++++++++++++++++++++++++++++++++
SpringSecurity with SpringDataJPA
++++++++++++++++++++++++++++++++
 => There is no direct provision to work with SpringData JPA or Spring ORM based authentication info provider, so we need to implement most of
    the logic manually as we do in other spring boot layer apps by taking seperate repository interfaces, service classes,model classes etc....
      eg: auth.inMemoryAuthentication(), auth.jdbcAuthentication() like this there is no direct template to work with SpringDataJPA(ORM)
            based authentication provider.

=> userregistration page(thymleaf)
                                                BCryptPasswordEncoder
                                                  |
      formpage ----> DS ----> Controller ----pwd----> Service --EncodedPwd----> Repository ------> DB S/w
                        (Security)

1. Create a project using the following starters
      a. thymleaf starter
      b. mysql-driver
      c. springdatajpa
      d. spring web
      e. spring security

Note: From Maven repository download the following dependancy and add it to pom.xml file
      <dependency>
            <groupId>org.thymeleaf.extras</groupId>
            <artifactId>thymeleaf-extras-springsecurity5</artifactId>
      </dependency>

2. Create a model class as shown below

```
@Table(name = "SECURITY_USERS")
@Entity
@Data
public class UserDetails {
      @GeneratedValue(strategy = GenerationType.IDENTITY)
      @Id
      private Integer uid;

      @Column(length = 20, unique = true, nullable = false)
      private String uname;

      @Column(length = 20, unique = true, nullable = false)
      private String pwd;

      @Column(length = 20, unique = true, nullable = false)
      private String email;

      private boolean status = true;

      @ElementCollection(fetch = FetchType.EAGER)
      @CollectionTable(name = "SECURITY_ROLES", joinColumns = @JoinColumn(name = "USER_ID", referencedColumnName = "uid"))
      @Column(name = "role")
```

```
      private Set<String> roles;

}
```

3. Create userregistration.html to take the inputs from the user

```
++++++++++++++++++++
user_registration.html
++++++++++++++++++++
<html xmlns:th="https://thymeleaf.org">

<form th:action="@{/user/register}"  th:object="${userInfo}" method="POST">
    <table border="0" bgcolor="cyan"  align="center">
        <tr>
               <td>  username :: </td>
               <td>  <input type="text" th:field="*{uname}" /> </td>
        </tr>
         <tr>
               <td> password :: </td>
               <td>  <input type="password" th:field="*{pwd}" /> </td>
        </tr>
        <tr>
               <td> email :: </td>
               <td>  <input type="text" th:field="*{email}" /> </td>
        </tr>
        <tr>
               <td> Roles: </td>
               <td>  <input type="checkbox" th:field="*{roles}" value="CUSTOMER"
checked > CUSTOMER
                     <input type="checkbox" th:field="*{roles}" value="MANAGER" >
MANAGER
                </td>
        </tr>
        <tr>
               <td> <input type="submit"  value="register"> </td>
               <td> <input type="reset"  value="cancel"> </td>
        </tr>
    </table>
</form>
```

4. set up the application.properties file with the following key-value pair data

```
++++++++++++++++++++
application.properties
++++++++++++++++++++
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/octbatch
spring.datasource.username=root
spring.datasource.password = root123

spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect
spring.jpa.properties.hibernate.format_sql =true
```

5. Create a ENDPOINTS to collect the value from the user, upon loading the page

```
++++++++++++++++++
UserController.java
++++++++++++++++++
```

```
@Controller
@RequestMapping("/user")
public class UserController {

	@Autowired
	private IUserService service;

	@GetMapping("/register")//for loading the page
	public String showRegistrationPage(@ModelAttribute("userInfo") UserDetails
details) {
		System.out.println("UserController.showRegistrationPage()");
		return "user_register";
	}

	@PostMapping("/register")//for submitting the page with loaded values
	public String registerUser(@ModelAttribute("userInfo") UserDetails details,
Map<String, Object> map) {
		System.out.println("UserController.registerUser()");
		String resultMsg = service.regsiter(details);
		map.put("message", resultMsg);
		return "user_registerd_success";
	}
}
```

6. Create a Service layer as shown below(Provide Encryption also)

```
++++++++++++++++
IUserService.java
++++++++++++++++

package in.ineuron.nitin.service;

import org.springframework.security.core.userdetails.UserDetailsService;

import in.ineuron.nitin.model.UserDetails;

public interface IUserService extends UserDetailsService{
	public String regsiter(UserDetails details);
}

++++++++++++++++++++
UserServiceImpl.java
++++++++++++++++++++

package in.ineuron.nitin.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Service;

import in.ineuron.nitin.repository.IUserDetailsRepo;

@Service("userService")
public class UserServiceImpl implements IUserService {

	@Autowired
```

```
	private IUserDetailsRepo repo;

	@Autowired
	private BCryptPasswordEncoder encoder;

	@Override
	public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
		System.out.println("UserServiceImpl.loadUserByUsername()");
		return null;
	}

	@Override
	public String regsiter(in.ineuron.nitin.model.UserDetails details) {
		System.out.println("UserServiceImpl.regsiter()");

		details.setPwd(encoder.encode(details.getPwd()));

		return repo.save(details).getUid() + " UserId is registered";
	}

}

7. Create html to display the success data
++++++++++++++++++++++++++++
user_registered_success.html
++++++++++++++++++++++++++++
<html xmlns:th="https://thymeleaf.org">
<b>hello</b>
<h1 style="color:green;text-align:center"><span th:text="${message}" /> </h1>
<br>
<b><a th:href="@{/user/register}">Add another user</a></b>


++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++
Adding the custom login page to the project to perform authentication and
authorization using SpringDataJPA
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++

Control flow
  a. POST + /login
        => request springboot security app
        => Security Config class sends the data to loadUserByUsername(String
username) method of UserDetailsService class.


1. Different URLS
	/login + GET  => Default authentication form based for authentication login
page.
	/login + POST => To process default form based authentication login page
submission

2. Adding handlerMethods to process the custom login page
   @RequestMapping("/showLogin")
   public String showLoginPage() {
      return "custom_login";
   }
```

JWT and Gmail,Facebook authentication

Sunday-> Junit,HttpUnit and mockito