Contents

[Persiapan 2](#_Toc169266568)

[Spring Starter Project 4](#_Toc169266569)

[Membuat Object 7](#_Toc169266570)

[Di spring boot 7](#_Toc169266571)

[AutoWired 8](#_Toc169266572)

[Menggunakan XML 8](#_Toc169266573)

[Singleton Object vs Prototype 10](#_Toc169266574)

[Singleton 10](#_Toc169266575)

[Setter Injection 11](#_Toc169266576)

[Primitive type(int, Sting, dll) 11](#_Toc169266577)

[Reference Type 13](#_Toc169266578)

[Constructor Injector 14](#_Toc169266579)

[Autowired di bean 15](#_Toc169266580)

[byName 17](#_Toc169266581)

[byType 18](#_Toc169266582)

[Spring Boot JDBC 18](#_Toc169266583)

[Spring MVC 23](#_Toc169266584)

[User Input 25](#_Toc169266585)

[Model and ModelMap 28](#_Toc169266586)

[Model Attribute 29](#_Toc169266587)

[Model Attribute di Method Level 31](#_Toc169266588)

[Contoh Membuat Project Spring MVC 32](#_Toc169266589)

[Post Mapping 37](#_Toc169266590)

[Get Mapping 38](#_Toc169266591)

[Spring ORM (Object Relational Mapping) 39](#_Toc169266592)

[Spring Hibernate Configuration 40](#_Toc169266593)

[Dao (Data Access Object) 41](#_Toc169266594)

[Add and Fetch 43](#_Toc169266595)

[Add/insert data 43](#_Toc169266596)

[Fetch/get data 44](#_Toc169266597)

[Spring Data JPA 45](#_Toc169266598)

[Fetch/get data 46](#_Toc169266599)

[Add data 46](#_Toc169266600)

[Query Dsl 47](#_Toc169266601)

[Query Annotation 47](#_Toc169266602)

[REST 47](#_Toc169266603)

[GetMapping 47](#_Toc169266604)

[PathVariable/ Fetch 49](#_Toc169266605)

[Post data 49](#_Toc169266606)

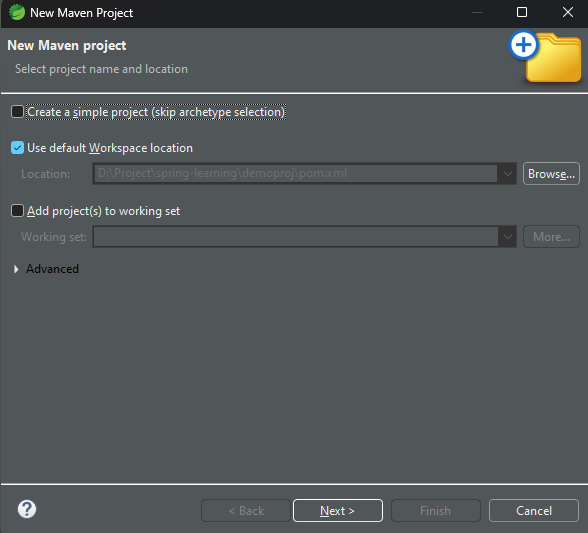
[Return XML 49](#_Toc169266607)

[Produces Atributes 50](#_Toc169266608)

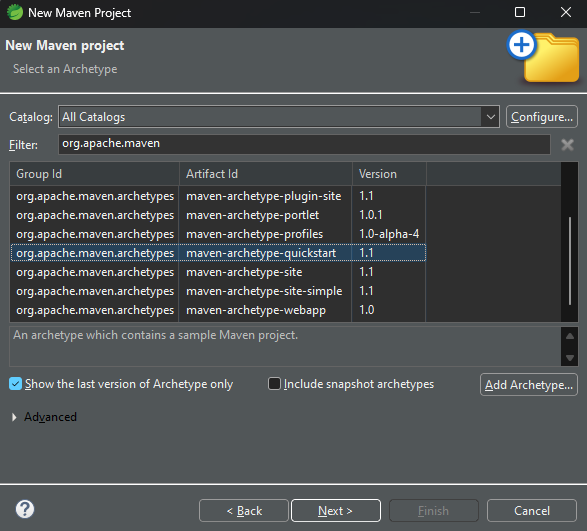
# Persiapan

Sebelum memulai project download dulu kebutuhan dependencies, Download Dependencies, menggunakan maven project

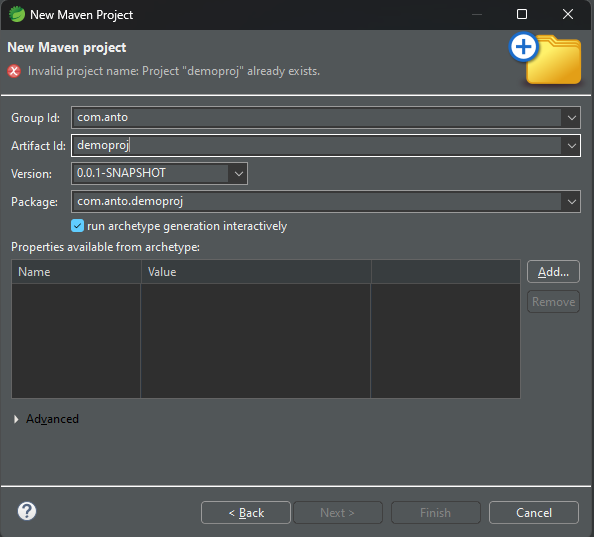
File > new maven project, next



Pilih maven-archtype-quickstart dan next



Isi Group-id = com.anto, artifact Id = demoproj, lalu Finish



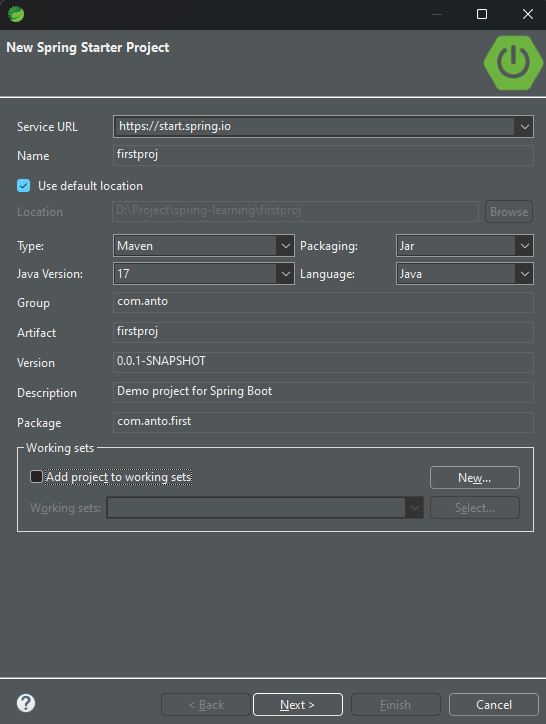
Akan muncul console, ketik Y

Untuk tambah dependencies buka pom.xml

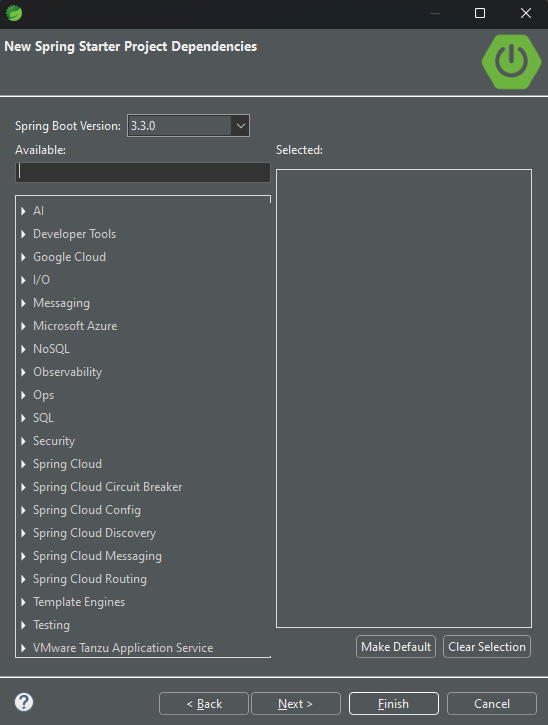
Untuk tau spring dependencies nya, buka <https://mvnrepository.com/search?q=spring> , pilih spring context, untuk versi tutorial ini kita menggunakan versi 6.1.8

Copy paste code dependency yg ada di tab Maven ke pom.xml di dalam tab <dependencies>

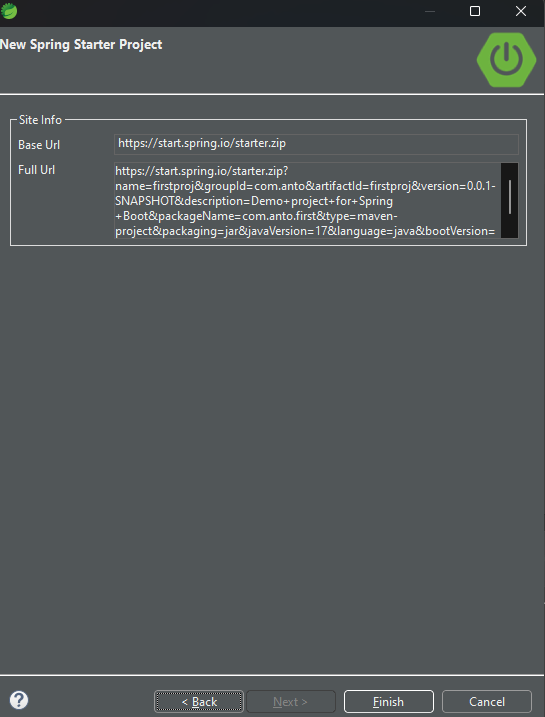
# Spring Starter Project



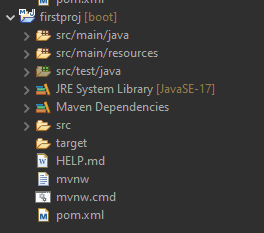
Isi Name, Type Maven, Group = com.anto, Artifact = fistproj, package = com.anto.first lalu next



Untuk ini, sementara langsung next saja



Finish, lalu tunggu proses download selesai



Buka src/main/java, lalu com.anto.first, FirstprojApplication.java

# Membuat Object

## Di spring boot

FirstprojApplication.java

package com.anto.first;

import org.springframework.boot.SpringApplication;

import org.springframework.context.ApplicationContext;

import org.springframework.boot.autoconfigure.SpringBootApplication;

*@SpringBootApplication*

public class FirstprojApplication {

public static void main(String[] args) {

ApplicationContext context = SpringApplication.*run*(FirstprojApplication.class, args);

// old method to create object

// Alien obj = new Alien();

// inject object in spring boot

Alien obj = context.getBean(Alien.class);

obj.code();

}

}

Pertama buat class bernama Alien terlebih dahulu, bisa dengan cara klik kanan agar muncul suggestion



Alien.java

package com.anto.first;

import org.springframework.stereotype.Component;

*@Component*

public class Alien

{

public void code()

{

System.***out***.println("Im Coding...");

}

}

Tambahkan @Component di atas nama class dan buat 1method, kemudian di FirstprojApplication import ApplicationContext

## AutoWired

Cara lain membuat object yg di handle spring dengan cara menambahkan @Autowired

Alien.java

package com.anto.first;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Component;

*@Component*

public class Alien

{

*@Autowired*

Laptop lap; // object ini akan otomatis dibuat oleh spring

public void code()

{

lap.compile();

}

}

Laptop.java

package com.anto.first;

import org.springframework.stereotype.Component;

*@Component*

public class Laptop {

public void compile()

{

System.***out***.println("Compiling....");

}

}

## Menggunakan XML

App.java

package com.anto.SpringDemo;

// import org.springframework.beans.factory.BeanFactory;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

// import org.springframework.core.io.FileSystemResource;

public class App

{

public static void main( String[] args )

{

ApplicationContext context= new ClassPathXmlApplicationContext("spring.xml");

Alien obj = (Alien)context.getBean("alien");

obj.code();

}

}

Alien.java

package com.anto.SpringDemo;

public class Alien

{

public void code()

{

System.***out***.println("Im coding...");

}

}

spring.xml

<?**xml** version=*"1.0"* encoding=*"UTF-8"*?>

<**beans** xmlns=*"http://www.springframework.org/schema/beans"*

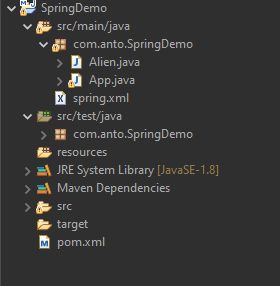
xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd"*>

<**bean** id=*"alien"* class=*"com.anto.SpringDemo.Alien"*></**bean**>

</**beans**>

Struktur file



\*untuk file spring.xml nya, bisa drag and drop file .xml ke folder src/main/java agar struktur nya seperti diatas

# Singleton Object vs Prototype

## Singleton

Alien.java

package com.anto.SpringDemo;

public class Alien

{

int age = 18;

public Alien()

{

System.***out***.println("Alien object created...");

}

public void code()

{

System.***out***.println("Im coding...");

}

}

App.java

package com.anto.SpringDemo;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class App

{

public static void main( String[] args )

{

ApplicationContext context= new ClassPathXmlApplicationContext("spring.xml");

Alien obj1 = (Alien)context.getBean("alien");

obj1.code();

obj1.age = 20;

System.***out***.println(obj1.age); // 20

Alien obj2 = (Alien)context.getBean("alien");

obj2.code();

System.***out***.println(obj2.age); // 20

}

}

Pada contoh diatas, meski kita define age =18 di class Alien, saat di main, kita deklarasi ulang age =20.

Saat membuat obj2 juga, age = 20 karena by default Spring beans menggunakan konsep singleton, obj1 dan obj2 me-refer ke object yg sama

Scope Prototype

Agar kita bisa menggunakan 2 objek pada file .xml tambahkan scope=*"prototype"*

Bila tidak kita tulis scope nya apa, berarti menggunakan “singleton”

spring.xml

<?**xml** version=*"1.0"* encoding=*"UTF-8"*?>

<**beans** xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd"*>

<**bean** id=*"alien"* class=*"com.anto.SpringDemo.Alien"* scope=*"prototype"*></**bean**>

</**beans**>

App.java

public static void main( String[] args )

{

ApplicationContext context= new ClassPathXmlApplicationContext("spring.xml");

Alien obj1 = (Alien)context.getBean("alien");

obj1.code();

obj1.age = 20;

System.***out***.println(obj1.age); // 20

Alien obj2 = (Alien)context.getBean("alien");

obj2.code();

System.***out***.println(obj2.age); // 18

}

Dengan menggunakan scope prototype kita dapat membuat 2 object yg berbeda

# Setter Injection

## Primitive type(int, Sting, dll)

Kita dapat men-set value properti pada bean xml, syarat nya di class harus dibuat setter getter dan default constructor

App.java

package com.anto.SpringDemo;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class App

{

public static void main( String[] args )

{

ApplicationContext context= new ClassPathXmlApplicationContext("spring.xml");

Alien obj1 = (Alien)context.getBean("alien");

obj1.code();

System.***out***.println(obj1.getAge()); // 10

}

}

Alien.java

package com.anto.SpringDemo;

public class Alien

{

private int age = 18;

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

public Alien()// default constructor

{

System.***out***.println("Alien object created...");

}

public void code()

{

System.***out***.println("Im coding...");

}

}

spring.xml

<?**xml** version=*"1.0"* encoding=*"UTF-8"*?>

<**beans** xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd"*>

<**bean** id=*"alien"* class=*"com.anto.SpringDemo.Alien"*>

<**property** name=*"age"* value=*"10"*></**property**>

</**bean**>

</**beans**>

## Reference Type

Me-inisialisasi type ref dari suatu class/object

App.java

public class App

{

public static void main( String[] args )

{

ApplicationContext context= new ClassPathXmlApplicationContext("spring.xml");

Alien obj1 = (Alien)context.getBean("alien");

obj1.code();

System.***out***.println(obj1.getAge()); // 10

}

}

Alien.java

public class Alien

{

private int age = 18;

private Laptop laptop; //memanggil class Laptop

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

public Laptop getLaptop() {

return laptop;

}

public void setLaptop(Laptop laptop) {

this.laptop = laptop;

}

public Alien()

{

System.***out***.println("Alien object created...");

}

public void code()

{

System.***out***.println("Im coding...");

laptop.compile(); // menggunakan method compile() yg ada di class laptop

}

}

Laptop.java

package com.anto.SpringDemo;

public class Laptop

{

public void compile()

{

System.***out***.println("Code Compiled...");

}

}

spring.xml

<**bean** id=*"alien"* class=*"com.anto.SpringDemo.Alien"*>

<**property** name=*"age"* value=*"10"*></**property**>

<**property** name=*"laptop"* ref=*"laptop"*></**property**>

</**bean**>

<**bean** id=*"laptop"* class=*"com.anto.SpringDemo.Laptop"*> <!-- id disini mengacu pada property dengan ref="laptop" -->

</**bean**>

pada Alien.java kita ingin menggunakan method compile() milik class Laptop, agar Laptop di kenal kita harus menambahkan bean dan property di spring.xml

# Constructor Injector

Menambahkan constructor with field di bean

Alien.java

public class Alien

{

private int age;

private Laptop laptop; //memanggil class Laptop

public Alien(int age) {// contructor with Field

this.age = age;

}

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

public Laptop getLaptop() {

return laptop;

}

public void setLaptop(Laptop laptop) {

this.laptop = laptop;

}

public void code()

{

System.***out***.println("Im coding...");

laptop.compile(); // menggunakan method compile() yg ada di class laptop

}

}

Spring.xml

<**bean** id=*"alien"* class=*"com.anto.SpringDemo.Alien"*>

<**constructor-arg** value=*"12"*></**constructor-arg**>

<**property** name=*"laptop"* ref=*"laptop"*></**property**>

</**bean**>

Saat program di jalankan, maka constructor(file xml) akan mengset variable this.age dengan value yg ada di XML

Kapan pakai Constructor injector & Setter injector?

Gunakan Constructor Injector kalau nilainya wajib ada/ required

Gunakan Setter Injector kalau nilainya tidak wajib

# Autowired di bean

Pertama buat interface Computer dan ubah Alien.java

Computer.java

ppackage com.anto.SpringDemo;

public interface Computer {

void compile();

}

Buat Class Desktop.java dan meng-implements interface Computer

package com.anto.SpringDemo;

public class Desktop implements Computer

{

public void compile()

{

System.***out***.println("Code Compiled in Desktop");

}

}

Ubah class Laptop.java dengan meng-implement Computer

package com.anto.SpringDemo;

public class Laptop implements Computer

{

public void compile()

{

System.***out***.println("Code Compiled in Laptop");

}

}

Ubah Alien.java dengan memanggil property/interface Computer, juga buat setter dan getter

package com.anto.SpringDemo;

public class Alien

{

private int age;

private Computer com;

public Alien()

{

System.***out***.println("Alien object created...");

}

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

public Computer getCom() {

return com;

}

public void setCom(Computer com) {

this.com = com;

}

public void code()

{

System.***out***.println("Im coding...");

com.compile();

}

}

Di spring.xml, set property untuk com

<**bean** id=*"alien"* class=*"com.anto.SpringDemo.Alien"*>

<**property** name=*"age"* value=*"10"*></**property**>

<**property** name=*"com"* ref=*"desktop"*></**property**>

</**bean**>

<**bean** id=*"laptop"* class=*"com.anto.SpringDemo.Laptop"*>

</**bean**>

<**bean** id=*"desktop"* class=*"com.anto.SpringDemo.Desktop"*>

</**bean**>

Bila di run maka outputnya:

Code Compiled in Laptop

10

Kenapa “Code compiled in Laptop” karena pada spring.xml kita membuat property com yg me-ref ke laptop

Bila kita ubah ref nya ke desktop, <**property** name=*"com"* ref=*"desktop"*></**property**> maka outputnya akan “Code Compiled in Desktop”

Bila kita membuat nama ref yg sama juga dengan nama property (sama2 com):

<**bean** id=*"alien"* class=*"com.anto.SpringDemo.Alien"*>

<**property** name=*"age"* value=*"10"*></**property**>

<**property** name=*"com"* ref=*"com"*></**property**>

</**bean**>

<**bean** id=*"com"* class=*"com.anto.SpringDemo.Laptop"*>

</**bean**>

Outputnya juga tetap sama “Code compiled in Laptop”, bila nama nya sama kita bisa singkat penulisan nya

Autowired, kita memberitau spring, jika ingin mengakses properti, dan jika ada bean yg available, otomatis konek. Ada 2 macam Autowired

## byName

pada contoh diatas, Alien.java property name private Computer com; adalah com dan bean name/id dari <**bean** id=*"com"* juga com, sehingga tanpa define tag <property> di xml, program akan otomatis di konek/sambungkan ke autowired byName

<**bean** id=*"alien"* class=*"com.anto.SpringDemo.Alien"* autowire=*"byName"*>

<**property** name=*"age"* value=*"10"*></**property**>

</**bean**>

<**bean** id=*"com"* class=*"com.anto.SpringDemo.Laptop"*>

</**bean**>

## byType

pada contoh diatas, Alien.java type property private Computer com; adalah Computer maka di xml nya, program akan mencari class yg meng-implement Computer, pada contoh di bawah class nya adalah Desktop

<**bean** id=*"alien"* class=*"com.anto.SpringDemo.Alien"* autowire=*"byType"*>

<**property** name=*"age"* value=*"10"*></**property**>

</**bean**>

<**bean** id=*"desktop"* class=*"com.anto.SpringDemo.Desktop"*>

</**bean**>

Output nya “Code Compiled in Desktop”

Jika ada 2 class di tag bean yg implement Computer, maka akan error, untuk mengatasinya tambahkan primary=*"true"*

<**bean** id=*"alien"* class=*"com.anto.SpringDemo.Alien"* autowire=*"byType"*>

<**property** name=*"age"* value=*"10"*></**property**>

</**bean**>

<**bean** id=*"com"* class=*"com.anto.SpringDemo.Laptop"* primary=*"true"*>

</**bean**>

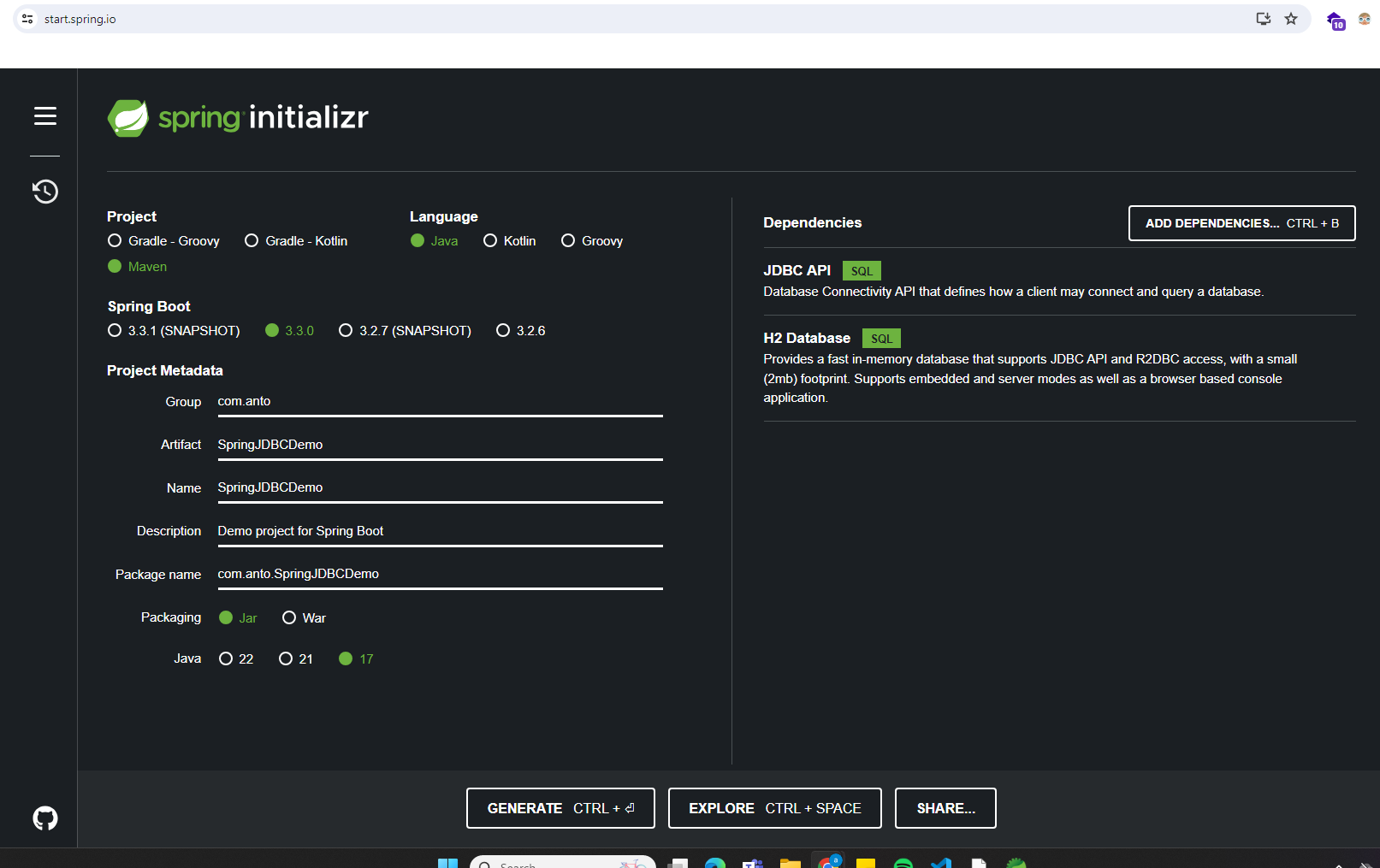
<**bean** id=*"desktop"* class=*"com.anto.SpringDemo.Desktop"*>

</**bean**>

Sehingga program akan mennjalankan yg Laptop

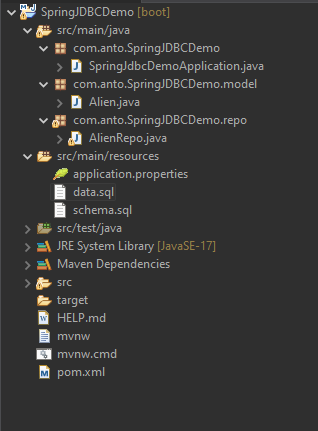
# Spring Boot JDBC

JDBC (Java Database Connectivity), buka pada browser <https://start.spring.io/>



Set pada project= Maven, Spring Boot version 3.3.0, isian Metadata sesuaikan gambar diatas. Untuk dependencies kita gunakan JDBC API, dan H2 Database(untuk tutorial ini)

Struktur file:



main()

package com.anto.SpringJDBCDemo;

import org.springframework.boot.SpringApplication;

import org.springframework.context.ApplicationContext;

import com.anto.SpringJDBCDemo.model.Alien;

import com.anto.SpringJDBCDemo.repo.AlienRepo;

import org.springframework.boot.autoconfigure.SpringBootApplication;

*@SpringBootApplication*

public class SpringJdbcDemoApplication {

public static void main(String[] args) {

ApplicationContext context = SpringApplication.*run*(SpringJdbcDemoApplication.class, args);

Alien alien1 = context.getBean(Alien.class);

alien1.setId(111);

alien1.setName("Navin");

alien1.setTech("Java");

AlienRepo repo = context.getBean(AlienRepo.class);

repo.save(alien1);

System.***out***.println(repo.findAll());

}

}

Alien.java

package com.anto.SpringJDBCDemo.model;

import org.springframework.context.annotation.Scope;

import org.springframework.stereotype.Component;

*@Component*

*@Scope*("prototype")

public class Alien {

private int id;

private String name;

private String tech;

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getTech() {

return tech;

}

public void setTech(String tech) {

this.tech = tech;

}

*@Override*

public String toString() {

return "Alien [id=" + id + ", name=" + name + ", tech=" + tech + "]";

}

}

AlienRepo.java

package com.anto.SpringJDBCDemo.repo;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.jdbc.core.JdbcTemplate;

import org.springframework.jdbc.core.RowMapper;

import org.springframework.stereotype.Repository;

import com.anto.SpringJDBCDemo.model.Alien;

*@Repository*

public class AlienRepo {

private JdbcTemplate template;

public JdbcTemplate getTemplate() {

return template;

}

*@Autowired*

public void setTemplate(JdbcTemplate template) {

this.template = template;

}

public void save(Alien alien)

{

String sql = "insert into alien (id,name,tech) values (?,?,?)";

int rows = template.update(sql, alien.getId(), alien.getName(), alien.getTech());

System.***out***.println(rows + " row/s affected");

}

public List<Alien> findAll()

{

String sql = "select \* from alien";

RowMapper<Alien> mapper = (rs, rowNum) -> {

Alien a = new Alien();

a.setId(rs.getInt(1));

a.setName(rs.getString(2));

a.setTech(rs.getString(3));

return a;

};

List<Alien> aliens = template.query(sql, mapper);

return aliens;

}

}

Schema.sql

create table alien (

id int primary key,

name varchar(50),

tech varchar(25)

);

Data.sql

insert into alien(id, name, tech) values (101, 'Kiran', 'BlockChain');

insert into alien(id, name, tech) values (102, 'Harsh', 'AI');

insert into alien(id, name, tech) values (103, 'Sushil', 'IoT');

Output:

1 row/s affected

[Alien [id=101, name=Kiran, tech=BlockChain], Alien [id=102, name=Harsh, tech=AI], Alien [id=103, name=Sushil, tech=IoT], Alien [id=111, name=Navin, tech=Java]]

Penjelasan:

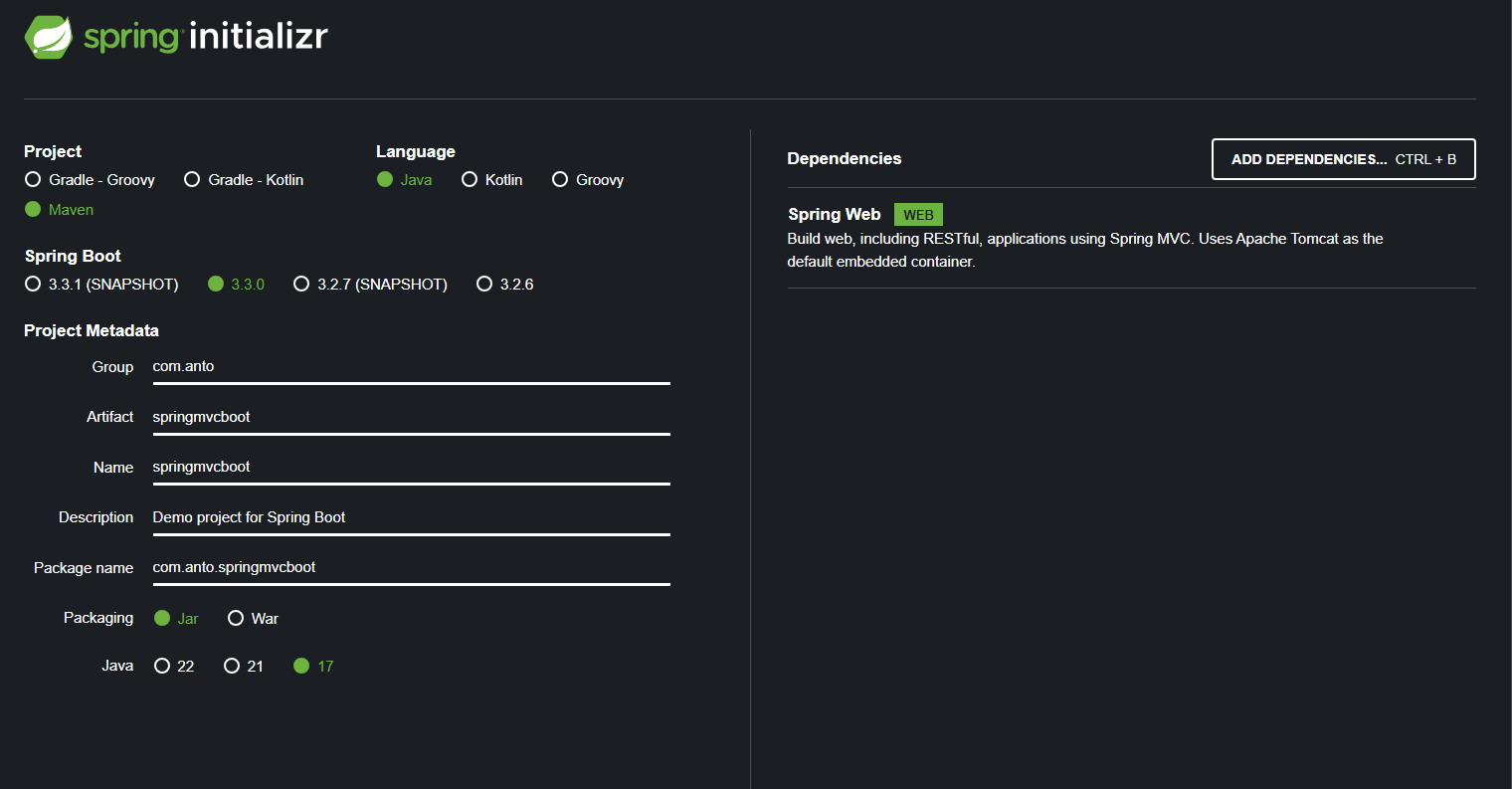
Karena kita menggunakan h2, maka database akan di simpan di memory, h2 akan otomatis mencreate table yg ada di schema.sql dan isi di data.sql

@Autowired juga memudahkan kita, jadi saat pertama object dibuat, akan langsung memanggil method dibawah nya, pada contoh diatas @Autowired ditulis diatas method setTemplate, artinya saat object dari AlienRepo dibuat otomatis di panggil method setTemplate

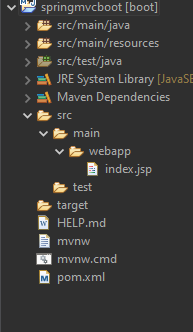
@Component, memberi tanda pada Spring bahwa class ini merupakan bean, jadi ketika menuliskan context.getBean(AlienRepo.class); program tau yg di maksud adalah Class/Bean AlienRepo

# Spring MVC

Untuk tutorial mvc kita menggunakan:



Struktur file nya



Buat folder main/webapp/index.jsp, (kalau tidak ada jsp di new file, install Eclipse Enterprise Java and Web di marketplace)

<%@ **page** language=*"java"* contentType=*"text/html; charset=ISO-8859-1"*

pageEncoding=*"ISO-8859-1"*%>

<!**DOCTYPE** html>

<**html**>

<**head**>

<**meta** charset=*"ISO-8859-1"*>

<**title**>Insert title here</**title**>

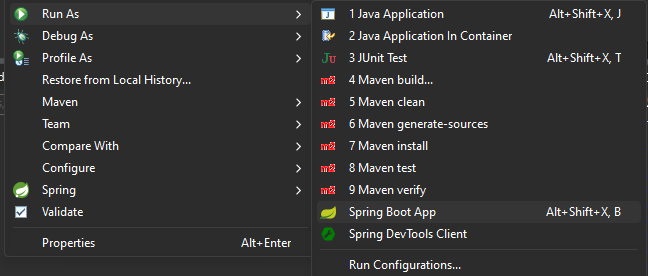
</**head**>

<**body**>

Welcome Hello World

</**body**>

Run as spring boot app



Buat class HomeController .java di src/main/java

package com.anto.springmvcboot;

import org.springframework.stereotype.Controller;

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

*@Controller*

public class HomeController

{

*@RequestMapping*("/") //ketika kita mengakses home page("/") jalankan method dibawah

public String home()

{

System.***out***.println("home page requested");

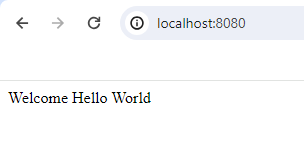
return "index.jsp";

}

}

Tambahkan TomcatJasper ke pom.xml dari <https://mvnrepository.com/artifact/org.apache.tomcat/tomcat-jasper/10.1.24>, untuk versi nya dapat dilihat pada Maven Dependencies, cari yg depan nya ‘tomcat-embeded-xxx.jar

Maka ketika kita run akan di jalankan file.jsp nya



## User Input

Index.jsp

<%@ **page** language=*"java"* contentType=*"text/html; charset=ISO-8859-1"*

pageEncoding=*"ISO-8859-1"*%>

<!**DOCTYPE** html>

<**html**>

<**head**>

<**meta** charset=*"ISO-8859-1"*>

<**title**>Insert title here</**title**>

</**head**>

<**body**>

Welcome Hello World

<**form** action=*"add"*>

Enter First Number:

<**input** type=*"text"* name=*"num1"*><**br**>

Enter Second Number:

<**input** type=*"text"* name=*"num2"*><**br**>

<**input** type=*"submit"*>

</**form**>

</**body**>

</**html**>

Src/main/java/HomeController.java

package com.anto.springmvcboot;

import org.springframework.stereotype.Controller;

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

import jakarta.servlet.http.HttpServletRequest;

import jakarta.servlet.http.HttpSession;

*@Controller*

public class HomeController

{

*@RequestMapping*("/") //ketika mengakses home page("/") jalankan method dibawah

public String home()

{

System.***out***.println("home page requested");

return "index.jsp";

}

*@RequestMapping*("add")

public String add(HttpServletRequest req)

{

// cara 1 pakai HttpServletRequest

int i = Integer.*parseInt*(req.getParameter("num1"));

int j = Integer.*parseInt*(req.getParameter("num2"));

int num3 = i+j;

HttpSession session = req.getSession();

session.setAttribute("num3", num3);

return "result.jsp";

}

}

Cara 2

*@RequestMapping*("add")

public String add(*@RequestParam*("num1")int i, *@RequestParam*("num2") int j, HttpSession session)

{

// cara 2 mengunakan @RequestParam nya Spring

int num3 = i+j;

session.setAttribute("num3", num3);

return "result.jsp";

}

Tanpa HttpSession, menggunakan ModelAndView:

*@RequestMapping*("add")

public ModelAndView add(*@RequestParam*("num1")int i, *@RequestParam*("num2") int j)

{

ModelAndView mv = new ModelAndView();

mv.setViewName("result.jsp");

int num3 = i+j;

mv.addObject("num3", num3);

return mv;

}

Src/main/webapp/result.jsp

<%@ **page** language=*"java"* contentType=*"text/html; charset=ISO-8859-1"*

pageEncoding=*"ISO-8859-1"* isELIgnored=*"false"*%>

<!**DOCTYPE** html>

<**html**>

<**head**>

<**meta** charset=*"ISO-8859-1"*>

<**title**>Insert title here</**title**>

</**head**>

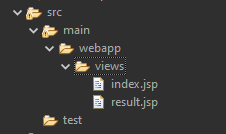
<**body**>

Result is: ${num3}

</**body**>

</**html**>

Dengan prefix dan sufix



Pada HomeController kita tidak perlu mendefine .jsp / nama extension nya

*@Controller*

public class HomeController

{

*@RequestMapping*("/")

public String home()

{

System.***out***.println("home page requested");

return "index";

}

*@RequestMapping*("add")

public ModelAndView add(*@RequestParam*("num1")int i, *@RequestParam*("num2") int j)

{

ModelAndView mv = new ModelAndView();

mv.setViewName("result");

//atau

ModelAndView mv = new ModelAndView("result");

int num3 = i+j;

mv.addObject("num3", num3);

return mv;

int num3 = i+j;

mv.addObject("num3", num3);

return mv;

}

}

Caranya dengan manambahkan view.prefix dan view sufix di src/main/resources/application.properties

spring.application.name=springmvcboot

spring.mvc.view.prefix= /views/

spring.mvc.view.suffix= .jsp

prefix disini adalah nama folder, dan suffix adalah extension dari nama file

## Model and ModelMap

Meggunakan Model:

import org.springframework.ui.Model;

*@Controller*

public class HomeController

{

*@RequestMapping*("/")

public String home()

{

System.***out***.println("home page requested");

return "index";

}

*@RequestMapping*("add")

public String add(*@RequestParam*("num1")int i, *@RequestParam*("num2") int j, Model m)

{

int num3 = i+j;

m.addAttribute("num3", num3);

return "result";

}

}

Menggunakan ModelMap:

import org.springframework.ui.ModelMap;

*@RequestMapping*("add")

public String add(*@RequestParam*("num1")int i, *@RequestParam*("num2") int j, ModelMap m)

{

int num3 = i+j;

m.addAttribute("num3", num3);

return "result";

}

## Model Attribute

Cara manual/cara biasa (di HomeController):

Index.jsp

<**html**>

<**head**>

<**meta** charset=*"ISO-8859-1"*>

<**title**>Insert title here</**title**>

</**head**>

<**body**>

Welcome Hello World

<**form** action=*"addAlien"*>

Enter your id:

<**input** type=*"text"* name=*"aid"*><**br**>

Enter your name:

<**input** type=*"text"* name=*"aname"*><**br**>

<**input** type=*"submit"*>

</**form**>

</**body**>

</**html**>

src/main/java/model/Alien.java

package com.anto.springmvcboot.model;

public class Alien {

private int aid;

private String aname;

public String getAname() {

return aname;

}

public void setAname(String aname) {

this.aname = aname;

}

public int getAid() {

return aid;

}

public void setAid(int aid) {

this.aid = aid;

}

*@Override*

public String toString() {

return "Alien [aid=" + aid + ", aname=" + aname + "]";

}

}

src/main/java/HomeController.java

*@Controller*

public class HomeController

{

*@RequestMapping*("addAlien")

public String addAlien(*@RequestParam*("aid")int aid, *@RequestParam*("aname") String aname, Model m)

{

Alien a = new Alien();

a.setAid(aid);

a.setAname(aname);

m.addAttribute("alien", a);

return "result";

}

}

src/main/webapp/views/result.jsp

<%@ **page** language=*"java"* contentType=*"text/html; charset=ISO-8859-1"*

pageEncoding=*"ISO-8859-1"* isELIgnored=*"false"*%>

<!**DOCTYPE** html>

<**html**>

<**head**>

<**meta** charset=*"ISO-8859-1"*>

<**title**>Insert title here</**title**>

</**head**>

<**body**>

Result is: ${alien}

</**body**>

</**html**>

Menggunakan ModelAttribute:

*@RequestMapping*("addAlien")

public String addAlien(*@ModelAttribute*("resultAlien") Alien a)

{

return "result";

}

result.jsp

<**html**>

<**head**>

<**meta** charset=*"ISO-8859-1"*>

<**title**>Insert title here</**title**>

</**head**>

<**body**>

Result is: ${resultAlien}

</**body**>

</**html**>

Disini @ModelAttribute otomatis men-set Attribute aid dan aname, serta otomatis membuat model dan menambahkan nya

Tanpa custom name

*@RequestMapping*("addAlien")

public String addAlien(*@ModelAttribute* Alien a)

{

return "result";

}

}

<**html**>

<**head**>

<**meta** charset=*"ISO-8859-1"*>

<**title**>Insert title here</**title**>

</**head**>

<**body**>

Result is: ${alien}

</**body**>

</**html**>

## Model Attribute di Method Level

<**html**>

<**head**>

<**meta** charset=*"ISO-8859-1"*>

<**title**>Insert title here</**title**>

</**head**>

<**body**>

Result is: ${alien}

Welcome ${name}

</**body**>

</**html**>

Misal kita ingin menampilkan name di result.jsp

*@Controller*

public class HomeController

{

*@ModelAttribute*

public void modelData(Model m)

{

m.addAttribute("name", "Aliens");

}

*@RequestMapping*("/")

public String home()

{

System.***out***.println("home page requested");

return "index";

}

*@RequestMapping*("addAlien")

public String addAlien(*@ModelAttribute* Alien a)

{

return "result";

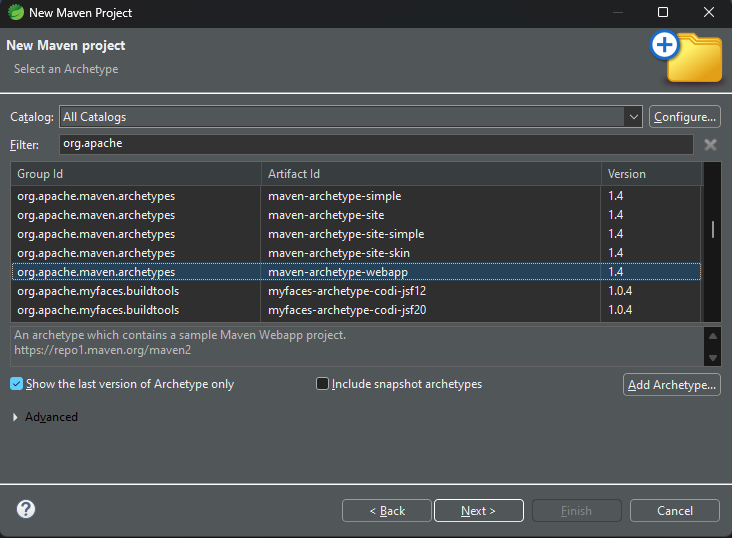
}

}

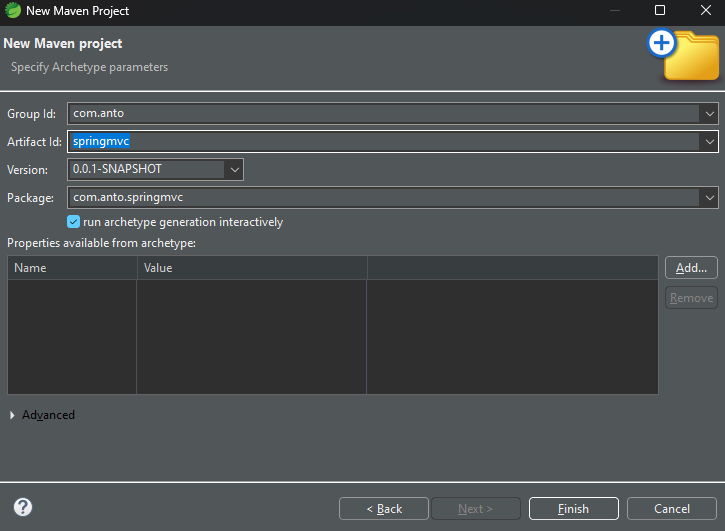
Kita dapat menambahkan anotation @ModelAttribute diatas method modeldata. Annotasi ini akan menajalankan method dibawah nya, sebelum RequestMapping

# Contoh Membuat Project Spring MVC

New > Maven Project

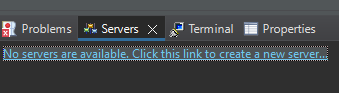


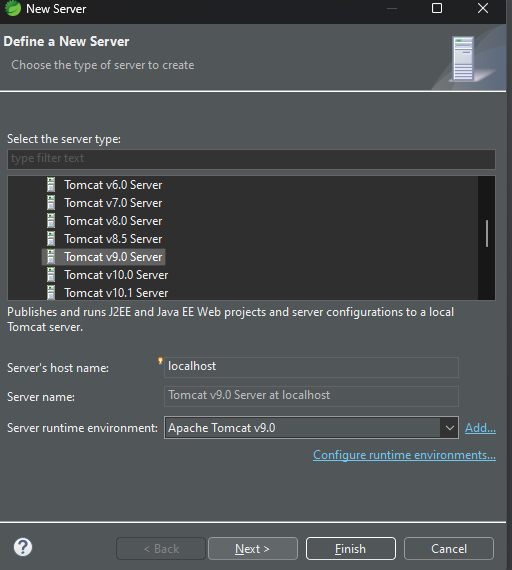
Cari yg maven-archtype-webapp

set groupId, ArtifactId lalu finish

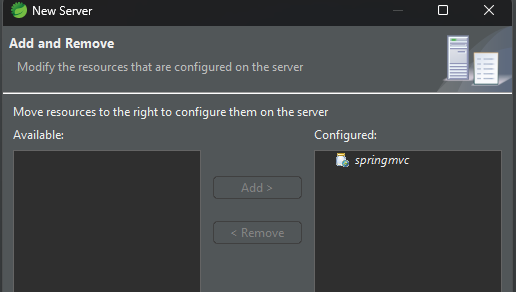
Tambah server tomcat dengan cara Windows,Preference, Server, Runtime Environtments, Add, Pilih Apache> Tomcat 9 (Bisa di download dulu Tomcat nya)

Untuk memnculkan View/Tab Server, bisa lewat Windows>Perspective>Open Perspective>Other>Java EE. Lalu bisa klik new,





Pilih Tomcat 10.1 saja, lalu next



Add project nya ke kanan, lalu finish

Klik Kanan lagi di project, pilih properties> Java Build Path> Tab Libraries> Add Libraries> Server Runtime> Apache Tomcat 9>Finish> Apply. Maka seharusnya error di index.jsp nya sudah hilang

Tambahkan Spring dependency di pom.xml, bisa di download di <https://mvnrepository.com/artifact/org.springframework/spring-webmvc/6.1.8>

Run Project dengan cara, klik kanan nama project>Run As> Run on Server, next, finish(karena sudah set server tomcat sblumnya)

Buka file web.xml di src/main/webapp/WEB-INF/web.xml, lalu paste kan berikut:

<!**DOCTYPE** web-app PUBLIC

"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"

"http://java.sun.com/dtd/web-app\_2\_3.dtd" >

<**web-app**>

<**servlet**>

<**servlet-name**>anto</**servlet-name**>

<**servlet-class**>org.springframework.web.servlet.DispatcherServlet</**servlet-class**>

</**servlet**>

<**servlet-mapping**>

<**servlet-name**>anto</**servlet-name**>

<**url-pattern**>/</**url-pattern**>

</**servlet-mapping**>

</**web-app**>

Buat file antp-servlet.xml di src/main/webapp/WEB-INF/anto-servlet.xml

<?**xml** version=*"1.0"* encoding=*"UTF-8"*?>

<**beans** xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:ctx=*"http://www.springframework.org/schema/context"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns:mvc=*"http://www.springframework.org/schema/mvc"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

*https://www.springframework.org/schema/beans/spring-beans-2.5.xsd*

*http://www.springframework.org/schema/mvc*

*http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd*

*http://www.springframework.org/schema/context*

*http://www.springframework.org/schema/context/spring-context-2.5.xsd "*>

<**ctx:component-scan** base-package=*"com.anto"*></**ctx:component-scan**>

<**ctx:annotation-config**></**ctx:annotation-config**>

<**bean** class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>

<!-- <property name="viewClass" value="org.springframework.web.servlet.view.JstlView"></property>-->

<**property** name=*"prefix"* value=*"/views/"*></**property**>

<**property** name=*"suffix"* value=*".jsp"*></**property**>

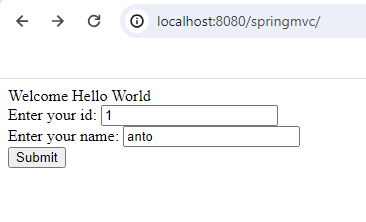
</**bean**>

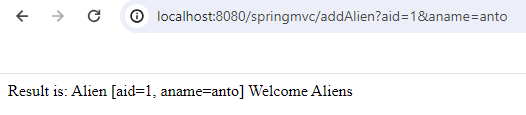
</**beans**>

Pastikan pada klik kanan project, properties, deployment assembly, add, maven dependencies

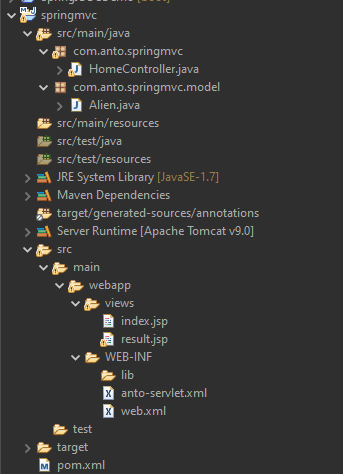
Kemudian klik kanan project, run as , run on server

Output:





Struktur project:



Untuk controller dan model, dapat di copy dari project sblumnya (springmvc)

# Post Mapping

Mensubmit form mengunakan method post

<**html**>

<**head**>

<**meta** charset=*"ISO-8859-1"*>

<**title**>Insert title here</**title**>

</**head**>

<**body**>

Welcome Hello World

<**form** action=*"addAlien"* method=*"post"*>

Enter your id:

<**input** type=*"text"* name=*"aid"*><**br**>

Enter your name:

<**input** type=*"text"* name=*"aname"*><**br**>

<**input** type=*"submit"*>

</**form**>

</**body**>

</**html**>

Bisa juga dengan me-restrict dari sisi method, jadi method ini hanya menerima method POST

HomeController.java

*@RequestMapping*(value="addAlien", method = *RequestMethod*.***POST***)

public String addAlien(*@ModelAttribute* Alien a)

{

return "result";

}

Atau

*@PostMapping*(value="addAlien")

public String addAlien(*@ModelAttribute* Alien a)

{

return "result";

}

# Get Mapping

Alien.java, tambahkan constructor

package com.anto.springmvcboot.model;

public class Alien {

private int aid;

private String aname;

public Alien(int aid, String aname) {

super();

this.aid = aid;

this.aname = aname;

}

HomeController.java

*@GetMapping*("getAliens")

public String getAliens(Model m)

{

List<Alien>aliens = Arrays.*asList*(new Alien(101,"Navin"), new Alien(102,"Rose"));

m.addAttribute("result", aliens);

return "showAliens";

}

/src/main/webapp/views/showAliens.jsp

<%@ **page** language=*"java"* contentType=*"text/html; charset=ISO-8859-1"*

pageEncoding=*"ISO-8859-1"* isELIgnored=*"false"*%>

<!**DOCTYPE** html>

<**html**>

<**head**>

<**meta** charset=*"ISO-8859-1"*>

<**title**>Insert title here</**title**>

</**head**>

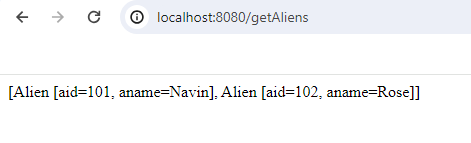
<**body**>

${result}

</**body**>

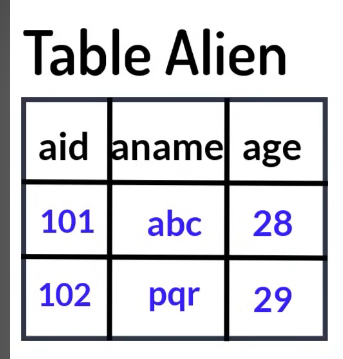
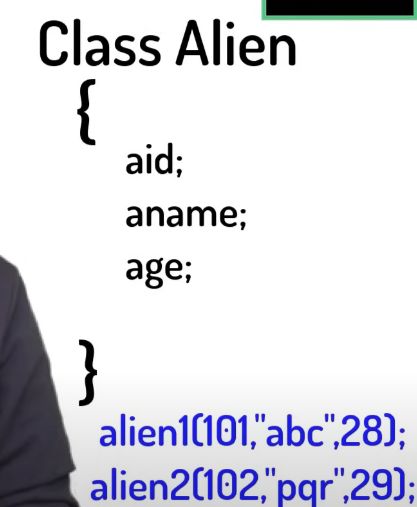
</**html**>

Ketika kita akses url: <http://localhost:8080/getAliens>, maka outputnya



# Spring ORM (Object Relational Mapping)

Menghubungkan antara table Db dengan Class



# Spring Hibernate Configuration

Untuk ini kita menggunakan project Spring saja (springmvc)

Pom.xml, tambahkan dependency dari <https://mvnrepository.com/artifact/org.hibernate.orm/hibernate-core>

Tambahkan dependency dari

<https://mvnrepository.com/artifact/org.springframework/spring-orm>

Tambahkan dependency dari

<https://mvnrepository.com/artifact/org.springframework/spring-tx>

Tambahkan

<https://mvnrepository.com/artifact/com.mysql/mysql-connector-j>

Tambahkan

<https://mvnrepository.com/artifact/com.mchange/c3p0/0.10.1>

buka anto-servlet.xml, tambahkan:

xmlns:tx=*"http://www.springframework.org/schema/tx"*

dan

*http://www.springframework.org/schema/tx*

*http://www.springframework.org/schema/tx/spring-tx.xsd*

final anto-servlet.xml

<?**xml** version=*"1.0"* encoding=*"UTF-8"*?>

<**beans** xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:ctx=*"http://www.springframework.org/schema/context"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns:mvc=*"http://www.springframework.org/schema/mvc"*

xmlns:tx=*"http://www.springframework.org/schema/tx"*

xsi:schemaLocation=*"*

*http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-2.5.xsd*

*http://www.springframework.org/schema/mvc*

*http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd*

*http://www.springframework.org/schema/tx*

*http://www.springframework.org/schema/tx/spring-tx.xsd*

*http://www.springframework.org/schema/context*

*http://www.springframework.org/schema/context/spring-context-2.5.xsd"*>

<**ctx:component-scan** base-package=*"com.anto"*/>

<**ctx:annotation-config**/>

<**bean** id=*"myDataSource"* class=*"com.mchange.v2.c3p0.ComboPooledDataSource"* destroy-method=*"close"*>

<**property** name=*"driverClass"* value=*"com.mysql.cj.jdbc.Driver"*/>

<**property** name=*"jdbcUrl"* value=*"jdbc:mysql://localhost:3306/anto"*/>

<**property** name=*"user"* value=*"root"*/>

<**property** name=*"password"* value=*""*/>

<**property** name=*"minPoolSize"* value=*"5"*/>

<**property** name=*"maxPoolSize"* value=*"10"*/>

<**property** name=*"maxIdleTime"* value=*"30000"*/>

</**bean**>

<**bean** id=*"sessionFactory"* class=*"org.springframework.orm.hibernate5.LocalSessionFactoryBean"*>

<**property** name=*"dataSource"* ref=*"myDataSource"*/>

<**property** name=*"packagesToScan"* value=*"com.anto.springmvc.model"*/>

<**property** name=*"hibernateProperties"*>

<**props**>

<**prop** key=*"hibernate.dialect"*>org.hibernate.dialect.MySQLDialect</**prop**>

<**prop** key=*"hibernate.show\_sql"*>true</**prop**>

</**props**>

</**property**>

</**bean**>

<**bean** id=*"myTransactionManager"* class=*"org.springframework.orm.hibernate5.HibernateTransactionManager"*>

<**property** name=*"sessionFactory"* ref=*"sessionFactory"*/>

</**bean**>

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

<**bean** class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>

<!-- <property name="viewClass" value="org.springframework.web.servlet.view.JstlView"></property>-->

<**property** name=*"prefix"* value=*"/views/"*/>

<**property** name=*"suffix"* value=*".jsp"*/>

</**bean**>

</**beans**>

## Dao (Data Access Object)

Buat new class sebagai layer, kalau nama table nya Alien, baiknya kita buat class dengan nama AlienDao di dalam package dao

/src/main/java/com/anto/springmvc/dao/AlienDao.java

package com.anto.springmvc.dao;

import java.util.List;

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Component;

import com.anto.springmvc.model.Alien;

import org.springframework.transaction.annotation.Transactional;

// import org.springframework.stereotype.Repository;

// import jakarta.transaction.Transactional;

*@Component*

public class AlienDao {

*@Autowired*

private SessionFactory sessionFactory;

*@Transactional*

public List<Alien> getAliens()

{

Session session = sessionFactory.getCurrentSession();

List<Alien> aliens= session.createQuery("from Alien", Alien.class).list();//from Alien disini adalah nama class

return aliens;

}

}

Pada class Alien tambahkan @Entity dan @Id:

*@Entity*

public class Alien {

*@Id*

private int aid;

private String aname;

HomeController.jsp

*@GetMapping*("getAliens")

public String getAliens(Model m)

{

m.addAttribute("result", dao.getAliens());

return "showAliens";

}

src/main/webapp/views/showAliens.jsp

<%@ **page** language=*"java"* contentType=*"text/html; charset=ISO-8859-1"*

pageEncoding=*"ISO-8859-1"* isELIgnored=*"false"*%>

<!**DOCTYPE** html>

<**html**>

<**head**>

<**meta** charset=*"ISO-8859-1"*>

<**title**>Insert title here</**title**>

</**head**>

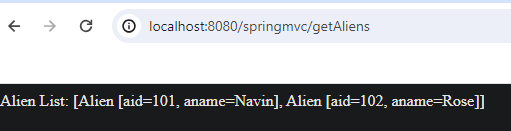
<**body**>

Alien List: ${result}

</**body**>

</**html**>

Output:



## Add and Fetch

### Add/insert data

Index.jsp

<**html**>

<**head**>

<**meta** charset=*"ISO-8859-1"*>

<**title**>Insert title here</**title**>

</**head**>

<**body**>

Welcome Hello World

<**form** action=*"addAlien"*>

Enter your id:

<**input** type=*"text"* name=*"aid"*><**br**>

Enter your name:

<**input** type=*"text"* name=*"aname"*><**br**>

<**input** type=*"submit"*>

</**form**>

</**body**>

</**html**>

HomeController.java

*@RequestMapping*("addAlien")

public String addAlien(*@ModelAttribute*("result") Alien a)

{

dao.addAlien(a);

return "showAliens";

}

AlienDao.java

*@Transactional*

public void addAlien(Alien a)

{

Session session = sessionFactory.getCurrentSession();

// session.save(a); -> deprecated

session.persist(a);

}

showAliens.jsp

<**html**>

<**head**>

<**meta** charset=*"ISO-8859-1"*>

<**title**>Insert title here</**title**>

</**head**>

<**body**>

Alien List: ${result}

</**body**>

</**html**>

### Fetch/get data

Index.jsp

<**form** action=*"getAlien"* method=*"get"*>

Enter your id:

<**input** type=*"text"* name=*"aid"*><**br**>

<**input** type=*"submit"*>

</**form**>

</**body**>

</**html**>

HomeController.java

*@GetMapping*("getAlien")

public String getAlien(*@RequestParam*("aid") int aid, Model m)

{

m.addAttribute("result", dao.getAlien(aid));

return "showAliens";

}

AlienDao.java

*@Transactional*

public Alien getAlien(int aid) {

Session session = sessionFactory.getCurrentSession();

Alien a = session.get(Alien.class, aid);

return a;

}

# Spring Data JPA

Untuk ini kita pakai project Spring Boot (nama project: springmvcboot)

Install dependency <https://mvnrepository.com/artifact/com.mysql/mysql-connector-j/8.3.0>

Install dependency <https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-data-jpa/3.3.0>

untuk yg spring-boot-starter-data-jpa hapus tag <version> nya

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

Application.properties

spring.application.name=springmvcboot

spring.mvc.view.prefix=/views/

spring.mvc.view.suffix=.jsp

spring.datasource.url=jdbc:mysql://localhost:3306/anto

spring.datasource.username=root

spring.datasource.password=

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect

untuk hibernate.dialect, sesuaikan dengan versi mysql nya. Karena kita pakai mysql-connector versi 8, pakai ‘org.hibernate.dialect.MySQL8Dialect’

/src/main/java/com/anto/springmvcboot/AlienRepo.java

package com.anto.springmvcboot;

import org.springframework.data.jpa.repository.JpaRepository;

import com.anto.springmvcboot.model.Alien;

public interface AlienRepo extends JpaRepository<Alien, Integer>{// Integer adalah type primary key

}

Alien.java

Tambahkan @Entity dan @Id dan constructor Alien(){}

*@Entity*

public class Alien {

*@Id*

private int aid;

private String aname;

public Alien() {} // default consructor

public Alien(int aid, String aname) {

super();

this.aid = aid;

this.aname = aname;

}

HomeController.java

*@Controller*

public class HomeController

{

*@Autowired*

AlienRepo repo;

*@GetMapping*("getAliens")

public String getAliens(Model m)

{

m.addAttribute("result", repo.findAll());

return "showAliens";

}

Output:

[Alien [aid=101, aname=Navin], Alien [aid=102, aname=Rose], Alien [aid=103, aname=Kiran], Alien [aid=104, aname=Clorinde], Alien [aid=105, aname=chiori], Alien [aid=106, aname=Kaveh]]

## Fetch/get data

*@GetMapping*("getAlien")

public String getAlien(*@RequestParam* int aid, Model m)

{

// m.addAttribute("result", repo.getOne(aid)); // getOne, Deprecated

m.addAttribute("result", repo.getReferenceById(aid));

return "showAliens";

}

## Add data

*@PostMapping*(value="addAlien")

public String addAlien(*@ModelAttribute* Alien a)

{

repo.save(a);

return "result";

}

## Query Dsl

Misal kita mau mencari data by name

Index.jsp

<**form** action=*"getAlienByName"* method=*"get"*>

Enter your name:

<**input** type=*"text"* name=*"aname"*><**br**>

<**input** type=*"submit"*>

</**form**>

HomeController.java

*@GetMapping*("getAlienByName")

public String getAlienByName(*@RequestParam* String aname, Model m)

{

m.addAttribute("result", repo.findByAname(aname));

return "showAliens";

}

AlienRepo.java

public interface AlienRepo extends JpaRepository<Alien, Integer>{

List<Alien> findByAnameOrderByAidDesc (String aname); //Query DSL(Domain Spesific Laguage)

}

JPA akan otomatis, perhatikan penulisan findByAname, harus dimulai dengan findBy/getBy, dilanjutkan dengan nama variable/kolom di Db? dengan huruf depan kapital. Selanjutnya bisa kita tambah dengan OrderBy Aid secara Desc

## Query Annotation

public interface AlienRepo extends JpaRepository<Alien, Integer>{

*@Query*("from Alien where aname= :name")

List<Alien> find(*@Param*("name") String aname);

}

# REST

Untuk Rest kita menggunakan Spring Boot (springmvcboot)

## GetMapping

Buat file controller baru

src/main/java/com/anto/springmvcboot/AlienController.java

package com.anto.springmvcboot;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.ResponseBody;

import com.anto.springmvcboot.model.Alien;

*@Controller*

public class AlienController

{

*@Autowired*

AlienRepo repo;

*@GetMapping*("aliens")

*@ResponseBody*

public List<Alien> getAliens()

{

List<Alien> aliens = repo.findAll();

return aliens;

}

}

Kalau di hit <http://localhost:8080/aliens> dari postman, outputnya:

## PathVariable/ Fetch

AlienController.java

*@GetMapping*("alien/{aid}")

*@ResponseBody*

public Alien getAlien(*@PathVariable*("aid") int aid) {

Alien alien = repo.findById(aid).orElse(new Alien(0, ""));

return alien;

}

Agar tidak menulis @ResponseBody di tiap method baru, tambahkan @RestController di awal class. Harus make sure dulu kita tidak membuat method lain selain untuk return json/xml. Jika mau campur method biasa dengan method api, pakai cara @ResponseBody

*@RestController*

public class AlienController

{

*@Autowired*

AlienRepo repo;

*@GetMapping*("aliens")

public List<Alien> getAliens()

{

List<Alien> aliens = repo.findAll();

return aliens;

}

*@GetMapping*("alien/{aid}")

public Alien getAlien(*@PathVariable*("aid") int aid) {

Alien alien = repo.findById(aid).orElse(new Alien(0, ""));

return alien;

}

}

## Post data

*@PostMapping*("alien")

public Alien addAlien(Alien alien) {

repo.save(alien);

return alien;

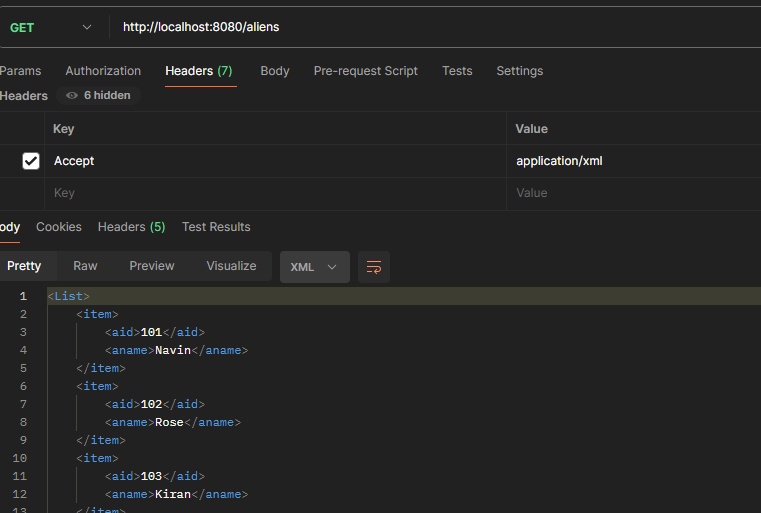
}

## Return XML

Di pom.xml tambahkan dependency dari <https://mvnrepository.com/artifact/com.fasterxml.jackson.dataformat/jackson-dataformat-xml/2.17.1>

Untuk versi nya samakan dengan versi jackson-core yg ada di Maven Dependencies

Postman nya:



## Produces Attributes

Meng-set return type yg akan di berikan ke client, by default return json

AlienController.java

*@GetMapping*(path="aliens", produces={"application/json"})

public List<Alien> getAliens()

{

List<Alien> aliens = repo.findAll();

return aliens;

}

Contoh datas, GET aliens, hanya bisa return xml

## Request Body & Consume Attributes

Bila client melakukan POST menggunakan body (json/xml) tambahkan @RequestBody

AlienController.java

*@PostMapping*("alien")

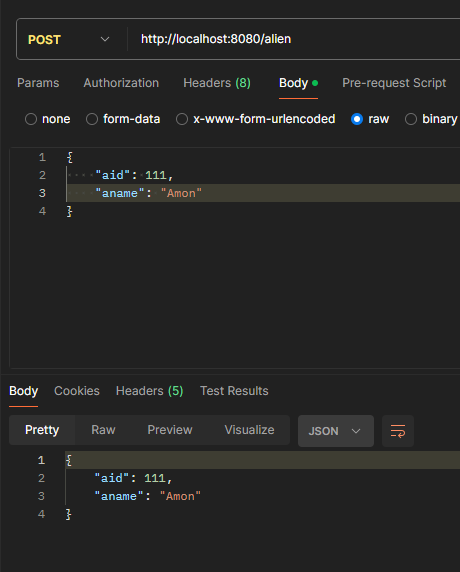
public Alien addAlien(*@RequestBody* Alien alien) {

repo.save(alien);

return alien;

}

Postman



Agar Endpoint hanya menerima jenis body tertentu(json/xml)

*@PostMapping*(path= "alien", consumes= {"application/json"})

public Alien addAlien(*@RequestBody* Alien alien) {

repo.save(alien);

return alien;

}

Contoh error menggunakan body xml

