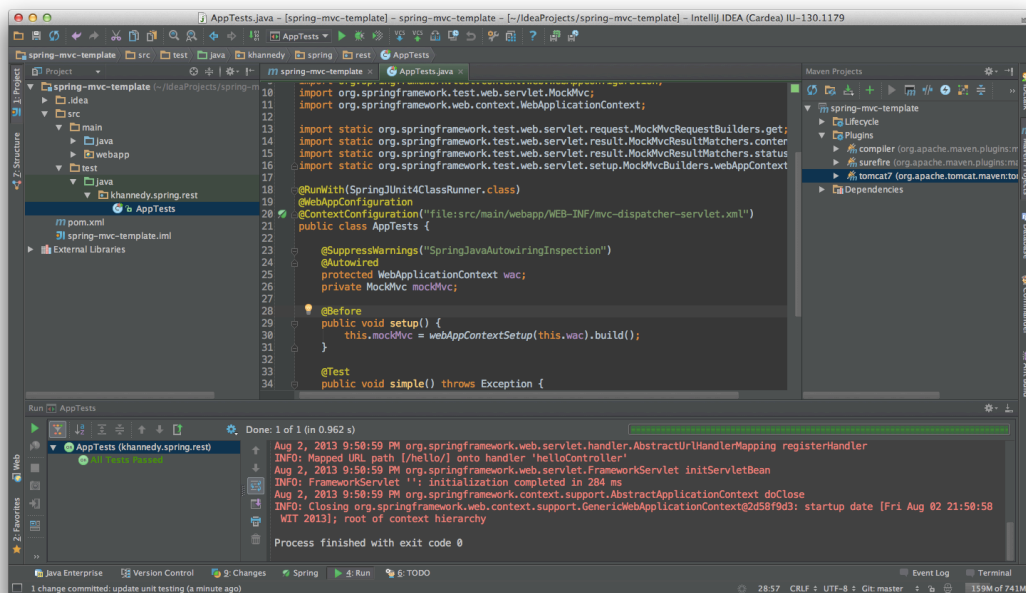


# Membangun Aplikasi RESTful Web Service [bagian 2] menggunakan Hibernate & MySQL

@khannedy

Eko Kurniawan Khannedy



## Daftar Isi

<b>Daftar Isi .....</b>	<b>2</b>
<b>Mau ngapain di buku ini? .....</b>	<b>3</b>
Nerusin Buku Sebelumnya .....	3
Jadi mo ngapain dibuku yang ke-2 ini? .....	3
<b>Yuk nerusin project sebelumnya yuk!.....</b>	<b>4</b>
Pake database MySQL aja ya .....	4
Bikin database dulu .....	4
Tambah dependency MySQL dan Hibernate .....	4
Konfigurasi Spring .....	5
Bikin entitas Barang .....	7
Bikin service untuk entitas Barang .....	9
Implementasi insert(Barang) .....	11
Implementasi update(Barang) .....	11
Implementasi delete(kode) .....	12
Implementasi find(kode) .....	12
Implementasi findAll().....	12
<b>Redesign BarangController.....</b>	<b>13</b>
Redesign RESTful insert() .....	13
Redesign RESTful find() .....	14
Redesign RESTful update().....	14
Redesign RESTful delete().....	15
Redesign RESTful findAll() .....	16
<b>Ngetest lagi.....</b>	<b>17</b>
<b>Tabel Barang .....</b>	<b>17</b>
<b>Insert beberapa Barang.....</b>	<b>17</b>
<b>Update barang dengan kode 10106031.....</b>	<b>17</b>
<b>Ngapus barang dengan kode 10106031 .....</b>	<b>18</b>
<b>Ngambil data barang .....</b>	<b>18</b>
<b>Ngambil semua data barang .....</b>	<b>18</b>
<b>Tugas selanjutnya! .....</b>	<b>20</b>
<b>Source Code .....</b>	<b>21</b>
<b>Buku siapa nih? .....</b>	<b>22</b>

## Mau ngapain di buku ini?

Beneran gak tau?

Berarti ente gak ngikutin buku bagian 1 nya, beuh wadezig!

Download dulu buku bagian 1 nya disini nih!

<http://eecchhoo.wordpress.com/2013/08/04/buku-gratis-membangun-aplikasi-restful-web-service-menggunakan-spring-web-mvc/>

## Nerusin Buku Sebelumnya

Dibuku sebelumnya, kita udah bikin aplikasi RESTful web service, udah sampai jadi malah. Cuma emang data nya gak disimpen di database, datanya cuma disimpen di Map (memory), jadi pas ente restart tuh aplikasi nya, ya ilang deh semua data nya

Capedeh!!!

Namanya juga belajar cuy!

## Jadi mo ngapain dibuku yang ke-2 ini?

Dibuku yang ke-2 ini, kita akan lanjutin dengan ganti semua data yang disimpen di Map, jadi disimpen ke database.

Tapi gak pake JDBC, soalnya gw males kalo harus pake SQL ☺

Jadi kita pake yang agak kerenan dikit, Hibernate ☺

Hibernate ini banyak dipake di perusahaan, jadi gak rugi deh kalo ente bisa Hibernate, dijamin kalo mo kerja pasti lancar, kalo gak lancar, bukan berarti ente bego, tapi ada yang lebih pinter dari ente ☺ #hehehe

Oke dah, yuk nerusin project sebelumnya ☺

## Yuk nerusin project sebelumnya yuk!

### Pake database MySQL aja ya

Nah untuk database nya, kita pake MySQL aja ya.

Kenapa pake MySQL, kenapa gak pake PostgreSQL atau Oracle, apa MySQL lebih baik dari database yang lain?

Enggak sih, cuma kebetulan di laptop gw cuma ada MySQL, jadi gw males kalo harus install yang lain ☺

### Bikin database dulu

Gw gak akan jelasin gimana install MySQL dan login ke MySQL, kalo bener2 gak ngerti, kebangetan deh!

Udah, pokoknya sekarang bikin aja databasenya, gw gak mau tau ente bisa atau enggak kek install MySQL.



```
mysql> create database rest_database;  
Query OK, 1 row affected (0.01 sec)  
  
mysql> 
```

Nama database nya **rest\_database**;

### Nambah dependency MySQL dan Hibernate

Sekarang kita tambahkan dependency MySQL dan Hibernate ke project yang udah kita selesain di buku sebelumnya.

Karena gw pake maven, jadi silahkan buka file pom.xml trus tambahkan kode ini didalam <dependencies>

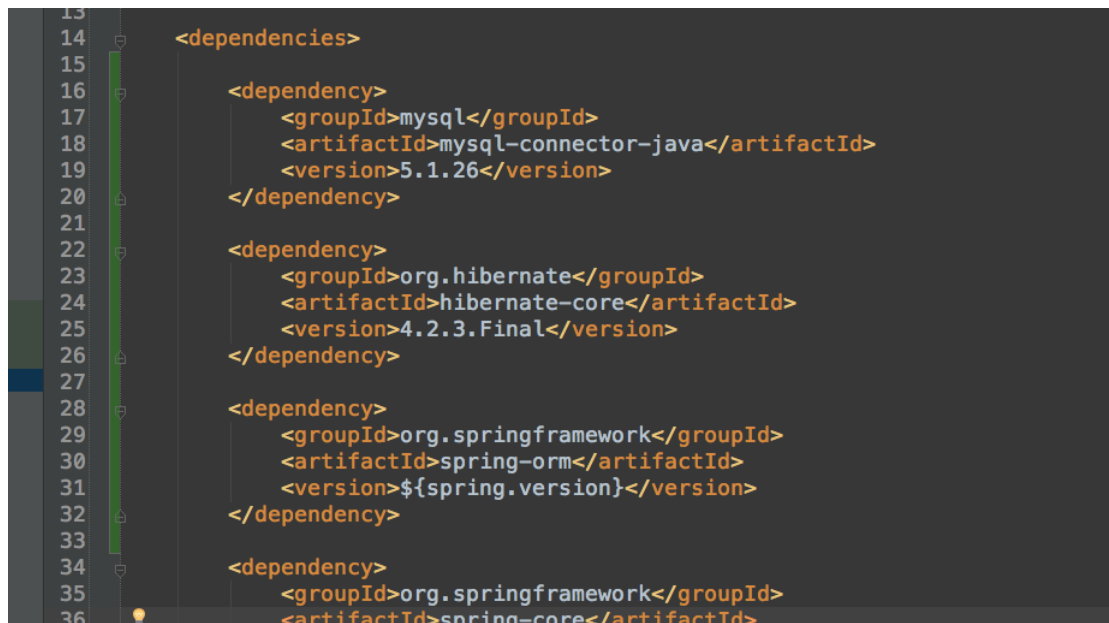
```
<dependency>  
  <groupId>mysql</groupId>  
  <artifactId>mysql-connector-java</artifactId>  
  <version>5.1.26</version>  
</dependency>  
<dependency>  
  <groupId>org.hibernate</groupId>  
  <artifactId>hibernate-core</artifactId>  
  <version>4.2.3.Final</version>
```

```

</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-orm</artifactId>
  <version>${spring.version}</version>
</dependency>

```

Jadi kayak gini nih :



```

13
14 <dependencies>
15
16   <dependency>
17     <groupId>mysql</groupId>
18     <artifactId>mysql-connector-java</artifactId>
19     <version>5.1.26</version>
20   </dependency>
21
22   <dependency>
23     <groupId>org.hibernate</groupId>
24     <artifactId>hibernate-core</artifactId>
25     <version>4.2.3.Final</version>
26   </dependency>
27
28   <dependency>
29     <groupId>org.springframework</groupId>
30     <artifactId>spring-orm</artifactId>
31     <version>${spring.version}</version>
32   </dependency>
33
34   <dependency>
35     <groupId>org.springframework</groupId>
36     <artifactId>spring-core</artifactId>

```

Enaknya pake Maven, dependency yang lain otomatis di download, misal Hibernate itu butuh dependency A, B, dan C, nah kita gak perlu download manual satu-satu tuh, karena nanti Maven bakal downloadin semuanya, jadi kita tinggal onggang onggang kaki aja, nunggu download selesai 😊

## Konfigurasi Spring

Karena kita pake Spring, jadi gw saranin jangan manual bikin object Hibernate, mending joinan ama Spring aja, lebih mudah dan gampang.

So, sekarang yuk kita ubah konfigurasi Spring nya.

Silahkan ubah file **mvc-dispatcher-servlet.xml** jadi kayak gini nih :

```

<beans xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:context="http://www.springframework.org/schema/context"
  xmlns:mvc="http://www.springframework.org/schema/mvc"
  xmlns:tx="http://www.springframework.org/schema/tx"
  xmlns="http://www.springframework.org/schema/beans"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context.xsd
    http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/tx/spring-tx.xsd">

```

```

<context:annotation-config/>
<mvc:annotation-config/>

<context:component-scan base-package="khannedy.spring.rest"/>

<tx:annotation-driven transaction-manager="transactionManager"/>

<bean class="com.mysql.jdbc.jdbc2.optional.MysqlDataSource" id="dataSource">
  <property name="user" value="root"/>
  <property name="password" value=""/>
  <property name="databaseName" value="rest_database"/>
  <property name="serverName" value="localhost"/>
  <property name="portNumber" value="3306"/>
</bean>

<bean class="org.springframework.orm.hibernate4.HibernateTransactionManager" id="transactionManager">
  <property name="sessionFactory" ref="sessionFactory"/>
</bean>

<bean class="org.springframework.orm.hibernate4.LocalSessionFactoryBean" id="sessionFactory">
  <property name="packagesToScan" value="khannedy.spring.rest.model"/>
  <property name="dataSource" ref="dataSource"/>
  <property name="hibernateProperties">
    <props>
      <prop key="hibernate.hbm2ddl.auto">update</prop>
      <prop key="hibernate.show_sql">true</prop>
      <prop key="hibernate.format_sql">true</prop>
    </props>
  </property>
</bean>
</beans>

```

Bhahahaha, pasti mlongo, apaan tuh?

Tapi yang udah ngerti gw acungin jempol, tapi masalahnya, kalo dah ngerti ngapain baca buku ini juga kalee :P

Tenang2 gw jelasin satu2 ☺

Untuk ngaktifin transaction management di Spring, kita harus gunakan perintah ini nih :

```

16
17 <tx:annotation-driven transaction-manager="transactionManager"/>
18

```

Disana disebutkan kalo transaction management adalah “transactionManager”, dimana itu adalah bean Hibernate yang kita buat di kode ini :

```

<bean class="org.springframework.orm.hibernate4.HibernateTransactionManager" id="transactionManager">
  <property name="sessionFactory" ref="sessionFactory"/>
</bean>

```

Nah tapi di HibernateTransactionManager itu, kita butuh sebuah property dengan nama sessionFactory, dimana itu adalah SessionFactory milik si Hibernate yang kita buat bean nya disini nih

```
<bean class="org.springframework.orm.hibernate4.LocalSessionFactoryBean" id="sessionFactory">
  <property name="packagesToScan" value="khannedy.spring.rest.model"/>
  <property name="dataSource" ref="dataSource"/>
  <property name="hibernateProperties">
    <props>
      <prop key="hibernate.hbm2ddl.auto">update</prop>
      <prop key="hibernate.show_sql">true</prop>
      <prop key="hibernate.format_sql">true</prop>
    </props>
  </property>
</bean>
```

Di SessionFactory diatas, kita nyebutin di property “packagesToScan” value-nya adalah “khannedy.spring.rest.model”, artinya semua kelas entitas bakal kita simpen di package itu.

Untuk property “dataSource” adalah DataSource koneksi ke database, nanti kita bahas, sekarang kita bahas dulu “hibernateProperties”, hibernateProperties itu adalah semua konfigurasi untuk Hibernate, ada beberapa konfigurasi yang kita sebut, yaitu :

- hibernate.hbm2ddl.auto = update, itu artinya nanti hibernate bakal otomatis bikinin kita tabel, dan tambah kolom yang gak ada di database, asik kan, gak perlu ribet bikin tabel manual :P
- hibernate.show\_sql = true, itu artinya nanti sintak SQL yang dieksekusi ama hibernate akan ditampilkan di console, jadi kita tau, apa aja yang hibernate lakuin sebenarnya
- hibernate.format\_sql = true, itu artinya perintah SQL yang nanti di tampilin di console, akan diformat rapih, jadi kita gampang bacanya, enggak satu baris panjang yang bikin mata kriting bacanya ☺

Sekarang balik lagi ke “dataSource”, itu adalah koneksi ke database, dimana kita buat dalam bean DataSource

```
<bean class="com.mysql.jdbc.jdbc2.optional.MysqlDataSource" id="dataSource">
  <property name="user" value="root"/>
  <property name="password" value=""/>
  <property name="databaseName" value="rest_database"/>
  <property name="serverName" value="localhost"/>
  <property name="portNumber" value="3306"/>
</bean>
```

Untuk konfigurasi database, silahkan sesuaikan dengan laptop masing2, gw gak akan banyak bahas kalo soal ini ☺

## Bikin entitas Barang

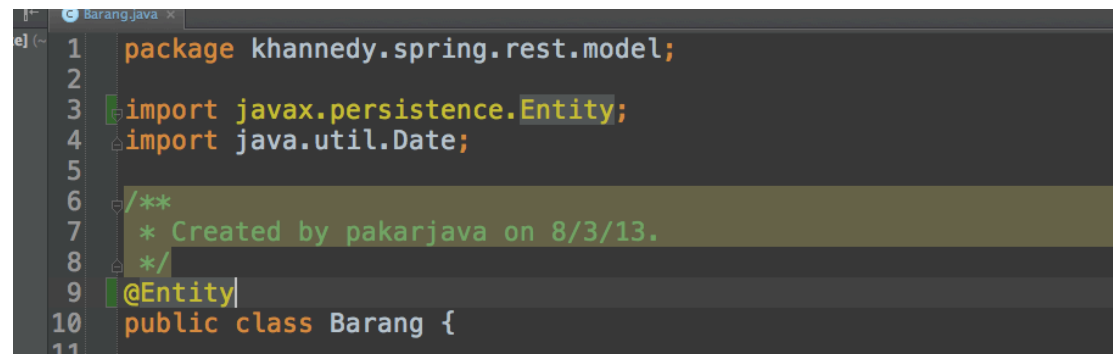
Dibuku sebelumnya kita udah bikin kelas model Barang, namun sayangnya kelas itu bukanlah kelas entitas.

Entitas itu apaan ya?

Entitas itu bahasa sederhananya, kelas yang representasi tabel yang ada di database, jadi kalo ente bikin tabel dengan kolom nama dan alamat, jadi ente juga harus bikin kelas tabel itu dengan atribut nama dan alamat, gitu bro!

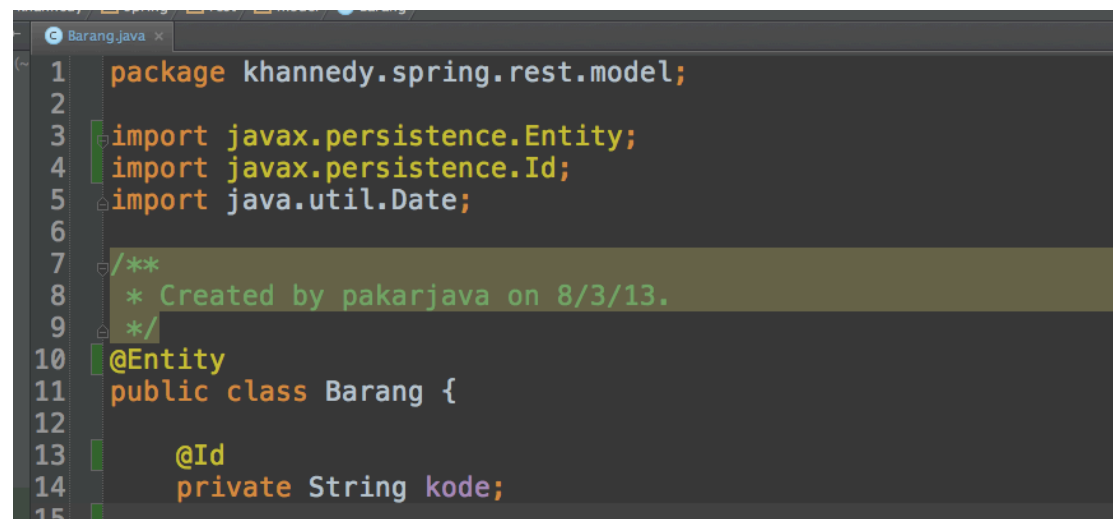
Oke, karena kelas Barang itu udah kita bikin sebelumnya, jadi sekarang kita cukup ngubah dikit2 aja.

Pertama tandain kelas Barang itu dengan tag `@Entity` di nama kelas nya, kayak gambar dibawah ini.



```
1 package khannedy.spring.rest.model;
2
3 import javax.persistence.Entity;
4 import java.util.Date;
5
6 /**
7  * Created by pakarjava on 8/3/13.
8  */
9 @Entity
10 public class Barang {
11
```

Setelah itu, di barang, kita mau jadiin kode sebagai Primary Key, jadi kita tambhin annotation `@Id` di atribut kode nya



```
1 package khannedy.spring.rest.model;
2
3 import javax.persistence.Entity;
4 import javax.persistence.Id;
5 import java.util.Date;
6
7 /**
8  * Created by pakarjava on 8/3/13.
9  */
10 @Entity
11 public class Barang {
12
13     @Id
14     private String kode;
15
```

Dan untuk yang lainnya, cukup gunakan `@Column`, kecuali untuk tanggalKadaluarsa, karena itu tipe datanya adalah Date, jadi kita perlu nambahin `@Temporal(DATE)`, jelasnya liat gambar dibawah ini :



```

1 package khannedy.spring.rest.model;
2
3 import javax.persistence.*;
4 import java.util.Date;
5
6 /**
7  * Created by pakarjava on 8/3/13.
8  */
9 @Entity
10 public class Barang {
11
12     @Id
13     private String kode;
14     @Column
15     private String nama;
16     @Column
17     private String kategori;
18     @Column
19     private Long harga;
20     @Column
21     private Integer stok;
22     @Column
23     private Boolean mudahTerbakar;
24     @Temporal(TemporalType.DATE)
25     @Column
26     private Date tanggalKadaluarsa;
27

```

## Bikin service untuk entitas Barang

Setelah kita update model barang jadi entitas, sekarang saatnya kita bikin service nya.

Service? Buat apaan?

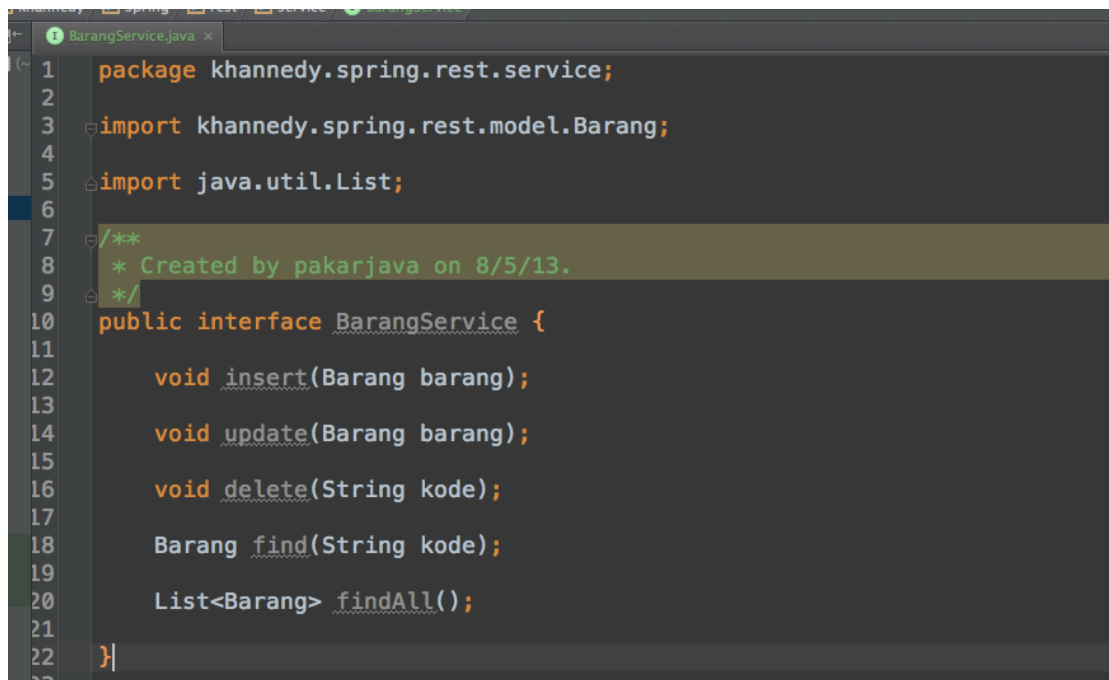
Simplenya gini, setiap kelas entitas itu biasanya punya Service, service itu gunanya untuk manipulasi kelas entitas tersebut ke database, kayak buat insert, update, delete, dan lain2.

Jadi sekarang silahkan bikin Service, nah perlu diketahui, untuk kelas Service, itu harus dibuat dalam 2 file, 1 file interface dan 1 file class (yang implements interface tersebut)

Kenapa harus gitu?

Hmmm, panjang sih kalo gw harus jelasin kenapa harus gitu, soalnya nih ngebahas tentang.....

Bah, ribet lah kalo gw jelasin dari awal, percuma, yang ada malah bingung. Percaya aja deh kalo hal ini justru mempermudah dibandingkan ente manual ngelakuinnya, jadi berterimakasih lah sama Spring Framework ☺



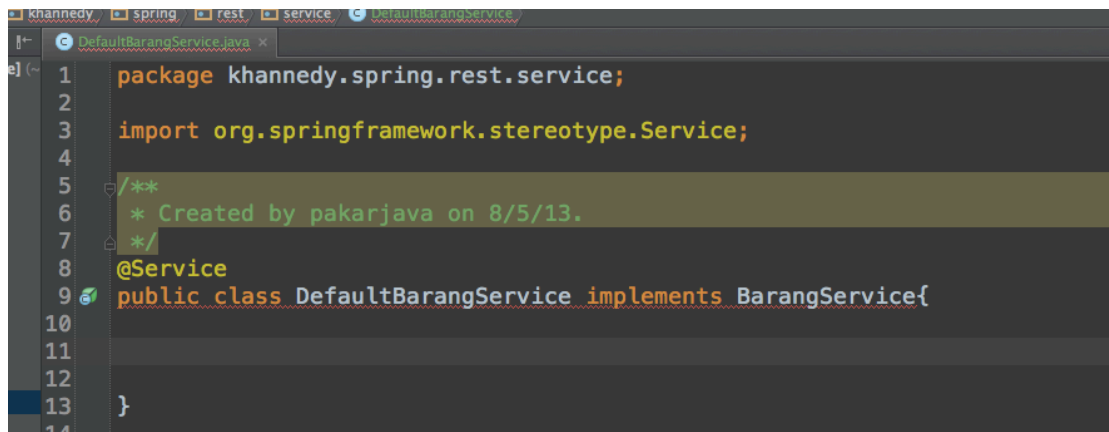
```

1 package khannedy.spring.rest.service;
2
3 import khannedy.spring.rest.model.Barang;
4
5 import java.util.List;
6
7 /**
8  * Created by pakarjava on 8/5/13.
9  */
10 public interface BarangService {
11
12     void insert(Barang barang);
13
14     void update(Barang barang);
15
16     void delete(String kode);
17
18     Barang find(String kode);
19
20     List<Barang> findAll();
21
22 }

```

Kurang lebih seperti digambar diatas tun Service nya. Oh iya, gw buat kelas service nya itu di package baru, namanya **"khannedy.spring.rest.service"**.

Sekarang bikin class service nya yang ngeimplement interface tersebut. Biasanya kalo gw lebih seneng pake nam Default, jadinya DefaultBarangService.



```

1 package khannedy.spring.rest.service;
2
3 import org.springframework.stereotype.Service;
4
5 /**
6  * Created by pakarjava on 8/5/13.
7  */
8 @Service
9 public class DefaultBarangService implements BarangService{
10
11
12
13 }

```

Di Spring, kalo kita bikin Service, itu harus ditandai dengan annotation `@Service`, kayak di gambar diatas tuh.

Sekarang saatnya kita implementasiin semua method yang ada di interface `BarangService` satu per satu, dimulai dari...

Eh, tapi perlu ada yang disiapkan dulu jeh, silahkan tambahin `SessionFactory` dulu, sorry hampir kelewat ☺

```

1 package khannedy.spring.rest.service;
2
3 import org.hibernate.SessionFactory;
4 import org.springframework.beans.factory.annotation.Autowired;
5 import org.springframework.stereotype.Service;
6
7 /**
8  * Created by pakarjava on 8/5/13.
9  */
10 @Service
11 public class DefaultBarangService implements BarangService{
12
13     @Autowired
14     private SessionFactory sessionFactory;
15
16 }
17

```

@Autowired itu artinya nanti atribut itu bakal di inject ama Spring, jadi kita gak perlu deklarasiin secara eksplisit (ngomong apa lagi)

Intinya, kita gak perlu bikin objek secara manual, cukup gunakan @Autowired, nanti itu semua di handle sama si Spring #mantap!

Sip, sekarang kita mulai!!!!

### Implementasi insert(Barang)

```

18
19 @Transactional
20 public void insert(Barang barang) {
21     // ngambil session hibernate
22     Session session = sessionFactory.getCurrentSession();
23
24     // simpen data barang ke database
25     session.save(barang);
26
27     // SELESAI!
28 }
29

```

Simple, cukup bikin kode kayak digambar diatas. Oh iya, kalo kita bikin method yang isinya itu manipulasi data (insert, update, delete), kita harus tambahkan annotation @Transactional di method nya, kayak gambar diatas.

Jangan lupa, kalo lupa nambahin @Transactional, dijamin tuh kode gak akan jalan 😊

### Implementasi update(Barang)

Untuk update barang, simple juga, kayak gini nih.

```

29
30 @Transactional
31 public void update(Barang barang) {
32     // ngambil session hibernate
33     Session session = sessionFactory.getCurrentSession();
34
35     // update data barang ke database
36     session.update(barang);
37
38     // SELESAI!
39 }

```

### Implementasi delete(kode)

Untuk delete barang berdasarkan kode, kayak gini implementasinya

```

40
41 @Transactional
42 public void delete(String kode) {
43     // ngambil session hibernate
44     Session session = sessionFactory.getCurrentSession();
45
46     // ngambil data barang
47     Barang barang = (Barang) session.get(Barang.class, kode);
48
49     // ngapus data barang di database
50     session.delete(barang);
51
52     // SELESAI!
53 }

```

### Implementasi find(kode)

Lanjut lagi, sekarang kita implementasiin untuk ngambil data barang berdasarkan kode. Sebenarnya udah ada di delete(kode), tinggal ambil sebagian kodenya aja 😊

```

54
55 @Transactional(readOnly = true)
56 public Barang find(String kode) {
57     // ngambil session hibernate
58     Session session = sessionFactory.getCurrentSession();
59
60     // ngambil data barang
61     Barang barang = (Barang) session.get(Barang.class, kode);
62
63     // SELESAI!
64     return barang;
65 }

```

Yang membedakan adalah di @Transactional, disana ditambahkan atribut readOnly=true, maksudnya kalo method ini melakukan operasi ke database, tapi cuma baca doank, gak sampai ngedit data. Gitu bro 😊

### Implementasi findAll()

Oke, yang terakhir adalah ngambil semua data barang.

```

68
69     @Transactional(readOnly = true)
70     public List<Barang> findAll() {
71         // ngambil session hibernate
72         Session session = sessionFactory.getCurrentSession();
73
74         // simpen data barang ke database
75         List<Barang> list = session.createCriteria(Barang.class).list();
76
77         // SELESAI!
78         return list;
79     }

```

## Redesign BarangController

Hehei!! Sekarang saatnya deh kita rombak total BarangController yang sebelumnya udah kita bikin ☺

Jangan bersedih ya ☺ #alah!

Kita mulai nyiapin dulu apa yang perlu kita tambah sebelum redesign BarangController-nya, simple, cukup tambahkan atribut BarangService.

```

21
22     @Controller
23     public class BarangController {
24
25         private Gson gson = new GsonBuilder().setDateFormat("dd/MM/yy
26         private Map<String, Barang> map = new HashMap<>();
27
28         @Autowired
29         private BarangService barangService;
30

```

Oh iya, untuk Map nya sekarang kita udah gak butuh lagi, jadi bisa ditendang sebagai atribut, wadezig!!!

```

1     */
2     @Controller
3     public class BarangController {
4
5         private Gson gson = new GsonBuilder().setDateFormat("dd/MM/y
6
7         @Autowired
8         private BarangService barangService;
9

```

Oke sip, sekarang kita redesign semua method RESTful web service nya, dimulai dari...

## Redesign RESTful insert()

Sekarang kita ubah nyimpen data ke Map jadi ke database via BarangService, jadinya hasil akhirnya kayak gini nih.

```

27 @ResponseBody
28 @RequestMapping(value = "/barang/insert", method = RequestMethod.POST)
29 public String insert(HttpServletRequest servletRequest, @RequestBody String json) {
30
31     // check hak akses
32     if (!checkAccess(servletRequest)) {
33         Status status = new Status();
34         status.setKode("408");
35         status.setPesan("Hak akses ditolak");
36         return gson.toJson(status);
37     }
38
39     // konversi dari json ke object barang
40     Barang barang = gson.fromJson(json, Barang.class);
41
42     // simpen data ke database
43     barangService.insert(barang);
44
45     // bikin status sukses
46     Status status = new Status();
47     status.setKode("200"); // kode terserah bisa berapa aja
48     status.setPesan("Sukses nyimpen data barang");
49
50     // bikin response json
51     return gson.toJson(status);
52 }

```

### Redesign RESTful find()

Selanjutnya method find(), jadinya kayak gini :

```

54 @ResponseBody
55 @RequestMapping(value = "/barang/find/{kode}", method = RequestMethod.GET)
56 public String find(HttpServletRequest servletRequest,
57     @PathVariable("kode") String kode) {
58
59     // check hak akses
60     if (!checkAccess(servletRequest)) {
61         Status status = new Status();
62         status.setKode("408");
63         status.setPesan("Hak akses ditolak");
64         return gson.toJson(status);
65     }
66
67     // ambil barang di map
68     Barang barang = barangService.find(kode);
69
70     // check apa barang ada atau enggak
71     if (barang == null) {
72         // kalo gak ada, return null aja
73         return null;
74     } else {
75         // kalo ada, convert jadi JSON
76         return gson.toJson(barang);
77     }
78 }
79

```

### Redesign RESTful update()

Untuk update jadinya kayak gini

```

80
81 @ResponseBody
82 @RequestMapping(value = "/barang/update", method = RequestMethod.PUT)
83 public String update(HttpServletRequest servletRequest, @RequestBody String json) {
84
85     // check hak akses
86     if (!checkAccess(servletRequest)) {
87         Status status = new Status();
88         status.setKode("408");
89         status.setPesan("Hak akses ditolak");
90         return gson.toJson(status);
91     }
92
93     // konversi dari json ke object barang
94     Barang barang = gson.fromJson(json, Barang.class);
95
96     // cek apa ada data barang dengan kode yang sama
97     if (barangService.find(barang.getKode()) != null) {
98         // kalo ada yang sama
99         // update data barang dengan yang baru
100        barangService.update(barang);
101
102        // bikin status sukses
103        Status status = new Status();
104        status.setKode("200"); // kode terserah bisa berapa aja
105        status.setPesan("Sukses mengubah data barang");
106
107        // bikin response json
108        return gson.toJson(status);
109    } else {
110        // kalo barang nya gak ada
111        // bikin status gagal
112        Status status = new Status();
113        status.setKode("404"); // kode terserah bisa berapa aja
114        status.setPesan("Barang gak ditemukan");
115
116        // bikin response json
117        return gson.toJson(status);
118    }
119 }

```

## Redesign RESTful delete()

Untuk delete, jadinya kayak gini

```

120 @ResponseBody
121 @RequestMapping(value = "/barang/delete/{kode}", method = RequestMethod.DELETE)
122 public String delete(HttpServletRequest servletRequest,
123                     @PathVariable("kode") String kode) {
124
125     // check hak akses
126     if (!checkAccess(servletRequest)) {
127         Status status = new Status();
128         status.setKode("408");
129         status.setPesan("Hak akses ditolak");
130         return gson.toJson(status);
131     }
132
133     // check apa barang ada atau enggak
134     if (barangService.find(kode) == null) {
135         // kalo gak ada, return status gagal
136         Status status = new Status();
137         status.setKode("404");
138         status.setPesan("Barang gak ditemukan");
139         return gson.toJson(status);
140     } else {
141         // hapus barang
142         barangService.delete(kode);
143
144         // kalo gak ada, return status sukses
145         Status status = new Status();
146         status.setKode("200");
147         status.setPesan("Barang berhasil dihapus");
148         return gson.toJson(status);
149     }
150 }
151 }
152

```

## Redesign RESTful findAll()

Dan yang terakhir, untuk method findAll() jadinya kayak gini

```

155
156 @ResponseBody
157 @RequestMapping(value = "/barang/findall", method = RequestMethod.GET)
158 public String findAll(HttpServletRequest servletRequest) {
159
160     // check hak akses
161     if (!checkAccess(servletRequest)) {
162         Status status = new Status();
163         status.setKode("408");
164         status.setPesan("Hak akses ditolak");
165         return gson.toJson(status);
166     }
167
168     // buat data list barang
169     List<Barang> list = barangService.findAll();
170
171     // return sebagai json
172     return gson.toJson(list);
173 }
174

```

Huh, artinya selesai juga deh 😊

Tinggal uji coba aja, kita test lagi deh 😊



## Ngetest lagi

Untuk ngetest lagi, gak perlu gw jelas lagi lah ya, kan udah dijelasin di buku yang pertama, tinggal ente coba lagi aja dari awal sampai akhir.

Bedanya pastiin datanya masuk ke database, misal kayak gininih.

## Tabel Barang

```
mysql> desc Barang;
+-----+-----+-----+-----+-----+-----+
| Field          | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| kode           | varchar(255)  | NO   | PRI | NULL    |       |
| harga          | bigint(20)    | YES  |     | NULL    |       |
| kategori       | varchar(255)  | YES  |     | NULL    |       |
| mudahTerbakar  | tinyint(1)    | YES  |     | NULL    |       |
| nama           | varchar(255)  | YES  |     | NULL    |       |
| stok           | int(11)       | YES  |     | NULL    |       |
| tanggalKadaluarsa | date         | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
7 rows in set (0.02 sec)

mysql>
```

## Insert beberapa Barang

```
mysql> select * from Barang;
+-----+-----+-----+-----+-----+-----+-----+
| kode      | harga | kategori | mudahTerbakar | nama      | stok | tanggalKadaluarsa |
+-----+-----+-----+-----+-----+-----+-----+
| 10106031 | 15000 | Minuman  | 0             | Sirup CBA | 1000 | 2020-12-29         |
| 10106032 | 15000 | Minuman  | 0             | Sirup CBA | 1000 | 2020-12-29         |
| 10106033 | 15000 | Minuman  | 0             | Sirup CBA | 1000 | 2020-12-29         |
+-----+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql>
```

## Update barang dengan kode 10106031

```
mysql> mysql> select * from Barang;
+-----+-----+-----+-----+-----+-----+-----+
| kode      | harga | kategori | mudahTerbakar | nama      | stok | tanggalKadaluarsa |
+-----+-----+-----+-----+-----+-----+-----+
| 10106031 | 15000 | Kategori Juga | 0             | Nama Diubah | 1000 | 2020-12-29         |
| 10106032 | 15000 | Minuman  | 0             | Sirup CBA | 1000 | 2020-12-29         |
| 10106033 | 15000 | Minuman  | 0             | Sirup CBA | 1000 | 2020-12-29         |
+-----+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql>
```

## Ngapus barang dengan kode 10106031

```
mysql> select * from Barang;
+-----+-----+-----+-----+-----+-----+-----+
| kode | harga | kategori | mudahTerbakar | nama | stok | tanggalKadaluarsa |
+-----+-----+-----+-----+-----+-----+-----+
| 10106032 | 15000 | Minuman | 0 | Sirup CBA | 1000 | 2020-12-29 |
| 10106033 | 15000 | Minuman | 0 | Sirup CBA | 1000 | 2020-12-29 |
+-----+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql>
```

## Ngambil data barang

Normal

Basic Auth

Digest Auth

OAuth 1.0

No environment

0

http://localhost:8080/spring-mvc-template/barang/find/10106032

GET

URL params

Headers (1)

Authorization

Basic ZWtvOkBraGFubmVkeQ==

Header

Value

Manage presets

Send

Preview

Add to collection

Reset

Body

Headers (4)

STATUS 200 OK

TIME 77 ms

Pretty

Raw

Preview

JSON

XML

```
1 {
2   "kode": "10106032",
3   "nama": "Sirup CBA",
4   "kategori": "Minuman",
5   "harga": 15000,
6   "stok": 1000,
7   "mudahTerbakar": false,
8   "tanggalKadaluarsa": "29/12/2020"
9 }
```

## Ngambil semua data barang

Normal

Basic Auth

Digest Auth

OAuth 1.0

No environment

0

http://localhost:8080/spring-mvc-template/barang/findall

GET

URL params

Headers (1)

Authorization

Basic ZWtvOkBraGFubmVkeQ==

Manage presets

Header

Value

Send

Preview

Add to collection

Reset

Body

Headers (4)

STATUS 200 OK

TIME 67 ms

Pretty

Raw

Preview

JSON

XML

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

[

{

"kode": "10106032",

"nama": "Sirup CBA",

"kategori": "Minuman",

"harga": 15000,

"stok": 1000,

"mudahTerbakar": false,

"tanggalKadaluarsa": "29/12/2020"

},

{

"kode": "10106033",

"nama": "Sirup CBA",

"kategori": "Minuman",

"harga": 15000,

"stok": 1000,

"mudahTerbakar": false,

"tanggalKadaluarsa": "29/12/2020"

}

]

## Tugas selanjutnya!

Sebenarnya nih aplikasi belon selesai

Belum selesai dimananya?

Masa lupa? Wadezig!

Di aplikasi sebelumnya kita tambahkan autentikasi pake BASIC AUTH, tapi sayangnya hardcode disini :

```
73 public boolean checkAccess(HttpServletRequest servletRequest) {  
74     // misal username dan password harus ...  
75     String username = "eko";  
76     String password = "@khannedy";  
77  
78     // buat object BasicAuth  
79     BasicAuth basicAuth = new BasicAuth(servletRequest);  
80  
81     // check username dan password  
82     return username.equals(basicAuth.getUsername()) &&  
83         password.equals(basicAuth.getPassword());  
84  
85 }  
86  
87
```

Jadi username nya cuma bisa “eko” dan passwordny “@khannedy”

Sekarang tugas ente, silahkan bikin autentikasi nya dengan cek ke database, jadi tugas ente selanjutnya adalah :

- Bikin entitas Pengguna
- Bikin service Pengguna
- Ubah pengecekan Basic Auth nya jadi via service ke database
- Selamat mencoba 😊

## Source Code

Source code project ini bisa di download disini nih :

<https://github.com/khannedy/hibernate-rest-final/archive/master.zip>

Selamat ngobrak ngabrik kodenya 😊

## Buku siapa nih?

Ini buku milik yang baca lah ☺

Tapi yang nulis itu namanya Eko Khannedy, JURAGAN MIE AYAM, CODER JAVA, dan juga sekarang lagi jadi SECRET AGENT di salah satu PAYMENT GATEWAY di INDONESIA ☺

Saya rutin nulis artikel di <http://eecchhoo.wordpress.com> dan juga nulis buku. Setiap buku baru yang saya tulis biasanya saya publish di blog saya itu.



Untuk tips dan trik seputar CODING JAVA, atau juga sedikit tentang JURAGANPRENEUR ☺ bisa follow twitter saya di <http://twitter.com/khannedy> , selain itu di twitter juga saya sering bagi2 buku java gratis dengan tag #BukuJavaGratis

Yang mau #BukuJavaGratis lainnya, silahkan follow twitter @khannedy, trus mention saya, nanti saya DM link2 buat download #BukuJavaGratis lainnya ☺

Semoga berguna ya

Salam JURAGAN JAVA ☺