Git Repo URL : https://github.com/ashokitschool/SBMS-39.git

=================

Spring Web MVC

=================

-> Spring Web MVC is one module in spring framework.

-> Using Spring Web MVC, we can develop below types of applications

1) Web Application ( C 2 B )

2) Distributed Application ( B 2 B )

-> We need to use 'springboot-starter-web' depenency to work with Spring Web MVC module.

-> 'web-starter' will provide embedded container by default. We no need to setup server manually.

- Apache (Default)

- Jetty

- Netty

-> Spring Web MVC supports multiple presentation technologies

Ex: JSP, Thymeleaf...

-> Spring Web MVC supports Form Binding. form data will be mapped to java object.

Note: When we develop java web app using servlets we need capture form data like below.

String phno = request.getParameter("phno");

Long ph = Long.parseLong(phno);

Note: We no need to write this logic in web mvc. It will take care of capturing form data and convert into corresponding data type and store into java object.

============================

Spring Web MVC Architecture

============================

=> In spring web mvc, below components will be involved...

1) DispatcherServlet : front controller/framework servlet

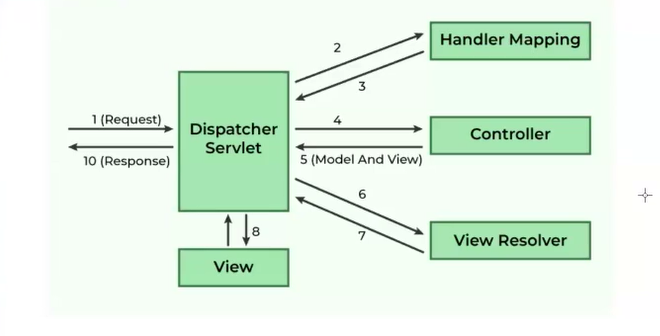
2) HandlerMapper : To identify request handler (controller)

3) Controller : Request Handler (spring bean) - we have to develop

4) ModelAndView : Model represents data & view represents UI page.

5) ViewResolver : To identify where view pages available in app

6) View : To render model data on view page.



1st step Every Request that you are sending to the web mvc application

Will be taken care by Dispatcher servlet

2nd 3rd step Dispatcher servlet will talk to handler mapping to identify

Which request is processed by which controller.

4th 5th step And controller will Return model and view object to dispatcher servlet

6th 7th step it is going to view resolver to identify the location of view pages to the view component

=========================================

Building First Web App using Spring Boot

==========================================

1) Create Boot app with below dependencies

a) web-starter

b) thymeleaf-starter

c) devtools

2) Create Controller class with required methods and map methods to HTTP methods with unique url patterns.

3) Create View Page (HTML + Thymeleaf) (under templates folder)

4) Run the application and test it.

----------------- Controller -------------------------------

@Controller

public class MsgController {

@GetMapping("/greet")

public ModelAndView getMsg2() {

ModelAndView mav = new ModelAndView();

mav.addObject("msg2", "Good Morning...!!");

mav.setViewName("index");

return mav;z

}

@GetMapping("/welcome")

public ModelAndView getMsg1() {

ModelAndView mav = new ModelAndView();

mav.addObject("msg1", "Welcome to Ashok IT");

mav.setViewName("index");

return mav;

}

}

------------------------index.html------------------------------

<html>

<body>

<p th:text="${msg1}"></p>

<p th:text="${msg2}"></p>

</body>

</html>

----------------------------------------------------------------

===========================

Spring Web MVC Assignments

===========================

1) Develop Spring Boot web app to retrieve products data from db table and display in UI page as a table.

=========================

Form Based Applications

=========================

=> In every web app many forms will be available

- login form

- register form

- search form

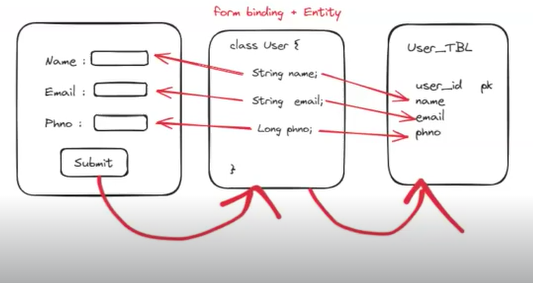
=> We need to capture form data and we need to perform operation on that data...

Note: Web MVC supports form binding.

=============================================================

Develop Boot web app to save and retrieve products data

==============================================================



1) Product.java (form binding + entity)

Integer pid; (PK, Auto\_Increment)

String pname;

Double price;

Integer qty;

2) ProductRepo.java (JpaRepository)

3) ProductService.java

- public boolean saveProduct(Product p);

- public List<Product> getProducts( );

4) ProductController.java

public ModelAndView loadForm( ); - GET

public ModelAndView saveProduct(Product p) - POST

public ModelAndView getAllProducts( ) - GET

5) View Pages

index.html - form to enter data

data.html - table to display data

=================================

Embedded Database in spring boot

=================================

=> Embedded databases are called temporary databases

=> H2 we can use as embedded database

=> When application starts h2 db will start and when application stopped h2 db also gets stopped.

Note: If application re-started then we will loose old data.

=> H2 db is used only for practice purpose.

====================================

How to use H2 DB in spring boot ?

====================================

### Step-1 : Add h2 dependency in pom.xml file

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

### Step-2 : Configure H2 datasource properties in application.properties file

spring.datasource.username=ashokit

spring.datasource.password=abc

spring.datasource.url=jdbc:h2:mem:sbms

spring.jpa.show-sql=true

server.port=8081

### Step-3 : Run the application and access h2-console in browser

URL : http://localhost:port/h2-console

============================================

How to change default container to jetty ?

============================================

=> When we add web starter then tomcat will become default embedded container to run boot application.

=> If we want to change from tomcat to jetty then we need to make below changes in pom.xml

### Step-1 : Exclude tomact from web-starter dependency

<dependency>

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

<artifactId>spring-boot-starter-web</artifactId>

<exclusions>

<exclusion>

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

<artifactId>spring-boot-starter-tomcat</artifactId>

</exclusion>

</exclusions>

</dependency>

### Step-2 : Add jetty starter

<dependency>

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

<artifactId>spring-boot-starter-jetty</artifactId>

</dependency>

### Step-3 : Run the application and observe console.

==============================================

What are the two ways of sent the data from controller to Ui.

===============================================

Index.html

<html>

<p th:text=”{msg}”></p>

</html>

// 1st Method \*\*\*prefable approach

@GetMapping(“/Welcome”)

Public String getGreetMsg(Model model){

Model.addAttribute(“msg”, “Hi , Hello”);

Return “index”; }

// 2nd Method

Public class MsgController {

@GetMapping(“/welcome”)

Public ModelAndView getWelcome(){

String msgText = “Good Morning…..!!”;

ModelAndView mav = new ModelAndView();

Mav.addObject(“msg”,msgText);

Mav,setViewName(“index”);

Return mav;

}

}

==============

Requirement :

==============

Develop Spring Boot web application with below functionalities

Registration Page : Name, Email, Pwd and Phno fields

Login Page : Email & Pwd fields

Dashboard Page : <msg>

Note: When user registered, application should send an email to the user.

Email Subject : Account Created - Ashok IT

Email Body : Congratuations.. you are onboard..

=================================

Email Sending using spring boot

=================================

=> To send emails using spring boot we have add 'mail-starter' dependency in pom.xml

=> We need to configure SMTP properties in application.properties file

Note: In SMTP props, we need to our gmail account credentials for authentication purpose.

Note: We need to generate "app password" for gmail for authentication.

URL To generate app pwd: https://g.co/kgs/f1ic3P9

=> Spring boot provided JavaMailSender to send emails

- SimpleMailMessage (plain text)

- MimeMessage (formats, attachments)

#smtp properties

spring.mail.host=smtp.gmail.com

spring.mail.port=587

spring.mail.username=monukumarrajbanshi1999@gmail.com

spring.mail.password=ivvv qixe oejs bvix

spring.mail.properties.mail.smtp.auth=true

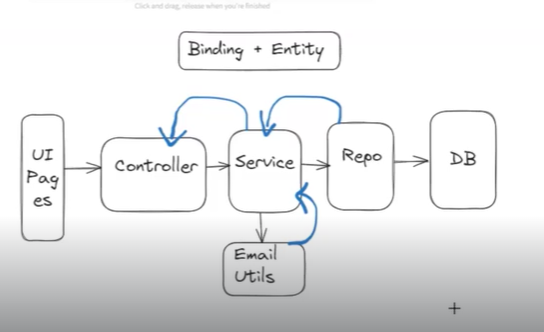
spring.mail.properties.mail.smtp.starttls.enable=true

===============================================

Requriement Solution

================================================

Diagram:-



Code

User.java

package in.ashokit.entity;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

import jakarta.persistence.Table;

*@Entity*

*@Table*(name="user\_dtls")

public class User {

*@Id*

*@GeneratedValue*(strategy = *GenerationType*.***IDENTITY***)

private Integer uid;

private String name;

private String email;

private String pwd;

private Long phno;

public Integer getUid() {

return uid;

}

public void setUid(Integer uid) {

this.uid = uid;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public String getPwd() {

return pwd;

}

public void setPwd(String pwd) {

this.pwd = pwd;

}

public Long getPhno() {

return phno;

}

public void setPhno(Long phno) {

this.phno = phno;

}

*@Override*

public String toString() {

return "User [uid=" + uid + ", name=" + name + ", email=" + email + ", pwd=" + pwd + ", phno=" + phno + "]";

}

}

UserRepo.java

package in.ashokit.repo;

import in.ashokit.entity.\*;

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

public interface UserRepo extends JpaRepository<User, Integer> {

public User findByEmailAndPwd(String email,String pwd);

}

EmailUtils.java

package in.ashokit.utils;

import org.apache.logging.log4j.message.SimpleMessage;

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

import org.springframework.mail.SimpleMailMessage;

import org.springframework.mail.javamail.JavaMailSender;

import org.springframework.stereotype.Component;

*@Component*

public class EmailUtils {

*@Autowired*

private JavaMailSender mailSender;

// **TODO** Logic

public boolean sendEmail(String to,String subject,String body) {

try {

SimpleMailMessage msg = new SimpleMailMessage();

msg.setTo(to);

msg.setSubject(subject);

msg.setText(body);

mailSender.send(msg);

return true;

}

catch (Exception e ){

e.printStackTrace();

}

return false ;

}

}

UserService.java

package in.ashokit.service;

import in.ashokit.entity.User;

public interface UserService {

public boolean saveUser(User user);

public User getUser(String email,String pwd);

}

UserServiceImpl.java

package in.ashokit.service;

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

import org.springframework.stereotype.Service;

import in.ashokit.entity.User;

import in.ashokit.repo.UserRepo;

import in.ashokit.utils.EmailUtils;

*@Service*

public class UserServiceImpl implements UserService {

*@Autowired*

private UserRepo userRepo;

*@Autowired*

private EmailUtils emailUtils;

*@Override*

public boolean saveUser(User user) {

// **TODO** Auto-generated method stub

User savedUser = userRepo.save(user);

if(savedUser.getUid()!=null)

{

String subject="Your Account is Created ";

String body="Congratulations, Welcome to board...";

emailUtils.sendEmail(user.getEmail(), subject, body);

}

return true;

}

*@Override*

public User getUser(String email, String pwd) {

// **TODO** Auto-generated method stub

return userRepo.findByEmailAndPwd(email, pwd);

}

}

UserController.java

package in.ashokit.controller;

import in.ashokit.entity.User;

import in.ashokit.service.\*;

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

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

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

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

*@Controller*

public class UserController {

*@Autowired*

private UserService userService;

// login - page - display

*@GetMapping*("/login")

public String login(Model model) {

model.addAttribute("user", new User());

return "index";

}

// login - button - handle

*@PostMapping*("/login")

public String handleLogin(User user , Model model) {

User userObj = userService.getUser(user.getEmail(), user.getPwd());

if(userObj==null)

{

model.addAttribute("emsg","Invalid Credentials ");

return "index";

}

else {

model.addAttribute("msg",userObj.getName()+", Welcome to Zomato...");

return "dashboard";

}

}

// register - page - display

*@GetMapping*("/register")

public String register(Model model) {

model.addAttribute("user", new User());

return "registerView";

}

// register - button - handle

*@PostMapping*("/register")

public String handleRegister(User user,Model model )

{

boolean status = userService.saveUser(user);

if(status) {

model.addAttribute("smsg","User Register");

}

else

{

model.addAttribute("emsg","Registration Failed ");

}

return "registerView";

}

// logout - method

*@GetMapping*("/logout")

public String logout(Model model) {

model.addAttribute("user",new User());

return "index";

}

}

============================================

Exception Handling in Boot web application

============================================

=> Exception : Unexpected and unwanted situation

=> When exception occurs program will be terminated abnormally

=> We need to handle exceptions for app graceful termination.

=> We have below keywords to handle exceptions in java

1) try

2) catch

3) throw

4) throws

5) finally

=> In spring boot we can handle exceptions in 2 ways

1. Controller/class Based (specific to class)

It is applicable only for particular controller class . it will take care of the exception occur only in the particular controller class.

1. Global Exception Handling (entire application)

But , in the project if we want to handle all the exception in the classes in the application . then we need to go for Global Exception

Handle.

Controller Based Handle

*@Controller*

public class MsgController {

*@GetMapping*("/welcome")

public String getWelcome(Model model) {

int i =10/0;

model.addAttribute("msg","Welcome to AbhiWebsite");

return "index";

}

// @ExceptionHandler(value =ArithmeticException.class)// for only divide by zero exception

*@ExceptionHandler*(value =Exception.class) // for all type of exception

public String handleException(Exception e,Model model) {

model.addAttribute("msg","Some Exception Occur ");

return "exView";

}

}

Global Exception Handle

*@ControllerAdvice*

public class AppExceptionHandler {

// it become global exception handle

*@ExceptionHandler*(value =Exception.class) // for all type of exception

public String handleException(Exception e,Model model) {

model.addAttribute("msg","Some Exception Occur ");

return "exView";

}

}

@ControllerAdvice //

public class AppExceptionHandler {

@ExceptionHandler(value = Exception.class)

public String handleAe(Exception e) {

// logic

return "exView";

}

}

What is user Defined Exception ?

Creating our own exception is called user defined exception

How to create User Defined Exception ?

package in.ashokit.controller;

public class ProductNotFoundException extends Exception{

public ProductNotFoundException() { // 0 param Constructor

// **TODO** Auto-generated constructor stub

}

public ProductNotFoundException(String msg) // param constructor

{

super(msg);

}

}

*@Controller*

public class ProductController {

*@GetMapping*("/product")

public String getProduct() {

try {

// logic

throw new ProductNotFoundException("Invalid Data");

}

catch (Exception e)

{

// loggers

}

return "index";

}

}

*@ExceptionHandler*(value = ProductNotFoundException.class)

public String handleProductEx(ProductNotFoundException pne)

{

//logic

return "exView";

}

For all user define exception

==================================================================

1) What is web mvc ?

2) Advantages with Web MVC

3) Web MVC Architecture

4) What is Embedded Container

5) How to develop boot web app

6) How to send data from controller to UI

7) Web MVC Form with Form Binding

8) Embedded Database (h2)

9) How to change default container

10) Email Sending using Spring Boot

11) Exception Handling in Web MVC

12) Login & Registration app

13) Product Store App (CRUD Ops)

===================================

Query Parameters (key-value)

===================================

=> Query Params are used to send data to server in URL

=> Query Params will represent data in key-value format

=> Query params will start with ?

=> Query Params will be seperated by &

=> Query Params will present at end of the URL

ex: www.youtube.com/watch?v=ljsdf79/

www.ashokit.in/course?name=sbms

www.ashokit.in/course?cname=sbms&tname=ashok

Note: We can read query parameters from URL using @RequestParam annotation in the controller.

================

Path Parameters

================

=> Path Params are used to send data to server in URL

=> Path Parameters will represent value directley

ex : www.youtube.com/c/AshokIT

=> Path Parameters will be seperated by '/'

=> Path Parameters can present anywhere in the URL

Ex : www.ashokit.in/course/{java}/info

=> We can read Path Parameters using @PathVariable annotation.

=========================

What is @ResponseBody ?

=========================

=> It is used to send direct response to client without any view page.

=> This can be used at controller class level and method level

Note: If we use at class level then it is applicable for all methods in that class

@Controller + @ResponseBody = @RestController

So , we know using @Controller we are sending model or view page as a response from method .

And @ResonseBody we are sending the direct data as a response from method

==================================================================

@Controller

public class MsgController {

@GetMapping("/welcome")

@ResponseBody

public String welcomeMethod(@RequestParam("name") String name) {

return name + ", Welcome to Ashok IT";

}

@GetMapping("/greet/{name}")

public String greetMethod(@PathVariable("name") String name, Model model) {

model.addAttribute("msg", name+", Good Morning");

return "index";

}

}

E.g:-

*@Controller*

public class MsgController {

// Query Parameter

*@GetMapping*("/welcome")

*@ResponseBody* // if we pass url like http://localhost:8080/welcome?name=Abhi

public String welcomeMethod(*@RequestParam*("name")String name)

{

return name + ", Welcome to Abhi Website";

}

// Path Parameter

*@GetMapping*("/greet/{name}") // link should look like this http://localhost:8080/greet/Abhi

public String greetMethod(*@PathVariable*("name")String name , Model model)

{

model.addAttribute("msg",name +", Good Morning ");

return "index";

}

}

=================

Form Validations

=================

=> To restrict users to provide valid information in the form

- Client Side Validations

- Server side validation

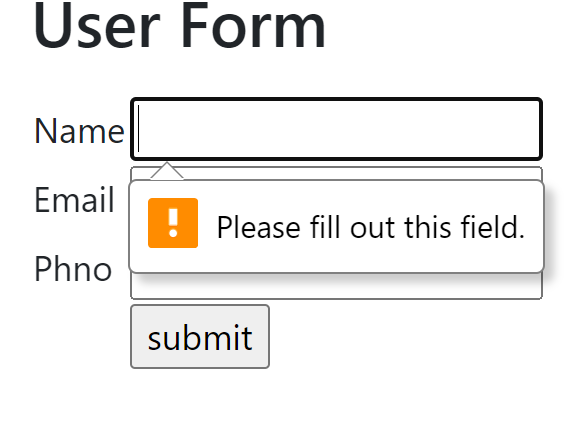
For e.g. in case you creating form and if u not enter any data and u submitted the form request is submitted to server with empty data

Wether the form having the correct data or not we need to do the validation.

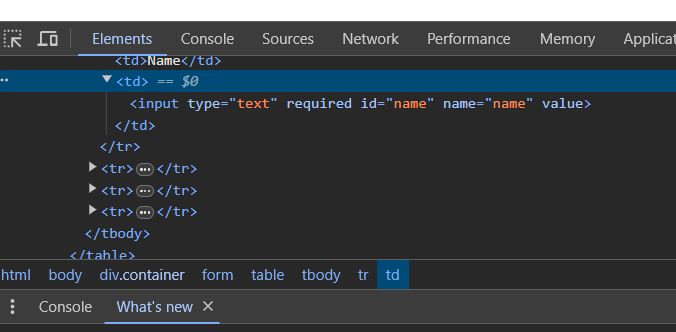
How to do client side validation ?

Simply on index.html

<td><input type="text" th:field="\*{name}" required> </td>



=> Client side validations will execute at browser. People can disable client side validations using inspect option in browser.



So, it is not recommended to using only client side validation we have to use client side as well as server side validation for best practice.

=> Server side validations will execute in our code. These are highly recommended in application.

=> To implement server side validations we will use below starter in pom.xml file

<dependency>

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

<artifactId>spring-boot-starter-validation</artifactId>

</dependency>

=> We can use below annotations to perform server side validations

@Valid

public String handleSubmit(@Valid User user, BindingResult result, Model model)

@NotEmpty

*@NotEmpty*(message="Name is mandatory")

private String name;

@NotNull

If there is long data type instead of using not empty notation we are using @NotNull.

*@NotNull*(message="phno is mandatory")

private Long phno;

@Size

*@Size*(min=3 ,max=12,message="Min 3 and Max 12 chars allowed ")

private String name;

@Email

*@Email*(message="Enter valid email")

private String email;

In thymlef code for display

<td><input type="text" th:field="\*{name}" > </td>

<td><p th:if="${#fields.hasErrors('name')}" th:errorclass="error" th:errors="\*{name}"></p></td>

Note: We will use these annotations at binding class.

@PostMapping("/user")

public String handleSubmit(@Valid User user, BindingResult result, Model model) {

if (result.hasErrors()) {

// validation failed

return "index";

} else {

// validation passed

System.out.println(user);

// logic to save in db

model.addAttribute("msg", "User Saved");

return "index";

}

}

========================================================================

Requirement : Develop spring boot web application to upload and download files.

=======================================================================

======================

Spring Boot with JSP

======================

=> JSP stands for Java Server Pages

=> JSP is used to develop presentation layer

Note: JSP will be translated to Servlet for execution..

=> Spring Web MVC supports JSP as presentation technology.

## Step-1 : Add tomcat-embed-jasper dependency in pom.xml file

<dependency>

<groupId>org.apache.tomcat.embed</groupId>

<artifactId>tomcat-embed-jasper</artifactId>

</dependency>

## Step-2 : Create jsp pages in below location

Location : src/main/webapp/pages/index.jsp

## Step-3 : Configure view resolver in application.properties file

spring.mvc.view.prefix=/pages/

spring.mvc.view.suffix=.jsp

=========================

Actuator in spring Boot

=========================

=> Used to monitor and manage our spring boot applications

=> Production ready features...

=> With the help of actuators we can get below details

- Health of App

- Beans loaded

- Metrics

- Loggers

- URL Mappings

- Config Props

- Thread Dump

- Heap Dump

=> To work with actuators we need to add below dependency

<dependency>

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

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

Note: With above dependency, By default /health will be exposed.

=> We need to write below property to expose other endpoints

management.endpoints.web.exposure.include=\*

=> We can exclude actuator endpoint like below

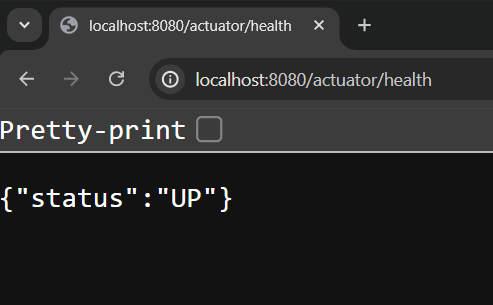
management.endpoints.web.exposure.exclude=beans

==============

Endpoint URLS

==============

/health : health of the app (UP or DOWN)



Its showing up means health of our application is good. By default it provide only health of the app . So, what we have to do get remaining functionality we have to enable them

Go to application.properties

Add-

management.endpoints.web.exposure.include=\*

after this you can see on the console that it will show exposing 14 endpoints beneath base path ‘/actuator’

/beans : Spring Beans loaded

What are the Springbeans are loaded by our application it will display.

/configprops : Properties loaded

It is used to specify what are the properties loaded by our application

/metrics

Metrics are the application that are going to display

/mappings : URL patterns of our application

What are url pattern are available in the application

/threaddump : Threads info

/heapdump : Heap info

/loggers : Logs of our applications

All the logs of our application

/shutdown : Stop server (HTTP POST Request)

=============================

What is shutdown endpoint ?

=============================

=> It is used to stop the application.

Note: We need to enable shutdown endpiont in our properties file like below

management.endpoint.shutdown.enabled=true

Note: Shutdown endpoint is mapped to POST request. We can send post request using POSTMAN software.

===================================

What are Profiles in Spring Boot ?

===================================

=> Environment means a platform which is used to run our application.

=> In Real-time one application contains multiple environments like below

- Local

- Dev

- QA

- UAT

- PILOT

- PROD

-> Local env is used for development purpose

-> DEV env is used by developers for integration testing

-> QA env is used by Testing team for system integration testing

-> UAT env is used by Client side team for testing (GO/No-GO)

-> PILOT env is used to test app with live data (Pre-Prod)

-> PROD env is used for live access.

=> Below properties will be changing from environment to environment.

- datasource properties

- smtp properties

- kafka properties

- redis properties

- payment-gateway properties

=> If we use single application.properties file to maintain config properties then maintanence will become difficult.

Note: to deploy code into env, everytime we have to change config props

=> To avoid this problem we will use Profiles in springboot

=> Using profiles we can maintain environment specific configuration.

application.properties ---- main file

application-dev.properties

application-qa.properties

application-uat.properties

application-prod.properties

=> We need to activate profile in main configuration file

spring.profiles.active=dev

========================================================================

1) Develop Java application to convert java object to json and json data to java object.

2) Develop Java application to convert java object to xml and xml data to java object.

========================================================================