**Modul Java Spring**

**Spring boot REST API dengan MYSQL(CRUD Barang)**

# Target Pembelajan

## Peserta bisa melakukan membuat REST API dengan java spring boot ,seperti: melihat list barang (*read*), simpan data barang (*create*), update data barang (*update*), dan menghapus data barang (*delete*.

## Fitur:

## API Data Barang

1. API Menambahkan data barang
2. API Mengubah data barang
3. API Menghapus data barang

## Laporan

1. API Menampilkan data barang.

## Proses Alur

## Peserta diharapkan mengikuti setiap langkah pembuatan REST API.

## Peserta menerapkan setiap fungsionalitas sesuai dengan requirement yang telah dibuat.

## Peserta akan menerapkan CRUD dan laporan Barang pada saat pembuatan REST API

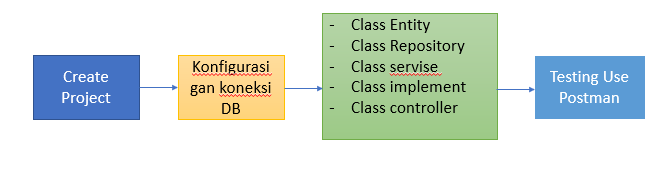
# Requirements

1. Mengerti dasar dari bahasa pemrograman JAVA
2. JDK 1.8  atau JDK 11
3. Eclipse, Intellij IDEA, Netbeans
4. Apache Maven : edit the system environment variables
5. Xampp : Mysql
6. Sistem Operasi Windows
7. Postman

Dapat download di https://drive.google.com/drive/folders/1cVZTSwjwPFwMVZF0tJkp8C5X7b-0ZIiI?usp=sharing

# Implementasi

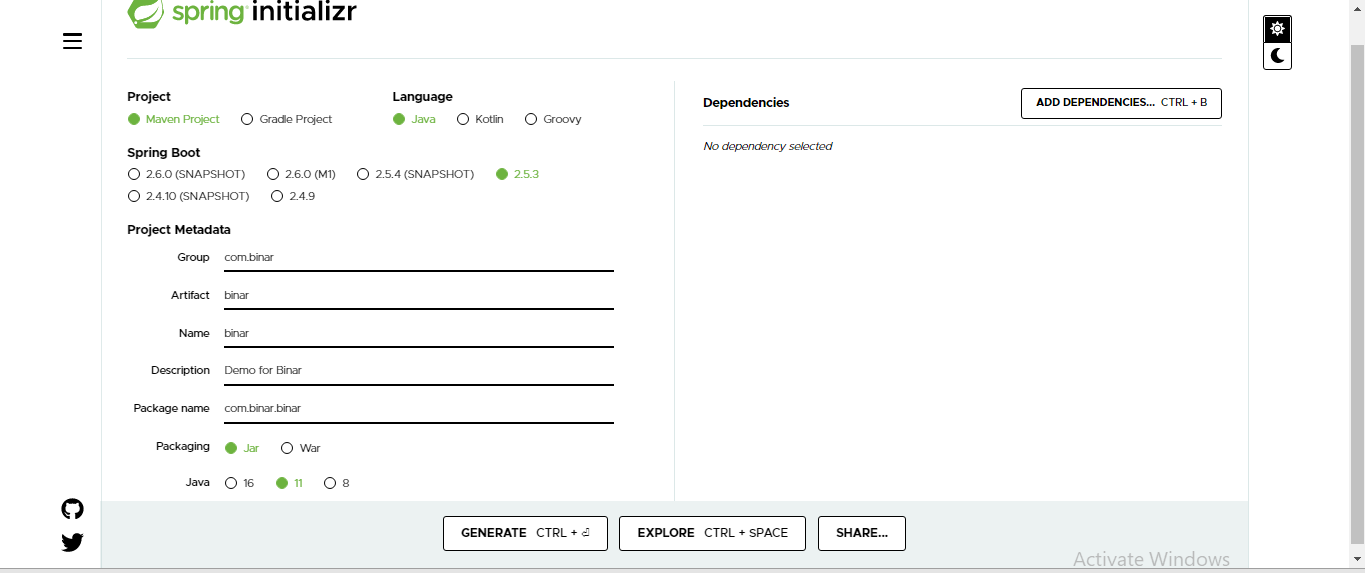
**Berikut langkah-langkah membangun REST API :**



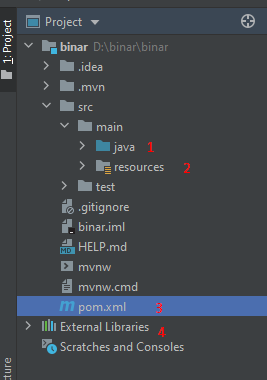
## Membuat Project

<https://start.spring.io/>

* pilih project
* languange : **java**
* version spring boot: **2.5.3**
* packaging :**jar**
* java : **11**



## Structure Project Spring boot



## Konfigurasi Spring Boot

Library yang kita butuhkan adalah JDBC API, MySQL Driver, dan Lombok.

* **JDBC API**

menyediakan jembatan untuk mengakses database melalui aplikasi

* **MySQL Driver**

Driver yang dibutuhkan aplikasi Java, karena DBMS MySQL

* **Lombok**

menyediakan method setter dan getter

Selanjutnya tambahkan library-library diatas pada file pom.xml, Liat structure Project.

|  |
| --- |
| …  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-jdbc</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>  … |

Reload pom.xml : klik kanan pom.xml -> maven-> reload project

## Konfigurasi Database

**Koneksi database di spring boot**

Lokasi path : src->main->resources-> application.properties

|  |
| --- |
| spring.datasource.username=root  spring.datasource.password=  spring.datasource.url=jdbc:mysql://localhost:3306/binar  spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  spring.datasource.initialization-mode=always  spring.jpa.show-sql=true  spring.jpa.hibernate.ddl-auto=update  spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect |

* **Never**: berarti tidak menjalankan file sql di dalam folder resources
* **Always**: selalu menjalankan file sql
* **Embedded**: digunakan untuk model database embed seperti H2

## Membuat Class Entitity

Entitas berfungsi untuk mapping tabel yang terdapat pada database.

|  |  |
| --- | --- |
| **Hibernite** | **Tampa** **Hibernite** |
| Generate Tabel dari aplikasi | Table dibuat manual di database |
| Contoh @Entity  @Table(name = "barang")  public class Barang implements Serializable {  @Id  @GeneratedValue(strategy=GenerationType.AUTO)  private long id;  @Column(name = "nama", nullable = false, length = 45)  private String nama;  @Column(name = "stok", nullable = false, length = 11)  private int stok;  @Column(name = "satuan", nullable = false, length = 45)  private String satuan;  @Column(name = "harga", nullable = false, length = 11)  private int harga;  }  } | Contoh CREATE TABLE `barang` (  `id` int(11) NOT NULL,  `nama ` text NOT NULL,  `stok` int(11) NOT NULL,  `satuan` text NOT NULL,  `harga` int(11) NOT NULL  ) ENGINE=InnoDB DEFAULT CHARSET=latin1; |
| Tambahkan pada pom.xml untuk depedency hibernite <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-jpa</artifactId>  </dependency> |  |

## Membuat Class Repository

Annotation @Repository menandai bahwa kelas tersebut merupakan sebuah Data Access Object (DAO).

@Autowired digunakan untuk melakukan dependency injection terhadap objek sessionFactory dari Hibernate.

|  |
| --- |
| Contoh class repository @Repository  public interface BarangRepo extends PagingAndSortingRepository<Barang, Long> {  @Query("select c from Barang c WHERE c.id = :id")  public Barang getbyID(@Param("id") Long id);  } |

## Membuat Class Service

Service merupakan interface yang mendefinisikan operasi-operasi pada sebuah entitas. Misal

* Operasi insert barang
* Operasi update barang
* Operasi delete barang
* Operasi List barang

|  |
| --- |
| public interface BarangService {  public Barang insert(Barang barang);  public Barang update(Barang barang);  public Barang delete(Barang barang);  public List<Barang> getAll();  } |

## Mebuat Class Implementasi dari Service

### Insert barang

|  |
| --- |
| try {  Barang obj = repo.save(barang);  map.put("data", obj);  map.put("statusCode", "200");  map.put("statusMessage", "Sukses");  return map;  } catch (Exception e) {  e.printStackTrace();  map.put("statusCode", "500");  map.put("statusMessage", e);  return map;  } |

### Update barang

|  |
| --- |
| try {    Barang obj = repo.getbyID(barang.getId());  obj.setNama(barang.getNama());  obj.setHarga(obj.getHarga());  obj.setSatuan(obj.getSatuan());  obj.setStok(obj.getStok());    repo.save(obj);  map.put("data", obj);  map.put("statusCode", "200");  map.put("statusMessage", "Sukses");  return map;  } catch (Exception e) {  e.printStackTrace();  map.put("statusCode", "500");  map.put("statusMessage", e);  return map;  } |

### Delete barang

|  |
| --- |
| try {  Barang obj = repo.getbyID(barang.getId());  repo.deleteById(obj.getId());  map.put("statusCode", "200");  map.put("statusMessage", "Delete Sukses");  return map;  } catch (Exception e) {  e.printStackTrace();  map.put("statusCode", "500");  map.put("statusMessage", e);  return map;  } |

### List barang

|  |
| --- |
| public Map getAll() {  List<Barang> list = new ArrayList<Barang>();  Map map = new HashMap();  try {  list = repo.getList();  map.put("data", list);  map.put("statusCode", "200");  map.put("statusMessage", "Delete Sukses");  return map;  } catch (Exception e) {  e.printStackTrace();  map.put("statusCode", "500");  map.put("statusMessage", e);  return map;  }  } |

## Membuat Class Controller API

Ditambahkan annotasi

* **@RestController**

 untuk menandai controller yang menyediakan layanan REST dengan tipe respon JSON

* **@RequestMapping(**"/path/path1")

Tambhakan depedency pada pom.xml, untuk request dan response REST API

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

Berikut controller rest API :

### Insert barang

Type : post : localhost:8080/v1/binar/save

Body :

{

    "nama":"nama1",

    "stok":"10",

    "satuan":"pcs",

    "harga":"1000"

}

### Update barang

Type : put : localhost:8080/v1/binar/update

Body :

{

    "id":"1",

    "nama":"nama2",

    "stok":"10",

    "satuan":"pcs",

    "harga":"1000"

}

### Delete barang

Type : delete

API url : localhost:8080/v1/binar/delete/2

### List barang

Type : get

Rest api url : localhost:8080/v1/binar/listpage

# Referensi

http://en.wikipedia.org/wiki/Apache\_Maven

http://www.tutorialspoint.com/maven/

<https://mvnrepository.com/>

<http://maven.apache.org/>

https://www.sinaungoding.com/operasi-crud-menggunakan-spring-boot-jdbc/