MySQL 5/8

Hibernate 5.4

Spring Boot 2.0

Angular 7

Don't Copy Paste and Waste Your Time, you will only fool yourself

Date : 14/02/2019
Prerequisites:
MySQL 5 or 8 Server
Gradle
Jdk 1.8 or above
Environment variable GRADLE_HOME pointing to the gradle folder that contains bin folder.
Environment variable JAVA_HOME pointing to the jdk folder that contains the bin folder.

Very important note : you are not supposed to include bin when you are specifying the home part.

on c:\ create a folder named as showcase
in it create a folder named as jpa
in jpa create a folder named as db
Note : db folder will contain our script file to create database

showcase.sql

```
drop table IF EXISTS downloadable;
drop table IF EXISTS hosted;
drop table IF EXISTS video;
drop table IF EXISTS screen_shot;
drop table IF EXISTS category;
drop table IF EXISTS project;

Create table project (
	code Int NOT NULL AUTO_INCREMENT,
	title Char(50) NOT NULL,
	category_code Int NOT NULL,
	domain Char(50) NOT NULL,
	synopsis Varchar(2000) NOT NULL,
	technologies Char(200) NOT NULL,
	UNIQUE (title),
 Primary Key (code)) ENGINE = InnoDB;

Create table category (
	code Int NOT NULL AUTO_INCREMENT,
	title Char(50) NOT NULL,
	UNIQUE (title),
 Primary Key (code)) ENGINE = InnoDB;

Create table screen_shot (
	code Int NOT NULL AUTO_INCREMENT,
	project_code Int NOT NULL,
	serial_number Int NOT NULL,
	title Char(100) NOT NULL,
	file Char(100) NOT NULL,
	note Varchar(300) NOT NULL,
	UNIQUE (file),
```

```
 Primary Key (code)) ENGINE = InnoDB;

Create table video (
        code Int NOT NULL AUTO_INCREMENT,
        project_code Int NOT NULL,
        serial_number Int NOT NULL,
        title Char(100) NOT NULL,
        file Char(100) NOT NULL,
        note Varchar(20) NOT NULL,
        duration Int NOT NULL,
 Primary Key (code)) ENGINE = InnoDB;

Create table hosted (
        project_code Int NOT NULL,
        url Char(100) NOT NULL,
 Primary Key (project_code)) ENGINE = InnoDB;

Create table downloadable (
        code Int NOT NULL AUTO_INCREMENT,
        project_code Int,
        download_type Char(1) NOT NULL,
        url Char(100) NOT NULL,
 Primary Key (code)) ENGINE = InnoDB;

Alter table screen_shot add Foreign Key (project_code) references project (code) on delete  restrict on
update  restrict;
Alter table video add Foreign Key (project_code) references project (code) on delete  restrict on update
restrict;
Alter table hosted add Foreign Key (project_code) references project (code) on delete  restrict on
update  restrict;
Alter table downloadable add Foreign Key (project_code) references project (code) on delete  restrict
on update  restrict;
Alter table project add Foreign Key (category_code) references category (code) on delete  restrict on
update  restrict;
```

---

create database showcasedb, then create user showcase with password as thinkingmachines and grant all privileges on showcasedb to use showcase

---

login as showcase user and use showcasedb and run showcase.sql using the source command as done earlier

---

Download Hibernate ORM API

https://sourceforge.net/projects/hibernate/files/hibernate-orm/5.4.1.Final/hibernate-release-5.4.1.Final.zip/download

unzip it and on c:\ create a folder named as hibernate-5.4.1, it should contain the folders that were unziped (documentation,lib,project etc in it)

In jpa folder create a folder named as libs
copy all jar files from c:\hibernate-5.4.1\lib\required folder to c:\showcase\jpa\libs folder
copy mysql.jar from c:\mysql to c:\showcase\jpa\libs folder

in jpa folder create the following structure src\main\java

in java folder create necessary folders for packages and create the following java files

```
package com.thinking.machines.showcase.dl.exceptions;
public class DAOException extends Exception
{
public DAOException(String message)
{
super(message);
}
}
```

For building
in jpa folder create the following build.gradle file

build.gradle

```
apply plugin: 'java'
jar {
archiveName 'showcasedl.jar'
}
dependencies {
compile fileTree(dir: 'libs' , include: '*.jar')
}
```

To compile your project, type gradle build while staying in jpa folder

If everything is correct, then the code will be compiled and in jpa a folder will be created by the name of build which will further contain a folder named as libs which should contain showcasedl.jar file

Thats it, now whenever you want to compile, type gradle build to build your project jar file

---

Create the following classes and build project using gradle build

```
package com.thinking.machines.showcase.dl.pojo;
import javax.persistence.*;
@Entity
@Table(name="category")
public class Category implements java.io.Serializable
{
@Id
@GeneratedValue(strategy=GenerationType.IDENTITY)
@Column(name="code")
private Integer code;
```

```
@Column(name="title",unique=true,nullable=false)
private String title;
public Category()
{
this.code=0;
this.title=null;
}
public void setCode(Integer code)
{
this.code=code;
}
public Integer getCode()
{
return this.code;
}
public void setTitle(String title)
{
this.title=title;
}
public String getTitle()
{
return this.title;
}
}
```

```
package com.thinking.machines.showcase.dl.pojo;
import javax.persistence.*;
@Entity
@Table(name="project")
public class Project implements java.io.Serializable
{
@Id
@GeneratedValue(strategy=GenerationType.IDENTITY)
@Column(name="code")
private Integer code;
@Column(name="title",unique=true,nullable=false)
private String title;
@ManyToOne
@JoinColumn(name="category_code")
private Category category;
@Column(name="domain",nullable=false)
private String domain;
@Column(name="synopsis",nullable=false)
private String synopsis;
@Column(name="technologies",nullable=false)
private String technologies;
public Project()
```

```
{
this.code=0;
this.title=null;
this.category=null;
this.domain=null;
this.synopsis=null;
this.technologies=null;
}
public void setCode(Integer code)
{
this.code=code;
}
public Integer getCode()
{
return this.code;
}
public void setTitle(String title)
{
this.title=title;
}
public String getTitle()
{
return this.title;
}
public void setCategory(Category category)
{
this.category=category;
}
public Category getCategory()
{
return this.category;
}
public void setDomain(String domain)
{
this.domain=domain;
}
public String getDomain()
{
return this.domain;
}
public void setSynopsis(String synopsis)
{
this.synopsis=synopsis;
}
public String getSynopsis()
{
return this.synopsis;
```

```
}
public void setTechnologies(String technologies)
{
this.technologies=technologies;
}
public String getTechnologies()
{
return this.technologies;
}
}
```

```
package com.thinking.machines.showcase.dl.pojo;
import javax.persistence.*;
@Entity
@Table(name="screen_shot")
public class ScreenShot implements java.io.Serializable
{
@Id
@GeneratedValue(strategy=GenerationType.IDENTITY)
@Column(name="code")
private Integer code;
@ManyToOne
@JoinColumn(name="project_code")
private Project project;
@Column(name="serial_number",nullable=false)
private Integer serialNumber;
@Column(name="title",nullable=false)
private String title;
@Column(name="file",nullable=false,unique=true)
private String file;
@Column(name="note",nullable=false)
private String note;
public ScreenShot()
{
this.code=0;
this.project=null;
this.serialNumber=0;
this.title=null;
this.file=null;
this.note=null;
}
public void setCode(Integer code)
{
this.code=code;
}
public Integer getCode()
{
```

```
return this.code;
}
public void setProject(Project project)
{
this.project=project;
}
public Project getProject()
{
return this.project;
}
public void setSerialNumber(Integer serialNumber)
{
this.serialNumber=serialNumber;
}
public Integer getSerialNumber()
{
return this.serialNumber;
}
public void setTitle(String title)
{
this.title=title;
}
public String getTitle()
{
return this.title;
}
public void setFile(String file)
{
this.file=file;
}
public String getFile()
{
return this.file;
}
public void setNote(String note)
{
this.note=note;
}
public String getNote()
{
return this.note;
}
}
```

```
package com.thinking.machines.showcase.dl.pojo;
import javax.persistence.*;
@Entity
```

```
@Table(name="video")
public class Video implements java.io.Serializable
{
@Id
@GeneratedValue(strategy=GenerationType.IDENTITY)
@Column(name="code")
private Integer code;
@ManyToOne
@JoinColumn(name="project_code")
private Project project;
@Column(name="serial_number",nullable=false)
private Integer serialNumber;
@Column(name="title",nullable=false)
private String title;
@Column(name="file",nullable=false,unique=true)
private String file;
@Column(name="note",nullable=false)
private String note;
@Column(name="duration",nullable=false)
private Integer duration;
public Video()
{
this.code=0;
this.project=null;
this.serialNumber=0;
this.title=null;
this.file=null;
this.note=null;
this.duration=0;
}
public void setCode(Integer code)
{
this.code=code;
}
public Integer getCode()
{
return this.code;
}
public void setProject(Project project)
{
this.project=project;
}
public Project getProject()
{
return this.project;
}
public void setSerialNumber(Integer serialNumber)
```

```
{
this.serialNumber=serialNumber;
}
public Integer getSerialNumber()
{
return this.serialNumber;
}
public void setTitle(String title)
{
this.title=title;
}
public String getTitle()
{
return this.title;
}
public void setFile(String file)
{
this.file=file;
}
public String getFile()
{
return this.file;
}
public void setNote(String note)
{
this.note=note;
}
public String getNote()
{
return this.note;
}
public void setDuration(Integer duration)
{
this.duration=duration;
}
public Integer getDuration()
{
return this.duration;
}
}
```

```
package com.thinking.machines.showcase.dl.util;
import java.util.*;
import org.hibernate.*;
import org.hibernate.boot.registry.*;
import org.hibernate.cfg.*;
import org.hibernate.service.*;
```

```
import com.thinking.machines.showcase.dl.pojo.*;
public class HibernateUtil
{
private static SessionFactory sessionFactory;
public static SessionFactory getSessionFactory()
{
if(sessionFactory==null)
{
try
{
Configuration configuration;
configuration=new Configuration();
Properties settings = new Properties();
settings.put(Environment.DRIVER, "com.mysql.jdbc.Driver");
settings.put(Environment.URL, "jdbc:mysql://localhost:3306/showcasedb?useSSL=false");
settings.put(Environment.USER, "showcase");
settings.put(Environment.PASS, "thinkingmachines");
settings.put(Environment.DIALECT, "org.hibernate.dialect.MySQL5Dialect");
settings.put(Environment.SHOW_SQL, "true");
settings.put(Environment.CURRENT_SESSION_CONTEXT_CLASS, "thread");
configuration.setProperties(settings);
configuration.addAnnotatedClass(Category.class);
configuration.addAnnotatedClass(Project.class);
configuration.addAnnotatedClass(ScreenShot.class);
configuration.addAnnotatedClass(Video.class);
ServiceRegistry serviceRegistry=new
StandardServiceRegistryBuilder().applySettings(configuration.getProperties()).build();
sessionFactory=configuration.buildSessionFactory(serviceRegistry);
}catch(Exception e)
{
e.printStackTrace();
}
}
return sessionFactory;
}
}
```

```
package com.thinking.machines.showcase.dl;
import org.hibernate.*;
import java.util.*;
import com.thinking.machines.showcase.dl.pojo.*;
import com.thinking.machines.showcase.dl.util.*;
import com.thinking.machines.showcase.dl.exceptions.*;
public class CategoryDAO
{
public void add(Category category) throws DAOException
{
```

```
Transaction transaction=null;
try(Session session=HibernateUtil.getSessionFactory().openSession())
{
transaction=session.beginTransaction();
session.save(category);
transaction.commit();
}catch(Exception exception)
{
if(transaction!=null)
{
transaction.rollback();
}
exception.printStackTrace();
throw new DAOException(exception.getMessage());
}
}
public void update(Category category) throws DAOException
{
Transaction transaction=null;
try(Session session=HibernateUtil.getSessionFactory().openSession())
{
transaction=session.beginTransaction();
session.update(category);
transaction.commit();
}catch(Exception exception)
{
if(transaction!=null)
{
transaction.rollback();
}
exception.printStackTrace();
throw new DAOException(exception.getMessage());
}
}
public void delete(Integer code) throws DAOException
{
Category category=null;
Transaction transaction=null;
try(Session session=HibernateUtil.getSessionFactory().openSession())
{
transaction=session.beginTransaction();
category=session.get(Category.class,code);
if(category!=null) session.delete(category);
transaction.commit();
}catch(Exception exception)
{
if(transaction!=null)
```

```
{
transaction.rollback();
}
exception.printStackTrace();
throw new DAOException(exception.getMessage());
}
if(category==null) throw new DAOException("Invalid code : "+code);
}
public Category getByCode(Integer code) throws DAOException
{
Category category=null;
Transaction transaction=null;
try(Session session=HibernateUtil.getSessionFactory().openSession())
{
category=session.get(Category.class,code);
}catch(Exception exception)
{
exception.printStackTrace();
throw new DAOException(exception.getMessage());
}
if(category==null) throw new DAOException("Invalid code : "+code);
return category;
}

public List<Category> getAll()
{
try(Session session=HibernateUtil.getSessionFactory().openSession())
{
return session.createQuery("from Category c order by c.title",Category.class).list();
}
}
}
```

```
package com.thinking.machines.showcase.dl;
import org.hibernate.*;
import java.util.*;
import com.thinking.machines.showcase.dl.pojo.*;
import com.thinking.machines.showcase.dl.util.*;
import com.thinking.machines.showcase.dl.exceptions.*;
public class ProjectDAO
{
public void add(Project project) throws DAOException
{
Transaction transaction=null;
try(Session session=HibernateUtil.getSessionFactory().openSession())
{
transaction=session.beginTransaction();
```

```
session.save(project);
transaction.commit();
}catch(Exception exception)
{
if(transaction!=null)
{
transaction.rollback();
}
exception.printStackTrace();
throw new DAOException(exception.getMessage());
}
}
public void update(Project project) throws DAOException
{
Transaction transaction=null;
try(Session session=HibernateUtil.getSessionFactory().openSession())
{
transaction=session.beginTransaction();
session.update(project);
transaction.commit();
}catch(Exception exception)
{
if(transaction!=null)
{
transaction.rollback();
}
exception.printStackTrace();
throw new DAOException(exception.getMessage());
}
}
public void delete(Integer code) throws DAOException
{
Project project=null;
Transaction transaction=null;
try(Session session=HibernateUtil.getSessionFactory().openSession())
{
transaction=session.beginTransaction();
project=session.get(Project.class,code);
if(project!=null) session.delete(project);
transaction.commit();
}catch(Exception exception)
{
if(transaction!=null)
{
transaction.rollback();
}
exception.printStackTrace();
```

```
throw new DAOException(exception.getMessage());
}
if(project==null) throw new DAOException("Invalid code : "+code);
}
public Project getByCode(Integer code) throws DAOException
{
Project project=null;
Transaction transaction=null;
try(Session session=HibernateUtil.getSessionFactory().openSession())
{
project=session.get(Project.class,code);
}catch(Exception exception)
{
exception.printStackTrace();
throw new DAOException(exception.getMessage());
}
if(project==null) throw new DAOException("Invalid code : "+code);
return project;
}
public List<Project> getAll()
{
try(Session session=HibernateUtil.getSessionFactory().openSession())
{
return session.createQuery("from Project p order by p.title",Project.class).list();
}
}
}
```

The Test Cases

In jpa folder create a folder named as testcases
In it create the testcases classes

Note : The testcases are not supposed to be compiled using gradle build, they are supposed to be compiled using the following

javac -classpath ..\libs*;..\build\libs*;. *.java

for execution also use the same classpath

```
import com.thinking.machines.showcase.dl.*;
import com.thinking.machines.showcase.dl.pojo.*;
import com.thinking.machines.showcase.dl.exceptions.*;
public class CategoryAddTestCase
{
public static void main(String gg[])
{
```

```
String title=gg[0];
try
{
Category category=new Category();
category.setTitle(title);
CategoryDAO categoryDAO=new CategoryDAO();
categoryDAO.add(category);
System.out.println("Category added with title : "+category.getCode());
}catch(DAOException daoException)
{
System.out.println(daoException.getMessage());
}
}
}
```

```
import com.thinking.machines.showcase.dl.pojo.*;
import com.thinking.machines.showcase.dl.exceptions.*;
import com.thinking.machines.showcase.dl.*;
import java.util.*;
public class CategoryListTestCase
{
public static void main(String gg[])
{
List<Category> categories=new CategoryDAO().getAll();
categories.forEach((category)->{
System.out.println(category.getCode()+","+category.getTitle());
});
}
}
```

```
import com.thinking.machines.showcase.dl.*;
import com.thinking.machines.showcase.dl.pojo.*;
import com.thinking.machines.showcase.dl.exceptions.*;
public class CategoryUpdateTestCase
{
public static void main(String gg[])
{
int code=Integer.parseInt(gg[0]);
String title=gg[1];
try
{
Category category=new Category();
category.setCode(code);
category.setTitle(title);
CategoryDAO categoryDAO=new CategoryDAO();
categoryDAO.update(category);
System.out.println("Category updated");
}catch(DAOException daoException)
```

```
{
System.out.println(daoException.getMessage());
}
}
}
```

```
import com.thinking.machines.showcase.dl.*;
import com.thinking.machines.showcase.dl.pojo.*;
import com.thinking.machines.showcase.dl.exceptions.*;
public class CategoryDeleteTestCase
{
public static void main(String gg[])
{
int code=Integer.parseInt(gg[0]);
try
{
CategoryDAO categoryDAO=new CategoryDAO();
categoryDAO.delete(code);
System.out.println("Category deleted");
}catch(DAOException daoException)
{
System.out.println(daoException.getMessage());
}
}
}
```

```
import com.thinking.machines.showcase.dl.*;
import com.thinking.machines.showcase.dl.pojo.*;
import com.thinking.machines.showcase.dl.exceptions.*;
public class CategoryGetByCodeTestCase
{
public static void main(String gg[])
{
int code=Integer.parseInt(gg[0]);
try
{
CategoryDAO categoryDAO=new CategoryDAO();
Category category=categoryDAO.getByCode(code);
System.out.println(category.getCode()+","+category.getTitle());
}catch(DAOException daoException)
{
System.out.println(daoException.getMessage());
}
}
}
```

```
import com.thinking.machines.showcase.dl.*;
import com.thinking.machines.showcase.dl.pojo.*;
import com.thinking.machines.showcase.dl.exceptions.*;
```

```
public class ProjectAddTestCase
{
public static void main(String gg[])
{
Integer categoryCode=Integer.parseInt(gg[0]);
String title=gg[1];
String domain=gg[2];
String synopsis=gg[3];
String technologies=gg[4];
try
{
CategoryDAO categoryDAO=new CategoryDAO();
Category category=categoryDAO.getByCode(categoryCode);
Project project=new Project();
project.setCategory(category);
project.setTitle(title);
project.setDomain(domain);
project.setSynopsis(synopsis);
project.setTechnologies(technologies);
ProjectDAO projectDAO=new ProjectDAO();
projectDAO.add(project);
System.out.println("Project added with code as : "+project.getCode());
}catch(DAOException daoException)
{
System.out.println(daoException.getMessage());
}
}
}
```

```
import com.thinking.machines.showcase.dl.pojo.*;
import com.thinking.machines.showcase.dl.exceptions.*;
import com.thinking.machines.showcase.dl.*;
import java.util.*;
public class ProjectListTestCase
{
public static void main(String gg[])
{
List<Project> categories=new ProjectDAO().getAll();
categories.forEach((project)->{
System.out.println(project.getCode()+","+project.getTitle()+","+project.getCategory().getCode()
+","+project.getCategory().getTitle()+","+project.getDomain()+","+project.getSynopsis()
+","+project.getTechnologies());
});
}
}
```

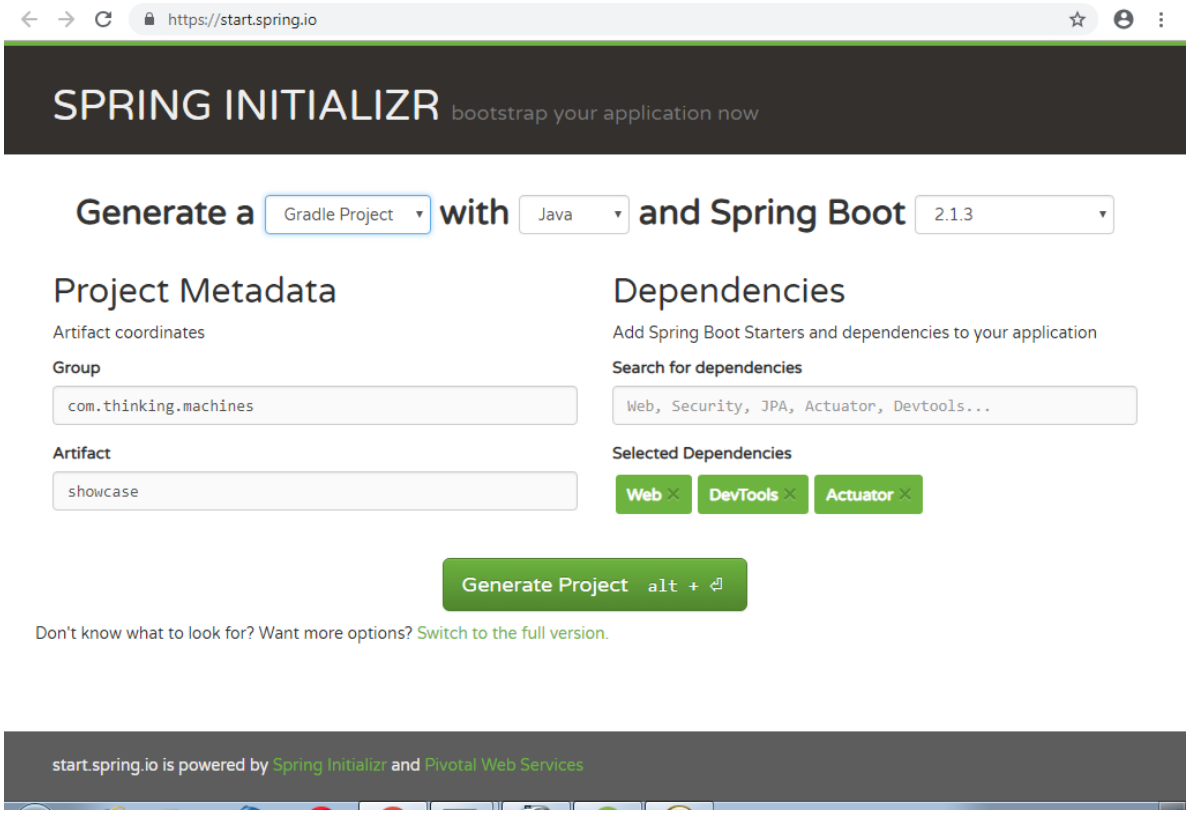
Assignment : Create remaining testcases for ProjectDAO, then create VideoDAO and ScreenShotDAO and its testcases.

Spring Boot 2.0

In parallel to jpa folder create a folder named as springboot,

Now visit https://start.spring.io/

Feed Data as follows, Note (Select Gradle as our build tool)



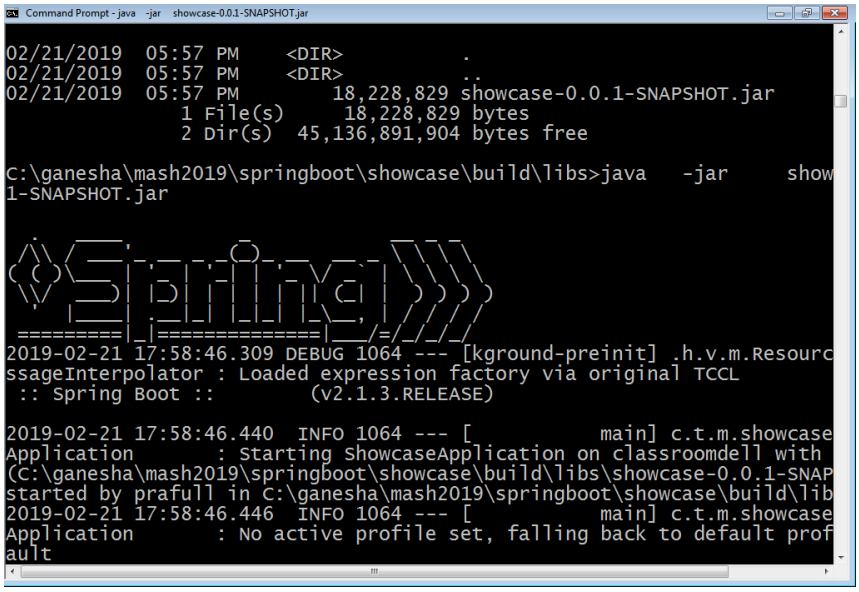
Select (Web,Actuator & DevTools as dependecies)
Click Generate Project button and a zip file show get downloaded, Unzip it and copy the folder (showcase, the one that contains src and gradle folder to our springboot folder)

Open command prompt, move to the showcase folder (the one that contains src folder) and type gradle build to build the project (It may take some time)

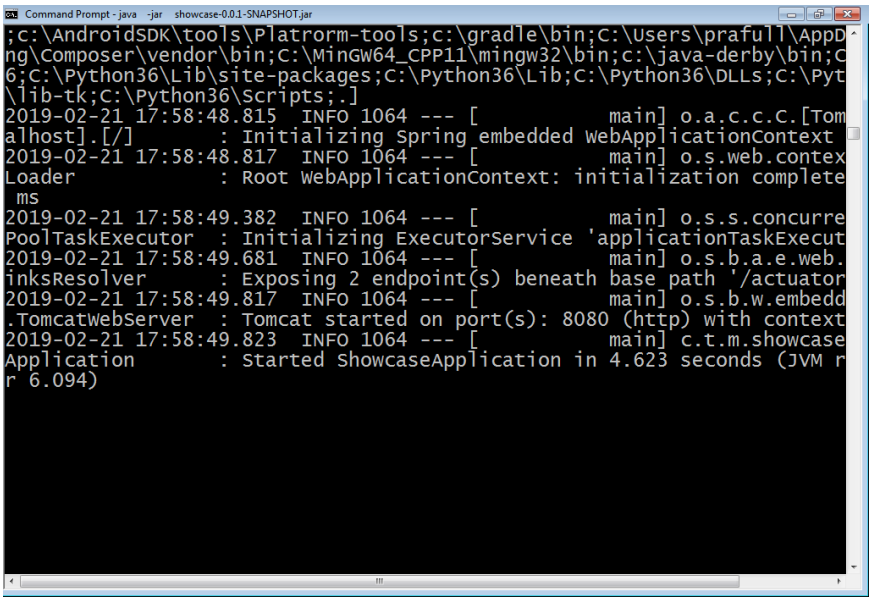
when the build is done, move to the build\libs folder and over there you should see a file named as showcase-0.0.1-SNAPSHOT.jar

While staying in the lib folder type
java -jar showcase-0.0.1-SNAPSHOT.jar

Initially the following should appear

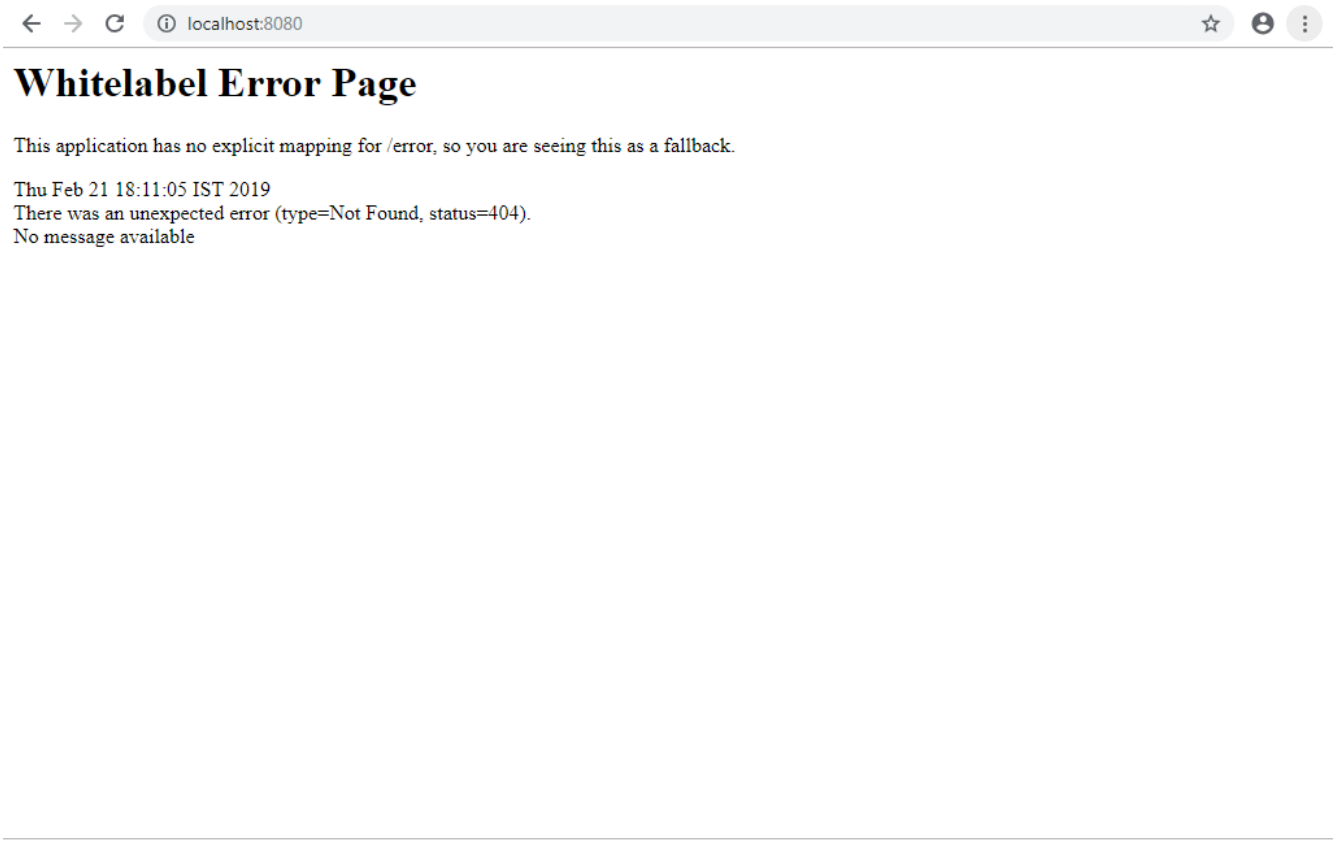


and eventually the following should appear



Embedded tomcat is up and listening on port 8080

Start a browser instance and type http://localhost:8080 and the following should appear



Now press ctrl c to end the tomcat application

Now let us create our first service

in src\main\java\com\thinking\machines\showcase folder create a folder named as services
and in services create a folder named as pojo
create the following classes in respective folders (accroding to package name)

```
package com.thinking.machines.showcase.services.pojo;
public class Category implements java.io.Serializable
{
private Integer code;
private String title;
public Category()
{
this.code=0;
this.title=null;
}
public void setCode(Integer code)
```

```
{
this.code=code;
}
public Integer getCode()
{
return this.code;
}
public void setTitle(String title)
{
this.title=title;
}
public String getTitle()
{
return this.title;
}
}
```

```
package com.thinking.machines.showcase.services;
import org.springframework.web.bind.annotation.*;
import com.thinking.machines.showcase.services.pojo.*;
import java.util.*;
@RestController
public class CategoryService
{
@GetMapping("/showcase/getCategories")
public List<Category> getAll()
{
List<Category> categories=new LinkedList<Category>();
Category category=new Category();
category.setCode(1);
category.setTitle("Desktop Application");
categories.add(category);
category=new Category();
category.setCode(2);
category.setTitle("Web Application");
categories.add(category);
category=new Category();
category.setCode(3);
category.setTitle("Mobile Application");
categories.add(category);
return categories;
}
}
```

Move to the folder that contains build.gradle file (the parent folder of src) and type gradle build
Now run the application using
java -jar showcase-0.0.1-SNAPSHOT.jar
Start browser and type url (http://localhost:8080/showcase/getCategories) and you will see the magic

The Project Part

Remove the pojo folder from services folder
create classes as discussed in the classroom session

```
package com.thinking.machines.showcase.model;
public class Category implements java.io.Serializable,Comparable<Category>
{
private Integer code;
private String title;
public Category()
{
this.code=0;
this.title=null;
}
public void setCode(Integer code)
{
this.code=code;
}
public Integer getCode()
{
return this.code;
}
public void setTitle(String title)
{
this.title=title;
}
public String getTitle()
{
return this.title;
}
public boolean equals(Object object)
{
if(!(object instanceof Category)) return false;
return this.code==((Category)object).code;
}
public int compareTo(Category category)
{
return this.code-category.code;
}
public int hashCode()
{
return this.code;
}
}
```

```
package com.thinking.machines.showcase.model;
import java.util.*;
```

```
public class ShowcaseModel implements java.io.Serializable
{
/*
We are assuming that there won't be many categories, hence a LinkedList
*/
private List<Category> categories=new LinkedList<Category>();
public ShowcaseModel()
{
}
public void setCategories(List<Category> categories)
{
this.categories=categories;
}
public void addCategory(Category category)
{
/*
Note : We are not throwing exception
Some scenario might demand generation of ModelException in case category exists.
*/
if(hasCategory(category)) return;
this.categories.add(category);
}
public boolean hasCategory(Category category)
{
for(Category c:categories)
{
if(c.getTitle().trim().equalsIgnoreCase(category.getTitle().trim())) return true;
}
return false;
}
public Category getCategoryByCode(int code)
{
/*
In some scenarios, you may have to create a clone and return its address
This has been discussed in the classroom session
*/
for(Category c:categories)
{
if(c.getCode()==code) return c;
}
return null;
}

public Category getCategoryByTitle(String title)
{
/*
In some scenarios, you may have to create a clone and return its address
```

```
This has been discussed in the classroom session
*/
for(Category c:categories)
{
if(c.getTitle().trim().equalsIgnoreCase(title.trim())) return c;
}
return null;
}
public void updateCategory(Category category)
{
for(Category c:categories)
{
if(c.equals(category))
{
c.setTitle(category.getTitle());
}
}
}
public void removeCategory(int code)
{
int i=0;
for(Category c:categories)
{
if(c.getCode()==code)
{
categories.remove(i);
}
i++;
}
}
public List<Category> getCategories()
{
/*
In some scenarios, you may have to return a cloned list
This has been discussed in the classroom session
*/
return this.categories;
}
}
```

```
package com.thinking.machines.showcase.services.pojo;
public class ServiceResponse implements java.io.Serializable
{
public Boolean success=true;
public Boolean isException=false;
public Boolean isError=false;
public Boolean hasResult=false;
```

```
public String exception="";
public String error="";
public Object result=null;
}
```

```
package com.thinking.machines.showcase.services;
import com.thinking.machines.showcase.model.*;
import com.thinking.machines.showcase.services.pojo.*;
import java.util.*;
import com.thinking.machines.showcase.dl.exceptions.*;
import com.thinking.machines.showcase.dl.*;
import org.springframework.web.bind.annotation.*;
import org.springframework.beans.factory.annotation.*;
@RestController
public class CategoryService
{
@Autowired
private ShowcaseModel showcaseModel;
@GetMapping("/showcase/getCategories")
public List<Category> getAll()
{
return showcaseModel.getCategories();
}
@PostMapping("/showcase/addCategory")
public  ServiceResponse add(@RequestBody Category category)
{
ServiceResponse serviceResponse=new ServiceResponse();
if(category==null || category.getTitle()==null)
{
serviceResponse.success=false;
serviceResponse.isException=true;
serviceResponse.exception="Category required.";
return serviceResponse;
}
if(showcaseModel.hasCategory(category))
{
serviceResponse.success=false;
serviceResponse.isException=true;
serviceResponse.exception="Category exists.";
return serviceResponse;
}
try
{
com.thinking.machines.showcase.dl.pojo.Category dlCategory;
dlCategory=new com.thinking.machines.showcase.dl.pojo.Category();
dlCategory.setTitle(category.getTitle());
CategoryDAO categoryDAO=new CategoryDAO();
```

```
categoryDAO.add(dlCategory);
category.setCode(dlCategory.getCode());
showcaseModel.addCategory(category);
serviceResponse.hasResult=true;
serviceResponse.result=category;
return serviceResponse;
}catch(DAOException daoException)
{
serviceResponse.success=false;
serviceResponse.isException=true;
serviceResponse.exception=daoException.getMessage();
return serviceResponse;
}catch(Throwable throwable)
{
serviceResponse.success=false;
serviceResponse.isError=true;
serviceResponse.error="Cannot perform operation.";
// some code to add error to log
return serviceResponse;
}
}
@PostMapping("/showcase/updateCategory")
public  ServiceResponse update(@RequestBody Category category)
{
ServiceResponse serviceResponse=new ServiceResponse();
if(category==null || category.getTitle()==null)
{
serviceResponse.success=false;
serviceResponse.isException=true;
serviceResponse.exception="Category required.";
return serviceResponse;
}
if(category.getCode()<=0)
{
serviceResponse.success=false;
serviceResponse.isException=true;
serviceResponse.exception="Category code is invalid.";
return serviceResponse;
}
Category c=showcaseModel.getCategoryByTitle(category.getTitle());
if(c!=null && c.getCode()!=category.getCode())
{
serviceResponse.success=false;
serviceResponse.isException=true;
serviceResponse.exception="Category exists.";
return serviceResponse;
}
```

```
try
{
com.thinking.machines.showcase.dl.pojo.Category dlCategory;
dlCategory=new com.thinking.machines.showcase.dl.pojo.Category();
dlCategory.setCode(category.getCode());
dlCategory.setTitle(category.getTitle());
CategoryDAO categoryDAO=new CategoryDAO();
categoryDAO.update(dlCategory);
showcaseModel.updateCategory(category);
serviceResponse.hasResult=true;
serviceResponse.result=category;
return serviceResponse;
}catch(DAOException daoException)
{
serviceResponse.success=false;
serviceResponse.isException=true;
serviceResponse.exception=daoException.getMessage();
return serviceResponse;
}catch(Throwable throwable)
{
serviceResponse.success=false;
serviceResponse.isError=true;
serviceResponse.error="Cannot perform operation.";
// some code to add error to log
return serviceResponse;
}
}
@PostMapping("/showcase/deleteCategory")
public ServiceResponse delete(@RequestParam(name="categoryCode") Integer code)
{
ServiceResponse serviceResponse=new ServiceResponse();
if(code<=0)
{
serviceResponse.success=false;
serviceResponse.isException=true;
serviceResponse.exception="Invalid code";
return serviceResponse;
}
Category category=showcaseModel.getCategoryByCode(code);
if(category==null)
{
serviceResponse.success=false;
serviceResponse.isException=true;
serviceResponse.exception="Invalid code";
return serviceResponse;
}
try
```

```
{
com.thinking.machines.showcase.dl.pojo.Category dlCategory;
CategoryDAO categoryDAO=new CategoryDAO();
categoryDAO.delete(code);
showcaseModel.removeCategory(code);
serviceResponse.hasResult=true;
serviceResponse.result=category;
return serviceResponse;
}catch(DAOException daoException)
{
serviceResponse.success=false;
serviceResponse.isException=true;
serviceResponse.exception=daoException.getMessage();
return serviceResponse;
}catch(Throwable throwable)
{
serviceResponse.success=false;
serviceResponse.isError=true;
serviceResponse.error="Cannot perform operation.";
// some code to add error to log
return serviceResponse;
}
}
@PostMapping("/showcase/getCategoryByCode")
public ServiceResponse getByCode(@RequestParam(name="categoryCode") Integer code)
{
ServiceResponse serviceResponse=new ServiceResponse();
if(code<=0)
{
serviceResponse.success=false;
serviceResponse.isException=true;
serviceResponse.exception="Invalid code";
return serviceResponse;
}
Category category=showcaseModel.getCategoryByCode(code);
if(category==null)
{
serviceResponse.success=false;
serviceResponse.isException=true;
serviceResponse.exception="Invalid code";
return serviceResponse;
}
serviceResponse.result=category;
serviceResponse.hasResult=true;
return serviceResponse;
}
```

```
}
```

```
package com.thinking.machines.showcase;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.*;
import org.springframework.context.annotation.*;
import java.util.*;
import com.thinking.machines.showcase.model.*;
@SpringBootApplication
public class ShowcaseApplication {
public static void main(String[] args) {
SpringApplication.run(ShowcaseApplication.class, args);
}
@Bean
public ShowcaseModel getShowcaseModel()
{
ShowcaseModel showcaseModel=new ShowcaseModel();
List<com.thinking.machines.showcase.dl.pojo.Category> dlCategories=new
com.thinking.machines.showcase.dl.CategoryDAO().getAll();
List<Category> categories=new LinkedList<Category>();
dlCategories.forEach((category)->{
Category c=new Category();
c.setCode(category.getCode());
c.setTitle(category.getTitle());
categories.add(c);
});
showcaseModel.setCategories(categories);
return showcaseModel;
}
}
```

While building some time is wasted because of unit testing, let us remove it for now
copy build.gradle as build.bck

Now remove the following line from build.gradle

testImplementation 'org.springframework.boot:spring-boot-starter-test'

From src folder remove / move the test folder

In showcase folder (the one that contains build.gradle), create a folder named as libs
in it copy all the jar files from jpa\libs and jpa\build\libs

Let us add a default home page (index.html) to our application

index.html

move to src\main\resource folder
in it create a folder named as public
in it create the following index.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Project showcase</title>
<link rel="stylesheet" type="text/css" href="css/showcase.css">
</head>
<body>
<h3 class='homepageHeading'>Project Showcase (This is a sample page for now)</h3>
<img src='images/thinkingmachines.png'>
</body>
</html>
```

Now move to resources\static folder in it create a folder named as css and images
in css folder create showcase.css as follows

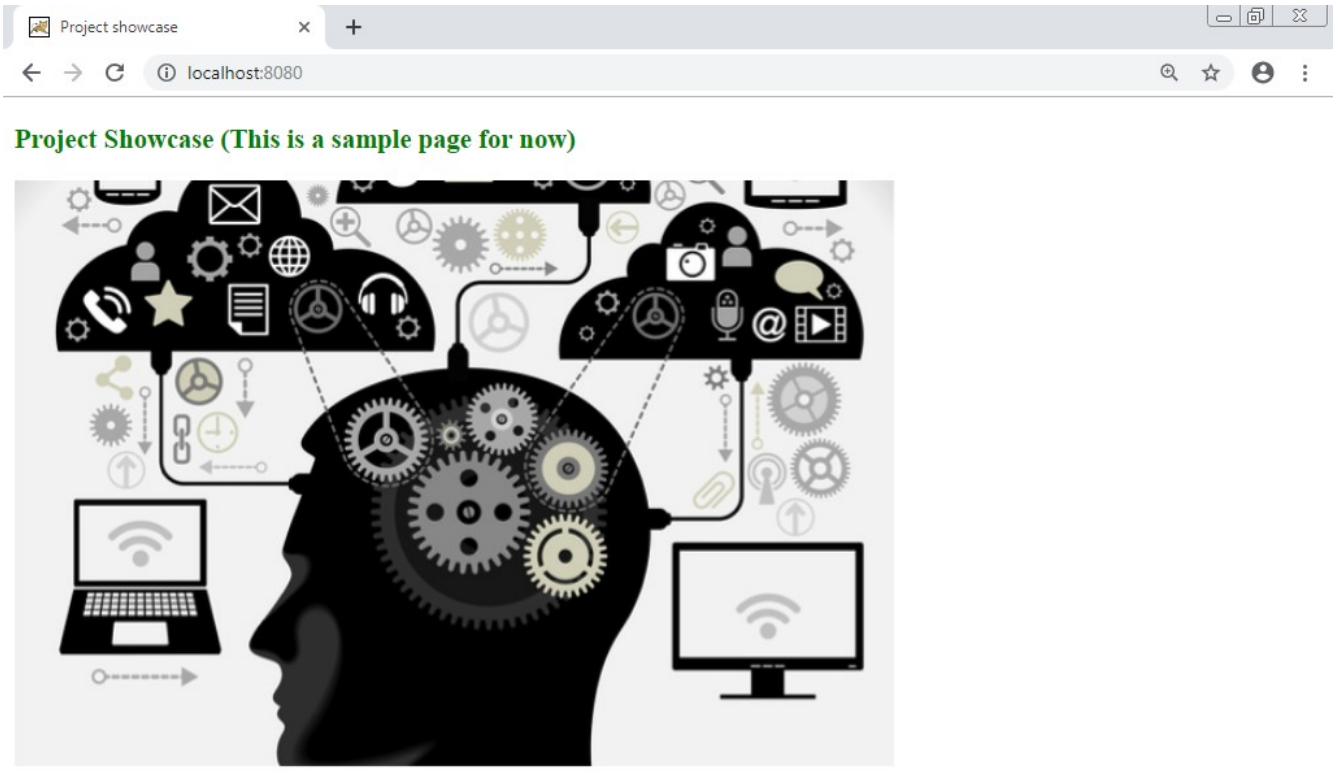
showcase.css

```
.homepageHeading
{
color: green;
}
```

---

the resources\static\images folder should contain a file named as thinkingmachines.png or whatever, just see to it that index.html contains a img tag pointing to this file

Now build and start application and now when you type http://localhost:8080 you should see the following



Assignment : Create a service with path as /showcase/categoryView

in it get the categories list from showcaseModel, and set it in request scope or session scope or somewhere so that it is available in jsp and see to it that the jsp gets processed.
CategoryView.jsp
JSP should make use of JSTL to iterate over the list and generate the necessary html so that the list of categories gets displayed.

Date : 12/3/2019

Now let us create a service that needs to dispatch request to jsp

First of all we need to understand that gradle should create a war (Web Application Archive) file, and not a jar file.

For that edit build.gradle and change the plugin factor from 'java' to 'war' as show below

```
plugins {
id 'org.springframework.boot' version '2.1.3.RELEASE'
id 'war'
}
```

Next where should we keep our jsp files ?

Create a folder named as webapp under src\main
in webapp create a folder named as WEB-INF
then create a folder of your choice in WEB-INF folder
in that folder keep your jsp files

I have create a folder named as views under WEB-INF and have placed the following jsp file in it

CategoryView.jsp

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Project showcase</title>
<link rel="stylesheet" type="text/css" href="css/showcase.css">
</head>
<body>
<h3>Categories</h3>
<table border='1'>
<c:forEach var="category" items="${categories}" varStatus='i'>
<tr>
<td>${i.count}</td>
<td>${category.title}</td>
</tr>
</c:forEach>
</body>
</html>
```

Now we need to edit application.properties file which resides in src\main\resource folder and we will have to add the following 2 settings as shown below

application.properties

spring.mvc.view.prefix: /WEB-INF/views/
spring.mvc.view.suffix: .jsp

Now whenever request arrives or is dispatched, for processing a jsp file, the server won't process it, instead the server will send its contents to the requesting client. To enable the server to understand that jsp's are supposed to be processed on server side, we will add the dependency of jsp parser in build.gradle as follows

add the following line to the dependecies section

implementation 'org.apache.tomcat.embed:tomcat-embed-jasper:9.0.16'

And if we are going to make use of JSTL then you will have to add the following dependency
implementation 'javax.servlet:jstl:1.2'

Now let us create our service method.
I would like to create a separate class for the services that would be dispatching request to jsp's
Create the following java file in appropriate folder (as per package name)

```
package com.thinking.machines.showcase.services;
import com.thinking.machines.showcase.model.*;
import com.thinking.machines.showcase.services.pojo.*;
import java.util.*;
import org.springframework.web.servlet.*;
import org.springframework.stereotype.*;
import org.springframework.web.bind.annotation.*;
import org.springframework.beans.factory.annotation.*;
@Controller
public class CategoryViewService
{
@Autowired
private ShowcaseModel showcaseModel;
@GetMapping("/showcase/categoriesView")
public ModelAndView getCategories()
{
System.out.println("getCategories for CategoryView.jsp got executed");
ModelAndView modelAndView=new ModelAndView();
modelAndView.addObject("categories",showcaseModel.getCategories());
modelAndView.setViewName("CategoryView");
return modelAndView;
}
}
```

---

delete the jar file from build\libs folder
build and this time a war file will be created, run the application using java -jar name_of_the_war and access the service using the following URL http://localhost:8080/showcase/categoriesView

Let us now start building the client side using Angular.
See to it that you have node 10.x installed, type node -v to check your version, I have 10.15.2 along with npm manager installed.

Step 1 → Install Angular/CLI (The super cool command line interface tools). For that type

npm install -g @angular/cli

It may take some time as many packages will be downloaded and installed

Now move to the folder that contains jpa and springboot folders (Our earlier work).
In that folder create a folder named as angular. Move into the angular folder and type

ng new showcase-app

Note : be particular about the lower cased application name (showcase-app)

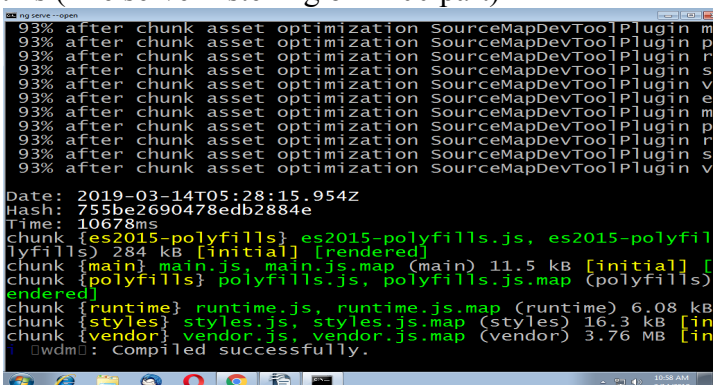
You will be asked if you would like to add angular routing, say yes to it and in case of selecting stylesheet format, just go for css and hit entered. Because of this a skeleton directory structure requried for angular application will be created by the name of showcase-app which would contain minimal necessary packages requried.

Now move into the showcase-app folder and type

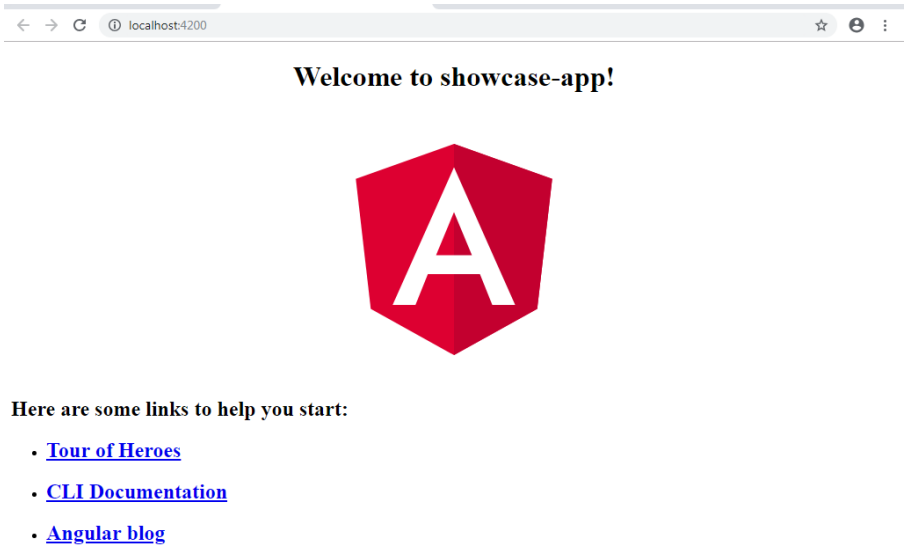
ng serve --open

Three things will happen because of the serve command.
A server will start and will start listening on port 4200, this server will be responsible for serving the angular application to the requesting client.
A directory observer will come into the picture. Its role will be observe necessary changes and will rebuild the app on changes.
Because of –open, the browser instance will be launched with initial url as http://localhost:4200

Now you should see this (The server listening on 4200 part)



And the browser



Thats it, everything is working properly. Now let us move ahead and create CRUD operations module for our Category entity.

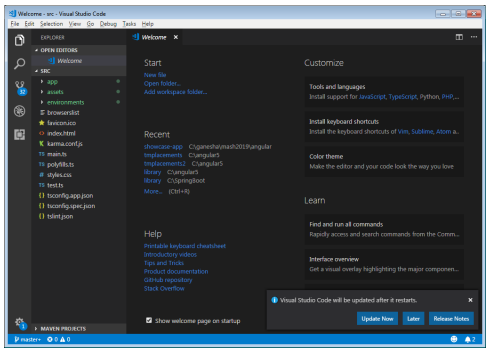
Note : Initially you will be going through incremental learning process, so be mentally ready to dump everything in the end and restructure the final version to implement best practices.

We will be using Visual Studio Code as our IDE. I hope you have gone through my TypeScript document from thinkingmachines.in

Now move to showcase-app\src folder and type (Note : code should be followed by space and a dot, to open the current directory in explorer)

code .

to launch the Visual Studio Code IDE, this is what you should see



Now let us make a small change and see how the browser gets refreshed automatically.

Expand the app node from the explorer tree, double click app.component.ts, the file that contains the component class and change the value of title property in class to My Projects and save and see the browser window (donot press refresh). The view should now be updated. Now let us remove other things from this page.
Note : After editing the file, do not forget to save it for the changes to be effective.
Edit the app.component.html and change it to the following

```
<!--The content below is only a placeholder and can be replaced.-->
<div style="text-align:center">
  <h1>
    {{ title }}@Thinking Machines
  </h1>
</div>

<router-outlet></router-outlet>
```

Now let us create our module for category.
I am assuming that there are some categories exists (as we have feed them using our jpa (testcases)or springboot services testing using restclient.

Now from command prompt, while staying in the showcase-app folder type the following to create necessary files for our category component

```
ng   generate   component  category
```

because of the above generate command html, spec.ts, ts and css files for our category component are created under category folder which resides in app folder. You can see that in your explorer part of Visual Studio Code IDE.
Initially we are not interested in fetching anything from the server. We will just setup our category objects in our component

right click the app node in explorer select new folder and name it as entity
now right click on entity node and select new file and name it as category.ts
write the following code in it.

```
export class Category
{
   code:number;
   title:string;
   constructor()
   {
      this.code=0;
      this.title="";

   }
}
```

Now let us edit our CategoryComponent class, our objective is to create an array of Category type objects and then loading those categories on the view through the component's view template.

Edit category.component.ts (it is under the app → category →)

I will guide you through the necessary changes.

Place the following import statement just below the first import statement at the top

import { Category } from '../entity/category'

Note, while typing necessary intellisense help should appear to enable you to understand that everything is going as desired.

Now add the following properties in the class CategoryComponent

categories:Category[];
title="Categories";

Now add the following code to ngOnInit method of the CategoryComponent class

```
//    this.categories=new Category[3]; // incorrect
    this.categories=[]; // correct or this.categories=new Array();
    let category:Category=null;
    category=new Category();
    category.code=1;
    category.title="Web Application";
    this.categories.push(category);
    category=new Category();
    category.code=2;
    category.title="Desktop Application";
    this.categories.push(category);
    category=new Category();
    category.code=3;
    category.title="Mobile Application";
    this.categories.push(category);
```

Now let us wire up the component's categories property to the view.

But first of all let us place the category component on the main (app) component's view.

For that edit app.component.html (it is under the app node)

add the following (highlighted part) to it

```
<!--The content below is only a placeholder and can be replaced.-->
<div style="text-align:center">
  <h1>
    {{ title }}@Thinking Machines
  </h1>
  <app-category>

  </app-category>
</div>

<router-outlet></router-outlet>
```

Save it and you should see the following in the browser

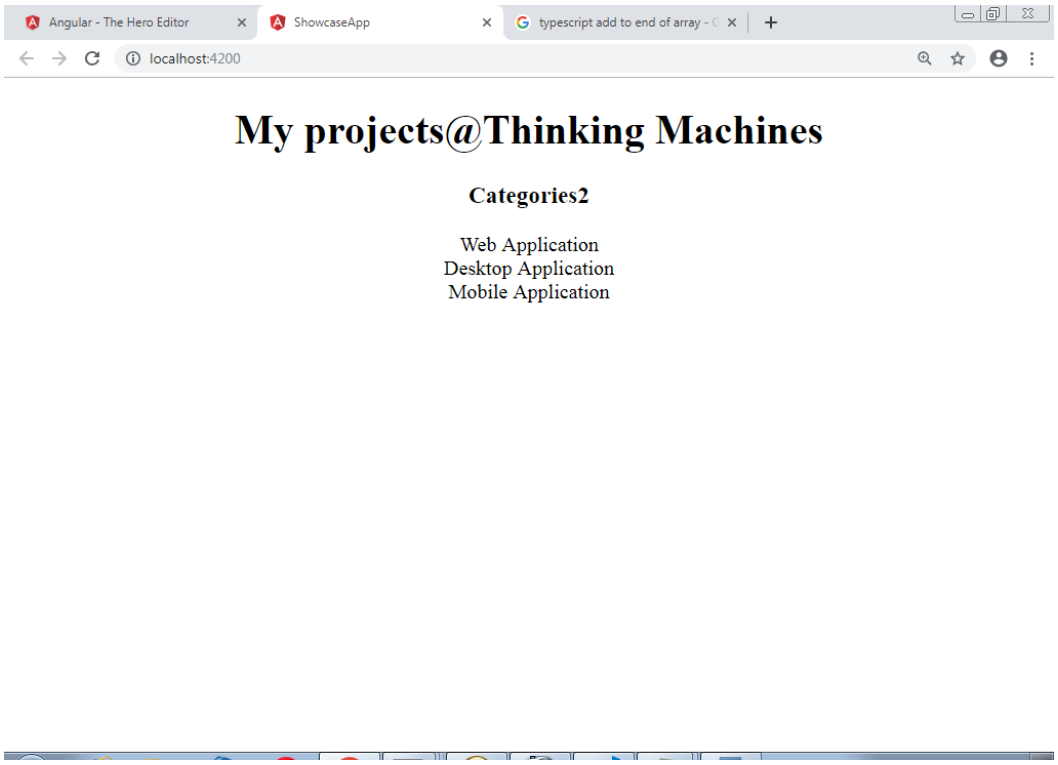
Now from where has this (category works!) taken.
See category.component.html and it has the following

```
<p>
  category works!
</p>
```

Now let us change (category.component.html) this to suit our needs

```
<h3>{{title}}</h3>
<div class='row' *ngFor='let category of categories'>
{{category.title}}<br/>
</div>
```

Thats it, now you should be able to see this on the browser



Now let us create a service
while staying in showcase-app folder type

ng generate service showcase

two files will be created in app foder, we are interested in showcase.service.ts

Its content should be as follows

```
import { Injectable } from '@angular/core';
import { of } from 'rxjs';
import { Observable } from 'rxjs';
import { Category } from '../app/entity/category';

@Injectable()
export class ShowcaseService {
  constructor() { }

  getCategories():Observable<any>
  {

    let categories:Category[];
    categories=[];
    let category:Category;
    category=new Category();
    category.code=1;
    category.title="Web application";
    categories.push(category);
    category=new Category();
    category.code=2;
    category.title="Desktop application";
    categories.push(category);
    category=new Category();
    category.code=3;
    category.title="Mobile application";
    categories.push(category);
    category=new Category();
    category.code=4;
    category.title="OS  application";
    categories.push(category);
    return of(categories);
  }
}
```

Now edit the category.component.ts, and its contents should be as follows (I have highlighted the changes)

```
import { Component, OnInit } from '@angular/core';
import { Category } from '../entity/Category';
import { ShowcaseService } from '../showcase.service';

@Component({
  selector: 'app-category',
  templateUrl: './category.component.html',
  styleUrls: ['./category.component.css'],
  providers: [ShowcaseService]
})
export class CategoryComponent implements OnInit {
categories:Category[];
title="Categories";
  constructor(private showcaseService:ShowcaseService) {
    this.categories=[];
  }

  ngOnInit() {
  this.showcaseService.getCategories().subscribe(data=>{
    this.categories=data;
  },error=>{
      alert(JSON.stringify(error));
  });
  }

}
```

Thats it for now. Now our component is pulling data from service and is unaware of the source of data. Next now we will change the service to fetch the data from our springboot application.

Date : 26/3/2019

Now let us fetch data from springboot application residing on embeded server

move to springboot\showcase\build\libs folder and run the application as we use to do earlier by typing

```
java    -jar    name_of_jar.war
```

Start your REST client application and verify that http://localhost:8080/showcase/getCategories serves the necessary JSON

Now the angular part.

Edit app.module.ts and add the following import statement at top

import { HttpClientModule } from '@angular/common/http'

then add HttpClientModule to imports array in the same file (app.module.ts)
Note : don't forget the , after the exising last entry

Now edit the showcase.service.ts and add the following import statement at top

import { HttpClient } from '@angular/common/http';

Then change the constructor of ShowcaseService to enable dependency injection for HttpClient

```
constructor(private httpClient:HttpClient)
{

}
```

Now in the onNgInit method remove or comment the existing code and write the following single line.
return this.httpClient.get("/showcase/getCategories");

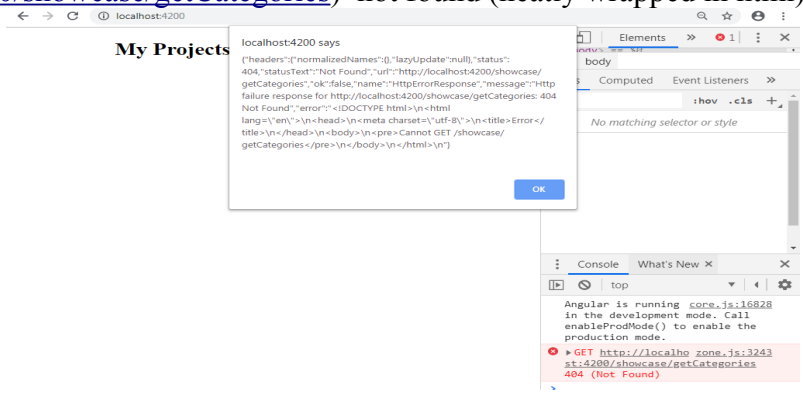
Now start angular server (ng serve)

Now start chrome, first of all open the developers panel

Visit http://localhost:4200 and now you won't get to see the data, instead because of our

alert(JSON.stringify(error));

in the error=> section of category.component.ts, you will get to see the error message (http://localhost:4200/showcase/getCategories) not found (neatly wrapped in html)



This is happening because, the request is being sent to http://localhost:4200 where as we want the

request to be sent to http://localhost:8080, for this, we will have to tell the server running on 4200 to divert all request starting with /showcase to server running on localhost 8080

For this, first of all stop the angular server.

Now in the showcase-app folder create the following file with specified contents

proxy.conf.json

```
{
"/showcase/*" : {
"target": "http://localhost:8080",
"secure": false,
"logLevel": "debug",
"changeOrigin": true
}
}
```

Now start the angular server using --proxy-config option as shown below

```
ng    serve    --proxy-config   proxy.conf.json
```

Now refresh the browser for http://localhost:4200 and you should see the data fetched from the spring boot application.

Assignment : Try to understand : How routing is implemented in Angular
Note : Remember that we were asked if we wanted the routing feature and we had said yes to it.
