TRAINING JAVA DAC

SESI 3 (19 Desember 2016)

1. Buat Folder dengan nama **belajar-web**
2. Copy File **pom.xml** dari project lain ke dalam folder **belajar-web**
3. Ganti :

   <packaging>jar</packaging> menjadi **<packaging>war</packaging>**

   <artifactId>bla-bla</artifactId> menjadi **<artifactId>aplikasi-web</artifactId>**

   <name> blab la bla </name> menjadi **<name>belajar-web</name>**
4. Buka File -> Import -> Existing Maven Project -> cari folder **belajar-web** -> Finish
5. Buat Folder **src** di dalam folder **belajar-web**
6. Buat Folder **main** di dalam folder **src**
7. Buat Folder **java** dan **webapp** di dalam folder **main**
8. Buat File **index.html** di dalam folder **webapp**
9. Refresh folder belajar-web di eclipse
10. Isi file index.html

```
<!DOCTYPE html>
<html>
	<head>
		<title>Aplikasi Belajar Web Java</title>
	</head>

	<body>
		<h1>Aplikasi Belajar Web Java</h1>
		<ul>
			<li>Halo Servlet</li>
		</ul>
	</body>
</html>
```

11. Jalankan perintah **mvn clean tomcat:run** di dalam cmd dengan root folder **belajar-web**
12. Buka url `[INFO] Running war on http://localhost:8080/aplikasi-web`
13. Download Tomcat di http://tomcat.apache.org/download-80.cgi
14. Klik link 32-bit/64-bit Windows Service Installer (pgp, md5, sha1)
15. Lalu setelah selesai download, langsung install dengan user dan password bebas
16. Jalankan perintah **mvn clean package** di dalam cmd

17. Isi file pom.xml di bawah tag dependencies

```
<build>
		<plugins>
			<plugin>
				<groupId>org.apache.maven.plugins</groupId>
				<artifactId>maven-war-plugin</artifactId>
				<configuration>

	<failOnMissingWebXml>false</failOnMissingWebXml>
				</configuration>
			</plugin>
		</plugins>
	</build>
```

18. Jalankan tomcatnya
19. Copy file war nya ke dalam folder

    C:\Program Files\Apache Software Foundation\Tomcat 8.0\webapps
20. Tomcat akan otomatis mendeploy
21. Lalu Jalankan url

```
[INFO] Running war on http://localhost:8080/aplikasi-web
```

22. Untuk undeploy dengan cara hapus file .war nya saja

Note!

Arsitektur Aplikasi Web:

- Server side : Tampilan dibuat di server, server akan mengeluarkan HTML dan CSS ->tampilan + data
- Client side: Server hanya mengeluarkan data (XML, JSON, dsb). Untuk menampilkan (table, grafik, peta, dsb) diurus di sisi client (menggunakan JavaScript Framework)

Jenis – jenis Framework web :

- Component Based : Berpikir dalam komponen + event. Mirip seperti aplikasi desktop
- Request / Action Based : Berpikir HTTP request / response. Mirip seperti aplikasi web pada umumnya (PHP, Perl, HTML, dsb)

Contoh Framework Java yang Action Based :

- Spring MVC (bagian dari Spring Framework)

- Struts 1 (tidak dikembangkan lagi)
- Webwork -> Struts 2 (tidak ada hubungan dengan Struts 1)

Contoh Framework Java yang Component Based:

- Java Server Faces (JSF)
  - Primefaces
  - RichFaces
- ZK
- GWT
- Vaadin

Rekomendasi : Client Side Architecture

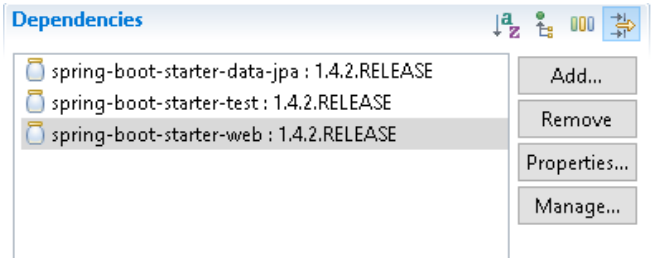
Kenapa?

- Fleksibel untuk aplikasi client : support tampilan web (brwoser) dan mobile (baik native maupun web)
- Perkembangan Javascript sangat pesat, lebih cepat daripada teknologi server side
- Fleksibel di sisi programming language. Di sisi server bias dibuat dengan Bahasa apa saja, yang penting struktur data yang dikeluarkan konsisten.

Javascript Framework:

- ExtJS
- AngularJS
- Dojo Toolkit
- EmberJS

23. Tambahkan :



24. Buat package **com.rahman.hadi.belajar.web.controller** ke dalam source **belajar-web/src/main/java**

25. Buat class **HaloController** di dalam package **com.rahman.hadi.belajar.web.controller**

HaloController.java

```
package com.rahman.hadi.belajar.springboot.controller;

import java.util.Date;
import java.util.HashMap;
import java.util.Map;

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

@RestController
public class HaloController {

	@RequestMapping("/hello")
	public Map<String, Object> halo(@RequestParam(value = "nama", required
= false) String nama){

		Map<String, Object> hasil = new HashMap<>();
		hasil.put("nama", nama);
		hasil.put("waktu", new Date());
		return hasil;
	}
}
```

26. Buat folder static di dalam folder resource

27. Buat file coba.html di dalam folder static

coba.html

```
<html>
	<head>
		<title>Halo Spring Boot</title>
		<meta charset="UTF-8">
		<meta name="viewport" content="width=device-width, initial-
scale=1.0"
	</head>
	<body>
		<div>Halo Spring Boot</div>
	</body>
</html>
```

28. Buat PesertaController.java di dalam package controller

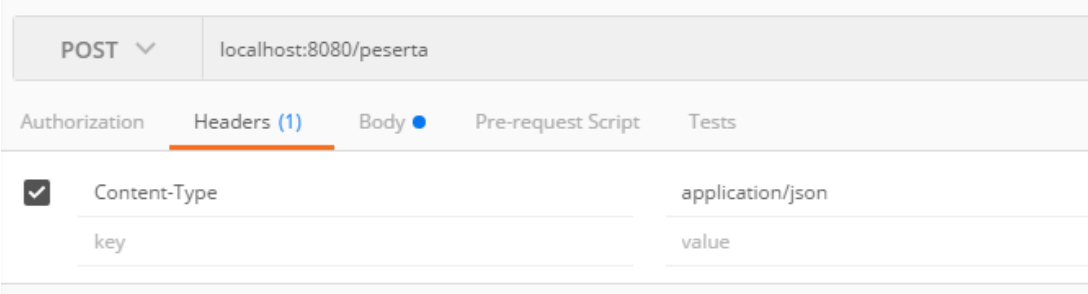
PesertaController.java

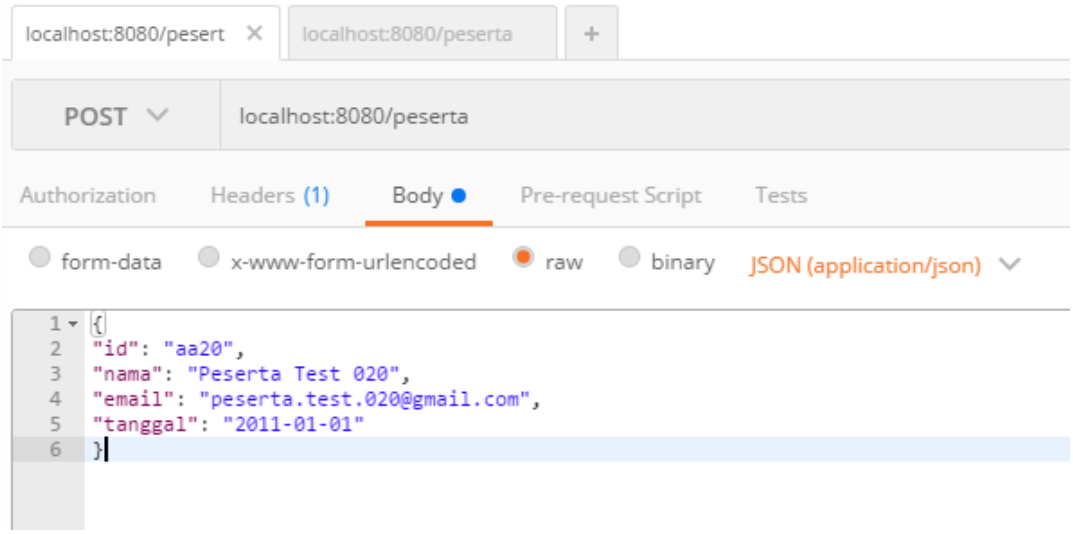
```
package com.rahman.hadi.belajar.springboot.controller;

import javax.validation.Valid;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestController;

import com.rahman.hadi.belajar.springboot.dao.PesertaDao;
import com.rahman.hadi.belajar.springboot.entity.Peserta;

@RestController
public class PesertaController {

	@Autowired
	private PesertaDao pd;

	@RequestMapping("/peserta")
	public Page<Peserta> cariPeserta(Pageable page){
		return pd.findAll(page);
	}

	@RequestMapping(value="/peserta", method = RequestMethod.POST)
	@ResponseStatus(HttpStatus.CREATED)
	public void insertPesertaBaru(@RequestBody @Valid Peserta p){
		pd.save(p);
	}

	@RequestMapping(value="peserta/{id}", method = RequestMethod.PUT)
	@ResponseStatus(HttpStatus.OK)
	public void updatePeserta(@PathVariable("id") String id, @RequestBody
@Valid Peserta p){
		p.setId(id);
		pd.save(p);
	}
```

PesertaController.java (Lanjutan)

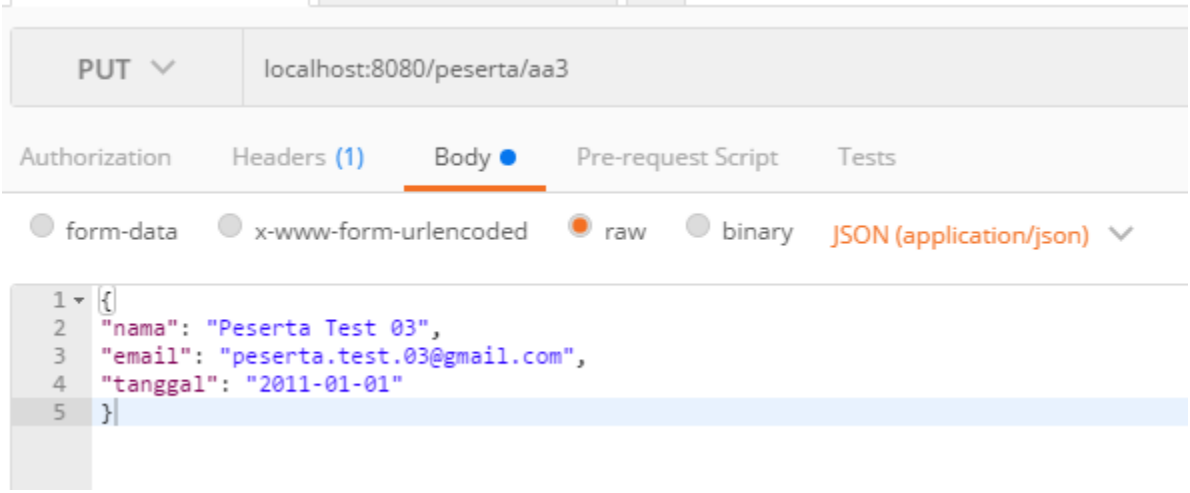
```
@RequestMapping(value="peserta/{id}", method = RequestMethod.GET)
	@ResponseStatus(HttpStatus.OK)
	public ResponseEntity<Peserta> cariPesertaByid(@PathVariable("id")
String id){
		Peserta hasil = pd.findOne(id);
		if(hasil == null){
			return new ResponseEntity<>(HttpStatus.NOT_FOUND);
		}
		return new ResponseEntity<>(hasil, HttpStatus.OK);
	}

	@RequestMapping(value="peserta/{id}", method = RequestMethod.DELETE)
	@ResponseStatus(HttpStatus.OK)
	public void hapusPeserta(@PathVariable("id") String id){
		pd.delete(id);
	}
}
```

29. Tambahkan **spring.jackson.serialization.indent_output=true** ke dalam **application.properties**
30. Install Ekstensi **Postman** di Google Chrome
31. Ikuti gambar di bawah ini (untuk CREATE)

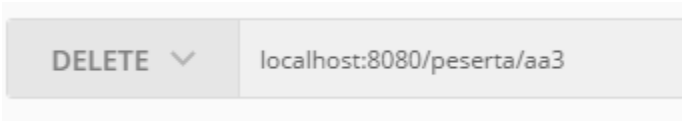




32. Jika Statusnya seperti : Status: 201 Created maka data berhasil ditambahkan ke dalam database.
33. Ikuti gambar di bawah ini (untuk EDIT)



34. Jika Statusnya seperti : Status: 200 OK maka data berhasil terupdate.
35. Ikuti gambar di bawah ini (untuk DELETE)



36. Jika Statusnya seperti : Status: 200 OK maka data berhasil terhapus.
37. Tambahkan atau replace file Peserta.java seperti kode di bawah ini:

Peserta.java

```
    @Id @GeneratedValue(generator = "uuid")
    @GenericGenerator(name="uuid",strategy="uuid2")
    private String id;

    @Column(nullable = false)
    @NotNull
    @NotEmpty
    @Size(min = 3, max = 50)
    private String nama;

    @Column(nullable = false, unique = true)
    @Email
    @NotNull
    @NotEmpty
    private String email;
}
```

38. Lalu tes kembali validasi di **Postman**.