=====================================
How to secure REST APIs using Spring Boot
=====================================

-> Security is very important for every web application

-> To protect our application & application data we need to implement security logic

-> Spring Security concept we can use to secure our web applications / REST APIs

-> To secure our spring boot application we need to add below starter in pom.xml file

```
            <dependency>
                  <groupId>org.springframework.boot</groupId>
                  <artifactId>spring-boot-starter-security</artifactId>
            </dependency>
```

Note: When we add this dependency in pom.xml file then by default our application will be secured with basic authentication. It will generate random password to access our application.

Note: Generated Random Password will be printed on console.

-> We need to use below credentials to access our application

Username : user

Password : <copy the pwd from console>

-> When we access our application url in browser then it will display "Login Form" to authenticate our request.

-> To access secured REST API from postman, we need to set Auth values in POSTMAN to send the request

===
How to override Spring Security Random Password
===

-> To override random credentials we can configre security credentials in application.properties file or application.yml file like

```
spring.security.user.name=ashokit
spring.security.user.password=ashokit@123
```

-> After configuring credentials like above, we need to give above credentials to access our application / api.

===================================
How to secure specific URL Patterns
==================================

-> When we add 'security-starter' in pom.xml then it will apply security filter for all the HTTP methods of our application.

-> But in reality we need to secure only few methods not all methods

```
            For Example

                        / login-page --> security not required

                        / transfer ---> security required

                        / balance ---> security required

                        /about-us ---> security not required
```

-> In order to achieve above requirement we need to Customize Security Configuration in our project like below

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

      @Bean
      public SecurityFilterChain securityFilter(HttpSecurity http) throws 
Exception{

            http.authorizeHttpRequests((request) -> request
                        .antMatchers("/","/login","/about", "/swagger-
ui.html").permitAll()
                        .anyRequest().authenticated()
            ).formLogin();

            return http.build();
      }
}
```

===
Spring Boot Security with JDBC Authentication
===

Step-1) Setup Database tables with required data

```
-- users table structure

CREATE TABLE `users` (
  `username` VARCHAR(50) NOT NULL,
  `password` VARCHAR(120) NOT NULL,
  `enabled` TINYINT(1) NOT NULL,
  PRIMARY KEY (`username`)
);
```

```
-- authorities table structure

CREATE TABLE `authorities` (
  `username` VARCHAR(50) NOT NULL,
  `authority` VARCHAR(50) NOT NULL,
  KEY `username` (`username`),
  CONSTRAINT `authorities_ibfk_1` FOREIGN KEY (`username`)
  REFERENCES `users` (`username`)
);
```

==================== Online Encrypt : https://bcrypt-generator.com/
==============================

```
-- insert records into table

insert into users values ('admin',
'$2a$12$0l.1TapVc7dR9mRwoCuWCO3GP4ekxrmfvtYVxx8VhXRbOznIrwNfu',  1);
insert into users values ('user',
'$2a$12$mZlgVUBTMMfZKQNhxvq4PO1u7syQ40RkVUzCDSSbwJmEr09KcJybW',  1);

insert into authorities values ('admin', 'ROLE_ADMIN');
insert into authorities values ('admin', 'ROLE_USER');
insert into authorities values ('user', 'ROLE_USER');
```

Step-2) Create Boot application with below dependencies

a) web-starter
b) security-starter
c) data-jdbc
d) mysql-connector
e) lombok
f) devtools

Step-3) Configure Data source properties in application.properties file

```
#MySQL database connection strings
spring.datasource.url=jdbc:mysql://localhost:3306/jrtp
spring.datasource.username=root
spring.datasource.password=root
```

Step-4) Create Rest Controller with Required methods

```
@RestController
public class UserRestController {

      @GetMapping(value = "/admin")
      public String admin() {
            return "<h3>Welcome Admin :)</h3>";
      }

      @GetMapping(value = "/user")
      public String user() {
            return "<h3>Hello User :)</h3>";
```

```
	}

	@GetMapping(value = "/")
	public String welcome() {
		return "<h3>Welcome :)</h3>";
	}

}
```

Step-5) Create Security Configuration class like below with Jdbc Authentication Manager

```
package in.ashokit;

import javax.sql.DataSource;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.authentication.builders.Authenticati
onManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;

@Configuration
@EnableWebSecurity
public class SecurityConfiguration {

	private static final String ADMIN = "ADMIN";
	private static final String USER = "USER";

	@Autowired
	private DataSource dataSource;


	@Autowired
	public void authManager(AuthenticationManagerBuilder auth) throws Exception {
	    auth.jdbcAuthentication()
	            .dataSource(dataSource)
	            .passwordEncoder(new BCryptPasswordEncoder())
	            .usersByUsernameQuery("select username,password,enabled from
users where username=?")
	            .authoritiesByUsernameQuery("select username,authority from
authorities where username=?");
	}

	@Bean
	public SecurityFilterChain securityConfig(HttpSecurity http) throws Exception
{

		http.authorizeHttpRequests( (req) -> req
				.antMatchers("/admin").hasRole(ADMIN)
				.antMatchers("/user").hasAnyRole(ADMIN,USER)
```

```
                        .antMatchers("/").permitAll()
                        .anyRequest().authenticated()
            ).formLogin();

            return http.build();
      }

}
```

```
===========
OAuth 2.0
===========
```

1) Create Spring Boot application with below dependencies

```
<dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-oauth2-client</artifactId>
</dependency>
<dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

2) Create OAuth app in Github.com

(Login --> Setting --> Developer Settings --> OAuth Apps --> Create App --> Copy Client ID & Secret)

3) Configure GitHub OAuth App client id & client secret in application.yml file like below

```
spring:
  security:
    oauth2:
      client:
        registration:
          github:
            clientId: <id>
            clientSecret: <secret>
```

4) Create Rest Controller with method

```
@RestController
public class WelcomeRestController {

      @GetMapping("/")
      public String welcome() {
            return "Welcome to Ashok IT";
      }
}
```

5) Run the application and test it.