" + id +
                ", name='" + name + '\'' +
                ", books=" + books +
                '}';
    }
}
```

Penjelasan kode diatas.

- `@OneToMany` adalah annotasi yang yang mendefinisikan relasi *one-to-many* sehingga class *BookCategory* akan mempunyai banyak *child Book.*
- `mappedBy` adalah property yang memberitahu Spring Data JPA bahwa *foreign key* dari relasi ini berada pada *class child* pada variable **`bookCategory.`**
- `CascadeType.ALL` adalah sebuah tipe *cascade* yang akan menyebarkan semua operasi *EntityManager* (PERSIST, REMOVE, REFRESH, MERGE, DETACH) ke entitas yang berelasi.

Selanjutnya, edit class *Book* dan tambahkan objek variable *BookCategory bookCategory*.

….

```
@ManyToOne
@JoinColumn
private BookCategory bookCategory;
```

…..

- `@ManyToOne` mendefinisikan relasi *many-to-one* antara dua entitas.
- `@JoinColumn` mengindikasikan bahwa entitas *Book* adalah yang mempunyai kunci (*foreign key*) yang menhubungkan dengan *class parent* yaitu *BookCategory*.

id.belajar.springdatajpa.entity.Book

```
package id.belajar.springdatajpa.entity;

import javax.persistence.*;

@Entity
public class Book {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String title;

    @Column(nullable = false)
    private String writer;

    @Column(nullable = false)
    private String isbn;

    @ManyToOne
    @JoinColumn
```

```java
private BookCategory bookCategory;

public Book() {
}

public Book(String title, String writer, String isbn) {
    this.title  = title;
    this.writer = writer;
    this.isbn = isbn;
}

public Long getId() {
    return id;
}

public void setId(Long id) {
    this.id = id;
}

public String getTitle() {
    return title;
}

public void setTitle(String title) {
    this.title = title;
}

public String getWriter() {
    return writer;
}

public void setWriter(String writer) {
    this.writer = writer;
}

public String getIsbn() {
    return isbn;
}

public void setIsbn(String isbn) {
    this.isbn = isbn;
}

public BookCategory getBookCategory() {
    return bookCategory;
}

public void setBookCategory(BookCategory bookCategory) {
    this.bookCategory = bookCategory;
}

@Override
public String toString() {
    return "Book{" +
            "id=" + id +
            ", title='" + title + '\'' +
            ", writer='" + writer + '\'' +
            ", isbn='" + isbn + '\'' +
            '}';
```

```
    }
}
```

id.belajar.springdatajpa.repository.BookCategoryRepository

```
package id.belajar.springdatajpa.repository;

import id.belajar.springdatajpa.entity.BookCategory;
import org.springframework.data.jpa.repository.JpaRepository;

public interface BookCategoryRepository extends JpaRepository<BookCategory,
Long> {
}
```

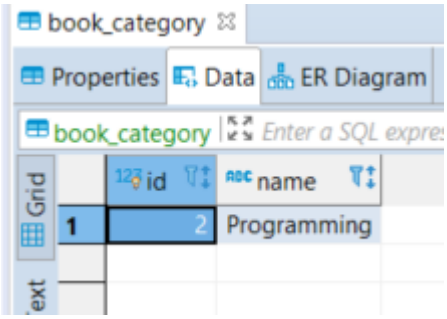
Untuk melakukan pengetesan, lakukan pada class *SpringDataJpaApplication* dan lakukan injeksi terhadap class BookCategoryRepository dan class BookRepository. Setelah itu kita masukan data BookCategory beserta dengan Book nya namun sebelumnya hapus terlebih dahulu data *Book* yang sudah ada didalam database.

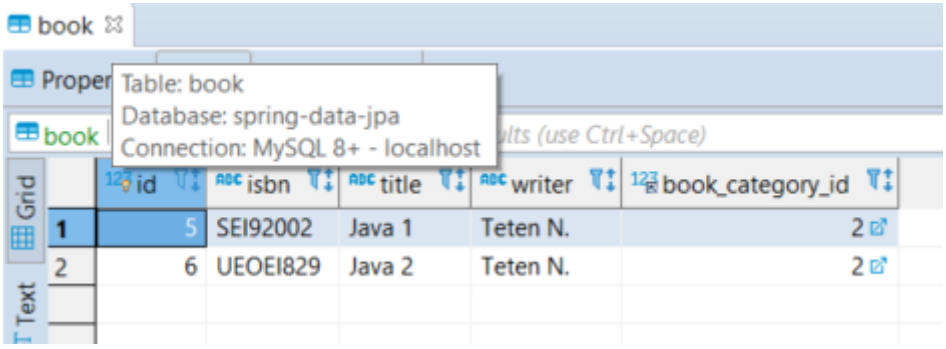
```
@Autowired
private BookRepository bookRepository;

@Autowired
private BookCategoryRepository bookCategoryRepository;

@Override
public void run(String... args) throws Exception {

   // create Book Category
   BookCategory bookCategory = bookCategoryRepository.save(new
BookCategory("Programming", new Book("Java 1","Teten N.","SEI92002"), new
Book("Java 2","Teten N.","UEOEI829")));

   LOG.info("BookCategory : "+bookCategory);

}
```

Jika dijalankan,maka dalam LOG akan terdapat keterangan data dari BookCategory beserta Book nya.

```
main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
main] i.b.s.SpringDataJpaApplication           : Started SpringDataJpaApplication in 5.408 seconds (JVM running for 6.909)
main] i.b.s.SpringDataJpaApplication           : BookCategory : BookCategory{id=2, name='Programming', books=[Book{id=5, title='Java 1', writer='Teten N.', isbn='SEI92002'}, Book{id=6, title='Java 2', writer='Teten N.', isbn='UEOEI829'}]}
```

Jika kita check dalam database, hasilnya akan sama dengan yang di LOG.





id.belajar.springdatajpa.SpringDataJpaApplication

```
@SpringBootApplication
public class SpringDataJpaApplication implements CommandLineRunner {

   private final Logger LOG =
LoggerFactory.getLogger(SpringDataJpaApplication.class);

   @Autowired
   private BookRepository bookRepository;

   @Autowired
   private BookCategoryRepository bookCategoryRepository;

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

   @Override
   public void run(String... args) throws Exception {

      // create Book Category
      BookCategory bookCategory = bookCategoryRepository.save(new
BookCategory("Programming", new Book("Java 1","Teten N.","SEI92002"), new
Book("Java 2","Teten N.","UEOEI829")));

      LOG.info("BookCategory : "+bookCategory);

   }
}
```

Many to Many Relationship

Setelah relasi *one-to-many* selanjutnya akan kita bahas mengenai relasi *many-to-many* dengan menggunakan contoh kasus relasi antara entitas *students* dan *courses.* Dimana satu *Students* bisa memiliki bayak *Courses* dan begitu juga satu *Courses* dapat memiliki banyak *Students*. Biasanya relasi ini akan membuat sebuah table yang menghubungkan kedua entitas seperti pada diagram ERD berikut.

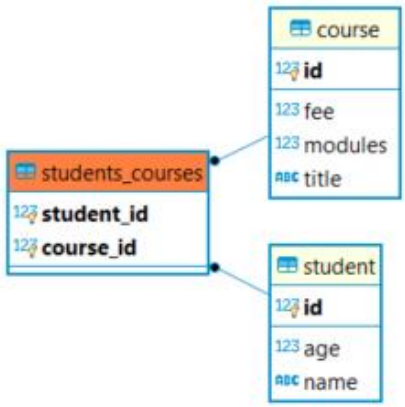


Table *Students_courses* diatas adalah sebuah table yang mempunyai dua *foreign key, student_id* dan *course_id* dimana *student_id* menghubungkan dengan table *Student* dan *course_id* menghubungkan dengan table *Course*.

Buat dua buah class dengan nama *Student* dan *Course* .

id.belajar.springdatajpa.entity.Student

```
@Entity
public class Student {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private int age;

    @ManyToMany(
            fetch = FetchType.LAZY,
            cascade = CascadeType.PERSIST
    )
    @JoinTable(
            name = "students_courses",
            joinColumns = {
                    @JoinColumn(name = "student_id", referencedColumnName =
"id", nullable = false, updatable = false)},
            inverseJoinColumns = {
                    @JoinColumn(name = "course_id", referencedColumnName = "id",
nullable = false, updatable = false)
            }
    )
    private Set<Course> courses = new HashSet<>();
```

```
    public Student() {
    }

    public Student(String name, int age) {
      this.name = name;
      this.age = age;
    }


    // Setter, Getter and toString
}
```

Penjelasan kode diatas.

- `FetchType.LAZY` konfigurasi yang tidak secara otomatis memuat data yang berada pada relasi nya sehingga untuk memuat harus menggunakan *getter method.*
- `CascadeType.PERSIST` artinya bahwa operasi *save()* dari Spring Data JPA *cascade* ke entitas yang berelasi.
- `@JoinTable` adalah anotasi yang akan membuat sebuah table baru dengan nama *students_courses* sesuai pada gambar ERD diatas. Menggunakan `@JoinColumn` untuk menghubungkan antara *foreign_key* dengan table referensi nya.

id.belajar.springdatajpa.entity.Course

```
@Entity
public class Course {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String title;
    private int modules;
    private double fee;

    @ManyToMany(
            mappedBy = "courses",
            fetch = FetchType.LAZY
    )
    private Set<Student> students = new HashSet<>();

    public Course() {
    }

    public Course(String title, int modules, double fee) {
      this.title = title;
      this.modules = modules;
      this.fee = fee;
    }


    // Setter, Getter and toString()
}
```

id.belajar.springdatajpa.repository.StudentRepository

```
package id.belajar.springdatajpa.repository;

import id.belajar.springdatajpa.entity.Student;
import org.springframework.data.jpa.repository.JpaRepository;

public interface StudentRepository extends JpaRepository<Student, Long> {
}
```

id.belajar.springdatajpa.repository.CourseRepository

```
package id.belajar.springdatajpa.repository;

import id.belajar.springdatajpa.entity.Course;
import org.springframework.data.jpa.repository.JpaRepository;

public interface CourseRepository extends JpaRepository<Course, Long> {
}
```

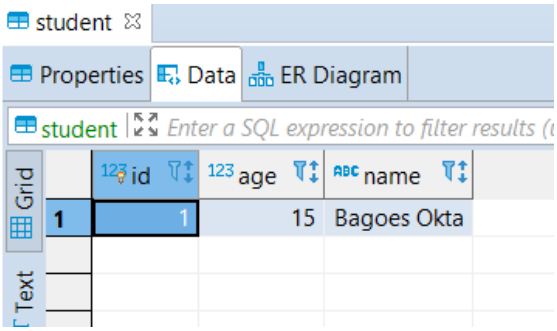
Pada class *SpringDataJPAApplication* lakukan injeksi pada class *StudentRepository* dan *CourseRepository* lalu buat object *student* dan *course* seperti pada source code berikut.

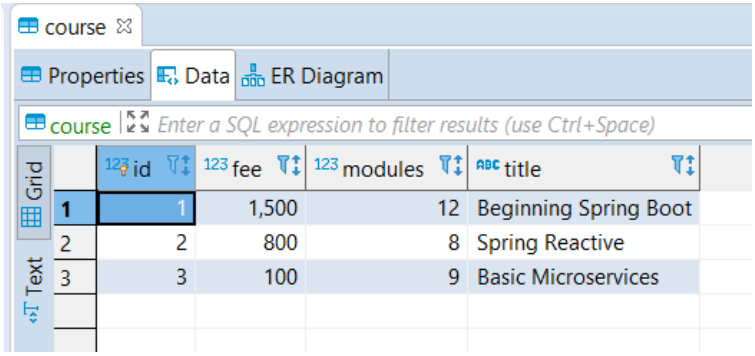
id.belajar.springdatajpa.SpringDataJpaApplication

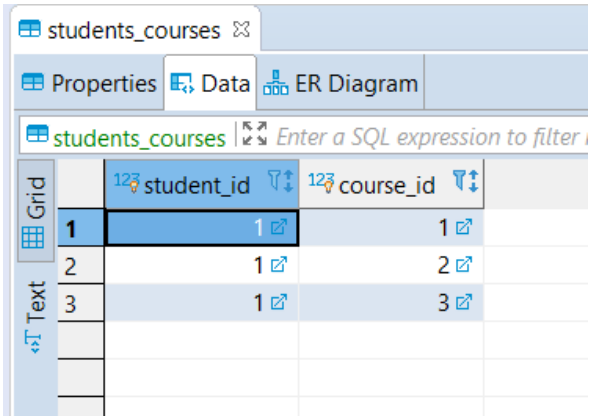
```
@SpringBootApplication
public class SpringDataJpaApplication implements CommandLineRunner {

   private final Logger LOG =
LoggerFactory.getLogger(SpringDataJpaApplication.class);

   @Autowired
   private StudentRepository studentRepository;

   @Autowired
   private CourseRepository courseRepository;

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

   @Override
   public void run(String... args) throws Exception {

      // create a student
      Student student = new Student("Bagoes Okta", 15);

      // save the student
      studentRepository.save(student);
```

```
        // create three courses
        Course course1 = new Course("Beginning Spring Boot",  12, 1500);
        Course course2 = new Course("Spring Reactive", 8, 800);
        Course course3 = new Course("Basic Microservices",9, 100);

        // save courses
        courseRepository.saveAll(Arrays.asList(course1, course2, course3));

        // add courses to the student
        student.getCourses().addAll(Arrays.asList(course1, course2, course3));

        // update the student
        studentRepository.save(student);

    }
}
```

Jalankan programnya, lalu cek menggunakan DBeaver pada table *students, courses,* dan *students_courses.*






students_courses
Properties
Data
ER Diagram
students_courses
Enter a SQL expression to filter
Grid
Text
student_id
course_id
1
1
1
2
1
3
1
3

Spring Web + Thymeleaf

Spring Web adalah salah satu projek yang membantu kita untuk membuat sebuah aplikasi web (yang bisa berjalan menggunakan *browser*) dan Spring Web ini pasti menggunakan annotasi @Controller pada level *Controller* nya. Sedangkan *Thymeleaf* adalah sebuah template engine khusus Java tempat kita menuliskan file-file HTML nya sama seperti JSP tapi *Thyemeleaf* adalah template engine yang paling baru didunia Java.

Buat sebuah projek dengan spefisikasi sebagai berikut:

Project	**Maven Project**
Versi Spring Boot	2.1.12
Group	id.belajar
Artifact	Spring-web
Dependencies	Spring-boot-starter-thymeleaf
	Spring-boot-starter-web
	Spring-boot-devtools
	Org.webjar Bootstrap

Dengan detail file *pom.xml* untuk projek ini seperti di bawah ini.

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<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
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
       <groupId>org.springframework.boot</groupId>
       <artifactId>spring-boot-starter-parent</artifactId>
       <version>2.1.12.RELEASE</version>
       <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>id.belajar</groupId>
    <artifactId>spring-web</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>spring-web</name>
    <description>Demo project for Spring Boot</description>

    <properties>
       <java.version>1.8</java.version>
       <bootstrap.version>4.2.1</bootstrap.version>
    </properties>

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

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

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <optional>true</optional>
        </dependency>

        <dependency>
            <groupId>org.webjars</groupId>
            <artifactId>bootstrap</artifactId>
            <version>${bootstrap.version}</version>
        </dependency>

    </dependencies>

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

</project>
```

Buat sebuah class *WelcomeController* pada layer Controller.

id.belajar.springweb.controller.WelcomeController

```
package id.belajar.springweb.controller;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;

import java.util.Arrays;
import java.util.List;

@Controller
public class WelcomeController {

    private String message = "Hello from Spring Web";

    private List<String> tasks = Arrays.asList("a", "b", "c", "d", "e", "f",
"g");
```

```
    @GetMapping("/")
    public String main(Model model) {
        model.addAttribute("message", message);
        model.addAttribute("tasks", tasks);

        return "welcome"; //view
    }

    // /hello?name=Jhon
    @GetMapping("/hello")
    public String mainWithParam(
            @RequestParam(name = "name", required = false, defaultValue = "")
                    String name, Model model) {

        model.addAttribute("message", name);

        return "welcome"; //view
    }

}
```

Penjelasan kode diatas :

- @Controller adalah annotasi yang menandakan bahwa kita akan membuat sebuah aplikasi Web.
- @GetMapping adalah akses web menggunakan method GET
- Model adalah *class* yang digunakan untuk mengirimkan data ke *View* dengan nama file *welcome.html*
- `@RequestParam` adalah annotasi untuk membuat sebuah parameter dalam suatu URL.

Lalu setelahnya kita membuat sebuah file html dengan nama *welcome.html*

src.resources.templates.welcome.html

```
<!DOCTYPE HTML>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-
to-fit=no">

    <title>Spring Boot Thymeleaf Hello World Example</title>

    <link rel="stylesheet"
th:href="@{webjars/bootstrap/4.2.1/css/bootstrap.min.css}"/>
    <link rel="stylesheet" th:href="@{/css/main.css}"/>

</head>

<body>

<nav class="navbar navbar-expand-md navbar-dark bg-dark fixed-top">
    <a class="navbar-brand" href="#">Spring Boot Web</a>
</nav>
```

```
<main role="main" class="container">

    <div class="starter-template">
        <h1>Spring Boot Web Thymeleaf Example</h1>
        <h2>
            <span th:text="'Hello, ' + ${message}"></span>
        </h2>
    </div>

    <ol>
        <li th:each="task : ${tasks}" th:text="${task}"></li>
    </ol>

</main>
<!-- /.container -->

<script type="text/javascript"
th:src="@{webjars/bootstrap/4.2.1/js/bootstrap.min.js}"></script>
</body>
</html>
```

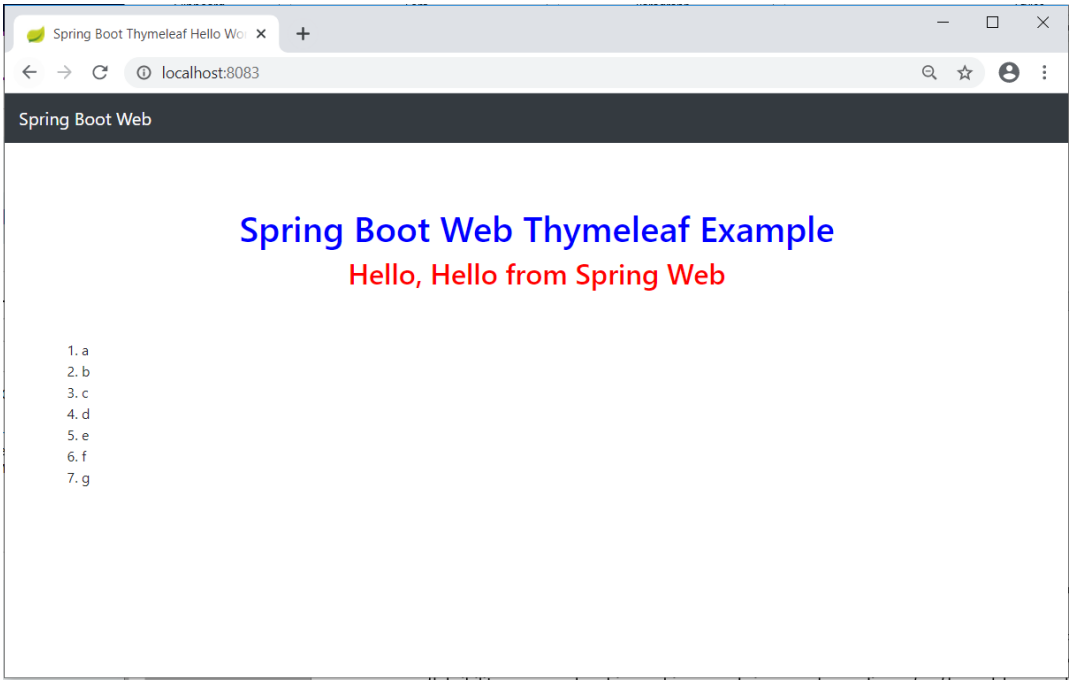
Dan buat file css nya dengan nama file ***main.css***

```
body {
    padding-top: 5rem;
}
.starter-template {
    padding: 3rem 1.5rem;
    text-align: center;
}

h1{
    color:#0000FF;
}

h2{
    color:#FF0000;
}
```

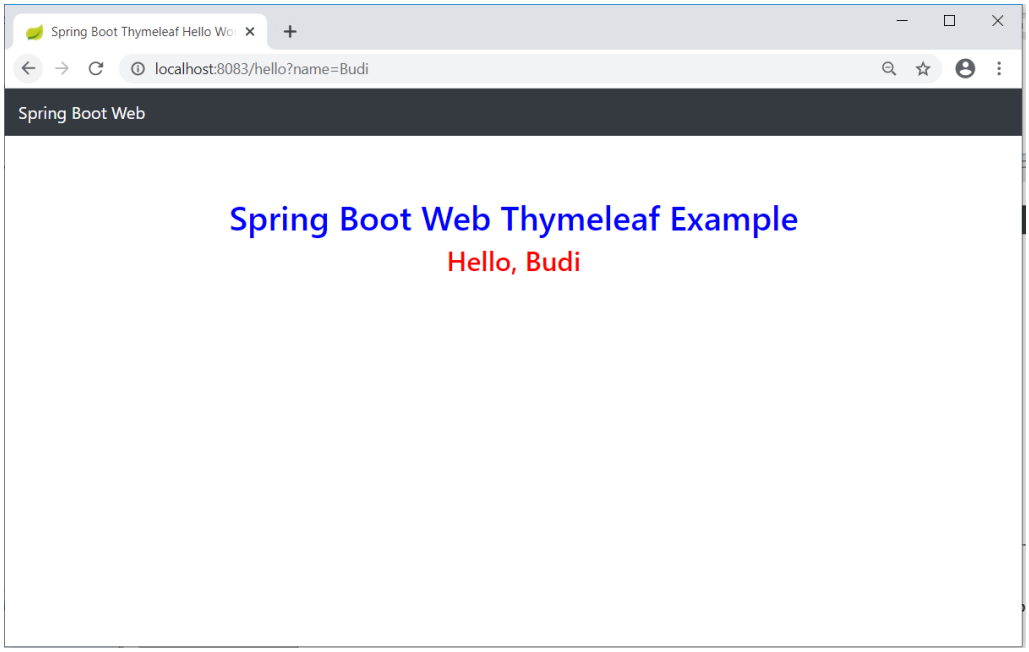
Aplikasi ini akan kita jalankan menggunakan port 8083 untuk itu tambahkan konfigurasinya pada file application.properties.

```
server.port=8083
```

jalankan aplikasi, kemudian akses menggunakan webbrowser dengan port 8083.



Apabila kita mengakses URL localhost:8083/hello?name=Budi, maka output nya akan seperti ini.



Membuat RESTful WebService

Sebelum kita membuat sebuah webservice berbasis restful, alangkah baiknya kita mengenal terlebih dahulu apa itu REST ? REST adalah singkatan dari *Representational State Transfer* adalah sebuah arsitektur layanan komunikasi berbasis web atau menggunakan protocol HTTP. Rest pertama kali diperkenalkan oleh Roy Fielding dalam disertasi dokternya pada tahun 2000. REST terkenal karena fleksibilitasnya yang luar biasa sehingga pada jaman sekaran di era *cloud* banyak layanan-layanan yang menggunakannya.

Langsung saja, akses web https://start.spring.io/ dan generate projek baru dengan spesifikasi sebagai berikut :

Project	**Maven Project**
Versi Spring Boot	2.1.12
Group	id.learn
Artifact	Webservice-restful
Dependencies	Spring Data JPA
	MySQL Driver
	Spring Web
	Project Lombok

Penjelasan :

Dalam projek ini kita menggunakan plugin *Project Lombok*, yaitu plugin yang membantu kita untuk membuat class POJO menjadi lebih cepat sehingga penulisan *default constructor, constructor parameter, getter, setter, toString.* Plugin ini akan digunakan di class-class model.

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<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
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.12.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>id.learn</groupId>
    <artifactId>webservices-restful</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>webservices-restful</name>
    <description>Demo project for Spring Boot</description>

    <properties>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>
        <dependency>
```

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

        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

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

</project>
```

Membuat Product Model

Dalam bab ini kita akan membuat sebuah webservice restful sederhana dengan fitur *Create, Retrieve, Update* dan *Delete* data Produk. Untuk itu kita akan awali dengan membuat table entity class Product yang nantinya akan menjadi table Product.

id.learn.webservicesrestful.model.Product

```
package id.learn.webservicesrestful.model;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.ToString;

import javax.persistence.*;

@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
```

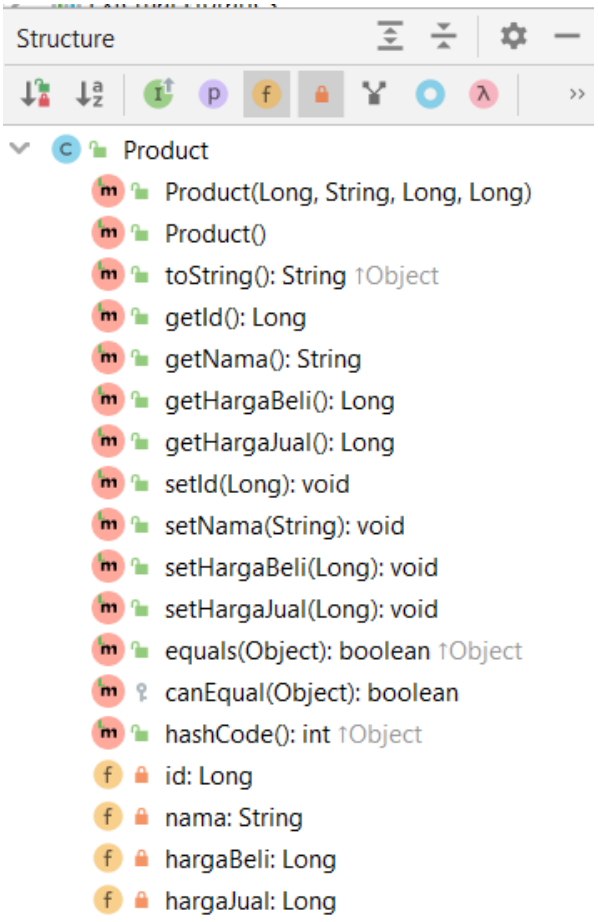
```java
@ToString
public class Product {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String nama;

    @Column(nullable = false)
    private Long hargaBeli;

    @Column(nullable = false)
    private Long hargaJual;


}
```

Penjelasan kode diatas :

Terlihat bahwa dalam class entity Product kali ini kita mengunakan *plugin Lombok*.

- @Data : untuk menggenerate *getter* dan *setter*
- `@NoArgsConstructor :` menyediakan konstruktor *default*
- `@AllArgsConstructor :` menyediakan kontruktor dengan parameter sebanyak jumlah atribut nya yaitu nama, harga beli dan harga jual.
- `@ToString` : untuk menggenerate method toString

Coba lihat attribute class pada IDE terlhat bahwa kita sudah berhasil menggenerate nya berdasarkan annotasi diatas.



Database dan JPA Properties

Pastikan kita sudah membuat database untuk webservice restful ini, misalkan kita ambil nama database nya yaitu **spring-webservice** .

src.main.resources.application.properties

```
#MySQL Connection
spring.datasource.url = jdbc:mysql://localhost:3306/spring-webservice
spring.datasource.username = root
spring.datasource.password = root

#JPA Properties
spring.jpa.database-platform=org.hibernate.dialect.MySQL5InnoDBDialect
spring.jpa.hibernate.ddl-auto=update
spring.datasource.driverClassName=com.mysql.jdbc.Driver

#HikariCP
```

```
spring.datasource.hikari.connection-timeout=20000
spring.datasource.hikari.minimum-idle=5
spring.datasource.hikari.maximum-pool-size=12
spring.datasource.hikari.idle-timeout=300000
spring.datasource.hikari.max-lifetime=120000
spring.datasource.hikari.auto-commit=true

server.port=8082
```

Membuat Product Repository

Untuk DAO nya kita akan menggunakan JPARepository untuk itu buatlah sebuah *interface* yang mendapatkan *inheritance* dari class JPARepository.

id.learn.webservicesrestful.repository.ProductRepository

```
package id.learn.webservicesrestful.repository;

import id.learn.webservicesrestful.model.Product;
import org.springframework.data.jpa.repository.JpaRepository;

public interface ProductRepository extends JpaRepository<Product, Long> {
}
```

Membuat Product Service

Untuk *logic* dari aplikasi ini dibuat di *layer service*. Jadi untuk enterprise khususnya Spring Boot biasanya programmer menggunakan 3 layer yaitu Controller, Service and Repository dan kadang-kadang suka ditambahkan layer Business untuk aplikasi dengan skala yang sangat besar. Dalam layer ini kita menggunakan annotasi @Service untuk service implementasinya. Tapi sebelumnya kita buat kontrak dulu dengan membuat interface Product Service.

id.learn.webservicesrestful.service.ProductService

```
package id.learn.webservicesrestful.service;

import id.learn.webservicesrestful.model.Product;

import java.util.List;

public interface ProductService {

    List<Product> findAllProducts();

    Product findProductById(Long id);

    Product saveorUpdateProduct(Product product);

    void deleteProduct(Long id);

}
```

Dalam kode diatas kita bisa melihat bahwa kita mendefinisikan kontrak atau method :

- `findAllProducts()` untuk mengambil semua data Products

- `findProductById(Long id)` untuk mencari data Product berdasarkan id
- `saveorUpdateProduct(Product product)`
- `deleteProduct(Long id)`

setelah kita mendefinisikan kontraknya selanjutnya kita akan membuat class yang mengimplementasikan kotrak diatas dengan nama class **ProductServiceImpl.**

id.learn.webservicesrestful.service.impl.ProductServiceImpl

```
package id.learn.webservicesrestful.service.impl;

import id.learn.webservicesrestful.exception.ProductNotFoundException;
import id.learn.webservicesrestful.model.Product;
import id.learn.webservicesrestful.repository.ProductRepository;
import id.learn.webservicesrestful.service.ProductService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class ProductServiceImpl implements ProductService {

    @Autowired
    ProductRepository productRepository;

    @Override
    public List<Product> findAllProducts() {
        return productRepository.findAll();
    }

    @Override
    public Product findProductById(Long id) throws Exception {
        Product product = productRepository.findById(id).orElse(new Product());

        return product;
    }

    @Override
    public Product saveorUpdateProduct(Product product) {
        return productRepository.save(product);
    }

    @Override
    public void deleteProduct(Long id) {
        Product product = productRepository.findById(id).orElse(new Product());
        productRepository.delete(product);
    }
}
```

Penjelasan kode diatas yaitu :

- `@Service` : adalah annotasi bahwa kita membuat komponen untuk layer Service
- `@Autowired` adalah annotasi yang menginjika kita untuk menginject sebuah class dalam hal ini yaitu kita menginject interface ProductRepository

- `List<Product> findAllProducts()` method yang digunakan untuk mengembalikan nilai product yang ada di database yang nantinya akan dikembalikan ke controller sebagai pemanggil dari method ini
- `Product findProductById(Long id)` method untuk mencari data single Product berdasarkan id yang diberikan dari Controller. Namun dalam method ini diberi logic jika data product nya tidak ada di database berdasarkan ID yang diberikan maka method ini mengembalikan object Product kosong.
- `Product saveorUpdateProduct(Product product)` method untuk menyimpan data produk baru atau memperbaharui data product.
- `deleteProduct(Long id)` method ini untuk menghapus data produk berdasarkan ID

Membuat Product Controller

Layer controller adalah layer dimana kita menyimpan setiap *endpoint* dari API yang telah kita buat. *Endpoint* adalah semacam alamat yang nantinya akan berkomunikasi langsung dengan rest client baik postman atau dari web aplikasi.

id.learn.webservicesrestful.controller.ProductController

```
package id.learn.webservicesrestful.controller;

import id.learn.webservicesrestful.model.Product;
import id.learn.webservicesrestful.service.ProductService;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import javax.validation.Valid;
import java.util.List;

@RestController
@RequestMapping("/api/products")
public class ProductController {

    @Autowired
    private ProductService productService;

    @GetMapping
    public ResponseEntity<List<Product>> getAllProducts() {
        return new ResponseEntity<>(productService.findAllProducts(),
HttpStatus.OK);
    }

    @PostMapping
    public ResponseEntity<Product> saveProduct(@Valid @RequestBody Product
product) {
        return new ResponseEntity<>(productService.saveorUpdateProduct(product),
HttpStatus.OK);
    }

    @GetMapping("/{id}")
    public ResponseEntity<Product> getOneProduct(@PathVariable Long id) throws
```

```
Exception {
        return new ResponseEntity<Product>(productService.findProductById(id),
HttpStatus.OK);
    }

    @DeleteMapping("/{id}")
    public String deleteProduct(@PathVariable Long id) {
        productService.deleteProduct(id);
        return "delete sukses";
    }
}
```

Penjelasan kode diatas :

- `@RestController` adalah annotasi yang menyatakan class tersebut akan membuat sebuah restful.
- `@RequestMapping("/api/products")` annotasi yang memberikan URL atau alamat dari rest class ini yaitu **`"/api/products"`** sebagai base URL.
- `@Autowired` annotasi untuk menginject. Dalam hal ini kita akan menginject layer service *ProductService* dimana pada service tersebut kita menyimpan *logic* dalam aplikasi ini. Jadi setiap data yang diterima pada layer controller kemudian akan diteruskan pada layer service sehingga controller tidak membuat *logic.*

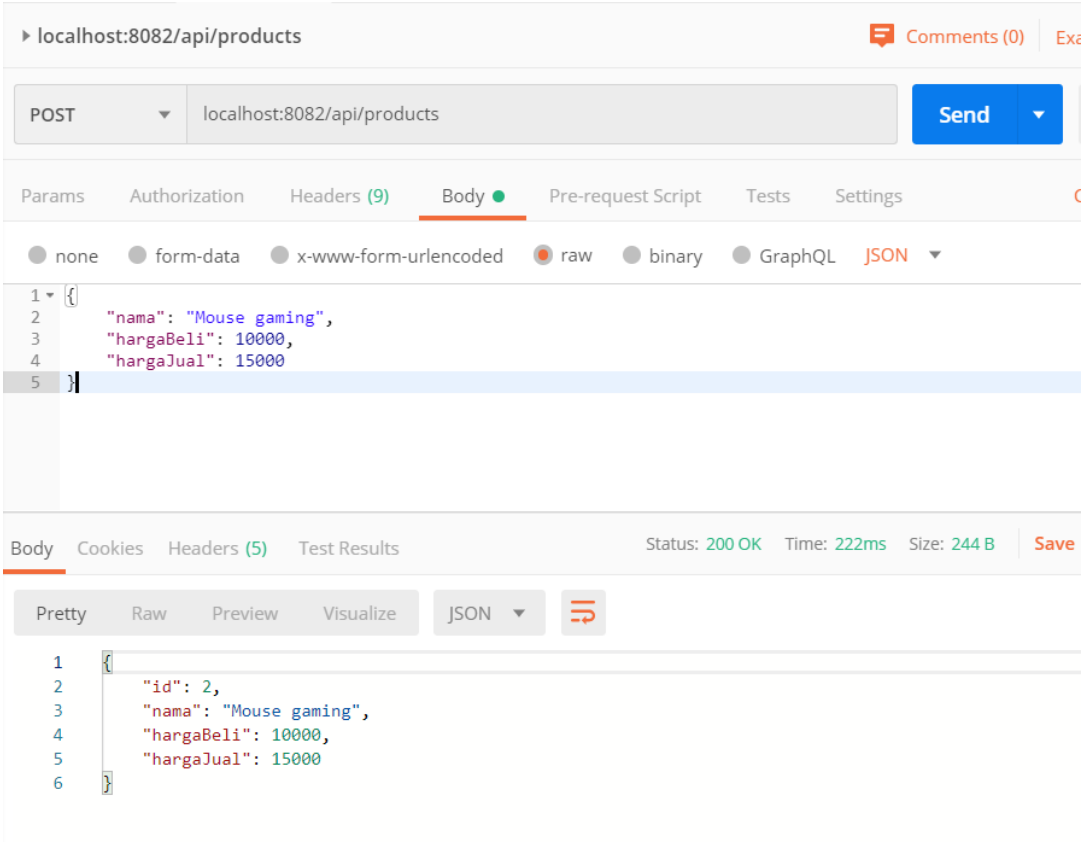
Untuk penjelasan setiap method dalam class ProductController akan dijelaskan sambil langsung mengetest menggunakan PostMan client yang sudah kalian download dan jangan lupa jalankan program ini dengan running di port **:8082** .

- `saveProduct(@Valid @RequestBody Product product)` adalah method yang berfungsi untuk menyimpan atau mengupdate data Product. Kita praktikan langsung buka postman nya kemudian ketikan data berikut :

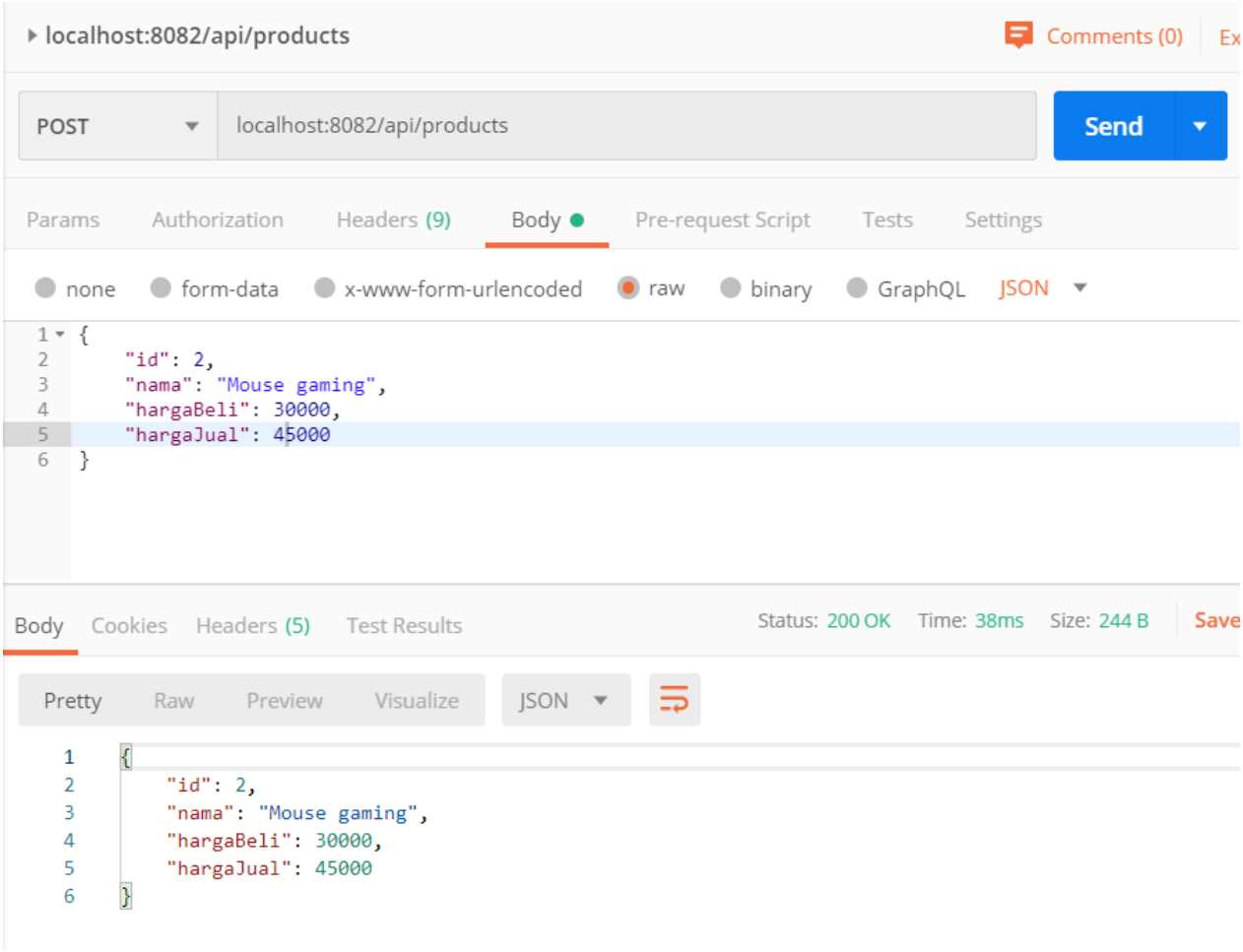
  URL : localhost:8082/api/products
  Method : POST
  Body -> RAW -> JSON dengan data berikut yang biasanya dinamakan ***payload***

  ```
  {
    "nama": "Mouse gaming",
    "hargaBeli": 10000,
    "hargaJual": 15000
  }
  ```

  Kemudian tekan tombol Send



Terlihat bahwa response di bagian bawah data yang diberikan sudah disimpan dalam database dengan id 2. Berikut kita akan simulasikan bahwa kita ingin mengupdate data Mouse gaming ini dengan mengupdate data harga nya. Masih menggunakan method diatas tapi dalam payload nya harus menyertakan **id** nya seperti berikut.

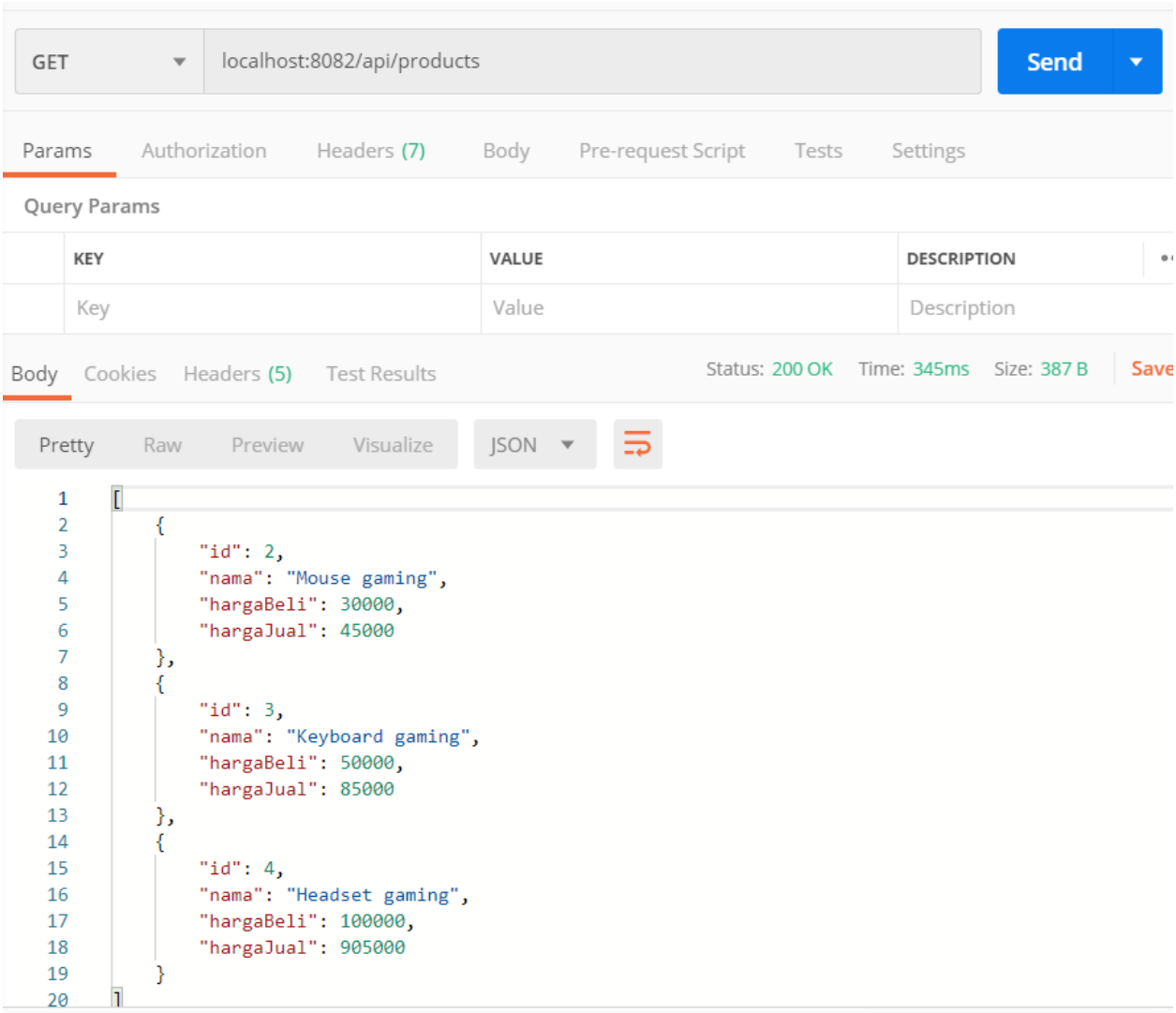


Operasi save dan update ini berdasarkan payload nya apakah menyertakan id atau tidak, jika tidak berarti method nya adalah save namun jika menyertakan id berarti method nya update.

- `getAllProducts()` adalah method yang digunakan untuk mengambil semua data yang ada pada table Product. Menggunakan method annotasi @GETMapping.

  URL : localhost:8082/api/products
  Method : GET

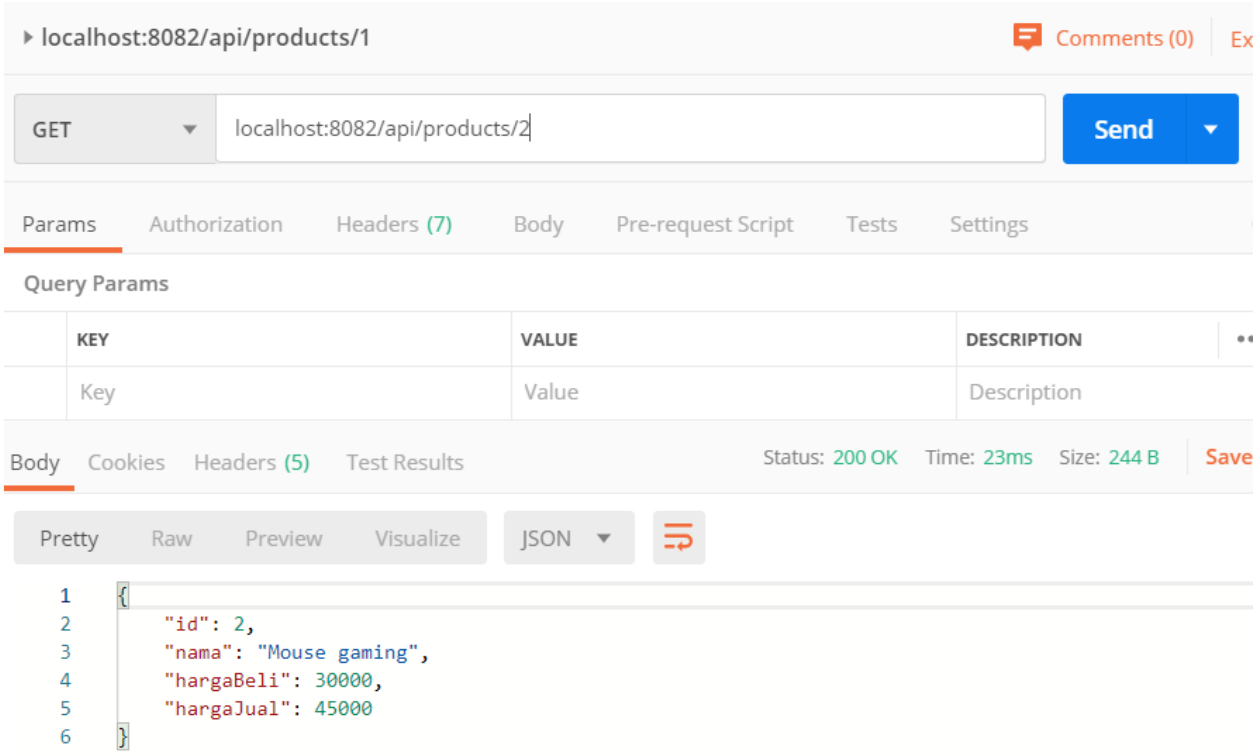
  Kemudian tekan button Send



- `getOneProduct(@PathVariable Long id)` method yang kita gunakan untuk mengambil satu data produk dengan memberikan parameter ID pada URL nya. Misalkan kita akan mengambil data Mouse Gaming diatas yang mempunyai id 2.

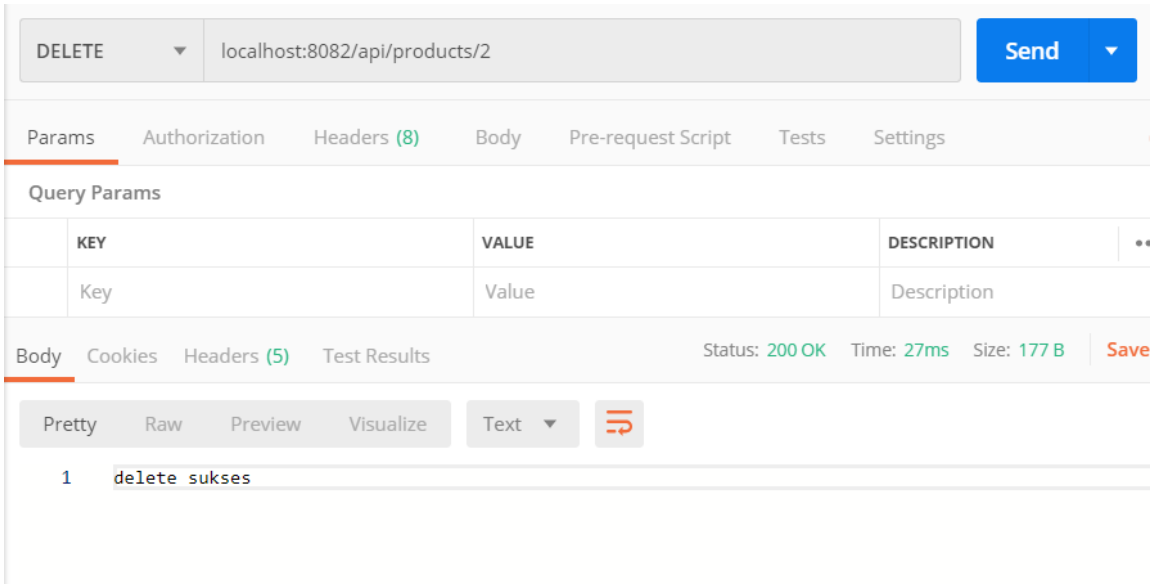
  URL : localhost:8082/api/products/2
  Method : GET

  Dan response dari postman adalah sebagai berikut

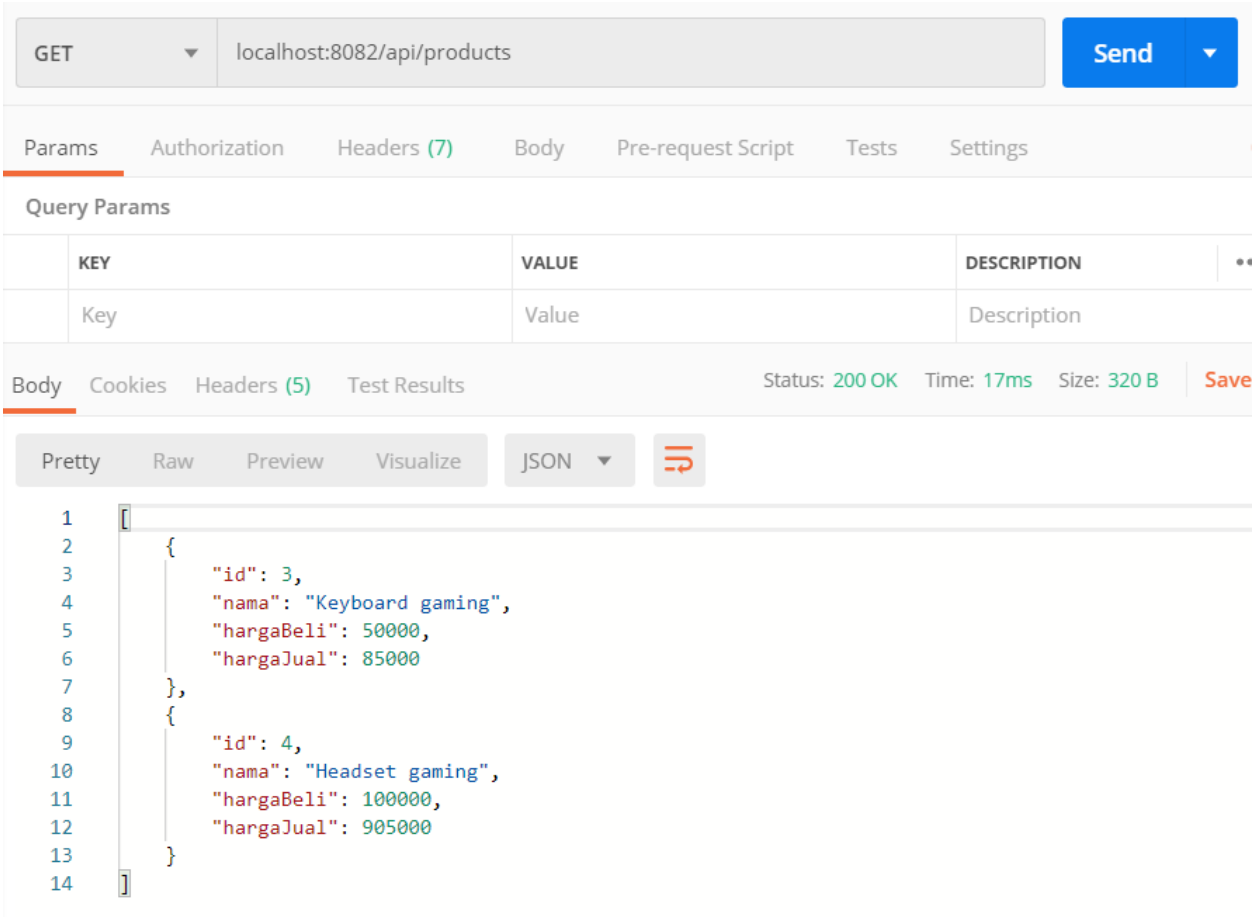


- `deleteProduct(Long id)` method yang digunakan untuk menghapus data produk di database dengan id yang diberikan. Misalkan kita akan menghapus data Mouse gaming yang memiliki id 2. Jika sukses menghapus data maka aplkasi akan mengembalikan status "delete sukses".

URL : localhost:8082/api/products/2
Method : DELETE



Kemudian apabila di cek kembali menggunakan method `getAllProducts()` terlihat bahwa data Mouse gaming sudah terhapus dari database.


GET
localhost:8082/api/products
Send
Params
Authorization
Headers (7)
Body
Pre-request Script
Tests
Settings
Query Params
KEY
VALUE
DESCRIPTION
Key
Value
Description
Body
Cookies
Headers (5)
Test Results
Status: 200 OK
Time: 17ms
Size: 320 B
Save
Pretty
Raw
Preview
Visualize
JSON
[
{
"id": 3,
"nama": "Keyboard gaming",
"hargaBeli": 50000,
"hargaJual": 85000
},
{
"id": 4,
"nama": "Headset gaming",
"hargaBeli": 100000,
"hargaJual": 905000
}
]

Spring Security

Security adalah salah satu fitur yang sangat penting dalam membangun sebuah aplikasi. Tanpa adanya fitur ini maka aplikasi yang dibuat menjadi tidak bagus. Biasanya fitur *security* dalam aplikasi menggunakan mekanisme *Authentication* dan *Authorization*. *Authentication* adalah fitur security yang menangani siapa user yang masuk ke dalam system. Sedangkan *Authorization* adalah setelah user tersebut berhasil masuk ke dalam system, lalu user tersebut bisa mengakses kemana saja.

Dalam *Spring Security* fitur tersebut sudah dibungkus dalam satu library yaitu *spring-boot-starter-security.* Dalam bab ini, masih menggunakan *source code* dala bab Membuat Webrestful. Edit file pom.xml dan tambahkan library tersebut.

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

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<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
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.12.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>id.learn</groupId>
    <artifactId>webservices-restful</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>webservices-restful</name>
    <description>Demo project for Spring Boot</description>

    <properties>
        <java.version>1.8</java.version>
    </properties>

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

        <dependency>
            <groupId>mysql</groupId>
```

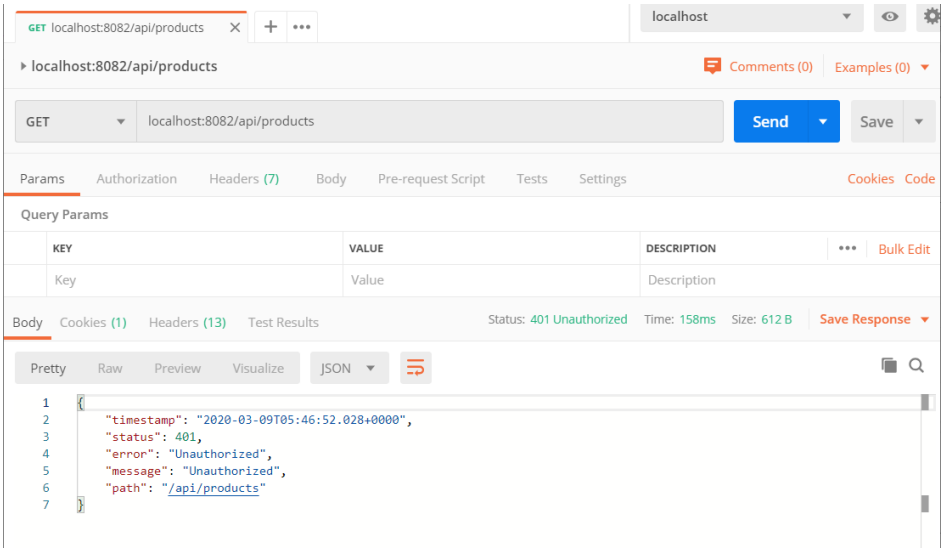
```
            <artifactId>mysql-connector-java</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-security</artifactId>
        </dependency>
    </dependencies>

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

</project>
```

Dalam library spring-boot-starter-security, terdapat tiga komponen penting yang dijadikan dalam satu bundle, yaitu :

- Spring Security Config
- Spring Security Core
- Spring Security Web

Basic Authentication

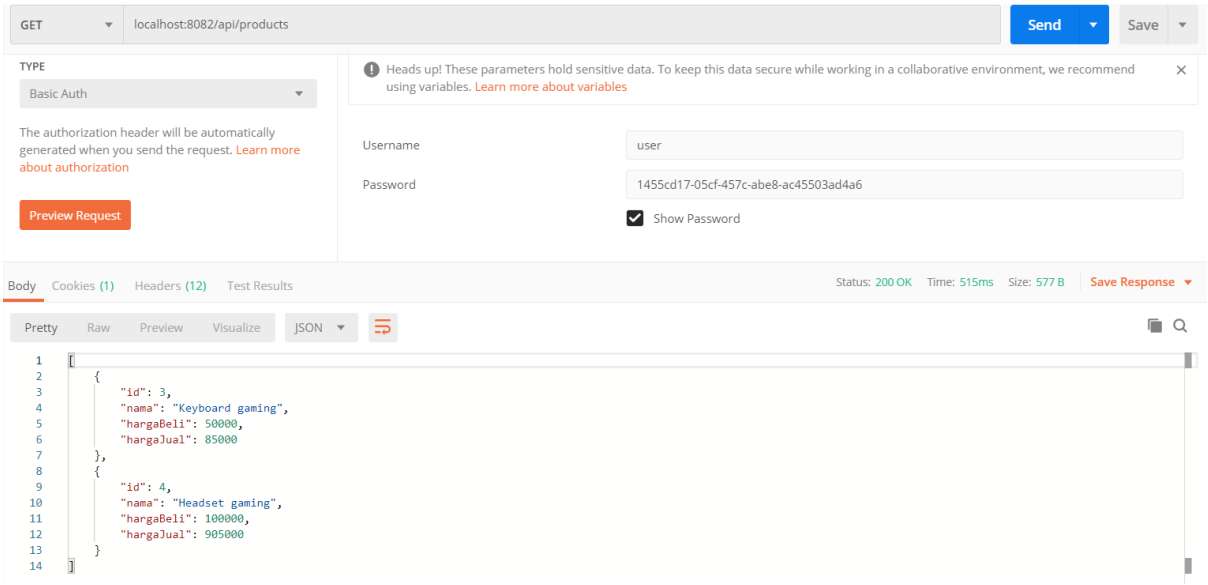
Secara *default* ketika kita menambahkan plugin spring boot security, maka semua *endpoint* yang telah dibuat bersifat *authenticated* atau user harus *login* terlebih dahulu. Bisa kita check, akses endpoint */api/products* dan hasil nya akan mengambalikan status 401 *Unauthorized*.



Ketika sebuah *resource* diamankan menggunakan *basic authentication* maka kita perlu mengirim *username* dan *password* untuk melakukan *request authentication*. Tapi jika kita tidak mengirim, maka secara default Spring Boot membuatkan password generator dengan menggunakan basic username yaitu *user* . Untuk password generator sendiri, bisa dilihat pada log seperti berikut.

```
2020-03-09 12:46:02.180  WARN 1868 --- [           main] aWebConfiguration$JpaWebMvcConfiguration : spring.jpa.open-in-view is enabled by de
2020-03-09 12:46:02.410  INFO 1868 --- [           main] .s.s.UserDetailsServiceAutoConfiguration :

Using generated security password: 1455cd17-05cf-457c-abe8-ac45503ad4a6

2020-03-09 12:46:02.535  INFO 1868 --- [           main] o.s.s.web.DefaultSecurityFilterChain     : Creating filter chain: any request, [org
2020-03-09 12:46:02.630  INFO 1868 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8082 (http) w
2020-03-09 12:46:02.630  INFO 1868 --- [           main] i.l.w.WebservicesRestfulApplication      : Started WebservicesRestfulApplication in
2020-03-09 12:46:51.982  INFO 1868 --- [nio-8082-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'd
2020-03-09 12:46:51.982  INFO 1868 --- [nio-8082-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2020-03-09 12:46:51.987  INFO 1868 --- [nio-8082-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 5 ms
```

Lakukan pengetesan kembali menggunakan postman client, masih menggunakan endpoint yang sama tapi dalam tab *Authorization* menggunakan tipe ***Basic Auth,*** masukan *username* dengan nilai *user* dan *password* dengan nilai dari *password generator* yang ada di log.



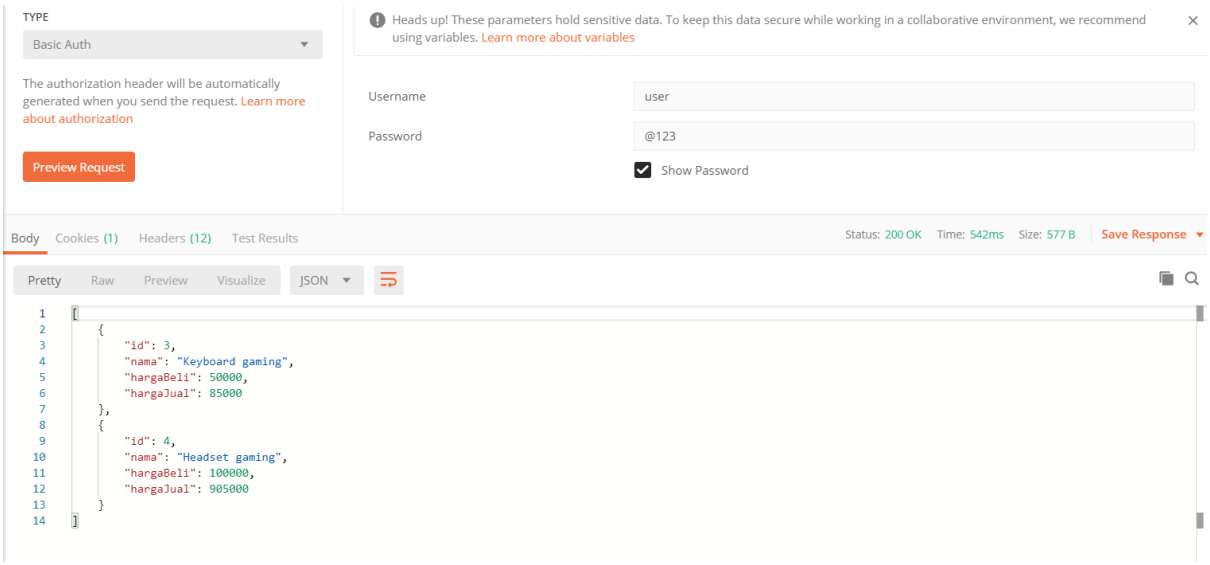
Lalu muncul pertanyaan, bagaimana jika kita ingin mendefinisikan sendiri username dan password nya sehingga tidak perlu menggunakan *password generator.* Untuk itu masukan konfigurasi berikut pada file *application.properties* nya.

```
#Security Config
spring.security.user.name=user
spring.security.user.password=@123
```

application.properties

```
#MySQL Connection
spring.datasource.url = jdbc:mysql://localhost:3306/spring-webservice
spring.datasource.username = root
spring.datasource.password = root

#JPA Properties
spring.jpa.database-platform=org.hibernate.dialect.MySQL5InnoDBDialect
spring.jpa.hibernate.ddl-auto=update
spring.datasource.driverClassName=com.mysql.jdbc.Driver

#HikariCP
spring.datasource.hikari.connection-timeout=20000
spring.datasource.hikari.minimum-idle=5
spring.datasource.hikari.maximum-pool-size=12
spring.datasource.hikari.idle-timeout=300000
spring.datasource.hikari.max-lifetime=120000
spring.datasource.hikari.auto-commit=true

#Security Config
spring.security.user.name=user
spring.security.user.password=@123

server.port=8082
```

Jalankan kembali aplikasinya, dan lakukan pengetesan kembali menggunakan postman dan jangan lupa gunakan username dan password yang sudah didefinisikan pada file *application.properties.*



Unit test dengan JUnit dan Mockito

Pada bab ini masih menggunakan *source code* restul API untuk mensimulasikan dalam penggunaan unit testing. Namun sebelumnya kita bahas dulu apa itu unit testing ?

Unit Testing adalah salah satu level dari proses testing dalam software development. Unit testing adalah pengujian dasar yang menguji setiap unit atau component baik itu dari segi functional atau behavior.

Sebagai programmer, sudah menjadi suatu kewajiban selain bisa memembuat program, juga bisa membuat unit testing dari program yang telah di tulis. Kenapa ? karena kalau tidak melakukan unit testing kita tidak bisa meyakinkan orang lain bahwa kode yang sudah kita tulis itu sudah sesuai dengan proses bisnis atau tidak dan bagaimana kualitas *code* yang ditulis. Namun kebanyakan programmer khusus nya pemula, suka melewati pembelajaran unit testing ini padahal hal ini sangat lah penting dalam *software development*. Di dalam bab ini akan dijelaskan dasar bagaimana membuat Unit Testing menggunakan *tools* yang sering digunakan yaitu JUnit dan Mockito.

Tambahkan Plugin Unit Test , Commons dan Mapper

Dalam bab ini kita akan menambahkan library atau plugin dalam pom.xml, yaitu untuk keperluan Unit Testing (JUnit dan Mockito) , apache commons dan zalando.

a. Plugin Unit Testing
   Tambahkan beberapa plugin berikut untuk membuat unit testing yaitu JUnit dan Mockito.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
    <exclusions>
        <exclusion>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
        </exclusion>
    </exclusions>
</dependency>
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-engine</artifactId>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-params</artifactId>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.junit.platform</groupId>
    <artifactId>junit-platform-launcher</artifactId>
    <version>1.3.2</version>
    <scope>test</scope>
</dependency>
<dependency>
```

```
        <groupId>org.junit.vintage</groupId>
        <artifactId>junit-vintage-engine</artifactId>
        <version>5.3.2</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.mockito</groupId>
        <artifactId>mockito-junit-jupiter</artifactId>
        <scope>test</scope>
    </dependency>
```

b. Plugin Apache Commons

Plugin ini akan kita gunakan untuk menghasilkan data palsu misalkan membuat nama, title,alamat menggunakan class *RandomStringUtils.* Karena generator fake data ini sangat diperlukan dalam unit testing juga.

```
    <dependency>
        <groupId>org.apache.commons</groupId>
        <artifactId>commons-lang3</artifactId>
        <version>3.9</version>
    </dependency>
```

c. Plugin Zalando

Yang terakhir adalah plugin zalando yaitu *mapper* yang berfungsi untuk mengubah file json menjadi object. Plugin ini nantinya akan digunakan untuk membuat unit testing dalam layer *controller*.

```
    <dependency>
        <groupId>org.zalando</groupId>
        <artifactId>problem-spring-web-starter</artifactId>
        <version>${problem-spring-web.version}</version>
        <type>pom</type>
    </dependency>
```

Untuk lengkapnya berikut isi dari file pom.xml yang telah kita modifikasi.

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<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
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.12.RELEASE</version>
```

```
    <relativePath/> <!-- lookup parent from repository -->
</parent>
<groupId>id.learn</groupId>
<artifactId>webservices-restful</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>webservices-restful</name>
<description>Demo project for Spring Boot</description>

<properties>
    <java.version>1.8</java.version>
    <problem-spring-web.version>0.25.0</problem-spring-web.version>
</properties>

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

    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>

    <!-- Random String generator -->
    <dependency>
        <groupId>org.apache.commons</groupId>
        <artifactId>commons-lang3</artifactId>
        <version>3.9</version>
    </dependency>

    <!-- For Testing-->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
        <exclusions>
            <exclusion>
                <groupId>junit</groupId>
                <artifactId>junit</artifactId>
            </exclusion>
        </exclusions>
    </dependency>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-engine</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
```

```
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-params</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.junit.platform</groupId>
            <artifactId>junit-platform-launcher</artifactId>
            <version>1.3.2</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.junit.vintage</groupId>
            <artifactId>junit-vintage-engine</artifactId>
            <version>5.3.2</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.mockito</groupId>
            <artifactId>mockito-junit-jupiter</artifactId>
            <scope>test</scope>
        </dependency>

        <!-- mapper -->
        <dependency>
            <groupId>org.zalando</groupId>
            <artifactId>problem-spring-web-starter</artifactId>
            <version>${problem-spring-web.version}</version>
            <type>pom</type>
        </dependency>
    </dependencies>

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

</project>
```

Membuat Data Fake Generator

Buatlah sebuah class dalam package **src/test** dengan nama **TestObjectFactory.** Dalam class tersebut kita membuat object dari class Product dan membuat object List dengan tipe Product juga.

id.learn.webserviceresterful

```
package id.learn.webservicesrestful;

import id.learn.webservicesrestful.model.Product;
import org.apache.commons.lang3.RandomStringUtils;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;
```

```
public class TestObjectFactory {

    public static Product createProduct() {
        final Product product = new Product();
        product.setId(new Random().nextLong());
        product.setNama(RandomStringUtils.randomAlphabetic(10));
        product.setHargaBeli(new Random().nextLong());
        product.setHargaJual(new Random().nextLong());

        return product;
    }

    public static List<Product> createProductList(final int size) {
        final List<Product> result = new ArrayList<>();
        for (int i = 0; i < size; i++) {
            result.add(createProduct());
        }
        return result;
    }


}
```

Penjelasan kode diatas :

- Method `createProduct()` berfungsi untuk membuat object dari class Product.dalam method tersebut terlihat kita menggunakan class *RandomStringUtils* yang merupakan salah satu *class* dari library `org.apache.commons.`
- Method `createProductList` berfungsi untuk membuat object List dengan tipe data Product dengan menggunakan parameter size sehingga kita bisa menentukan jumlah data dalam List tersebut.

Unit Testing Service Layer

Dimulai dari *service layer* dimana layer tersebut adalah logic dari aplikasi ini dan bisa berkomunikasi dengan repository atau database. Kenapa pada layer repository tidak di test ? karena kita menggunakan Spring Data JPA untuk layer repositorynya maka secara otomatis sudah ditest juga oleh pihak terkaitnya sehingga kita tidak perlu melakukan testing lagi untuk layer repository.

Buat sebuah class **ProductServiceTest** dengan base konfigurasi seperti dibawah ini.

id.learn.webserviceresftful.ProductServiceTest

```
@RunWith(SpringRunner.class)
public class ProductServiceTest{

  @InjectMocks
  private ProductService productService = new ProductServiceImpl();

  @Mock
  private ProductRepository productRepository;

  @Before
  public void setup() {
```

```
    MockitoAnnotations.initMocks(this);
    ReflectionTestUtils.setField(productService, "productRepository",
productRepository);
}


}
```

Penjelasan kode diatas :

- `@RunWith(SpringRunner.class)` adalah sebuah alias dari class SpringJunit4ClassRunner yang menghubungkan JUnit dan Spring TestContext. Dengan SpringRunner, kita dapat mengimplementasikan JUnit dan integration test.
- `@InjectMock` untuk membuat object dari class yang akan di test, dalam hal ini adalah ProductServiceImpl.
- `@Mock` membuat dependensi tiruan (*mock*) karena dalam class ProductServiceImpl terdapat dependensi ke ProductRepository. Sehingga nantinya kita seakan-akan berkomunikasi dengan database.
- `@Before` adalah annotasi untuk mengeksekusi pertama kali setiap dilakukan unit testing di panggil.
- `MockitoAnnotations.initMocks(this)` menginisialisasi setiap property atau *field* yang diberi annotasi @Mock.
- `ReflectionTestUtils` adalah salah satu bagian dari Spring Test Context yang merupakan kumpulan dari method-method utilitas berbasis refleksi yang dilakukan unit testing dan integration test untuk memanggil method private dan melakukan *injection.*

Testing findAll

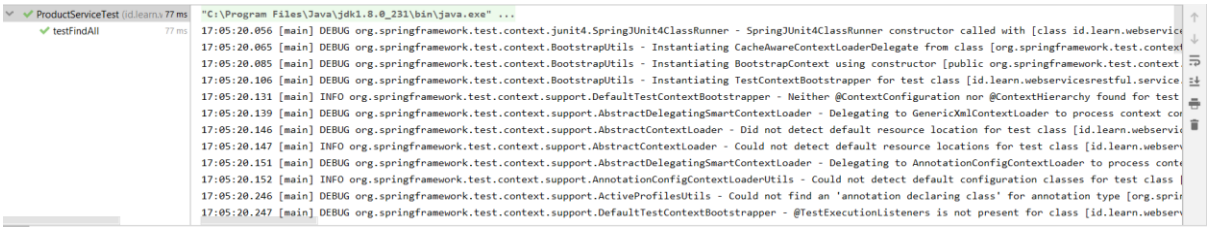
Didalam class **ProductServiceImpl** terdapat method **findAllProduct()**,method ini akan kita coba buatkan unit testing nya. Buatlah sebuah method dengan nama **testFindAll()** didalam class **ProductServiceTest**.

id.learn.webserservicerestful.ProductServiceTest

```
...

@Test
public void testFindAll() {
    final List<Product> datas = TestObjectFactory.createProductList(10);

    Mockito.when(productRepository.findAll()).thenReturn(datas);

    final List<Product> actual = productService.findAllProducts();

    MatcherAssert.assertThat(actual.size(), Matchers.equalTo(datas.size()));
}


...
```

Penjelasan kode diatas :

- @Test adalah annotasi JUnit yang menandakan bahwa method testFindAll adalah sebuah method yang digunakan untuk mengetest method tertentu, dalam hal ini method **findAllProduct**
- Kita siapkan dummy data dan simpan dalam variable *datas* dengan jumlah data sebesar 10 item.
- `Mockito.when(productRepository.findAll()).thenReturn(datas)` merupakan simulasi bahwa kita seakan-akan memangil method findAll dari productRepository dan hasil dari pemanggilan itu kemudian mengembalikan variable *datas* yang telah diinisialisasi.
- `final List<Product> actual = productService.findAllProducts()` adalah simulasi bahwa eksekusi berikutnya seakan-akan kita memanggil method findAllProducts() dalam productService.
- Yang terakhir, kita lakukan pencocokan jumlah data antara data yang diinisialisasi dalam object *datas* dengan object *actual* dengan menggunakan class MatcherAssert.assertThat(…) harus sama-sama melempar jumlah 10;
- Run unit test method ini, jika berhasil tampilannya akan sebagai berikut.



Testing getProductById

Didalam class **ProductServiceImpl** terdapat method **findProductById()**,method ini akan kita coba buatkan unit testing nya. Buatlah sebuah method dengan nama **testProductById()** didalam class **ProductServiceTest**.

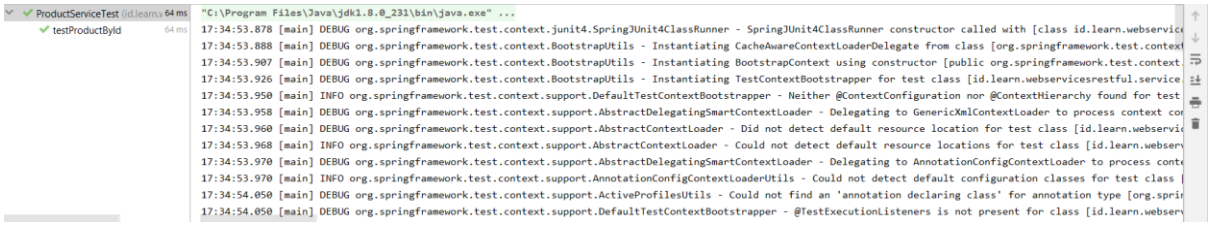
id.learn.webserverestful.ProductServiceTest

```
…

@Test
public void testProductById() throws Exception {
    final Long id = new Random().nextLong();
    final Product product = TestObjectFactory.createProduct();


Mockito.when(productRepository.findById(id)).thenReturn(Optional.of(product));

    final Product actual = productService.findProductById(id);

    MatcherAssert.assertThat(actual.getId(), Matchers.equalTo(product.getId()));
    MatcherAssert.assertThat(actual.getNama(),
Matchers.equalTo(product.getNama()));
    MatcherAssert.assertThat(actual.getHargaBeli(),
Matchers.equalTo(product.getHargaBeli()));
    MatcherAssert.assertThat(actual.getHargaJual(),
```

```
Matchers.equalTo(product.getHargaJual()));
}

...
```

Penjalasan kode diatas :

- `final Long id` karena kita akan mensimulasikan findProductById maka kita harus menyiapkan variable id dengan tipe data Long dan inisialisasi dengan nilai Long secara random / acak.
- Buat object Product dan inisialisasi dengan TestObjectFactory.
- `Mockito.when(productRepository.findById(id)).thenReturn(Optional.of(product))` eksekusi `productRepository.findById()` mock dan kembalikan object product tapi menggunakan class Optional. Karena jika kita lihat dalam method ProductService – findProductById itu melempar object Optional.
- `final Product actual` buat object actual seakan-akan kita mengakses method `productService.findProductById(id).`
- Terakhir lakukan pencocokan data antara objek *actual* dan objek *product*.
- Jika berhasil maka hasilnya akan seperti ini



Testing getProductByIdWithNullDataFromDB

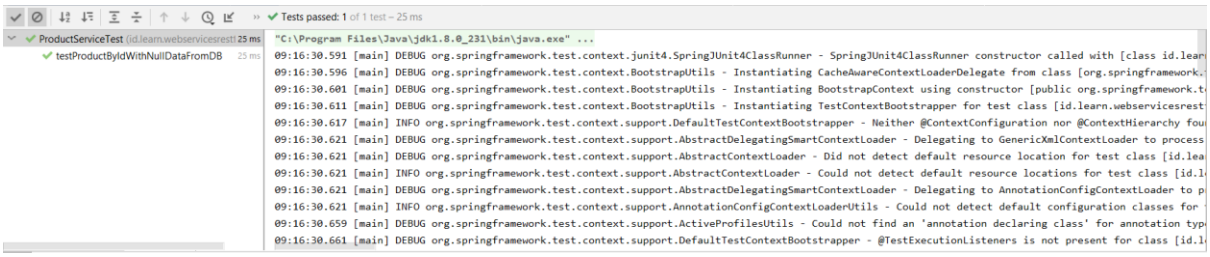
Sebenarnya test ini sama dengan method diatas yaitu mencari data single product berdasarkan ID namun disimulasikan seakan-akan data dari ID yang dicari itu tidak ada di database.

id.learn.webserverestful.ProductServiceTest

```
...

@Test
public void testProductByIdWithNullDataFromDB() throws Exception {
    final Long id = new Random().nextLong();

    Mockito.when(productRepository.findById(id)).thenReturn(Optional.empty());

    final Product actual = productService.findProductById(id);

    MatcherAssert.assertThat(actual, Matchers.nullValue());
}

...
```

Penjelasan kode diatas :

- Saat memanggil productRepository maka method tersebut mengembalikan data null dengan menggunakan `Optional.empty()`.
- `final Product actual` membuat object real seakan-akan kita mengakses method findProductById di ProductServiceImpl.
- Terakhir lakukan pencocokan antara object *actual* , lakukan pencocokan dengan nilai null karena itu yang kita akan test kemudian test.



Testing saveOrUpdateProduct

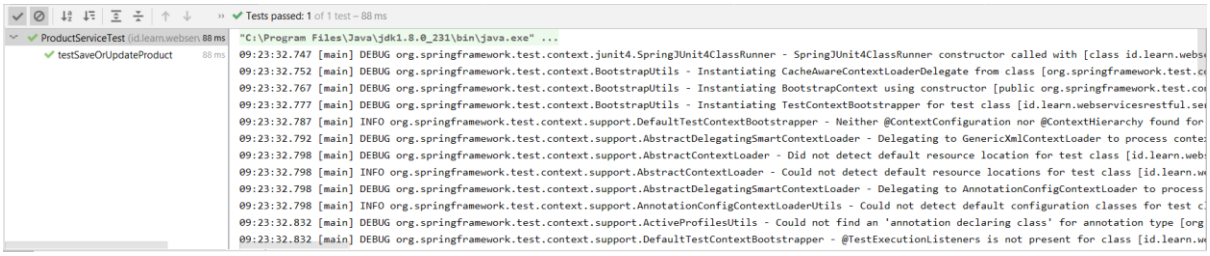
Didalam class **ProductServiceImpl** terdapat method **saveOrUpdateProduct()**,method ini akan kita coba buatkan unit testing nya. Buatlah sebuah method dengan nama **testSaveOrUpdateProduct()** didalam class **ProductServiceTest**.

id.learn.webserservicerestful.ProductServiceTest

```
...

@Test
public void testSaveOrUpdateProduct() {
    final Product product = TestObjectFactory.createProduct();

    Mockito.when(productRepository.save(product)).thenReturn(product);

    final Product actual = productService.saveorUpdateProduct(product);

    MatcherAssert.assertThat(actual, Matchers.notNullValue());
}

...
```

Penjelasan kode diatas :

- Kita membuat object *product* menggunakan factory dummy.
- Panggil productRepository.save() dan seakan-akan mengembalikan data product dari database
- Objek real *actual* dipanggil seakan-akan kita memanggil method saveorUpdateProduct dari layer service.
- Dan terakhir lakukan pencocokan antara object *actual* dengan `Matchers.notNullValue()` karena kita menghendaki nilai dari *product* itu tidak null. Kenapa kita bisa memastikan objek *product* itu tidak null ? karena kita sudah menginisialisasi menggunakan class *TestObjectFactory.*



Testing deleteProduct

Didalam class **ProductServiceImpl** terdapat method **deleteProduct()**,method ini akan kita coba buatkan unit testing nya. Buatlah sebuah method dengan nama **testDeleteProduct()** didalam class **ProductServiceTest**.

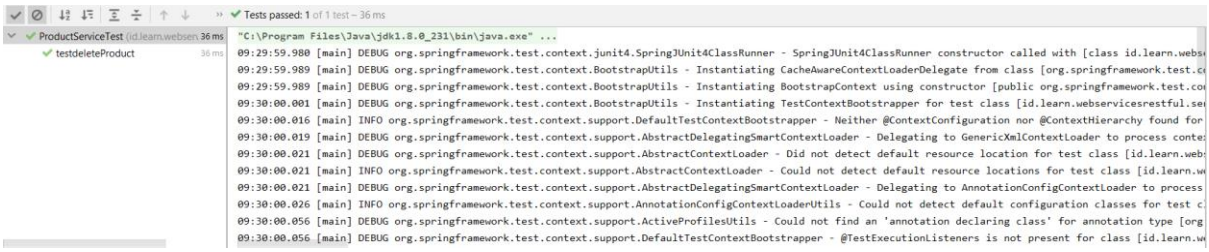
id.learn.webserserviceresful.ProductServiceTest

```
...

@Test
public void testdeleteProduct() {

    final Long id = new Random().nextLong();

    Product product = TestObjectFactory.createProduct();


Mockito.when(productRepository.findById(id)).thenReturn(Optional.of(product));
Mockito.doNothing().when(productRepository).delete(product);


    productService.deleteProduct(id);

    Mockito.verify(productRepository, times(1)).delete(product);

}

...
```

Penjelasan kode diatas :

- Object *id* akan digunakan sebagai simulasi id yang dilempar dari client
- Object *product* diinisialisasi oleh class *TestObjectFactory* jadi seakan-akan object ini tidak berisi null
- `productRepository.findById(id)` repository memanggil method findById dengan parameter dari object *id* dan mengembalikan object *product.*
- `Mockito.doNothing().when(productRepository).delete(product)` menggunakan method doNothing() karena pada saat melakukan eksekusi kode ini, tidak melempar data apapun.
- `productService.deleteProduct(id)` kita langsung eksekusi method ini karena sifat nya tidak mengembalikan tipe data apapun (void – bisa di cek di class ProductService).

- Mockito.*verify*(**productRepository**, *times*(1)).delete(product) adalah sintak untuk memastikan bahwa kode telah mengeksekusi



Untuk source ProductServiceTest secara lengkap seperti di bawah ini.

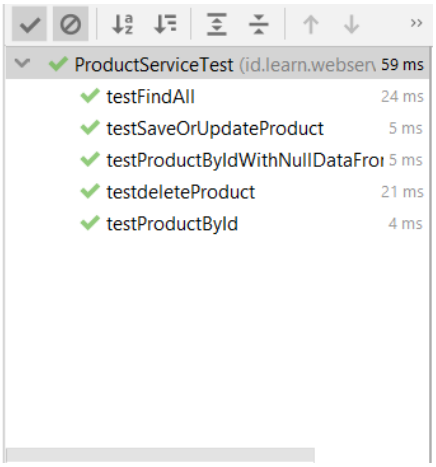
id.learn.webserservicerestful.ProductServiceTest

```
package id.learn.webservicesrestful.service;

import id.learn.webservicesrestful.TestObjectFactory;
import id.learn.webservicesrestful.model.Product;
import id.learn.webservicesrestful.repository.ProductRepository;
import id.learn.webservicesrestful.service.impl.ProductServiceImpl;
import org.hamcrest.MatcherAssert;
import org.hamcrest.Matchers;
import org.junit.Before;
import org.junit.Test;
import org.junit.jupiter.api.BeforeEach;
import org.junit.runner.RunWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.Mockito;
import org.mockito.MockitoAnnotations;
import org.springframework.test.context.junit4.SpringRunner;
import org.springframework.test.util.ReflectionTestUtils;

import java.util.List;
import java.util.Optional;
import java.util.Random;

import static org.mockito.Mockito.times;

@RunWith(SpringRunner.class)
public class ProductServiceTest{

    @InjectMocks
    private ProductService productService = new ProductServiceImpl();

    @Mock
    private ProductRepository productRepository;

    @Before
    public void setup() {
        MockitoAnnotations.initMocks(this);
        ReflectionTestUtils.setField(productService, "productRepository",
productRepository);
```

```
    }

    @Test
    public void testFindAll() {
        final List<Product> datas = TestObjectFactory.createProductList(10);

        Mockito.when(productRepository.findAll()).thenReturn(datas);

        final List<Product> actual = productService.findAllProducts();

        MatcherAssert.assertThat(actual.size(), Matchers.equalTo(datas.size()));
    }

    @Test
    public void testProductById() throws Exception {
        final Long id = new Random().nextLong();
        final Product product = TestObjectFactory.createProduct();


Mockito.when(productRepository.findById(id)).thenReturn(Optional.of(product));

        final Product actual = productService.findProductById(id);

        MatcherAssert.assertThat(actual.getId(),
Matchers.equalTo(product.getId()));
        MatcherAssert.assertThat(actual.getNama(),
Matchers.equalTo(product.getNama()));
        MatcherAssert.assertThat(actual.getHargaBeli(),
Matchers.equalTo(product.getHargaBeli()));
        MatcherAssert.assertThat(actual.getHargaJual(),
Matchers.equalTo(product.getHargaJual()));
    }

    @Test
    public void testProductByIdWithNullDataFromDB() throws Exception {
        final Long id = new Random().nextLong();


Mockito.when(productRepository.findById(id)).thenReturn(Optional.empty());

        final Product actual = productService.findProductById(id);

        MatcherAssert.assertThat(actual, Matchers.nullValue());
    }

    @Test
    public void testSaveOrUpdateProduct() {
        final Product product = TestObjectFactory.createProduct();

        Mockito.when(productRepository.save(product)).thenReturn(product);

        final Product actual = productService.saveorUpdateProduct(product);

        MatcherAssert.assertThat(actual, Matchers.notNullValue());
    }

    @Test
    public void testdeleteProduct() {
```

```
        final Long id = new Random().nextLong();

        Product product = TestObjectFactory.createProduct();


Mockito.when(productRepository.findById(id)).thenReturn(Optional.of(product));
        Mockito.doNothing().when(productRepository).delete(product);

        productService.deleteProduct(id);

        Mockito.verify(productRepository, times(1)).delete(product);

    }

}
```

Dan jika kita eksekusi secara keseluruhan maka hasilnya akan seperti berikut.



Unit Testing Controller Layer

Pada unit testing ini, kita seakan-akan mengetest menggunakan postman tapi versi Java nya menggunakan class *MockMVC*. Buat sebuah class baru dengan nama **ProductControllerTest** sebagai kerangka dasar sebagai berikut.

id.learn.webserviceresftful.controller.ProductServiceTest

```
@ExtendWith(SpringExtension.class)
@WebMvcTest(controllers = ProductController.class)
public class ProductControllerTest  {

    @MockBean
    private ProductService productService;

    @Autowired
    private MockMvc mockMvc;

    @Autowired
    private ObjectMapper objectMapper;
```

```
}
```

Penjelasan kode diatas :

- Karena kita akan mencoba menggunakan JUnit 5 maka tidak diperlukan lagi @RunWith tapi diganti dengan @ExtendWith. SpringExtension adalah class yang disediakan Spring 5 untuk dapat berintegrasi dengan Spring TestContext Framework. @ExtendWith annotasi menerima class yang mengimplementasi *Extension* interface.
- `@WebMvcTest` annotasi yang digunakan untuk menjalankan *application context* yang berisi hanya *beans* untuk melakukan *testing* sebuah *class controller / web*. Dan disini kita akan mengetest class ProductController.class
- Kita menggunakan *@MockBean* untuk membuat object palsu / *mock* dalam sebuah class controller.
- Annotasi `@WebMvcTest`digunakan untuk mengkonfigurasi secara otomatis object **`mockMvc.`**
- **`objectMapper`** akan digunakan untuk mengubah class menjadi JSON.

Testing testSaveOrUpdateProduct

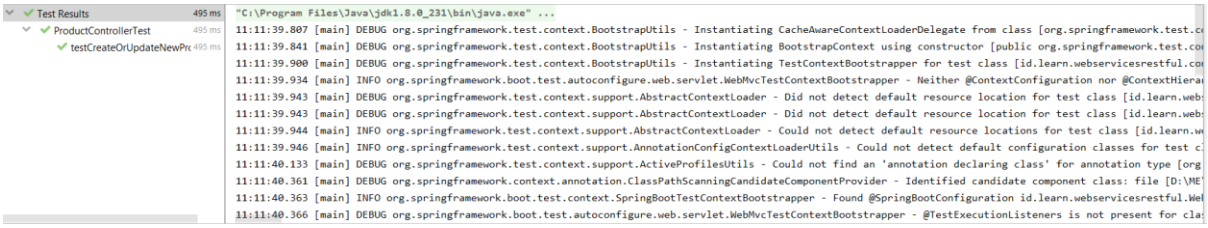
kita akan coba mensimulasikan method testSaveOrUpdateProduct() yang ada di class ProductController. Untuk itu tambahkan method baru dalam file **`ProductControllerTest`** sebagai berikut :

`id.learn.webserverservicerestful.controller.ProductControllerTest`

```
@Test
public void testCreateOrUpdateNewProduct() throws Exception {
    Product product = TestObjectFactory.createProduct();


Mockito.when(productService.saveorUpdateProduct(product)).thenReturn(product);

    mockMvc.perform(post("/api/products")
        .contentType("application/json")
        .content(objectMapper.writeValueAsString(product)))
        .andExpect(status().isOk());
}
```

Penjelasan kode diatas :

- pertama buat sebuah object *product* dengan menggunakan data factory
- kemudian akses **`productService`**`.saveorUpdateProduct(product)` dan simulasikan seakan-akan mengembalikan object atau data product yang telah dibuat diatas.
- Langkah terakhir adalah mensimulasikan bahwa kita sedang mengakses url **`/api/products`** menggunakan method POST dan untuk data yang akan kita kirim ke server sebelumnya kita convert menjadi JSON String menggunakan **`objectMapper`**`.writeValueAsString(), dan diharakan hasil akhir nya server mengemabalikan status OK.`



Testing testGetAllProducts

Pada method ini, akan disimulasikan bagaimana kita mengaskses url /api/products. Buatlah sebuah method test testGetAllProducts() pada class **`ProductControllerTest.`**

`id.learn.webserservicerestful.controller.ProductControllerTest`

```
…

@Test
public void testGetAllProducts() throws Exception {
    final List<Product> datas = TestObjectFactory.createProductList(10);
    Mockito.when(productService.findAllProducts()).thenReturn(datas);

    this.mockMvc.perform(get("/api/products"))
            .andExpect(status().isOk())
            .andExpect(jsonPath("$.size()", is(datas.size())));
}


…
```

Penjelasan kode diatas :

- Buat object *datas* yang berisi *collection* dari class Product menggunakan factory sebanyak 10 item.
- Akses service layer **`productService`**`.findAllProducts()` dan simulasikan seakan-akan mengembalikan object *datas.*
- Akses **`/api/products`** dan set hasil yang diharapkan yaitu status nya ok dan mengembalikan data dan lakukan pencocokan bahwa yang diberikan oleh server itu sama dengan jumlah data pada object *datas*.

Testing getOneProduct

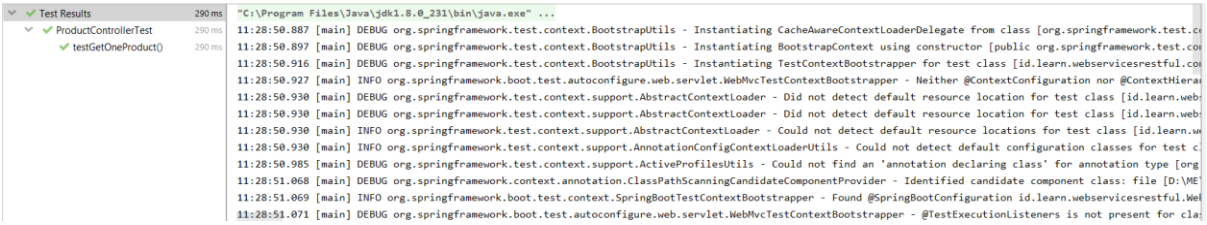
Sesuai dengan namanya, kita akan mengetest URL **`/api/products/{id}`** dan mengembalikan satu data atau object dari database. Edit kembali class **`ProductControllerTest`** dan buat method test baru dengan nama testGetOneProduct().

`id.learn.webserservicerestful.controller.ProductControllerTest`

```
...

@Test
public void testGetOneProduct() throws Exception {
    final Long id = new Random().nextLong();
    final Product product = TestObjectFactory.createProduct();

    Mockito.when(productService.findProductById(id)).thenReturn(product);

    mockMvc.perform(get("/api/products/{id}", id)
            .contentType("application/json"))
            .andExpect(status().isOk());
}
...
```

Penjelasan kode diatas :

- Karena kita akan mensimulasikan untuk mendapatkan object berdasarkan ID maka kita buat object *id* dengan tipe data *Long* menggunakan class random generator.
- Buat *object* Product
- Akses url **/api/products/{id}** dan berikan *id* yang telah dibuat, atur bahwa *contentType* nya adalah *application/json*. Terakhir kita mengharapkan bahwa hasilnya mengembalikan status OK.



Testing testDeleteProduct

Terakhir, kita akan coba melakukan testing penghapus dan data dengan mengakses **/api/products/{id}** menggunakan method DELETE.disamping itu, kita juga akan memerlukan *id* yang akan dilemparkan ke server. Pada class **ProductControllerTest** buat sebuah method test dengan nama testDeleteProduct.

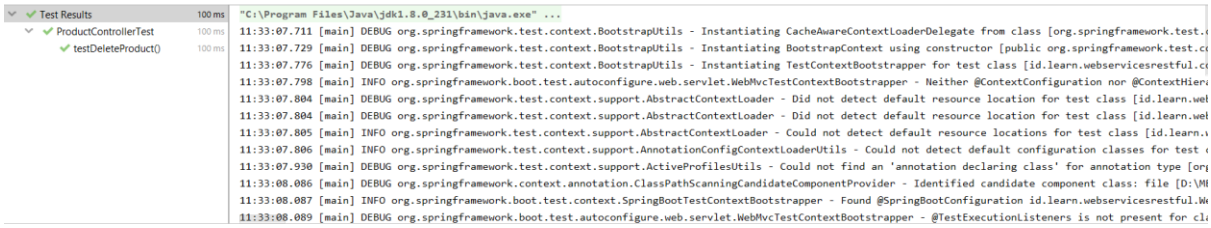
id.learn.webserservicerestful.controller.ProductControllerTest

```
...

@Test
public void testDeleteProduct() throws Exception {
    final Long id = new Random().nextLong();

    Mockito.doNothing().when(productService).deleteProduct(id);

    this.mockMvc.perform(delete("/api/products/{id}", id))
            .andExpect(status().isOk());
}
```

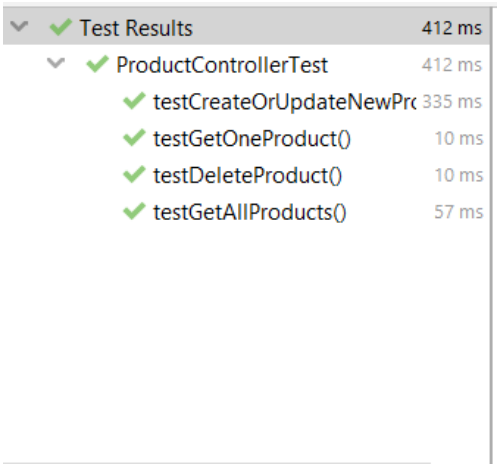
…

Penjelasan kode diatas :

- Karena kita akan mensimulasikan untuk mendapatkan object berdasarkan ID maka kita buat object *id* dengan tipe data *Long* menggunakan class random generator.
- Akses service layer **`productService`** pada `deleteProduct(id).` Karena method deleteProduct() tidak melemparkan apapun (*void*) maka gunakan *Mockito.doNothing()*.
- Terakhir, akses url **`/api/products/{id},`** sertakan *id* yang telah dibuat dan hasil yang diharapkan server mengembalikan status OK.



Dan jika ditest secara keseluruhan dengan melakukan pengetesan lansung pada class **`ProductControllerTest`** maka hasil yang kita harapkan adalah semuanya lolos test.



`id.learn.webserserviceresful.controller.ProductControllerTest`

```
package id.learn.webservicesrestful.controller;

import com.fasterxml.jackson.databind.ObjectMapper;
import id.learn.webservicesrestful.TestObjectFactory;
import id.learn.webservicesrestful.model.Product;
import id.learn.webservicesrestful.service.ProductService;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.Mockito;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;
```

```java
import org.springframework.boot.test.mock.mockito.MockBean;
import org.springframework.test.context.junit.jupiter.SpringExtension;
import org.springframework.test.web.servlet.MockMvc;

import static
org.springframework.test.web.servlet.request.MockMvcRequestBuilders.*;
import static
org.springframework.test.web.servlet.result.MockMvcResultMatchers.*;
import static org.hamcrest.CoreMatchers.is;
import java.util.List;
import java.util.Random;

@ExtendWith(SpringExtension.class)
@WebMvcTest(controllers = ProductController.class)
public class ProductControllerTest  {

    @MockBean
    private ProductService productService;

    @Autowired
    private MockMvc mockMvc;

    @Autowired
    private ObjectMapper objectMapper;

    @Test
    public void testGetAllProducts() throws Exception {
        final List<Product> datas = TestObjectFactory.createProductList(10);
        Mockito.when(productService.findAllProducts()).thenReturn(datas);

        this.mockMvc.perform(get("/api/products"))
                .andExpect(status().isOk())
                .andExpect(jsonPath("$.size()", is(datas.size())));
    }

    @Test
    public void testCreateOrUpdateNewProduct() throws Exception {
        Product product = TestObjectFactory.createProduct();


Mockito.when(productService.saveorUpdateProduct(product)).thenReturn(product);

        mockMvc.perform(post("/api/products")
            .contentType("application/json")
            .content(objectMapper.writeValueAsString(product)))
            .andExpect(status().isOk());
    }

    @Test
    public void testGetOneProduct() throws Exception {
        final Long id = new Random().nextLong();
        final Product product = TestObjectFactory.createProduct();

        Mockito.when(productService.findProductById(id)).thenReturn(product);

        mockMvc.perform(get("/api/products/{id}", id)
                .contentType("application/json"))
                .andExpect(status().isOk());
    }
```

```java
    @Test
    public void testDeleteProduct() throws Exception {
        final Long id = new Random().nextLong();

        Mockito.doNothing().when(productService).deleteProduct(id);

        this.mockMvc.perform(delete("/api/products/{id}", id))
                .andExpect(status().isOk());
    }
}
```

Generate Code Coverage dengan Jacoco dan Sonarqube

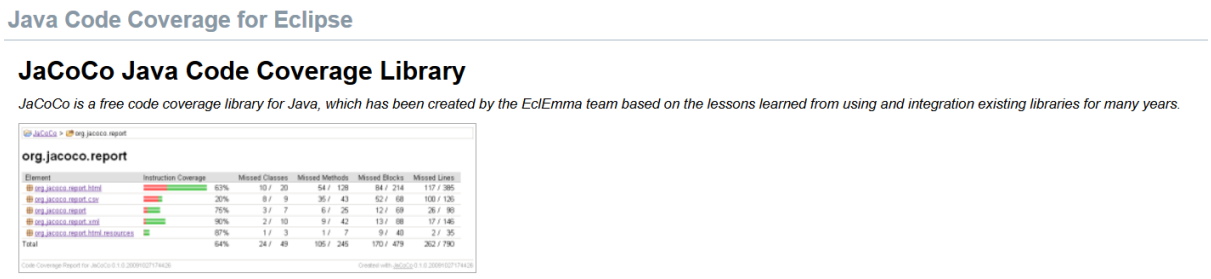
Apa itu Codecoverage ?

Codecoverage adalah sebuah *tools* yang mengukur efektivitas dari *unit testing,* dan untuk menunjukan seberapa lengkap *code* yang telah ditulis yang sesuai dengan *process business* yang telah di*cover* oleh unit test.

Dengan *coverage* kita bisa mengetahui berapa persen kode yang telah di testing dan yang belum, dan untuk mendapatkan hasil yang PASS atau berhasil minimal semua *code* yang tercover minimal 90%.

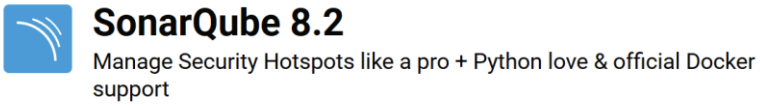
Jacoco and SonarQube

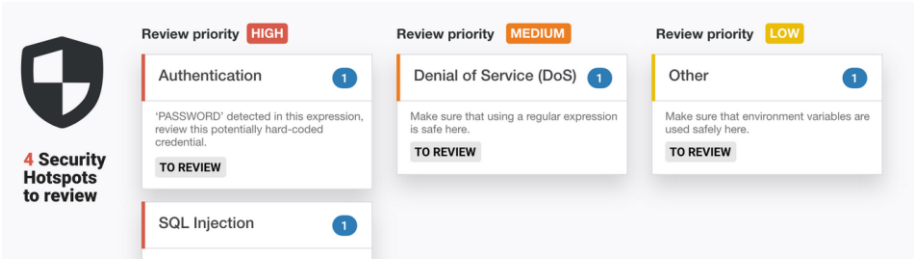
Jacoco adalah sebuah library *opensource* untuk aplikasi Java. Dibuat oleh tim EclEmma berdasarkan hasil riset mereka selama bertahun-tahun. Untuk detail nya bisa diakses di situs nya langsung di https://www.eclemma.org/jacoco/. Jadi aplikasi Jacoco akan melakukan *generate result* dari codecoverage yang telah kita buat pada *unit testing.*



Sonarqube adalah sebuah *tool* yang dilakukan untuk menginspeksi kualitas dari *code* yang telah ditulis untuk melihat *bug, code smells, security vulnerabilities.* Hasil report yang dihasilkan dari *jacoco* kemudian akan dibaca oleh *Sonarqube* yang selanjutkan akan ditampilkan secara informative menggunakan webbrowser.

Pada bab ini penulis menggunakan *Sonarqube free edition* sonarqube-7.9.1 Community Edition dan kalian bisa unduh secara langsung pada situs nya https://www.sonarqube.org/downloads/ .



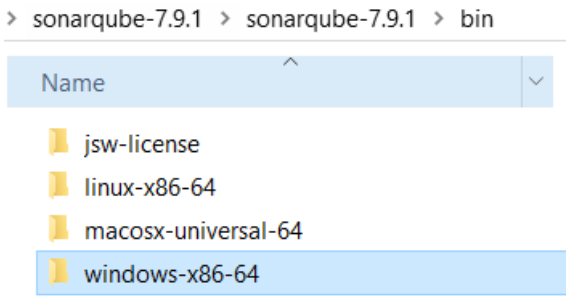


Setting WebService integrasi dengan Jacoco dan Sonarqube

Pada bab ini akan menggunakan kembali *code* yang telah dibuat *unit testing* pada bab sebelumnya yang focus pada unit testing *service layer* dan *controller layer*.

Start Local Server Sonarqube

Setelah *Sonarqube* di unduh kemudian ekstrak ke dalam folder. Untuk menjalankan server sonarqube masuk ke direktori sonarqube di **bin/** , disana terdapat tipe dari sonarqube sesuai dengan system operasi, karena penulis menggunakan system operasi windows, maka penulis memilih folder **windows-x86-64**.



Klik file **StartSonar.bat** dan akan muncul window berupa command line untuk melihat log dari server sonarqube.

wrapper | --> Wrapper Started as Console

wrapper | Launching a JVM...

jvm 1 | Wrapper (Version 3.2.3) http://wrapper.tanukisoftware.org

jvm 1 | Copyright 1999-2006 Tanuki Software, Inc. All Rights Reserved.

jvm 1 |

jvm 1 | 2020.03.02 13:12:16 INFO app[][o.s.a.AppFileSystem] Cleaning or creating temp directory D:\DEV_SOFTWARE\sonarqube-7.9.1\sonarqube-7.9.1\temp

jvm 1 | 2020.03.02 13:12:16 INFO app[][o.s.a.es.EsSettings] Elasticsearch listening on /127.0.0.1:9001

jvm 1 | 2020.03.02 13:12:16 INFO app[][o.s.a.ProcessLauncherImpl] Launch process[[key='es', ipcIndex=1, logFilenamePrefix=es]] from [D:\DEV_SOFTWARE\sonarqube-7.9.1\sonarqube-7.9.1\elasticsearch]: C:\Program Files\AdoptOpenJDK\jdk-11.0.5.10-hotspot\bin\java -XX:+UseConcMarkSweepGC -XX:CMSInitiatingOccupancyFraction=75 -XX:+UseCMSInitiatingOccupancyOnly -Des.networkaddress.cache.ttl=60 -Des.networkaddress.cache.negative.ttl=10 -XX:+AlwaysPreTouch -Xss1m -Djava.awt.headless=true -Dfile.encoding=UTF-8 -Djna.nosys=true -XX:-OmitStackTraceInFastThrow -Dio.netty.noUnsafe=true -Dio.netty.noKeySetOptimization=true -Dio.netty.recycler.maxCapacityPerThread=0 -Dlog4j.shutdownHookEnabled=false -Dlog4j2.disable.jmx=true -Djava.io.tmpdir=D:\DEV_SOFTWARE\sonarqube-7.9.1\sonarqube-7.9.1\temp -XX:ErrorFile=../logs/es_hs_err_pid%p.log -Xms512m -Xmx512m -XX:+HeapDumpOnOutOfMemoryError -Delasticsearch -Des.path.home=D:\DEV_SOFTWARE\sonarqube-7.9.1\sonarqube-7.9.1\elasticsearch -Des.path.conf=D:\DEV_SOFTWARE\sonarqube-7.9.1\sonarqube-7.9.1\temp\conf\es -cp lib/* org.elasticsearch.bootstrap.Elasticsearch

jvm 1 | 2020.03.02 13:12:16 INFO app[][o.s.a.SchedulerImpl] Waiting for Elasticsearch to be up and running

jvm 1 | OpenJDK 64-Bit Server VM warning: Option UseConcMarkSweepGC was deprecated in version 9.0 and will likely be removed in a future release.

jvm 1 | 2020.03.02 13:12:16 INFO app[][o.e.p.PluginsService] no modules loaded

jvm 1 | 2020.03.02 13:12:16 INFO app[][o.e.p.PluginsService] loaded plugin [org.elasticsearch.transport.Netty4Plugin]

jvm 1 | 2020.03.02 13:12:32 INFO app[][o.s.a.SchedulerImpl] Process[es] is up

wrapper | Wrapper Process has not received any CPU time for 27 seconds. Extending timeouts.

jvm 1 | 2020.03.02 13:12:32 INFO app[][o.s.a.ProcessLauncherImpl] Launch process[[key='web', ipcIndex=2, logFilenamePrefix=web]] from [D:\DEV_SOFTWARE\sonarqube-7.9.1\sonarqube-7.9.1]: C:\Program Files\AdoptOpenJDK\jdk-11.0.5.10-hotspot\bin\java -Djava.awt.headless=true -Dfile.encoding=UTF-8 -Djava.io.tmpdir=D:\DEV_SOFTWARE\sonarqube-7.9.1\sonarqube-7.9.1\temp --add-opens=java.base/java.util=ALL-UNNAMED --add-opens=java.base/java.lang=ALL-UNNAMED --add-opens=java.base/java.io=ALL-UNNAMED --add-opens=java.rmi/sun.rmi.transport=ALL-UNNAMED -Xmx512m -Xms128m -XX:+HeapDumpOnOutOfMemoryError -Dhttp.nonProxyHosts=localhost|127.*|[::1] -cp ./lib/common/*;D:\DEV_SOFTWARE\sonarqube-7.9.1\sonarqube-7.9.1\lib\jdbc\h2\h2-1.3.176.jar org.sonar.server.app.WebServer D:\DEV_SOFTWARE\sonarqube-7.9.1\sonarqube-7.9.1\temp\sq-process9553224268248928809properties

jvm 1 | 2020.03.02 13:13:00 INFO app[][o.s.a.SchedulerImpl] Process[web] is up

jvm 1 | 2020.03.02 13:13:00 INFO app[][o.s.a.ProcessLauncherImpl] Launch process[[key='ce', ipcIndex=3, logFilenamePrefix=ce]] from [D:\DEV_SOFTWARE\sonarqube-7.9.1\sonarqube-7.9.1]: C:\Program Files\AdoptOpenJDK\jdk-11.0.5.10-hotspot\bin\java -Djava.awt.headless=true -Dfile.encoding=UTF-8 -Djava.io.tmpdir=D:\DEV_SOFTWARE\sonarqube-7.9.1\sonarqube-7.9.1\temp

--add-opens=java.base/java.util=ALL-UNNAMED -Xmx512m -Xms128m -XX:+HeapDumpOnOutOfMemoryError -Dhttp.nonProxyHosts=localhost|127.*|[::1] -cp ./lib/common/*;D:\DEV_SOFTWARE\sonarqube-7.9.1\sonarqube-7.9.1\lib\jdbc\h2\h2-1.3.176.jar org.sonar.ce.app.CeServer D:\DEV_SOFTWARE\sonarqube-7.9.1\sonarqube-7.9.1\temp\sq-process133967801889261142000properties

jvm 1 | 2020.03.02 13:13:09 INFO app[][o.s.a.SchedulerImpl] Process[ce] is up

jvm 1 | 2020.03.02 13:13:09 INFO app[][o.s.a.SchedulerImpl] SonarQube is up

pada potongan log diatas di baris terakhir, terlihat bahwa server sonarqube sudah berhasil running.

Tambahkan Jacoco Plugin pada file Pom.xml

Tambahkan propertis berikut dalam file pom.xml project.

```
<jacoco.version>0.8.3</jacoco.version>
<sonar.java.coveragePlugin>jacoco</sonar.java.coveragePlugin>
<sonar.dynamicAnalysis>reuseReports</sonar.dynamicAnalysis>
<sonar.jacoco.reportPath>${project.basedir}/../target/jacoco.exec</sonar.jacoco.
reportPath>
<sonar.language>java</sonar.language>
```

Setelah itu, masih dalam file pom.xml tambahkam plugin **Jacoco** didalam tag *plugin.*

```
<plugin>
    <groupId>org.jacoco</groupId>
    <artifactId>jacoco-maven-plugin</artifactId>
    <version>${jacoco.version}</version>
    <configuration>
        <skip>${maven.test.skip}</skip>
        <destFile>${basedir}/target/coverage-reports/jacoco-unit.exec</destFile>
        <dataFile>${basedir}/target/coverage-reports/jacoco-unit.exec</dataFile>
        <output>file</output>
        <append>true</append>
        <excludes>
            <exclude>*MethodAccess</exclude>
        </excludes>
    </configuration>
    <executions>
        <execution>
            <id>jacoco-initialize</id>
            <goals>
                <goal>prepare-agent</goal>
            </goals>
            <phase>test-compile</phase>
        </execution>
        <execution>
            <id>jacoco-site</id>
            <phase>verify</phase>
            <goals>
                <goal>report</goal>
            </goals>
```

```
            </execution>
        </executions>
    </plugin>
```

Full *code* file pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<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
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.12.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>id.learn</groupId>
    <artifactId>webservices-restful</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>webservices-restful</name>
    <description>Demo project for Spring Boot</description>

    <properties>
        <java.version>1.8</java.version>
        <problem-spring-web.version>0.25.0</problem-spring-web.version>

        <jacoco.version>0.8.3</jacoco.version>
        <sonar.java.coveragePlugin>jacoco</sonar.java.coveragePlugin>
        <sonar.dynamicAnalysis>reuseReports</sonar.dynamicAnalysis>

<sonar.jacoco.reportPath>${project.basedir}/../target/jacoco.exec</sonar.jacoco.
reportPath>
        <sonar.language>java</sonar.language>

    </properties>

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

        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
```

```
            <optional>true</optional>
        </dependency>

        <!-- Random String generator -->
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-lang3</artifactId>
            <version>3.9</version>
        </dependency>

        <!-- For Testing-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
            <exclusions>
                <exclusion>
                    <groupId>junit</groupId>
                    <artifactId>junit</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-engine</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-params</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.junit.platform</groupId>
            <artifactId>junit-platform-launcher</artifactId>
            <version>1.3.2</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.junit.vintage</groupId>
            <artifactId>junit-vintage-engine</artifactId>
            <version>5.3.2</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.mockito</groupId>
            <artifactId>mockito-junit-jupiter</artifactId>
            <scope>test</scope>
        </dependency>

        <!-- mapper -->
        <dependency>
            <groupId>org.zalando</groupId>
            <artifactId>problem-spring-web-starter</artifactId>
            <version>${problem-spring-web.version}</version>
            <type>pom</type>
        </dependency>
    </dependencies>
```

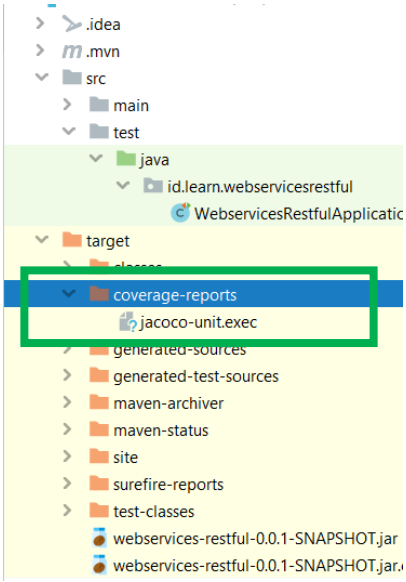
```
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
            <plugin>
                <groupId>org.jacoco</groupId>
                <artifactId>jacoco-maven-plugin</artifactId>
                <version>${jacoco.version}</version>
                <configuration>
                    <skip>${maven.test.skip}</skip>
                    <destFile>${basedir}/target/coverage-reports/jacoco-
unit.exec</destFile>
                    <dataFile>${basedir}/target/coverage-reports/jacoco-
unit.exec</dataFile>
                    <output>file</output>
                    <append>true</append>
                    <excludes>
                        <exclude>*MethodAccess</exclude>
                    </excludes>
                </configuration>
                <executions>
                    <execution>
                        <id>jacoco-initialize</id>
                        <goals>
                            <goal>prepare-agent</goal>
                        </goals>
                        <phase>test-compile</phase>
                    </execution>
                    <execution>
                        <id>jacoco-site</id>
                        <phase>verify</phase>
                        <goals>
                            <goal>report</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>

</project>
```

Konfigurasi diatas adalah konfigurasi yang dibutuhkan jacoco untuk menggenerate *codecoverage*. Buka terminal atau *command prompt* dan *build* projeknya dengan sintak berikut dan tunggu sampai proses *build* selesai.

mvn clean install

```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  25.581 s
[INFO] Finished at: 2020-03-02T13:20:39+07:00
[INFO] ------------------------------------------------------------------------
```

Kembali ke IDE, setelah projek di *build* lihat pada struktur folder nya di bawah folder ***target,*** terdapat folder ***coverage-reports*** yang berisi ***jacoco-unit.exec***. file itulah yang nantinya akan dibaca oleh *Sonarqube* untuk menginspeksi *code* yang sudah dibuat.



Integrasi Jacoco dan Sonarqube

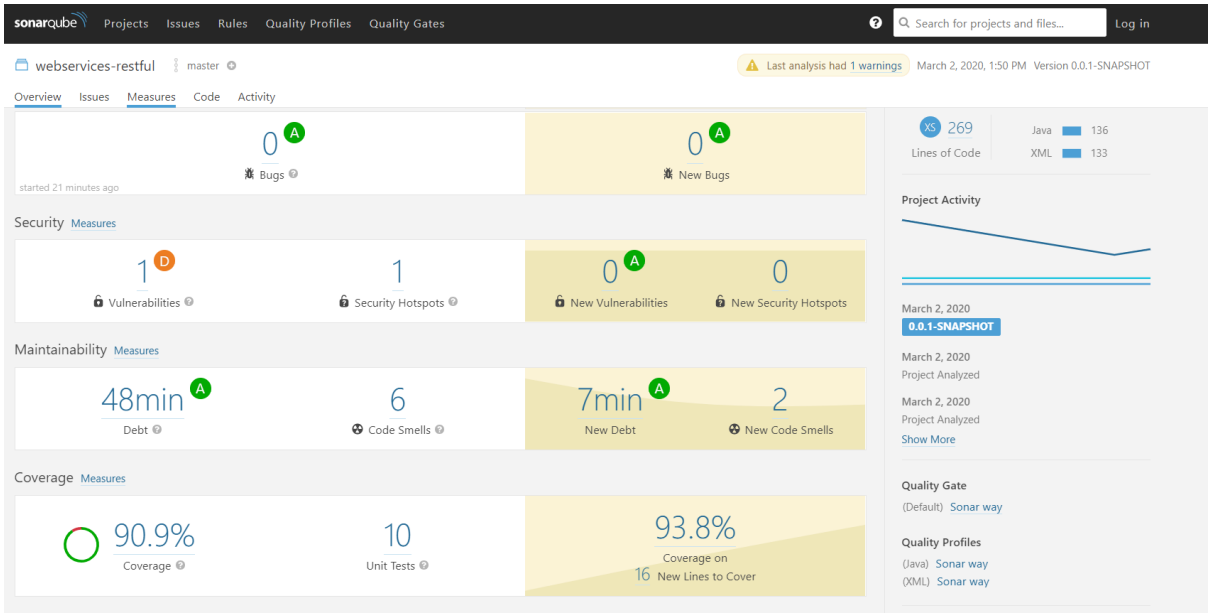
Masih dalam terminal atau *command prompt* yang sama, kemudian ketikan sintak berikut untuk melakukan generate code quality.

mvn sonar:sonar

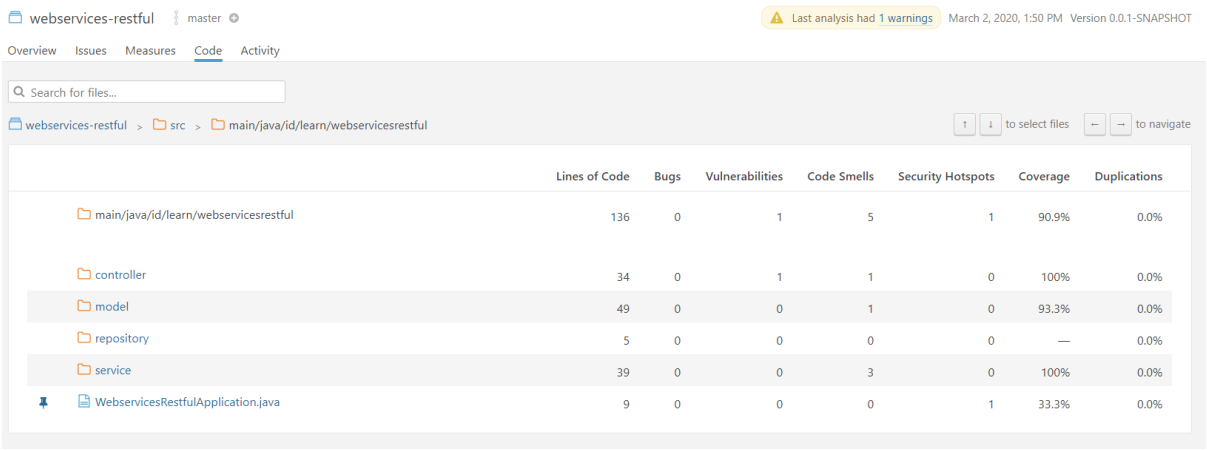
tunggu sampai selesai, sampai keluar link yang bisa kita gunakan untuk mengakses *codecoverage report.*

```
[INFO] Sensor XML Sensor [xml] (done) | time=149ms
[INFO] 1/1 source files have been analyzed
[INFO] ------------ Run sensors on project
[INFO] Sensor Zero Coverage Sensor
[INFO] Sensor Zero Coverage Sensor (done) | time=4ms
[INFO] Sensor Java CPD Block Indexer
[INFO] Sensor Java CPD Block Indexer (done) | time=57ms
[INFO] No SCM system was detected. You can use the 'sonar.scm.provider' property to explicitly specify it.
[INFO] 5 files had no CPD blocks
[INFO] Calculating CPD for 4 files
[INFO] CPD calculation finished
[INFO] Analysis report generated in 109ms, dir size=100 KB
[INFO] Analysis report compressed in 88ms, zip size=20 KB
[INFO] Analysis report uploaded in 622ms
[INFO] ANALYSIS SUCCESSFUL, you can browse http://localhost:9000/dashboard?id=id.learn%3Awebservices-restful
[INFO] Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
[INFO] More about the report processing at http://localhost:9000/api/ce/task?id=AXC272kOfHMaRK3MSnID
[INFO] Analysis total time: 9.691 s
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  02:06 min
[INFO] Finished at: 2020-03-02T13:29:15+07:00
[INFO] ------------------------------------------------------------------------
```

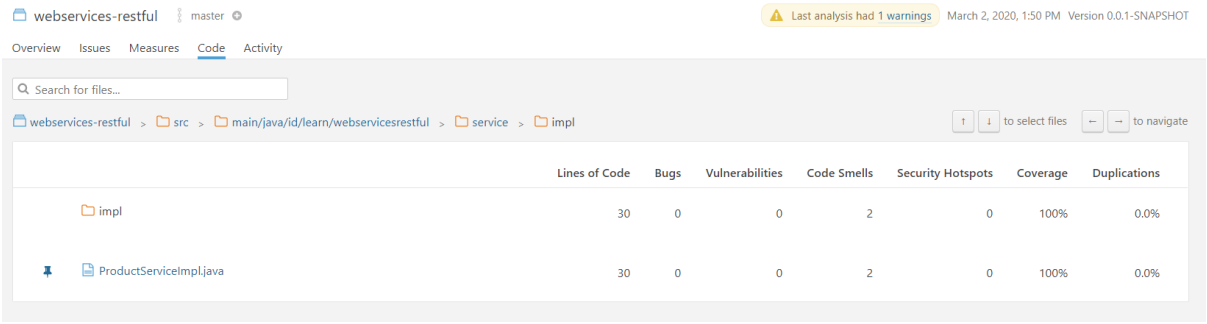
Copy link tersebut kemudian *paste* pada browser.

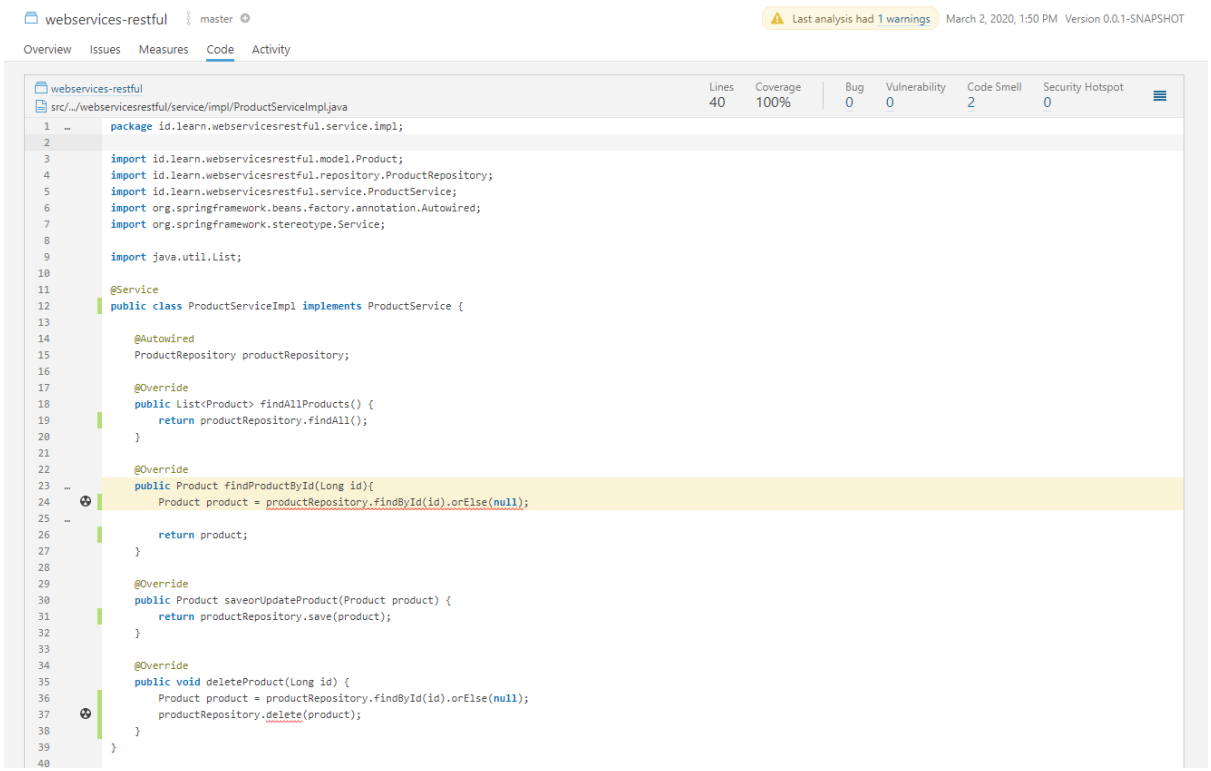


Pada *windows* tersebut kita bisa melihat informasi dari *code* yang telah ditulis, mulai dari *security, maintainability* dan ***coverage***. Terlihat *codecoverage* bernilai 90.9% yang artinya bahwa kode kita sudah bisa dikatakan berkualitas. Jika angka tersebut di klik maka akan masuk ke halaman detail nya.



Dan untuk melihat lebih detail lagi, bisa menekan folder-folder diatas misalkan melihat *codecoverage* pada class **ProductServiceImpl**.




webservices-restful
master
Last analysis had 1 warnings
March 2, 2020, 1:50 PM Version 0.0.1-SNAPSHOT
Overview
Issues
Measures
Code
Activity
webservices-restful
src/.../webservicesrestful/service/impl/ProductServiceImpl.java
Lines 40
Coverage 100%
Bug 0
Vulnerability 0
Code Smell 2
Security Hotspot 0
1 package id.learn.webservicesrestful.service.impl;
2
3 import id.learn.webservicesrestful.model.Product;
4 import id.learn.webservicesrestful.repository.ProductRepository;
5 import id.learn.webservicesrestful.service.ProductService;
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.stereotype.Service;
8
9 import java.util.List;
10
11 @Service
12 public class ProductServiceImpl implements ProductService {
13
14 @Autowired
15 ProductRepository productRepository;
16
17 @Override
18 public List<Product> findAllProducts() {
19 return productRepository.findAll();
20 }
21
22 @Override
23 public Product findProductById(Long id){
24 Product product = productRepository.findById(id).orElse(null);
25
26 return product;
27 }
28
29 @Override
30 public Product saveorUpdateProduct(Product product) {
31 return productRepository.save(product);
32 }
33
34 @Override
35 public void deleteProduct(Long id) {
36 Product product = productRepository.findById(id).orElse(null);
37 productRepository.delete(product);
38 }
39 }
40

Spring AOP (*Abstract Oriented Programming*)

Pengertian AOP

AOP adalah singkatan dari *Aspect Oriented Programming* yaitu sebuah paradigm yang memungkinkan kita untuk melakukan *cross-cutting-concern. cross-cutting-concern* adalah fungsi-fungsi umum yang dibutuhkan oleh aplikasi tapi tidak ada hubungan dengan proess bisnis pada aplikasi tersebut. Adapun contoh-contoh *cross-cutting-concern* diantaranya :

- Performance Monitoring
- Logging
- Transaction Management
- Caching
- Security
- Handling Error
- Dan Lain-lain.

Namun dalam bab ini kita akan menggunakan untuk membuat *Logger* pada layer *Controller*. Seperti yang sudah kita tahu bahwa ketika kita membuat Logger secara Manual menggunakan

```
private final Logger log = LoggerFactory.getLogger(this.getClass());
```

sangatlah merepotkan, setiap log harus di definisikan pada setiap method yang ingin diketahui dalam Log. Dengan menggunakan AOP maka proses ini akan kita rubah menjadi otomatis, jadi kita tidak perlu membuat Logger disetiap method nya. Namun sebelumnya tambahkan library AOP terlebih dahulu.

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

Dan tambahkan konfigurasi untuk Logging nya pada file *application.properties.*

id.learn.webserverservicerestful.aspect.LoggingAspect

```
…

#Logging Config
logging.level.org.springframework.web=INFO
logging.level.org.hibernate=ERROR
logging.level.id.learn.webservicesrestful=DEBUG
```

Membuat LoggingAspect

Class *LoggingAspect* nantinya akan kita gunakan untuk membuat Logger pada setiap class yang berada dalam layer Controller. Terdapat beberapa istilah yang harus kita pahami dalam AOP, yaitu :

- *Joinpoint* adalah suatu titik pengeksekusikan pada aplikasi, misalnya pada saat pemanggilan method, inisialisasi class, field assignment atau inisiasi object
- *Advice* adalah kode yang akan dieksekusi oleh *Joinpoint*. Ada beberapa macam *Joinpoint* yaitu *before advice* dan *after advice*.
- *Pointcut* adalah kumpulan beberapa *Joinpoint* yang mendefinisikan kapan *Advice* akan dijalankan.
- *Aspect* adalah kombinasi antara *Advice* dan *Pointcut*. Kombinasi ini yang nantinya akan menghasilkan *logic* yang harus dieksekusi oleh aplikasi.
- *Target* adalah objek yang dimodifikasi oleh AOP.

id.learn.webserserviceresful.aspect.LoggingAspect

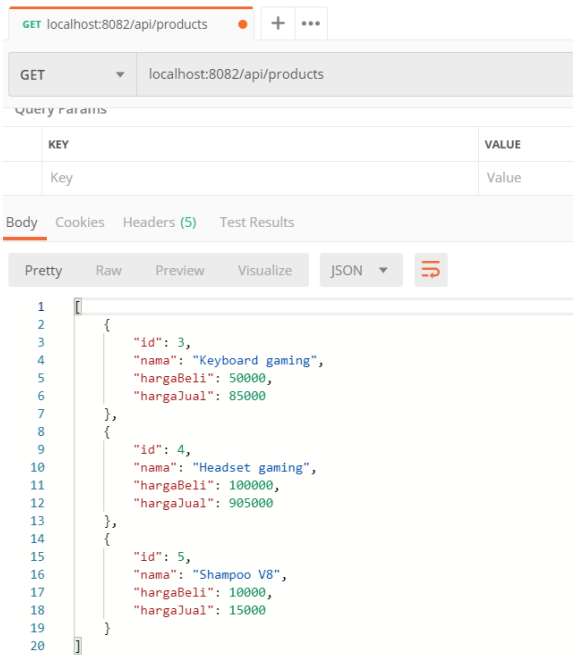
```
package id.learn.webservicesrestful.aspect;

import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Pointcut;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Component;

import java.util.Arrays;

@Aspect
@Component
public class LoggingAspect {

    private final Logger log = LoggerFactory.getLogger(this.getClass());


    @Pointcut("within(@org.springframework.stereotype.Repository *)" +
            " || within(@org.springframework.stereotype.Service *)" +
            " || within(@org.springframework.web.bind.annotation.RestController
*)")
    public void springBeanPointcut() {

    }

        @Pointcut("within(id.learn.webservicesrestful.controller..*)")
    public void applicationPackagePointcut() {

    }

        @Around("applicationPackagePointcut() && springBeanPointcut()")
    public Object logAround(ProceedingJoinPoint joinPoint) throws Throwable {
        if (log.isDebugEnabled()) {
            log.debug("Enter: {}.{}() with argument[s] = {}",
joinPoint.getSignature().getDeclaringTypeName(),
                    joinPoint.getSignature().getName(),
Arrays.toString(joinPoint.getArgs()));
        }
        try {
            Object result = joinPoint.proceed();
```

```
            if (log.isDebugEnabled()) {
                log.debug("Exit: {}.{}() with result = {}",
joinPoint.getSignature().getDeclaringTypeName(),
                        joinPoint.getSignature().getName(), result);
            }
            return result;
        } catch (IllegalArgumentException e) {
            log.error("Illegal argument: {} in {}.{}()",
Arrays.toString(joinPoint.getArgs()),
                    joinPoint.getSignature().getDeclaringTypeName(),
joinPoint.getSignature().getName());
            throw e;
        }
    }
}
```

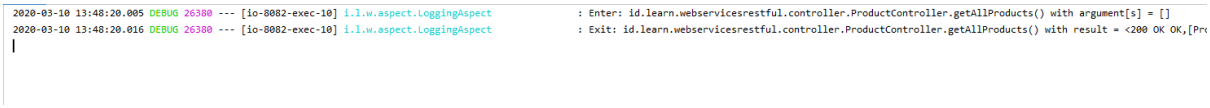
Penjelasan method-method diatas yaitu :

- *springBeanPointCut()* adalah *Pointcut* yang mencocokan untuk semua repository, services atau controller
- *applicationPackagePointcut()* adalah *Pointcut* yang mencocokan dengan semua *Spring Beans* dalam package *main*.
- *logAround()* adalah *Advice* yang menghasilkan Logger ketika memasuki dan keluar dari sebuah method.

Lalu kita jalankan aplikasi dan lakukan *hit* menggunakan Postman Client.



Lalu kita cek pada Log aplikasi, jika seperti pada gambar di bawah ini,maka Logger menggunakan AOP berhasil dilakukan.



2020-03-10 13:48:20.005 DEBUG 26380 --- [io-8082-exec-10] i.l.w.aspect.LoggingAspect : Enter: id.learn.webservicesrestful.controller.ProductController.getAllProducts() with argument[s] = []

2020-03-10 13:48:20.016 DEBUG 26380 --- [io-8082-exec-10] i.l.w.aspect.LoggingAspect : Exit: id.learn.webservicesrestful.controller.ProductController.getAllProducts() with result = <200 OK OK,[Product(id=3, nama=Keyboard gaming, hargaBeli=50000, hargaJual=85000), Product(id=4, nama=Headset gaming, hargaBeli=100000, hargaJual=905000), Product(id=5, nama=Shampoo V8, hargaBeli=10000, hargaJual=15000)],[]>

Deploy Webservice menggunakan Docker dan Docker-compose

Docker adalah sebuah tools containe yang bersifat opensource yang ditujukan untuk para *developer* atau *sysadmin* untuk mengemas, membangun dan menjalankan aplikasi dimanapun di dalam sebuah container sesuai dengan konfigurasi yang diinginkan.

Dulu penulis sebelum mengenal docker, penulis selalu menggunakan Vagrant untuk development. Jika teman-teman belum tahu, Vagrant adalah sebuah tool virtualisasi yang menggunakan Oracle Virtual Box. Jadi menjalankan OS didalam OS dan sudah pasti sangat banyak memakan *memory*. Beberapa perbedaan dapat dilihat pada table berikut .

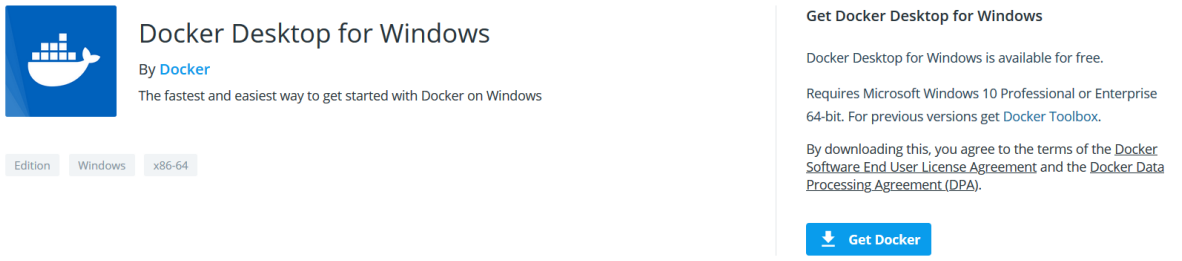
	Docker	Vagrant
Virtualization	Linux container	Virtual machine
Resource isolation	Weak	Strong
OS	Linux	Linux, Windows, MacOS, ...
Starting time	Seconds	Minutes
Size	100M+	1G+
CM integration	No	Yes
Building image time	Short (mins)	Long (10+ mins)
Hosted number	>50	<10
Deployment tools	CoreOS, Mesos,...	Terraform
Public images	Yes (Docker Hub)	Yes (Vagrant Cloud)

Dari table diatas bisa kita lihat banyak sekali perbedaan antara Docker dan Vagrant. Adapun perbedaan yang sangat mencolok adalah pada pemakaian *resource* atau *memory*. Docker cenderung menggunakan *memory* lebih sedikit daripada *Vagrant.* Jika menggunakan Vagrant maka kita wajib menginstall virtual machine seperti Oracle Virtualbox atau VMware, berbeda dengan docker yang hanya menggunakan Linux container sehingga tidak perlu menggunakan virtual machine.

Mengunduh Docker

Docker tersedia dengan berbagai variant sesuai dengan Sistem Operasi seperti Windows 10, Mac OS dan Linux. Pada bab ini penulis menggunakan versi Docker untuk windows 10, namun apabila teman-teman menggunakan Sistem Operasi yang berbeda bisa akses langsung disitus mengenai cara installasinya. Disana sudah terdapat cara-cara instalasi sesuai dengan Sistem Operasi.

Unduh Docker Installer pada alamat ini https://hub.docker.com/editions/community/docker-ce-desktop-windows/ dengan ukuran file kurang dari 1GB kemudian klik tombol Get Docker.



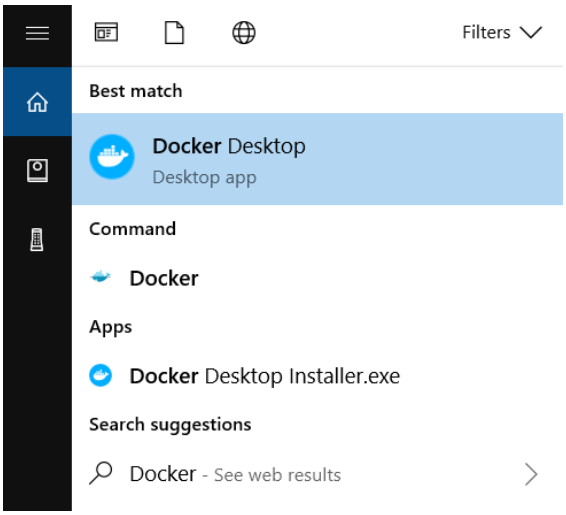
Adapun spesifikasi yang diperlukan untuk dapat menggunakan Docker adalah sebagai berikut.

- Versi untuk windows yaitu minimal harus menggunakan Windows 10 64Bit dengan versi Pro, Enterprise atau Education
- Fitur Hyper-V dan Container Windows harus *enable*.
- Minimal RAM atau *memory* sebesar 4GB.
- Pada BIOS, *Hardware Virtualization* harus di *enable.*

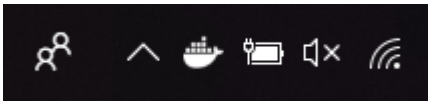
Langkah selanjutnya yaitu melakukan proses instalasi Docker.

- Tekan dua kali pada file Docker installer yang telah selesai diunduh.
- Ikuti instruksi sesuai dengan tampilan *wizard* seperti *accept license, authorize the installer* dan lain-lain.
- Ketika windows popup, izinkan Docker untuk menggunakan password pada computer Anda. Hal ini digunakan Docker untuk melakukan instalasi komponen *networking*, membuat link ke desktop dan mengatur fitur Hyper-V VM.
- Setelah selesai klik **Finish**

Untuk memulai docker, klik icon docker pada Dekstop atau cari Docker pada hasi pencarian sepertin ini.



Tunggu sampai proses *loading* nya selesai,



Setelah itu untuk mengecek kita bisa menggunakan sintak berikut untuk memastikan docker sudah *running* atau belum.

docker version

Apabila response nya seperti dibawah ini maka Docker telah berhasil berjalan

Client: Docker Engine - Community

Version:	19.03.5
API version:	1.40
Go version:	go1.12.12
Git commit:	633a0ea
Built:	Wed Nov 13 07:22:37 2019
OS/Arch:	windows/amd64
Experimental:	false

Menyiapkan Image untuk Deploy Aplikasi

Sebelum dapat mengunduh image, terlebih dahulu kita harus terdaftar pada Docker Hub. Docker Hub adalah sebuah marketplace Docker Images. Disana kita bisa mengunduh image seperti JDK, Maven, MySQL dan lain-lain yang kita perlukan untuk menjalankan aplikasi. *Sign Up* di situs nya https://hub.docker.com/ setelah berhasil lakukan *Sign In* pada Docker Hub dan Docker Local yang sudah terinstall.

Karena aplikasi yang akan dideploy berbasis Java, maka imags yang akan digunakan adalah :

- OpenJDK Versi 11
- MySQL 5.7.28

Untuk itu, lakukan proses unduh images menggunakan terminal atau *command prompt* pada local computer menggunakan sintak berikut.

docker pull openjdk:11
docker pull mysql:5.7.28

jika kedua images tersebut sudah berhasil diunduh, lakukan pengecekan dengan sintak

docker images

dan hasilnya docker images yang tadi diunduh telah berhasil terdaftar.

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
users-mysql	latest	9ae4d9e25dcf	4 days ago	516MB
maven	3.6.3-jdk-11	e378090eb05d	3 weeks ago	638MB
logstash	7.6.0	799d4bde3bdd	3 weeks ago	805MB
kibana	7.6.0	b36db011e72c	3 weeks ago	1.01GB
elasticsearch	7.6.0	5d2812e0e41c	3 weeks ago	790MB
sonarqube	7.9.2-community	0956756d5b96	4 weeks ago	483MB
kong	alpine	c60ce2ccc6b3	3 months ago	129MB
maven	slim	7eedc6c1832f	3 months ago	419MB
postgres	9.6.16	47b69eada093	3 months ago	250MB
haproxy	latest	26a3b0a2daac	3 months ago	92.2MB
couchbase	latest	45225cae486f	3 months ago	964MB
golang	1.13.4-stretch	d30a8b8b92e2	3 months ago	763MB
openjdk	8	09df0563bdfc	3 months ago	488MB
openjdk	8u232-stretch	09df0563bdfc	3 months ago	488MB
openjdk	11	243e95d792e3	3 months ago	605MB
openjdk	11.0.5-stretch	243e95d792e3	3 months ago	605MB
redis	latest	dcf9ec9265e0	3 months ago	98.2MB
mysql	5.7.28	1e4405fe1ea9	3 months ago	437MB
nginx	latest	231d40e811cd	3 months ago	126MB
cassandra	latest	5c4da61080b6	3 months ago	324MB
consul	latest	61c55d0793c6	3 months ago	117MB
envoyproxy/envoy-alpine	latest	137714143dc8	3 months ago	40.9MB
mongo	latest	965553e202a4	4 months ago	363MB
jenkins	alpine	2ad007d33253	2 years ago	223MB

Proses Deploy Aplikasi

Untuk melakukan proses deploy ada beberapa hal yang harus dilakukan diantaranya melakukan profiling application properties khususnya yang berhubungan dengan koneksi ke database, membuat Dockerfile untuk membuat images dan membuat Docker-compose untuk mengautomasi proses *mounting* docker container.

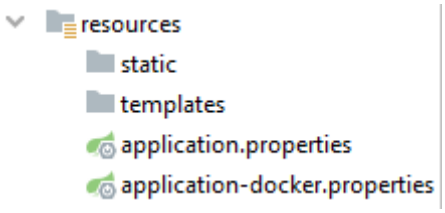
Merubah Profile Aplikasi

Pada bab ini juga masih menggunakan *sourcecode* webrestful terakhir yang kita gunakan untuk membuat *codecoverage* pada bab sebelumnya. Buka *sourcecode* nya kembali menggunakan IDE, dan lakukan langkah-langkah sebagai berikut :

a. Pada directory ***src/resources*** buat satu buah profiles application.properties dengan nama

   application-docker.properties

   sehingga akan ada dua file profile yaitu default dan docker.



application.properties, digunakan untuk *pointing* ke koneksi database yang ada di local atau untuk kebutuhan *development*
application-docker.properties, digunakan untuk *pointing* ke koneksi database yang ada didocker container yang akan dibuat di tahap selanjutnya.

b. Edit file ***application-docker.properties*** menjadi seperti ini.

application-docker.properties

```
#Spring Profile
spring.profiles.include=docker

#MySQL Connection
spring.datasource.url=jdbc:mysql://dbcontainer:3306/webrestful
spring.datasource.username=sa
spring.datasource.password=webrestful

#HikariCP
spring.datasource.hikari.connection-timeout=20000
spring.datasource.hikari.minimum-idle=5
spring.datasource.hikari.maximum-pool-size=12
spring.datasource.hikari.idle-timeout=300000
spring.datasource.hikari.max-lifetime=120000
spring.datasource.hikari.auto-commit=true

#JPA Properties
spring.jpa.database-platform=org.hibernate.dialect.MySQL5InnoDBDialect
spring.jpa.hibernate.ddl-auto=update
spring.datasource.driverClassName=com.mysql.jdbc.Driver

server.port=8082
```

Terlihat diatas pada bagian ***url*** menggunakan container **`dbcontainer`** dan database **`webrestful`** yang akan dibuat pada tahap berikutnya.

Membuat Docker Network

Kita akan membuat sebuah docker network dengan nama **`mysql-webrestful`** yang nantinya akan digunakan untuk menghubungkan aplikasi yang sudah di *deploy* dengan database mysql dari container yang sudah dimounting. Ketik sintak berikut.

docker network create mysql-webrestful

jika berhasil maka docker akan mengembalikan hash dari network tersebut

```
λ docker network create mysql-webrestful
80dcd0d1f5bcb062e130148a6b58ccf02785a3abaa6a0833168320527436a914
```

Untuk mengecek docker network yang telah dibuat, ketikan sintak berikut:

docker network list

docker akan mengeluarkan list network yang sudah dibuat

```
λ docker network list
NETWORK ID          NAME                            DRIVER              SCOPE
02dcc8d33c5a        bridge                          bridge              local
b04a3cd7051e        employee-jdbc_employee-mysql    bridge              local
aaf5d37b28a6        employee-mysql                  bridge              local
2fe069b1d4c3        employee-mysql-docker           bridge              local
f1c8c42c075c        host                            host                local
                    mysql_test_default              bridge              local
80dcd0d1f5bc        mysql-webrestful                bridge              local
18321b95e822        none                            null                local
```

Membuat File .sh untuk Container DependOn

Biasanya dengan konfigurasi *default,* ketika mengeksekusi *docker-compose up* maka semua image akan di *mount* secara berbarengan atau *parallel.* Untuk versi docker-compose 3.3 ke atas walau menggunakan tag *depends_on* tapi tetap tidak dapat menggunakan *container dependensi*. Untuk itu sesuai dengan saran dari situs *docker* kita akan memanipulasi dengan menggunakan file .sh. pada *root* buat sebuah file ***wait-for.sh.***

wait-for.sh

```
#!/usr/bin/env sh

if [ "$#" -ne 1 ]; then
    >&2 echo "Usage: $0 host:port"
    exit -1
fi

ARGUMENT=$1
HOST="$(echo $ARGUMENT | cut -d ':' -f1)"
PORT="$(echo $ARGUMENT | cut -d ':' -f2)"
MAX_RETRY=90

echo "Testing connection to host $HOST and port $PORT."

count=0
while [ $count -lt $MAX_RETRY ]
do
    count=$((count+1))
```

```
    nc -z $HOST $PORT
    result=$?
    if [ $result -eq 0 ]; then
        echo "Connection is available after $count second(s)."
        exit 0
    fi
    echo "Retrying..."
    sleep 1
done

>&2 echo "Timeout occurred after waiting $MAX_RETRY seconds for $HOST:$PORT."
exit -1
```

Membuat Dockerfile

Dockerfile adalah sebuah file yang akan dibaca oleh Docker sebagai rujukan untuk melakukan proses *deploy* ke dalam bentuk docker images. Untuk mempermudah, edit nama aplikasi nya menjadi ***webrestful*** dengan menambahkan konfigurasi dalam file pom.xml pada bagian tag plugin.

```
<plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
    <configuration>
        <finalName>webrestful</finalName>
    </configuration>
</plugin>
```

Full *code* file pom.xml

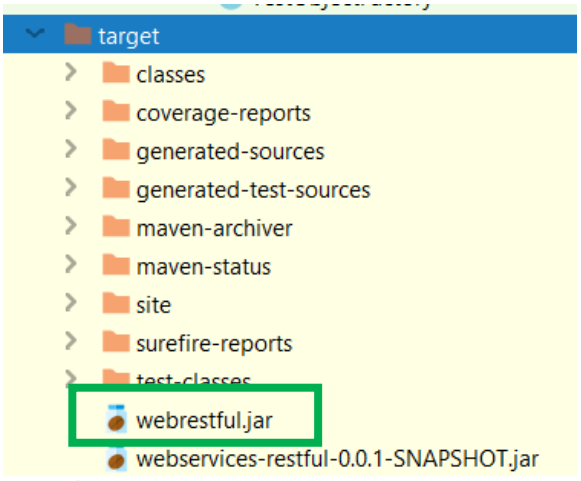
```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.12.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>id.learn</groupId>
    <artifactId>webservices-restful</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>webservices-restful</name>
    <description>Demo project for Spring Boot</description>

    <properties>
        <java.version>1.8</java.version>
        <problem-spring-web.version>0.25.0</problem-spring-web.version>

        <jacoco.version>0.8.3</jacoco.version>
        <sonar.java.coveragePlugin>jacoco</sonar.java.coveragePlugin>
        <sonar.dynamicAnalysis>reuseReports</sonar.dynamicAnalysis>

<sonar.jacoco.reportPath>${project.basedir}/../target/jacoco.exec</sonar.jacoco.
```

```
reportPath>
        <sonar.language>java</sonar.language>

    </properties>

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

        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>

        <!-- Random String generator -->
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-lang3</artifactId>
            <version>3.9</version>
        </dependency>

        <!-- For Testing-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
            <exclusions>
                <exclusion>
                    <groupId>junit</groupId>
                    <artifactId>junit</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-engine</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-params</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.junit.platform</groupId>
            <artifactId>junit-platform-launcher</artifactId>
            <version>1.3.2</version>
```

```
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.junit.vintage</groupId>
            <artifactId>junit-vintage-engine</artifactId>
            <version>5.3.2</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.mockito</groupId>
            <artifactId>mockito-junit-jupiter</artifactId>
            <scope>test</scope>
        </dependency>

        <!-- mapper -->
        <dependency>
            <groupId>org.zalando</groupId>
            <artifactId>problem-spring-web-starter</artifactId>
            <version>${problem-spring-web.version}</version>
            <type>pom</type>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <finalName>webrestful</finalName>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.jacoco</groupId>
                <artifactId>jacoco-maven-plugin</artifactId>
                <version>${jacoco.version}</version>
                <configuration>
                    <skip>${maven.test.skip}</skip>
                    <destFile>${basedir}/target/coverage-reports/jacoco-
unit.exec</destFile>
                    <dataFile>${basedir}/target/coverage-reports/jacoco-
unit.exec</dataFile>
                    <output>file</output>
                    <append>true</append>
                    <excludes>
                        <exclude>*MethodAccess</exclude>
                    </excludes>
                </configuration>
                <executions>
                    <execution>
                        <id>jacoco-initialize</id>
                        <goals>
                            <goal>prepare-agent</goal>
                        </goals>
                        <phase>test-compile</phase>
                    </execution>
                    <execution>
                        <id>jacoco-site</id>
                        <phase>verify</phase>
```

```
                <goals>
                    <goal>report</goal>
                </goals>
            </execution>
        </executions>
    </plugin>
</plugins>
</build>

</project>
```

Selanjutnya, build project nya dengan sintak ***mvn clean install*** dan tunggu sampai proses *buildnya* selesai .



File ***webrestful.jar*** selanjutnya akan dideploy ke Docker menggunakan Dockerfile. Untuk itu, buat sebuah file dengan nama ***Dockerfile*** dalam directory root aplikasi **webrestful**.

Dockerfile

```
FROM openjdk:11
ADD target/webrestful.jar webrestful.jar
ADD wait-for.sh wait-for.sh
EXPOSE 8082
ENTRYPOINT ["java", "-Dspring.profiles.active=docker","-jar", "webrestful.jar"]
```

Keterangan kode diatas :

- **FROM** images yang akan digunakan oleh aplikasi yang akan kita deploy
- **ADD** proses menambahkan file .jar local ke docker
- **EXPOSE** aplkasi yang dideploy akan *listen* dalam port 8082
- **ENTRYPOINT** adalah sintak untuk menjalankan aplikasi yang sudah dalam bentuk images. Terlihat bahwa untuk koneksi internet kita menggunakan properties aplikasi ***docker*** .

Pada terminal atau *command prompt* yang sama, ketikan sintak berikut untuk proses deploy image dengan nama **webrestful** dan tunggu sampai proses deploy berhasil.

docker build -t webrestful .

Sending build context to Docker daemon 41.7MB

Step 1/5 : FROM openjdk:11

 ---> 243e95d792e3

Step 2/5 : ADD target/webrestful.jar webrestful.jar

 ---> 9edbc46f55cc

Step 3/5 : ADD wait-for.sh wait-for.sh

 ---> 3f0cb49ed345

Step 4/5 : EXPOSE 8082

 ---> Running in 2d8c0cd24511

Removing intermediate container 2d8c0cd24511

 ---> e5c90c6c51ad

Step 5/5 : ENTRYPOINT ["java", "-Dspring.profiles.active=docker","-jar", "webrestful.jar"]

 ---> Running in a51c800fa606

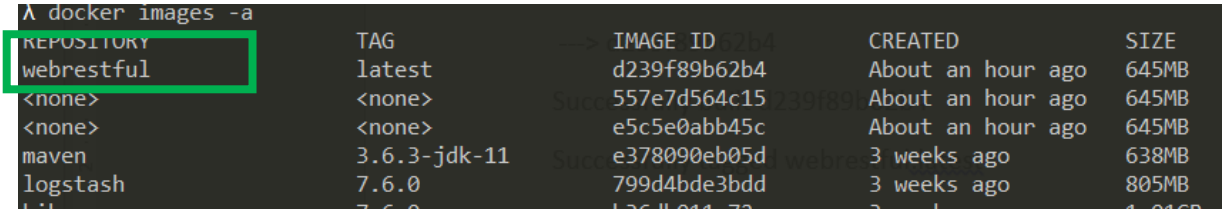
Removing intermediate container a51c800fa606

 ---> f92513300b1e

Successfully built f92513300b1e

Successfully tagged webrestful:latest

Kemudian cek apakah proses deploy sudah berhasil atau belum dengan ***docker images -a***



Membuat Docker-compose

Pada saat menjalankan mysql container, kita menjalankan secara manual, bayangkan jika ada 10 images yang akan di *mounting* ke container, berarti akan ada 10 tahapan manual yang harus kita lakukan, untuk mengotomasi proses *mounting* ini ada satu fitur yang dinamakan *docker-compose.* Dengan docker-compose, berapapun jumlah images yang akan dimounting, cukup dengan satu kali sintak yaitu.

docker-compose up

buat sebuah file dengan nama **docker-compose.yml** pada directory root aplikasi webrestful.

Docker-compose.yml

```
version: '3.3'

services:
  webrestful:
    container_name: webrestful
    build: .
    image: webrestful
    ports:
      - "8087:8082"
    command: ["-c", "wait-for.sh dbcontainer:3306 && java -jar -
Dspring.profiles.active=docker webrestful.jar"]
    networks:
      -  mysql-webrestful
    depends_on:
      - dbcontainer

  dbcontainer:
    container_name: dbwebrestful
    image: mysql:5.7.28
    networks:
      -  mysql-webrestful
    volumes:
      - db-data-mysql:/data/mysql
    environment:
      MYSQL_ROOT_PASSWORD: 'webrestful'
      MYSQL_DATABASE: 'webrestful'
      MYSQL_USER: 'sa'
      MYSQL_PASSWORD: 'webrestful'
      MYSQL_ROOT_HOST: "%"
    ports:
      - "3306:3306"

# Volumes
volumes:
  db-data-mysql:

# Networks
networks:
  mysql-webrestful:
```

Penjelasan sintak diatas adalah sebagai berikut:

- **version: "3.7"** adalah versi docker-compose yang akan digunakan
- **services** adalah daftar images yang akan di *mounting* ke dalam bentuk container, terlihat bahwa kita akan *mounting* aplikasi **webrestful** dengan **database Mysql**
- **port : - "8089:8086"** adalah port yang digunakan untuk mengatur akses dari luar supaya bisa terkoneksi dengan container webrestul yang listen di port 8086.

- **`links: - mysql-webrestful,`** image **webrestful** akan dieksekusi setelah image `mysql-webrestful` selesai di *mounting.*
- **`networks: - mysql-webrestful,`** network yang akan digunakan antara container ***webrestul*** dan ***mysql*.**
- **`volumes:`** `db-data:/var/lib/mysql` , penyimpanan yang digunakan oleh MySQL sehingga jika nantinya container mysql di unmounts terus di mount lagi, data nya tidak hilang.
- **`ports :`** terlihat bahwa kita menggunakan port 8087:8082 yang artinya bahwa kita akan mengakses aplikasi menggunakan *postman* dengan port 8087 tapi oleh docker kemudian akan diarahkan ke port 8082 dimana aplikasi *listen* pada port itu.
- **`Command`** : adalah tag yang digunakan untuk mengeksekusi file ***wait-for.sh*** yang sudah diupload didalam images *webrestful.* Sehingga dengan *command* ini bisa menggunakan *dependensi container* yaitu container *webrestful* akan menunggu sampai *container dbcontainer* selesai *up*.

Ketik **docker-compose up -d** dan tunggu sampai proses *mounting* nya selesai.

```
λ docker-compose up -d
Creating network "webservices-restful_mysql-webrestful" with the default driver
Creating dbwebrestful ... done
Creating webrestful   ... done
```

apabila dicek dengan **docker ps –a** terlihat bahwa ada dua container yang berhasil di mounting.

```
λ docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                                NAMES
4e3c4df5eff1        webrestful          "java -Dspring.profi…"   43 seconds ago      Up 42 seconds       0.0.0.0:8087->8082/tcp               webrestful
c207df253301        mysql:5.7.28        "docker-entrypoint.s…"   44 seconds ago      Up 43 seconds       0.0.0.0:3306->3306/tcp, 33060/tcp    dbwebrestful
```

Test Dengan Postman Client

Terakhir lakukan pengetesan aplikasi menggunakan *postman* dan jangan lupa untuk mengakses menggunakan port **8087.**

Get_All	GET	localhost:8087/api/products
SaveOrUpdate	POST	localhost:8087/api/products
GET_One	GET	localhost:8087/api/products/{id}
Delete	DELETE	localhost:8087/api/products/{id}

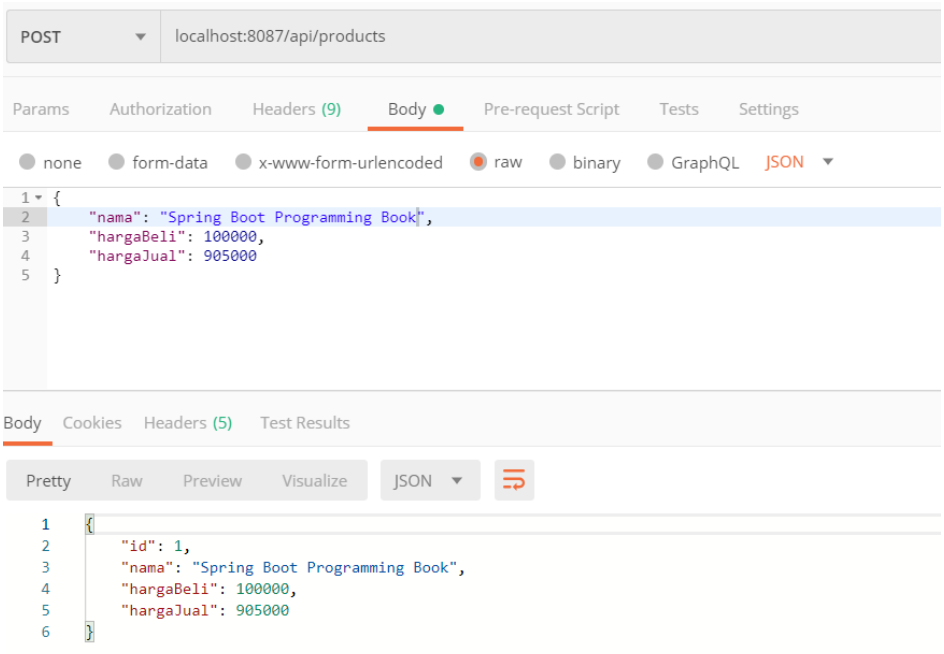


Figure 1- Post Data

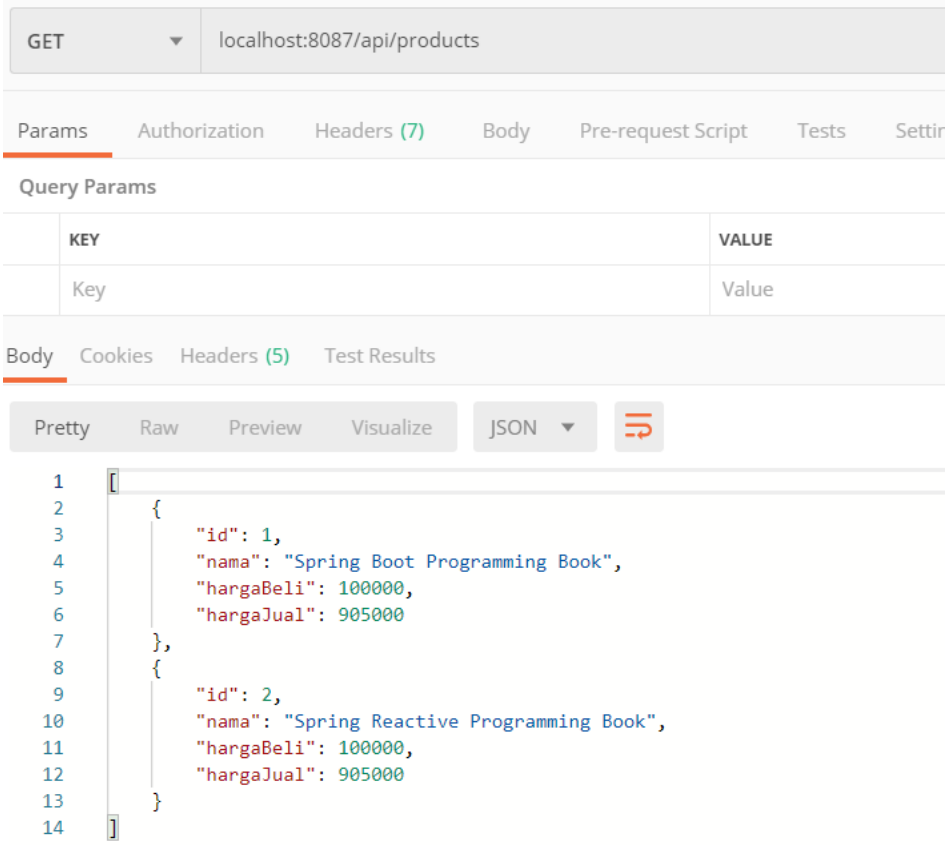


Figure 2- Get All

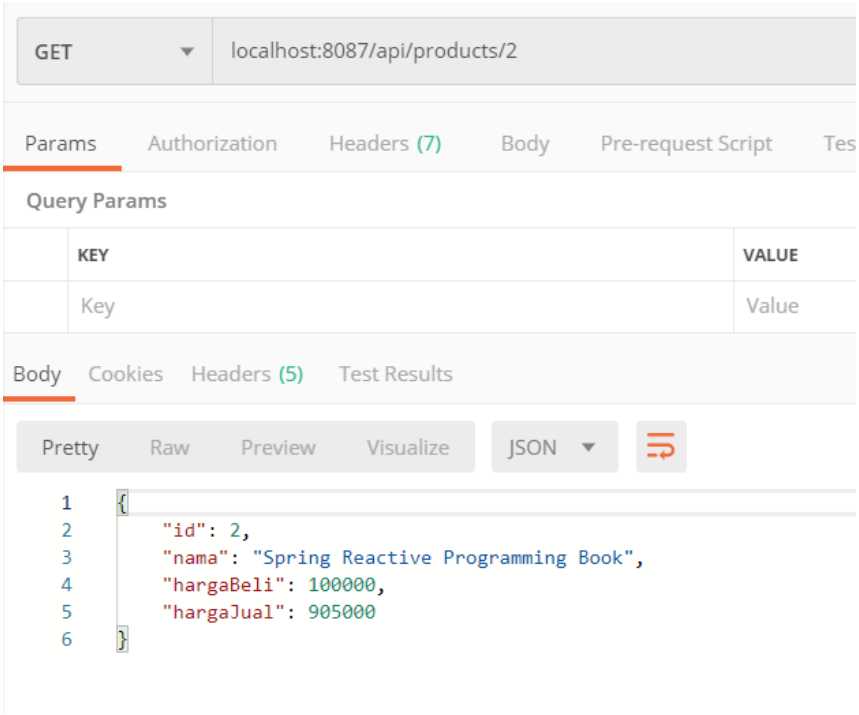


Figure 3- Get One Object

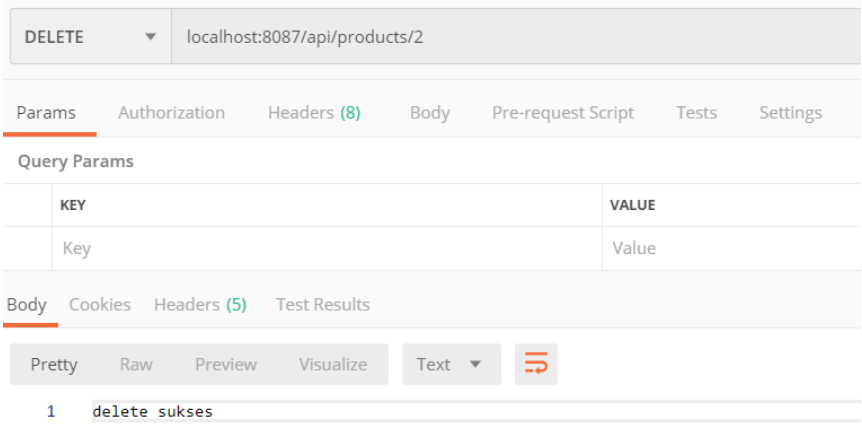


Figure 4- Delete Object

Tentang Penulis

Teten Nugraha, Lahir di Garut pada tanggal 18 Desember 1990. Lulus di perguruan tinggi swasta di Bandung pada tahun 2012 dengan program studi Sistem Informasi.

Penulis saat ini bekerja di perusahan Asuransi swasta yang sehari-hari bekerja menggunakan banyak Stack teknologi salah satunya Java dan arsitektur microservices.

Penulis dapat dijumpai pada account sebagai berikut :

- Web : https://medium.com/backend-habit
- Linkedin : https://www.linkedin.com/in/teten-nugraha/
- Github: https://github.com/teten777
- Email : teten.nugraha18@gmail.com
- Telegram : @TetenNugraha

*** silahkan jika ada yang kurang mengerti dari penjelasan diatas atau jika ada yang ingin meminta source, dapat langsung menghubungi penulis*

Daftar Pustaka

Walls, Craig. 2015. *Spring Boot in Action.* Manning Publications

Karanam, Ranga Rao. 2019. *Mastering Spring 5*. Packt Publishing

Karanam, Ranga Rao. 2018. *Spring: Microservices with Spring Boot*. Packt Publishing

Bloch, Joshua. 2017. *Effective Java, 3rd Edition*. Addison-Wesley Professional

https://www.javaguides.net

https://mkyong.com/

https://www.baeldung.com/

https://medium.com/backend-habit